# COST-EFFECTIVE CLOUD DATA PROCESSING

By

Willis Lang

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy
(Computer Sciences)

at the
UNIVERSITY OF WISCONSIN-MADISON
2012

Date of final oral examination: July 9, 2012
The dissertation is approved by the following members of the Final Oral Committee:
Jignesh M. Patel, Professor, Computer Sciences
Jeffrey F. Naughton, Professor, Computer Sciences
Anhai Doan, Associate Professor, Computer Sciences
Christopher Ré, Assistant Professor, Computer Sciences
Jonathan Eckhardt, Associate Professor, Management and Human Resources

# Dedication

*For my parents, Ching-Ling and Walter Lang*

## Acknowledgements

I'd like to thank three groups of people that helped make this journey possible: my advisor, Professor Jignesh M. Patel; my family; and the committee and various friends/collaborators that I've gotten to know along the way.

Jignesh's instruction, enthusiasm, and friendship have arguably been the sole reason that this body of work exists today. From the first day he took me on as a new graduate student at Michigan, to the conclusion of my graduate studies at Wisconsin, and continuing as working colleagues into the future, I have enjoyed and look forward to continue working with Jignesh. Through the numerous trials, successes, and failures, I consider myself lucky that Jignesh has been there as a mentor and companion on this journey.

Throughout my life, my family has provided the love, support, and advice that helped make sure I was able to cross the finish line in this marathon. My parents, I cannot describe in words the sacrifices that they have made throughout my life to give me all the opportunities possible. My brother Josh has always been there with help at a moments notice. My cousin Q and my aunt and uncle have provided invaluable support and advice. Finally, Jane, my love, thank you for being by my side through it all.

David, Jeff, Chris, Anhai, and Jag, thank you for all your mentoring and guidance. I've gotten to know so many friends throughout the years who have helped me through this journey. GSL and HP Labs friends and collaborators: Srinath, Nikhil, Eric, Alan, Rimma, Dimitris, Stavros, and Mehul. Friends from Wisconsin: Avrilia, Ian, Jessie, Jae, Spyros, and RK. And finally, fellow dbslaves from Michigan: Arnab, Dan, YY, Cong, Bin, YunYao, Sandeep, and Mike, it was indeed a special group.

# Table of Contents

# List of Tables

**Table**

# List of Figures

**Figure**

Lang, Willis (Ph.D., Computer Science)

Cost-Effective Cloud Data Processing

Thesis directed by Professor, Computer Sciences Jignesh M. Patel

We are headed towards an increasingly data-driven society due to the vast amounts of data that are both more readily available, and more valuable than ever. Consumer and enterprise services such as social networking sites and traditional retailers are working together to gather and use data to increase the efficiency and effectiveness of their business operations. Additionally, the health-care industry is also taking advantage of readily available research, drug, and treatment data to increase both the efficiency of services and the quality of care for patients. In short, the value of "big data" analytics is now widely recognized across all sectors of society.

The IT industry and academic researchers have raced to develop new systems that enable the extraction of insights from vast data repositories. These systems are being built and run in (public and private) "cloud" clusters and housed in large data warehouses. Unfortunately, to-date, little attention has been paid to the rising costs of deploying and running such systems ($10M-100Ms per data warehouse). Two major cost components in operating such big data systems are: a) the cost to purchase the computing hardware and b) the cost to power the hardware. Together, these two components make up roughly 88% of the monthly ownership cost.

This dissertation provides a comprehensive look at the relationship between the following three factors: (i) the performance of the data processing system; (ii) the allocation of hardware resources; and (iii) the energy consumption of the system. We ask questions like: "How can we meet user performance targets while minimizing the hardware provisioning costs?" and, "How much does the latency and energy consumption of a database query change if we decrease the computing resources that are assigned to process that query?" The key challenges involve modeling the relationships between performance, hardware, and energy, as well as developing frameworks to effectively trade-off one for another. We find that we can modulate the performance and hardware/energy costs of data processing in a controlled way by changing the way

that the software uses its hardware resources, and/or by changing the way we build our servers/clusters for data-processing systems. The main contributions of this dissertation involve the identification, formulation, and evaluation of models and frameworks for desirable trade-offs between hardware/energy costs and data processing performance. The implications of this thesis is that the methods presented here can be used to reduce the overall dollar cost of running big data analysis in the cloud, while meeting any applicable workload performance targets.

# Chapter 1

# Introduction

Many aspects of society are increasingly data driven as nearly every facet of both enterprise and consumer services rely on data processing and analysis. For example, in the consumer services space, large social networks like Facebook are able to generate 30 billion pieces of information a month. By arming themselves with consumer information, it has been estimated that retailers in the private sector may be able to raise their operating margins by 60%. The public sector can also benefit from this shift; for example, by leveraging health-care data, it has been estimated that the national health-care expenditure can be reduced by 8% [119].

To maintain acceptable levels of performance in this "Big Data"-era, practitioners of Big Data analytics have rushed to develop larger and more powerful data-processing systems. Typically, these data-processing systems require large clusters of servers that are deployed in a "cloud" data center (public or private[1] ). As the number of servers in a data-processing cluster increase, this has a direct effect on the total cost of ownership (TCO) of the cloud cluster. Furthermore, with perpetually rising energy costs, using more hardware for analysis amplifies today's increasing energy bills. Figure 1.1 presents a recently published cost breakdown of a modern 46,000 server Amazon datacenter [81]. In this breakdown, the four major cost components presented are: the 36 month amortized server costs, direct energy and power-related infrastructure costs, networking infrastructure costs, and other miscellaneous infrastructure costs. From Figure 1.1, it is clear that the two largest cost components are the server (which excludes network switches and networking infrastructure) costs and energy costs. These two costs make up 88% of the monthly costs, almost

[1] Public clouds refer to hosted clusters shared by customers and open to the public. Private clouds refer to clusters not generally available to the public and generally are owned and used by the same institution/corporate entity.

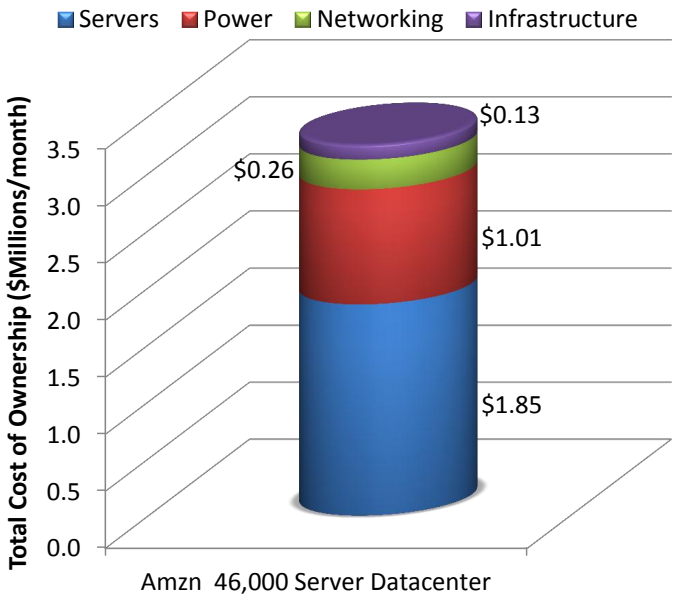



Figure 1.1: Four major cost components of a modern datacenter: server costs amortized over 36mo., direct energy and power-related infrastructure costs, networking infrastructure, and other miscellaneous infrastructure costs [81]

$3M/month, in a modern 46,000 server data center [81]. It is for this reason that this thesis targets server and energy costs for optimization.

Consequently, from the observations from Figure 1.1, IT professionals that manage data-processing infrastructure are faced with two optimization goals: (1) Keep up with the increasing load on their infrastructure and provide acceptable levels of data-processing performance; and (2) Maintain low costs by prudently deploying computing software and infrastructure as well as increasing the system energy efficiency. *The broad goals of this dissertation are to understand how the two main cost components of data processing, server hardware and energy, contribute to performance, and then formulating and evaluating models and frameworks to trade performance for lower costs in a controlled manner.*

## 1.1 Key Challenges

The key challenge when optimizing for performance and cost is that they are interconnected, often in a complex way. Traditionally, given a data processing system and increasing amounts of data, the simplest means of maintaining throughput or latency performance has been to buy new/more hardware. Such a

solution increases costs through more server hardware as well as the corresponding increase in the power footprint of the IT infrastructure. This dissertation considers how performance is affected by both sources of cost: server costs and the energy costs of data processing.

### 1.1.1 Cost-effective Server Provisioning

One question we want to address for large-scale data processing environments is how IT managers should configure and purchase servers for their clusters while adhering to their users' performance requirements. As mentioned above, one straightforward approach to maintaining user performance with increasing data volume (or user load) is to simply purchase more hardware. However, there are two approaches to managing this cost: (1) Determining the minimum cluster resources that must be deployed for the user load; and/or (2) Configuring the server components such that we can try to minimize the purchase cost while delivering performance. Both of these approaches come with considerable challenges. For instance, in the first approach, how do we characterize the users' workloads and performance objectives and then formulate a cost optimization? With the second approach, using less expensive servers may require a larger cluster deployment which, for some workloads, suffers from diminishing returns in performance. This dissertation addresses some of the key challenges associated with cost/performance trade-offs.

### 1.1.2 Energy-aware Data Processing

The biggest change in the industry-standard, Transaction Processing Council (TPC) benchmarks in over two decades is now well underway – namely the addition of an energy-efficiency metric along with traditional performance metrics [167]. Managing the energy consumption of data processing infrastructure has now become necessary because of numerous factors like: (1) In a modern data center, the cost of electricity has already surpassed $1M a month in some cases, and is expected to equal if not dominate monthly amortized server costs in the near future [81, 141]; (2) Future CPUs are likely to require that software systems work within a tight power budget [31]; and (3) Emerging legislative conservation policies may limit the energy consumption of data centers [27]. In the same way that energy efficiency forced a radical shift in automotive design, we are at the cusp of shifting toward energy-efficient designs of next

generation data processing infrastructure [59, 96].

Unfortunately, reducing the energy consumption of the system when running a data processing task is typically accompanied by a drop in performance. In most environments, performance has been, and will continue to be, a first-class optimization goal. Luckily, we can take advantage of two key facts to tackle this problem. First, data processing systems are typically provisioned for peak load, which, on average delivers higher performance than what the user needs. Secondly, hardware manufacturers as well as the architecture and systems research communities have been studying and developing various types of power/performance trade-off mechanisms that are accessible to software systems [121]. Thus, we have a trade-off opportunity to use various hardware and software techniques to reduce performance to user-acceptable levels in controlled way so that we can minimize the energy consumption (while meeting performance targets). However, a lack of understanding of the relationship between energy consumption and performance is the first step that must be addressed before pursuing any trade-off solutions.

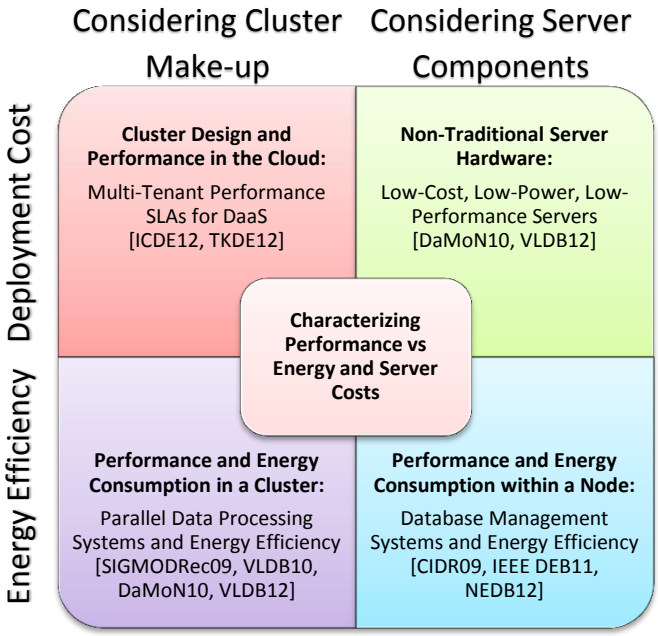



Figure 1.2: A breakdown of the approaches taken to tackle balancing data processing cost with performance.

**Summary:** With these key challenges of cost/performance and energy/performance trade-offs, the work in this dissertation can be roughly partitioned along these two categories – *server costs* versus *energy*

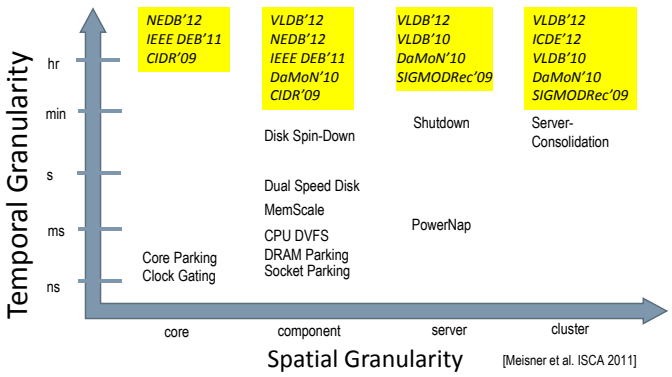



Figure 1.3: Current and future hardware mechanisms available to data processing systems [121]. Plots the temporal granularity at the mechanism can be switched and the spatial granularity of the target of the mechanism. The work presented in this dissertation covers the entire space of available mechanisms.

*costs*. Further, we can also consider data processing in a *single server* (the traditional DBMS environments) versus *scale-out, shared-nothing parallel data processing* in clusters. These dimensions are visualized in Figure 1.2 which also provides an overview of the technical contributions of this dissertation.

## 1.2 Overview of Technical Contributions of this Dissertation

In Figure 1.2, the four broad areas of focus covered by this dissertation are illustrated in a quad chart. The two dimensions of this quad chart are: (1) trading performance for lower server cost or higher energy efficiency, and (2) examining trade-off opportunities in a cluster environment or within a single server.

To explore trading off performance at the cluster and server level, we leverage power/performance and cost/performance hardware mechanisms at various spatial granularities. This is shown in Figure 1.3 which is a map of the hardware mechanisms exploitable by data processing systems [121]. This map shows that there are mechanisms that target server components, all the way to cluster-level mechanisms. We can see that as the spatial granularity increases, so does the mechanism switching time (plotted on the y-axis). For example, as we have shown in Figure 1.3, parking a processing core of a CPU is a mechanism that operates at around the micro-second time scale, while shutting down a server can take up to a minute. This increase in the mechanism's switching time affects the performance trade-off capability because the hardware mechanism can significantly reduce data processing performance. This dissertation presents work that covers the entire

space of these available mechanisms.

Below is a brief overview of the work covered by each of the quadrants of Figure 1.2 and the mechanism space shown in Figure 1.3.

### 1.2.1 Cost-effective Cluster Design and Performance Objectives in the Cloud

The focus of this work is on tackling the deployment cost problem by examining how to provision and use cluster resources efficiently in a multi-tenant cloud environment, represented by the upper-left quadrant of Figure 1.2. The "mechanism" that we employ here can be categorized as cluster-level management and falls in the "cluster" column of Figure 1.3. As traditional and mission-critical relational database workloads migrate to the cloud in the form of Database-as-a-Service (DaaS), there is an increasing motivation to provide performance goals in Service Level Objectives (SLOs) [12]. Providing such performance goals is challenging for DaaS providers as they must balance the performance that they can deliver to tenants against the data center's operating costs [40]. In general, aggressively aggregating tenants on each server reduces the operating costs but degrades performance for the tenants, and vice versa. This work presents a framework that takes as input the tenant workloads, their performance SLOs, and the server hardware that is available to the DaaS provider, and outputs a cost-effective recipe that specifies how much hardware to provision and how to schedule the tenants on each hardware resource. An evaluation of the proposed method shows that it produces effective solutions that can reduce the costs for the DaaS provider while meeting performance goals.

This work was accepted into the Proceedings of the IEEE International Conference on Data Engineering (ICDE), 2012 [109], as well as invited for submission for a special issue (Best Papers of ICDE) of the IEEE Transactions on Knowledge and Data Engineering (TKDE) journal [110]. The content of this work can be found in Chapter 2 of this dissertation.

### 1.2.2 The Costs of Using Non-Traditional Server Hardware

This work examines the impact of changing the configuration of server nodes in a cluster on the price/performance of parallel database analytics workloads, represented by the upper-right quadrant of Fig-

ure 1.2. The work here considers using different components, servers, and ways to provision clusters; it covers the corresponding three columns of Figure 1.3. The high cost associated with powering servers has introduced new challenges in improving the energy efficiency of clusters running data processing jobs. Traditional high-performance servers are largely energy inefficient due to various factors such as the over-provisioning of resources. There has been an increasing movement to replace traditional high-performance server nodes with low-power low-end nodes in clusters. This approach has recently been touted as a solution to the cluster cost/energy problem [13, 145, 174]. However, the key tacit assumption that drives such a solution is that the scale-out of such low-power cluster nodes results in a constant scaleup in performance. Here, we challenge the validity of such an assumption using measured price and performance results from a low-power Atom-based node and a traditional Xeon-based server and a number of published parallel scaleup results. Our results show that in most cases, computationally complex queries exhibit disproportionate scaleup characteristics which potentially makes scale-out with low-end nodes an expensive and lower performance solution.

This work was accepted into the Proceedings of the International Workshop on Data Management on New Hardware (DaMoN), 2010 and awarded best paper [108]. The content of this work can be found in Chapter 3 of this dissertation.

While some of the results regarding costs are dependent on the current market for commodity hardware, our results in this chapter focus on the impact of poor scalability in traditional database analytics tasks. Our energy efficiency results in this work show that despite the poor scalability of large wimpy node clusters, in certain cases, they can still be markedly more energy efficient than clusters of traditional high-performance servers. This provides our segue into the discussions on the energy efficiency of data processing.

### 1.2.3 Energy-aware Parallel Data Processing

This portion of the dissertation investigates some opportunities and challenges that arise in energy-aware computing in a cluster of servers running data-intensive workloads – this covers the bottom-left quadrant of Figure 1.2. The operational mechanisms we consider here range from cluster-level, to server-level, as

well as using different components in the server. Thus, the following three topics cover the three right-most columns of Figure 1.3.

***On the Energy-efficient Design of Clusters***

Although a number of recent studies have investigated the energy efficiency of DBMSs [140,141,171, 181], none of these studies have looked at the architectural design space of energy-efficient parallel DBMS clusters. There are many challenges to increasing the energy efficiency of a DBMS cluster, including dealing with the inherent scaling inefficiency of parallel data processing, and choosing the appropriate energy-efficient hardware. In this discussion, we experimentally examine and analyze a number of key parameters related to these challenges for designing energy-efficient database clusters. We explore the cluster design space using empirical results and propose a model that considers the key bottlenecks to energy efficiency in a parallel DBMS. This study represents a key first step in designing energy-efficient database clusters, which is increasingly important given the trend toward custom parallel database appliances.

This work was accepted into the Proceedings of the International Conference on Very Large Data Bases (VLDB), 2012 [101] The content of this work can be found in Chapter 4 of this dissertation.

***On the Management of MapReduce Clusters***

Since the area of cluster-level energy management has attracted significant research attention over the past few years, a number of broad classes of techniques have been studied [39, 112, 165]. One class of techniques to reduce the energy consumption of clusters is to selectively power down nodes during periods of low utilization to increase energy efficiency. One can think of a number of ways of selectively powering down nodes, each with varying impact on the workload response time and overall energy consumption. Since the MapReduce framework is becoming "ubiquitous", the focus of this discussion is on developing a framework for systematically considering various MapReduce node power down strategies, and their impact on the overall energy consumption and workload response time. (The methods examined here, however, can also be applied to other distributed computing frameworks.)

This discussion closely examined two extreme techniques that can be accommodated in this framework. The first is based on a recently proposed technique called "Covering Set" (CS) that keeps only a small fraction of the nodes powered up during periods of low utilization [112]. At the other extreme is a technique

that is proposed in this study, called the All-In Strategy (AIS). AIS uses all the nodes in the cluster to run a workload and then powers down the entire cluster. Using both actual evaluation and analytical modeling, this work brings out the differences between these two extreme techniques and we show the scenarios under which AIS (or CS) is the right energy-saving strategy.

This work was accepted into the Proceedings of the International Conference on Very Large Data Bases (VLDB), 2010 [104] The content of this work can be found in Chapter 5 of this dissertation.

***Exploiting Data Replication Schemes for Energy Efficiency***

Drawing on the insight that servers in a cluster are often underutilized [140], this makes it attractive to consider powering down some servers and redistributing their load to others. Of course, powering down servers naïvely will render data stored only on the powered-down servers inaccessible. While data replication can be exploited to power down servers without losing access to data, unfortunately, care must be taken in the design of the replication and server power-down schemes to avoid creating load imbalances on the remaining "live" servers. Accordingly, this work analyzes the interaction between energy management, load balancing, and replication strategies for data-intensive cluster computing. In particular, it is shown that Chained Declustering [90] – a replication strategy proposed more than 20 years ago – can support very flexible energy management schemes.

This work was accepted for publication into an issue of the SIGMOD Record, 2009 [106] and was also released as an extended University of Wisconsin-Madison technical report [107]. The content of this work can be found in Chapter 6 of this dissertation.

### 1.2.4 Energy-aware Database Management Systems

As we have discussed, there is a growing, real, and urgent demand for energy-efficient database processing. Database query processing engines must now consider becoming energy-aware, else they risk missing many opportunities for significant energy savings. While other recent work has focused on solely optimizing for energy efficiency [171, 181], such methods are only practical if they also consider performance requirements specified in SLAs. The focus of this discussion is on the design and evaluation of a general framework for query optimization that considers both performance constraints *and* energy consump-

tion as first-class optimization criteria. This framework recognizes and exploits the evolution of modern computing hardware that allows hardware components to operate in different energy and performance states (Figure 1.3) [121]. The optimization framework considers these states and uses an energy consumption model for database query operations. An energy model was also built for an actual commercial DBMS. Using this model the query optimizer can pick query plans that meet traditional performance goals (e.g., specified by SLAs), but result in lower energy consumption. Through numerous experimental evaluations, it is shown that the system-wide energy savings can be significant and point toward greater opportunities with upcoming energy-aware technologies on the horizon. While the discussion will focus on a single-server DBMS environment (bottom-right quadrant of Figure 1.2), this framework is generic and can also apply to distributed parallel data processing environments.

This work is a compilation of a number of publications and presentations: the Proceedings of the Conference on Innovative Data Systems Research (CIDR), 2009 [103] an issue of the IEEE Data Engineering Bulletin, 2011 [102] and presented at the New England Database Summit (NEDS), 2012 [105]. These works can be found in Chapter 7 of this dissertation.

## 1.3 Summary

This dissertation tackles the problem of data processing cost through two main avenues of: (1) **managing server deployment costs**, and (2) **managing the energy consumption of data processing systems**. For both of these avenues of cost management, this dissertation further breaks down the categories into two approaches: (i) **cluster-centric solutions**, and (ii) **server-centric solutions**. *The over-arching challenge is to be able to balance potential trade-offs of performance for lower costs in a controllable, predictable, and significant manner.* In all four combinations of approaches, analysis of the performance/cost relationship is studied to develop optimization and/or operational frameworks for trading performance for lower server deployment cost and lower energy cost.

# Chapter 2

## Cost-effective Cluster Provisioning and Performance Objectives in the Cloud

Traditional relational database workloads are quickly moving to the cloud in the form of Database-as-a-Service (DaaS). Such cloud deployments are projected to surpass the "on-premises" market by 2014 [146]. As this move to the cloud accelerates, increasing numbers of mission-critical workloads will also move to the cloud, and in turn will demand that the cloud service provider furnish some assurances on meeting certain quality-of-service metrics. Some of these metrics, such as uptime/availability, have been widely adopted by DaaS providers as Service Level Objective (SLOs) [12, 182]. (SLOs are specific objectives that are specified in the encompassing Service Level Agreement – SLA.) Unfortunately, performance-based SLOs have still not been widely adopted in DaaS SLAs. Performance-based SLOs have been proposed in other (non-DaaS) cloud settings [85], and in the near future it is likely that DaaS users will demand these SLOs (especially if they are running mission-critical database applications that require a certain level of performance). DaaS providers may also provide performance-based SLOs as a way to differentiate their services from their competitors.

DaaS providers want to promise high performance to their tenants, but this goal can often conflict with the goal of minimizing the overall operating costs. Data centers that house database services can have high fixed monthly costs that impact the DaaS providers' bottom line [38, 80]. For a DaaS provider, servicing the same tenants with fewer servers decreases the amortized monthly costs [158]. Hence, consolidation via *multi-tenancy* (where multiple database tenants are run on the same physical server) is a straight-forward way to increase the cost-effectiveness of the DaaS deployment.

In a traditional single tenant database setting, two key factors that determine performance are: a) The

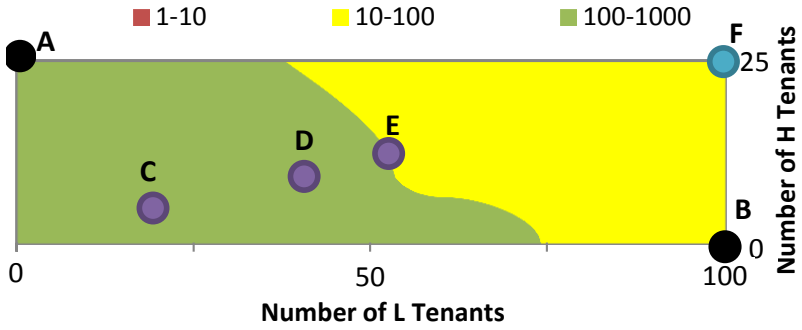
workload characteristics; and b) The server hardware on which the database management system (DBMS) is being run. In a multi-tenant setting, the degree of multi-tenancy becomes an additional factor that impacts performance, both for the overall system and the performance that is experienced by each individual tenant. In general, increasing the degree of multi-tenancy decreases per-tenant performance, but reduces the overall operating cost for the DaaS provider.

Hence, the important question for a DaaS provider is how to balance multi-tenancy with performance-based SLOs. This chapter focuses on posing this question and presenting an initial answer. There are many open questions that need to be answered beyond the work here, which points to a rich direction of future work (see Section 9.2, Chapter 9).

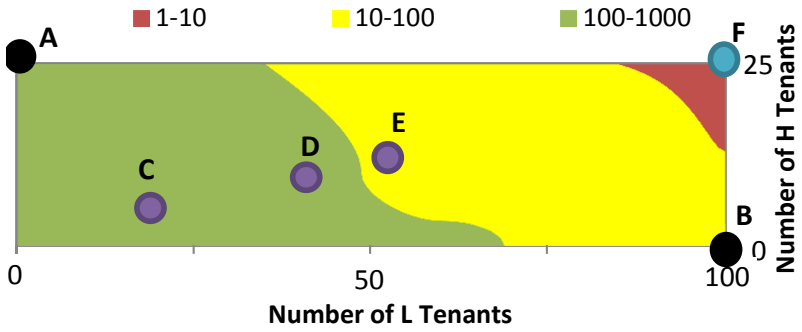
In this study, we propose a general DaaS provisioning and scheduling framework that optimizes for operating costs while adhering to desired performance-based SLOs. Developing a framework to optimize DBMS clusters for performance-based SLOs is challenging because of a number of specific issues, namely: (a) The DaaS provider may have a number of different hardware SKUs (Stock Keeping Units) to choose from, and needs to know how many machines of each SKU to provision for a given set of tenants – thus the provider needs *a hardware provisioning policy*; and (b) The DaaS provider also needs to know an efficient mapping of the tenants to the provisioned SKUs that meets the SLOs for each tenant while minimizing the overall cost of provisioning the SKUs – thus the DaaS provider needs a *tenant scheduling policy*. Note that the tenants on the same server may have different performance requirements, and the tenants may interfere with each other, making the mapping of tenants to the SKUs challenging.

## 2.1 Motivating Illustrative Experiments

Let us consider a concrete example to illustrate these issues. Assume that a DaaS provider has many tenants that have workloads that are like TPC-C scale factor 10. The performance metric that is of interest here is transactions per second (tps). Assume that the DaaS provider has 10,000 tenants split into two classes: '**H**' and '**L**'. The tenants in the **H** class are associated with a high performance SLO of 100 tps, whereas the tenants in the **L** class are associated with a lower performance SLO of 10 tps (and presumably a lower price). Assume that 20% of the tenants (2000 tenants) belong to the class H and the remaining (8000




(a) Performance (tps) for class H tenants




(b) Performance (tps) for class L tenants

Figure 2.1: Average performance seen by tenants in class H (100tps) and class L (10tps) on TPC-C scale factor 10 database, as the tenant mix is varied. In both figures, circles annotated with the same letter correspond to the same operating point.

tenants) belong to the class L. For this example, imagine that there is only one SKU, and assume that all the tenants have the same query workload characteristics (i.e., all tenants have the same query workload, and issue queries to the server with the same frequency).

To find a hardware provisioning policy and the associated tenant scheduling policy, we first need to understand how the performance of the tenants in class H (and class L) changes for a workload that consists of a mix of these tenants. In other words, we need to characterize the performance that *each* tenant sees for varying mixes of tenants from the two classes, when these tenants are scheduled on the same server. We capture this performance trait in a *SKU performance characterizing model*.

To produce the SKU performance characterizing model, we first benchmark the server SKU for a *homogeneous* mix of tenants. This benchmark shows that we can accommodate around 25 tenants of class H (100 tps). Scheduling more than 25 tenants results in the tps dropping below 100 tps, and hence breaks

the performance SLO. Similarly, we find that this SKU can accommodate up to 100 tenants of class L (10 tps). Points A and B in Figure 2.1 correspond to the findings from this homogeneous benchmark. (Below we describe what Figure 2.1 shows in more detail.)

The homogeneous benchmark above defines the boundaries of how many tenants of each class we can pack on a given server. Next, we need to characterize the space to allow for an arbitrary mix of tenants. We note that while it is possible that an optimal hardware provisioning policy and associated tenant scheduling policy could only have SKUs with homogeneous tenants (i.e., no SKU has a mix of tenants from the two classes), it is also possible that the optimal policy has a mix of tenants from the two classes on some or all the SKUs. This may be the case if different tenant workloads have different resource utilizations (memory vs. disk vs. CPU) on a SKU. Thus, the SKU performance characterizing model must also consider *heterogeneous* mixes of tenants.

To complete the SKU performance characterizing model, we need to benchmark the server for varying mixes of tenants from the two performance classes, and measure the throughput that each tenant in each class sees. Figure 2.1 shows the SKU performance characterizing model for an actual SSD-based server SKU using experimental results for the 100 tps and the 10 tps TPC-C tenant classes. (See Section 5.2 for details.)

In Figure 2.1, the performance of the class H tenants is shown in Figure 2.1(a), while the performance that the class L tenants experience is shown in Figure 2.1(b).

First, consider a homogeneous tenant scheduling policy that uses only the points A (25 100tps tenants), and B (100 10tps tenants). In this case, the DaaS provider needs to provision 160 SSD-based servers for the 10,000 tenants (80 for the H class tenants, and 80 for the L class tenants).

But, could we do better than using a homogeneous tenant scheduling policy? To answer this question, we need to systematically explore the entire space of tenant workload mixes, and the associated hardware provisioning (to compute the operating cost). Essentially, we need to explore the entire space shown in Figure 2.1. Note that some of the points in this space are not feasible "solutions", as they violate the performance SLOs. For example, at the operating point F in Figure 2.1(a), the H class tenants see a performance level that is below 100 tps, since the point F is in the yellow zone that corresponds to 10-100 tps. In Fig-

ure 2.1(b), at point F, the L class tenants do not reach a satisfactory performance either.

On the other hand, in Figure 2.1, the operating points C, D, and E are all feasible, but they result in different hardware provisioning policies, which in turn impacts the overall operating costs. In this case, the operating point E is the most cost-effective of these three operating points, because it only requires 143 SKUs (14 H tenants and 56 L tenants per SKU). In contrast, the operating point D (10 H and 40 L tenants per SKU) and the operating point C (5 H and 20 L tenants per SKU) require 200 and 400 SKUs respectively. Notice that the policy from point E results in 17 fewer servers required than the homogeneous policy from point A and B.

The problem illustrated above becomes even more complicated if the DaaS provider has a mix of SKUs to choose from. In this case, assume that the DaaS provider has another SKU that is less expensive, but has lower overall performance on this workload. In this case, the DaaS provider needs to consider the cost ratio between the two different SKUs and the relative performance differences, and provision hardware that reduces the overall operating cost. Note that the lowest cost feasible operating point could involve deploying a mix of the two (or, in general, more) SKUs, as shown by various examples in Section 2.4. Thus, the overall optimization problem involves finding a mix of SKUs to deploy for a given set of tenants belonging to different performance-based SLO classes, along with a tenant scheduling policy for each deployed SKU. Here, we present and evaluate a solution to this problem.

## 2.2 Technical Contributions

The above example not only highlights the difficult optimization problem that our framework tackles but also the two component solution: the output of our framework is an **SLO compliant tenant scheduling policy** and a **cost minimizing hardware provisioning policy**. In this chapter, we fully define the optimization framework to provide these provisioning and scheduling solutions, discuss how to solve the optimization problem, and show interesting policies that arise from empirical results.

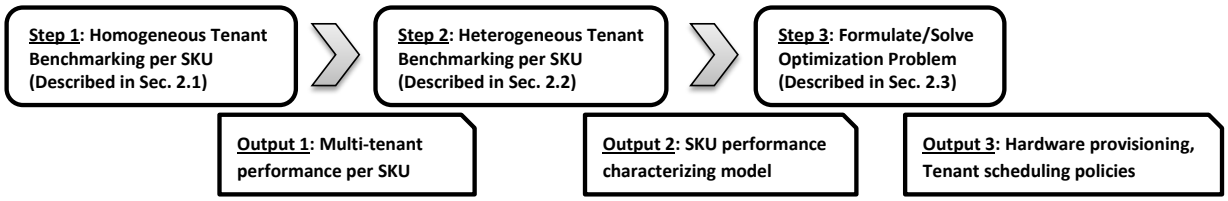



Figure 2.2: A work-flow diagram for using our cost-optimization framework.

## 2.3 Performance SLO Framework

In this section, we describe our optimization framework, which has three steps as shown in Figure 2.2. Recall that the goal of this framework is to provide hardware provisioning and tenant scheduling policies that minimize the costs to DaaS providers while satisfying the performance-related specifications in tenant SLOs.

In the first step in Figure 2.2 (described in Section 2.3.1), we benchmark the performance of each server SKU in a *homogeneous* multi-tenant environment. At the end of this step, we understand the tenant performance for each tenant class on each hardware SKU, producing Output 1 in Figure 2.2. From this first step, for a specific performance level, we can determine the maximum number of tenants of a given class that can be scheduled on a specific server SKU, such that the performance SLOs can be satisfied for each tenant. Essentially, in this step we find points like A and B in Figure 2.1 for every tenant class for every hardware SKU.

The next step, marked as Step 2 in Figure 2.2, uses Output 1 to compute the boundaries of the space of mixed class workloads that should be considered. Then, for each hardware SKU this space is characterized by running actual benchmarks. In other words, Step 2 computes Figure 2.1 for every hardware SKU as Output 2. Now, we understand the impact of scheduling a workload with tenants that have different SLO requirements on the same server box. This step is discussed in more detail in Section 2.3.2.

The last step in Figure 2.2 takes as input the set of SKU performance characterizing models (i.e., Output 2) and computes an optimal strategy to deploy the workload. This step uses an optimization method that takes as input (i) A set of performance SLOs; (ii) A set of hardware SKUs with specific costs and performance characteristics; (iii) A set of tenants with different performance SLOs to be scheduled; and

computes the hardware provisioning and the tenant scheduling policies that minimize costs while satisfying all SLOs. This step is discussed in more detail in Section 2.3.3.

### 2.3.1 Characterizing Multi-Tenant Performance

This section discusses the first step in our framework that is shown in Figure 2.2.

**Workload and Performance Metric**

To make the discussion concrete, in this work, we use TPC-C as a model workload, which has also been used before to study DaaS [47].

Each of our TPC-C transactions were implemented as stored procedures within SQL Server. Our application driver issued stored procedure calls to SQL Server via .NET connections from network-attached clients. Like prior studies [94], we maintained the full transaction mix ratio as dictated by TPC but eliminated think-time pauses, implemented each tenant with a single remote application driver, and did not scale the number of clients with warehouses. As a performance metric, we use the throughput of the *new-order* transactions, as is done for reporting TPC-C results[1] .

**Hardware SKUs**

Table 2.1 shows the two server SKUs, *ssdC* and *diskC*, that we use in this study. Both servers are identical except for the storage subsystem. Both server SKUs are configured with low-power Nehalem-based L5630 Intel processors (dual quad cores), and 32GB DDR3 memory, running Windows Server 2008R2 and the latest internal version of SQL Server. The OS and the DBMS are installed on a separate 10K RPM 300GB SAS drive. In the *ssdC* configuration, all the data files and log files of the database are stored on three Crucial C300 256GB SSDs while in the *diskC* configuration, these are stored on three 10K RPM 300GB SAS drives.

We note that the storage subsystem has a big impact on the RDBMS performance in a multi-tenant environment, since the load imposed on the hardware when serving independent tenant requests naturally leads to randomized data access. This behavior is in contrast to traditional single-tenant environments where

[1] Disclaimer: While we have used the TPC-C benchmark as a representative workload in this work, the results presented are not audited or official results, and, in fact, were not run in a way that meets all of the benchmark requirements. Consequently, these results should not be used as a basis to determine SQL Server's performance on this or any other related benchmarks.

| | ssdC | diskC |
|---|---|---|
| CPU | 2X Intel L5630 | 2X Intel L5630 |
| RAM | 32GB DDR3 | 32GB DDR3 |
| OS Storage | 10K SAS 300GB | 10K SAS 300GB |
| DB Data | 2X Crucial C300 256GB | 2X 10K SAS 300GB |
| DB Log | Crucial C300 256GB | 10K SAS 300GB |
| RAID Cntlr w/BBC | YES | YES |
| Cost | $4,500 | $4,000 |

Table 2.1: Two server configurations (SKUs)

the DBMS schedules data accesses to be as sequential as possible.

**Multi-Tenancy and Performance**

There are many ways to deploy a DaaS on a cluster with multi-tenancy [16, 17, 42, 47, 146]. We list four main approaches to housing tenants that have emerged recently in decreasing order of complexity: (1) all tenant data are stored together within the same database and the same tables with extra annotation such as 'TenantID' to differentiate the records from different tenants [16, 17]; (2) tenants are housed within a single database, but with separate schemas to differentiate their tables and provide better schema-level security; (3) each tenant is housed in a separate database within the same DBMS instance (for even greater security); (4) each tenant has a separate Virtual Machine (VM) with an OS and DBMS, which allows for resource control via VM management [47].

We use option 3 to implement multi-tenancy, since this option above provides a good trade-off between wasted resources due to extra OSs in the VM method (option 4), and the complex manageability and security issues associated with options 1 and 2 [47]. Looking at the other options is an interesting direction for future work.

In our experiments, we consider a workload comprised of 1GB TPC-C tenants with 10 warehouses. We recorded the average per-tenant TPC-C transactions per second achieved on both hardware SKUs for varying degrees of multi-tenancy over a time span of 100s. These results are shown in Figure 2.3.

There are a few important observations from Figure 2.3. First, on our hardware SKUs, the only way tenants can achieve a performance of 100tps is if their datasets almost completely fit in memory. Note

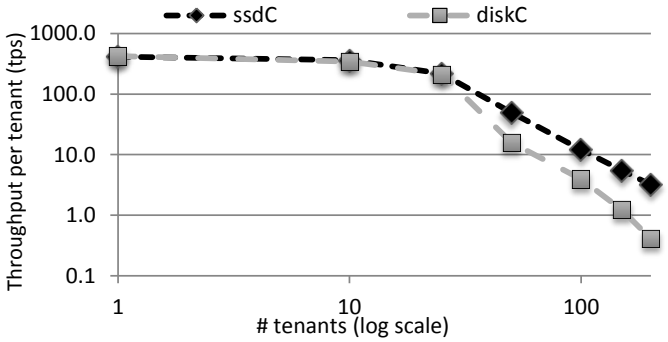



Figure 2.3: Performance for the ssdC and diskC SKUs (see Table 2.1) as we increase the number of tenants on a single SKU.

the drop-off in tps when the number of tenants is increased beyond 25 (i.e., after the combined tenant size crosses $25GB$). Second, when the datasets fit completely in memory, the cheaper *diskC* server can deliver the same per-tenant performance as the more expensive *ssdC* server since the storage subsystem is not the bottleneck. Finally, notice that at the lower performance levels, the *ssdC* server can support significantly more concurrent tenants than the *diskC* server. This behavior is due to the better random I/O performance of the SSD storage compared to the mechanical disk storage. For instance, in Figure 2.3, the measured log disk utilization at 10 tenants for the *ssdC* and *diskC* SKUs was 39% and 41% respectively. As we increased the number of tenants to 25, the log disk utilization increased to 50% and 66% for these two SKUs respectively. Finally, at 50 tenants and beyond, the log disk utilization is saturated at more than 95% for both SKUs.

The curve shown in Figure 2.3 defines the maximum number of tenants that each SKU can support while maintaining a specific performance level per tenant. This homogeneous multi-tenant benchmarking is a necessary first step since it defines the boundaries of the performance that the DaaS provider can promise in their SLOs.

**Definition 2.3.1.** *Let the set $S = \{s_1, s_2, ..., s_k\}$ represent the $k$ SLOs published by a DaaS provider.*

Typically, $k > 1$ since different tenants may require (and be willing to pay for) different levels of performance. Given a set of tenants with different SLOs to schedule on a cluster, a natural scheduling policy is to schedule the tenants of each class on the type of server that can handle the most number of tenants of that class. However, this approach ignores the relative cost of different SKUs, as well as the possibility of

scheduling tenants of different classes on the same server to reduce the overall provisioning and operating costs. The next step (Section 2.3.2) is to determine the behavior of a single SKU when loaded with tenants that are associated with different SLOs.

### 2.3.2 Characterizing Heterogeneous SLOs

A number of mechanisms can be used to provide different performance SLOs on the same server. One simple mechanism is resource governance whereby tenants are allocated specific amounts of critical resources like CPU and DBMS buffer pages to limit their resource consumption. Another mechanism is to use an admission control server that throttles incoming tenant requests accordingly. Studying the different mechanisms to implement performance SLOs is an interesting topic, but is orthogonal to our optimization framework, and hence beyond the scope of this work.

To avoid the additional complexity of an admission control server, we chose to simulate a buffer pool resource governance mechanism on top of SQL Server. In our method, we start separate SQL Server instances within each physical server with one instance for each SLO class $s_i$ (there are $k$ of these as per Definition 2.3.1). All tenants that belong to the same SLO class $s_i$ are assigned to the same SQL Server instance. The performance of each SQL Server instance is throttled by limiting the amount of main memory that is allocated to it. The amount of main memory that is allocated to each SQL Server instance (SLO class) is an average of two factors. The first factor is the fraction of the tps requirements for that SLO class compared to the aggregate total tps across all the SLO classes. The second factor is the ratio of tenants in that SLO class to the total number of tenants. This memory allocation method provides a balance between allocating memory purely based on tps and purely based on the number of tenants. (We experimented with other methods, but found that this method provided the best overall behavior allowing us to pack far more tenants per SKU than other simpler methods.)

Recall that Figure 2.3 characterizes the performance of the server SKUs *ssdC* and *diskC* when all the tenants on a SKU have equal access to resources. Given tenants with different SLOs (Definition 2.3.1), we need to characterize the performance delivered by each server SKU to each tenant class $s_i$. For this purpose, we use a SKU performance characterizing function, which is described next.

**Definition 2.3.2.** *For a given SKU, let* $\vec{b} = [b_1 \ b_2 \ ... \ b_k]^T$ *where* $b_i$ *represents the number of tenants of class* $s_i$ *scheduled on the server. For this server, the SKU performance characterizing function,* $f(\vec{b})$*, represents the performance delivered over a specific time interval for different tenant scheduling policies. Here* $f(\vec{b}) = [\phi_1 \ \phi_2 \ ... \ \phi_k]^T$ *where* $\phi_i$ *is the random variable representing the performance achieved by the tenants of class* $s_i$ *scheduled on the server.*

Using this definition for function $f$, it is possible to provide the performance SLOs in the same way as the current uptime SLAs. For instance, say that for a given SKU with a load defined by $\vec{b}$, we determine that the distribution of the measured performance over 100 seconds for the tenants of class $s_i$ (say, a 100tps class) is normal, with an average of 130 tps and a standard deviation of 10tps; that is, $\phi_i \sim N(130, 10)$. Then, according to the definition of a normal distribution, for all the 100tps tenants that are scheduled on this server, we can guarantee the desired performance 99.6% of the time.

The ability to provide such guarantees makes our formulation of the SKU characterizing function $f$ very powerful in defining performance SLOs. In practice, fully characterizing $f$ is likely to be very challenging and one has to simplify this function. Here, we consider the following simplification of $f$ to a boolean characterizing function (exploring other options is an interesting direction for future work).

**Definition 2.3.3.** *Given a certain server SKU and* $\vec{b}$ *from Definition 2.3.2, a simplified boolean SKU performance characterizing function* $\hat{f}(\vec{b})$ *returns true if all the tenants achieve their respective SLO performance based on a set of summary statistics of the random variables and false otherwise.*

As a simplification for our experiments, we ignored other statistics such as variance and defined $\hat{f}(\vec{b})$ in terms of the average transactions per second over 100s. For example, consider Figure 2.1, we plotted $f(\vec{b}) = [E[\phi_1] \ E[\phi_2] \ ... \ E[\phi_k]]^T$ for *ssdC* (see Table 2.1) for two SLO classes, $S = \{100tps, 10tps\}$.

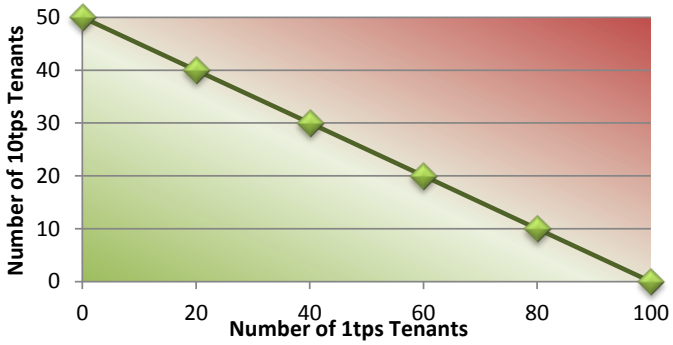
Having defined the SKU performance characterizing function, the next question is to find acceptable operating zones that deliver the promised performance to each tenant in each class $s_i$. Again, using Figure 2.1 as an example, we wish to compute the area in both sub-figures where both the 100tps tenants and the 10tps tenants meet their performance requirements. This area defines the acceptable "operating zone" for the *ssdC* SKU, and is distinguished from the other areas using Definition 2.3.3.

To evaluate the function $\hat{f}$, a systematic search of the tenant scheduling space is performed as follows: We first start by scheduling the maximum number of highest-performance tenants as determined by the benchmarking step in Section 2.3.1. Then, we systematically substitute a fixed small number of these highest-performance tenants with low-performance tenants (if there are more than two tenant classes, in this step, we can iterate through fixed size combinations of the lower performance tenant classes). For each sample, we run a benchmark with the current mix of tenants, and record the observed per-tenant performance. If the observed performance satisfies all tenant SLOs, then $\hat{f}$ returns true for this tenant scheduling policy and for all other scheduling policies where there are fewer tenants in any of the classes. If $\hat{f}$ returns true, we also try adding more low-performance tenants (iteratively in every low performance class) and repeat the experiment. We keep pushing up the number of the tenants in the low performing tenant class(es) until $\hat{f}$ returns false, in which case we know we have reached the boundary of the $\hat{f}$ function. Thus, we determine a tenant scheduling "frontier", so that $\hat{f}$ is true on one side of the frontier and false on the other side. (As part of future work, it would be interesting to consider obtaining this frontier via other methods such as augmenting the query optimizer module to generate/estimate this frontier [30, 55].)

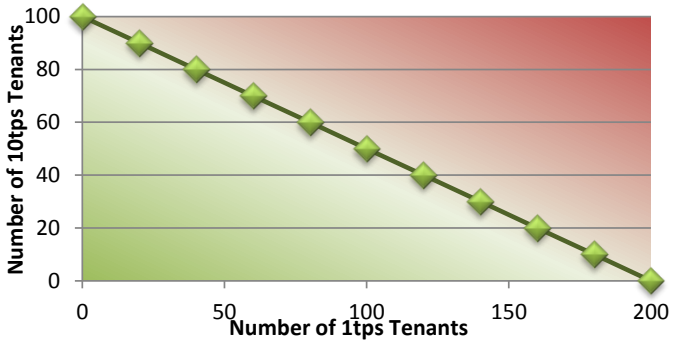
**Frontier for the SLO mix – 10tps and 1tps** Consider the SLO set $S = \{10tps, 1tps\}$, and the SKUs ssdC and diskC (see Table 2.1). The frontiers for this case are shown in Figure 2.4 as the solid black line. The diamond points in this figure represent some of the actual benchmark tests that were run. The points that lie *above* a frontier line represent tenant scheduling policies that *fail* to meet tenant SLOs ($\hat{f} = false$), whereas the points that lie *on* the frontier line will satisfy all tenant SLOs. The area below the frontier line contains scheduling policies that will satisfy tenant SLOs but potentially waste resources (i.e., are potentially over-provisioned).

An interesting point about the performance characteristics shown in Figure 2.4 is that the bottleneck for the points in the frontier is the log disk. Each database has a log file and as more tenants are added, the I/Os to the log disk become more random, and each log I/O becomes relatively more expensive. As a result, if we look at the pure 10tps case (upper left point in the graph) and remove $x$ of the 10tps tenants, we can add far fewer than $10x$ 1tps tenants.

Having a linear frontier as is the case in Figure 2.4 implies that we can add/remove tenants of different




(a) Performance on the diskC SKU

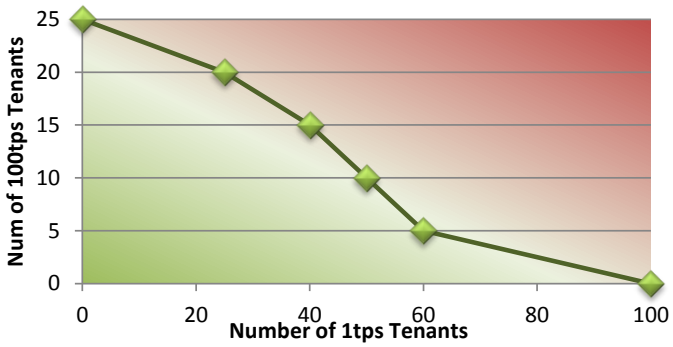



(b) Performance on the ssdC SKU

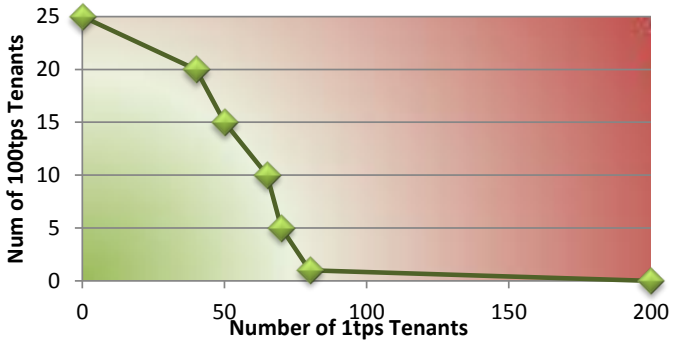
Figure 2.4: SKU performance characterizing functions for $S = \{10tps, 1tps\}$

classes to a server according to a constant ratio. For example, consider again the frontier for the *diskC* SKU (Figure 2.4(a)) and the *ssdC* SKU (Figure 2.4(b)). The slope of the lines in both graphs is $-\frac{1}{2}$, which implies that for any operating point along these two frontier lines, the DaaS provider can safely swap one 10tps tenant for two 1tps tenants. Thus, a linear frontier simplifies the tenant scheduling policies. As we discuss below, we may not always observe a linear frontier.

**Frontier for the SLO mix – 100tps and 1tps** Suppose that a DaaS provider wishes to publish a 100tps SLO. From Figure 2.3, we know that for both SKUs, we are limited to about 25 100tps tenants on either SKU. Figures 2.5(a) and (b) show the observed frontiers for both the *diskC* and *ssdC* SKUs respectively, for $S = \{100tps, 1tps\}$. The frontiers are no longer linear and show that if we start from the case of only 100tps tenants (upper left point in both graphs), the initial curve is convex and then tapers off into a concave shape. At the "only 100tps tenants" point, the system is memory bound (see Figure 2.3). As we move to the right along the frontier, the system now becomes log disk bound.




(a) Performance on the diskC SKU




(b) Performance on the ssdC SKU

Figure 2.5: SKU performance characterizing functions for $S = \{100tps, 1tps\}$

The initial shape of the frontier is convex since the log disk saturates a little beyond the proportions dictated by the line formed by connecting the two end points of the frontiers. For example, in Figure 2.5(a) as we move from the 25 100tps case to the right, we reach a point where there are 20 100tps tenants. If the frontier were linear, then we should only be able to add $5 \times 4 = 20$ 1 tps tenants, but we can add 25 1tps tenants before the log disk saturates.

Now consider the concave tail of the frontier in Figure 2.5. Again this has to do with the log disk. Consider the (bottom) right-most point in the frontier. Here we have only 1 tps tenants. At this point, the system is bottlenecked on the log disk. This behavior is captured in Figure 2.6, which plots the log disk performance ($y$ axis) of an *ssdC* server with one 100tps tenant as the number of 1tps tenants is varied ($x$ axis). The log write wait time is shown as a range by a vertical bar where the low point denotes the first quartile and the high point denotes the third quartile. The horizontal (green) bar denotes the average. The performance achieved by the 100tps tenant (shown on the right vertical axis) is plotted using round dots.

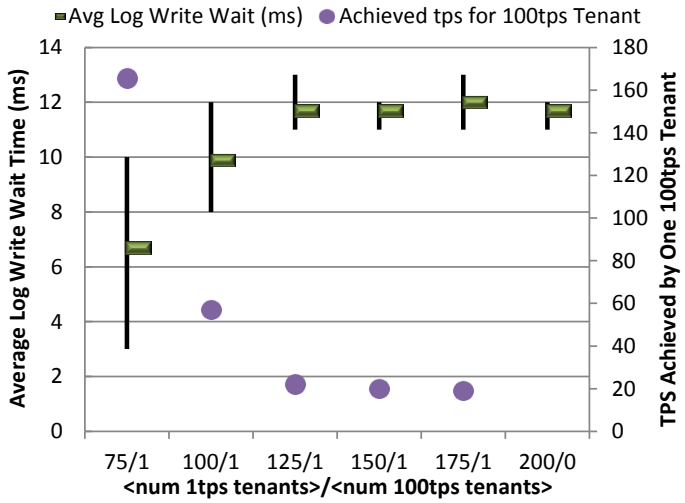
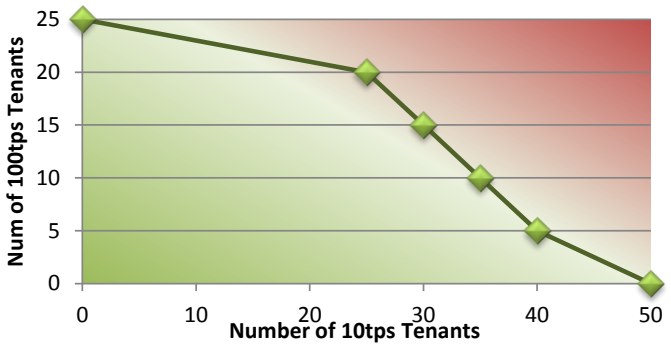


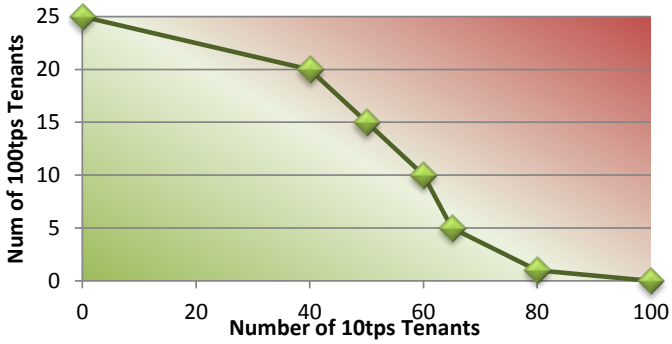

Figure 2.6: Average database log write wait time with vertical bars spanning the $1^{st}$ to the $3^{rd}$ quartiles, along with the average tps achievable by a single 100tps tenant on the *ssdC* SKU.

In Figure 2.6, we see that at the 200/0 point, the log disk writes takes an average of 12 ms (and the log disk is saturated at this point). If we move to the left from this point by dropping 25 1 tps tenants and adding one 100 tps tenant, then the 100tps tenant only achieves around 20 tps. As we continuously decrease the number of 1tps tenants by 25, we observe that the average log write wait time decreases only after 125 1tps tenants. The performance achieved by the 100tps tenant very closely follows with a jump at 100 1tps tenants. These results show why scheduling one 100tps tenant onto the server in Figure 2.5 requires a substantial drop in 1tps tenants. To summarize, *a high performance tenant requires disproportionately large headroom in log disk provisioning to process transactions with a high throughput. Thus, even though the tenants are all running the same workload, the sheer increased performance requirement of some tenants over others causes resource requirement disparities similar to tenants running different workloads.*

**Frontier for the SLO mix – 100tps and 10tps** Now let us consider a mix of 100tps and 10tps tenants, i.e., $S = \{100tps, 10tps\}$. The results for this case are shown in Figures 2.7(a) and (b) for the diskC and the ssdC SKUs respectively. For the same reasons as discussed above for the 100tps/1tps case, we observe the a knee near the lower right corner of the frontier line, and a convex shape near the upper left corner of the frontier line.




(a) Performance on the diskC SKU




(b) Performance on the ssdC SKU

Figure 2.7: SKU performance characterizing functions for $S = \{100tps, 10tps\}$

### 2.3.3 Putting It All Together

In the previous section, we described how to compute the SKU performance characterizing function for each SKU. We can now use these functions to formulate and solve the optimization problem for provisioning hardware and scheduling tenants that satisfy different performance SLOs (namely Step 3 in Figure 2.2).

**Definition 2.3.4.** *$M$ is a multiset $\{m_1, m_2, ..., m_p\}$ where each $m_j$ represents a server SKU defined by a pair $m_j = (\hat{f}_j, c_j)$ where function $\hat{f}_j$ is the simplified SKU characterizing function (defined in Definition 2.3.3) and $c_j$ represents the amortized monthly operating cost for a server.*

Note that since $M$ is a multiset, $m_j$ need not be unique. This allows a single server SKU to be scheduled with tenants in different ways.

Recall that we have the set of published SLOs as defined in Definition 2.3.1. We must now associate each tenant with its corresponding SLO.

**Definition 2.3.5.** *Let $t_i$ represent the set of tenants that subscribe to SLO $s_i$ as defined in Definition 2.3.1. We represent all tenants by the set $T = \cup_{i=1}^{k} t_i$.*

Using Definitions 2.3.1 to 2.3.5, the following definition describes the main optimization (minimization) problem.

**Problem Definition 1.** *Given the sets $S$, $T$, and multiset $M$, compute $a = [\alpha_1\ \alpha_2\ ...\ \alpha_p]$ and $B = [\vec{b_1}\ \vec{b_2}\ ...\ \vec{b_p}]^T$, where $\alpha_i$ is the needed number of servers of type $m_i$, and $\vec{b_i}$ is a vector of length $k$ indicating how many tenants of each of the $k$ SLO classes should be scheduled on an individual server of type $m_i$. The objective function $C = \Sigma_{i=1}^{p} \alpha_i c_i$ satisfies the following constraints:*
*Constraint 1 : $aB = [|t_1|\ |t_2|\ ...\ |t_k|]$ (cover all the tenants)*
*Constraint 2 : $\hat{f}_i(\vec{b_i})$ returns true for $1 \leq i \leq p$ (all SLOs are satisfied)*

Problem Definition 1 is a non-linear programming problem in the general case[2] . Here, we need to compute the following variables:

(1) $a$ – the number of servers used for each SKU. This vector determines the total cost for provisioning the servers.

(2) $B$ – the tenant scheduling policy.

The entire space of solutions does not need to be fully explored since the feasible regions are defined by the $\hat{f}$ characterizing functions and the curves defined by Constraint 1 of Problem Statement 1. Since our space of solutions is relatively small, a brute-force solver that explores the non-negative integer space bounded by these curves sufficed for our purposes.[3] Exploring other approaches is part of future work.

With this brute-force solver and the experimental results from Section 2.3.2, we now have the tools that we need to evaluate our framework.

[2] In simple cases, we can parameterize the problem into a linear programming problem, but this is increasingly onerous when faced with non-linear piecewise frontier functions that characterize the server SKUs. The approach we take to solving the non-linear programming problem is much more straight-forward.

[3] For a 5000 100tps tenant and 5000 10tps tenant problem, our single-threaded brute-force solver finds a solution within 80 seconds on a 2.67Ghz Intel i7 CPU.

| Scenario | SLO set | Tenant Ratio | diskC Amortized Cost |
|---|---|---|---|
| SC1 | $S_1$={10tps, 1tps} | 20:80 | $111 |
| SC2 | $S_1$={10tps, 1tps} | 50:50 | $111 |
| SC3 | $S_1$={10tps, 1tps} | 80:20 | $111 |
| SC4 | $S_1$={10tps, 1tps} | 20:80 | $88 |
| SC5 | $S_1$={10tps, 1tps} | 50:50 | $88 |
| SC6 | $S_1$={10tps, 1tps} | 80:20 | $88 |
| SC7 | $S_2$={100tps, 1tps} | 20:80 | $111 |
| SC8 | $S_2$={100tps, 1tps} | 50:50 | $111 |
| SC9 | $S_2$={100tps, 1tps} | 80:20 | $111 |
| SC10 | $S_2$={100tps, 1tps} | 20:80 | $88 |
| SC11 | $S_2$={100tps, 1tps} | 50:50 | $88 |
| SC12 | $S_2$={100tps, 1tps} | 80:20 | $88 |
| SC13 | $S_3$={100tps, 10tps} | 20:80 | $111 |
| SC14 | $S_3$={100tps, 10tps} | 50:50 | $111 |
| SC15 | $S_3$={100tps, 10tps} | 80:20 | $111 |
| SC16 | $S_3$={100tps, 10tps} | 20:80 | $88 |
| SC17 | $S_3$={100tps, 10tps} | 50:50 | $88 |
| SC18 | $S_3$={100tps, 10tps} | 80:20 | $88 |

Table 2.2: Experimental parameters for evaluating various scenarios. Tenant ratios divide 10,000 tenants across two SLOs for each scenario. The *ssdC* SKU amortized cost over 36 months is $125.

## 2.4 Evaluation

In this section we apply the framework described in Section 2.3 to hypothetical DaaS scenarios to illustrate the merits of the hardware provisioning and tenant scheduling policies obtained as solutions to the cost-optimization problem defined in Problem Definition 1.

In our evaluation, we assume that the hypothetical DaaS provider must accommodate a total of 10,000 tenants running TPC-C scale 10 workloads, with two available SKUs – *ssdC* and *diskC* – as described in Section 2.3.1. We varied the following three parameters to arrive at the 18 scenarios listed in Table 2.2.

(1) Published set of SLOs: We limited ourselves to three sets of SLOs discussed in Section 2.3.2, namely $S_1 = \{10tps, 1tps\}$, $S_2 = \{100tps, 1tps\}$, and $S_3 = \{100tps, 10tps\}$. We used average tps over 100s as the metric to determine if an SLO is satisfied or not.

(2) Tenant ratios: For each SLO set $S_i$, we varied the relative proportion of tenants belonging to one SLO

versus the other. We used three ratios in our scenarios – 20:80, 50:50 and 80:20. For instance, a 20:80 ratio for the SLO set $\{100tps, 1tps\}$ means that 2000 tenants are associated with the 100tps SLO while 8000 tenants are associated with the 1tps SLO.

(3) Relative costs between server SKUs: The true purchase costs of a single *ssdC* and *diskC* server are \$4,500 and \$4,000 respectively. Amortized over 36 months [80], we arrived at monthly costs of \$125 and \$111 respectively. Although in reality the *diskC* server is 10% less expensive than *ssdC*, we also considered a hypothetical *diskC* price point of \$3,150 (\$88 amortized, 30% less than *ssdC*) to consider what happens if the relative costs of the hard disks were lower (e.g., if we had used less expensive SATA3 disks). We note that this method of running our framework with different scenarios can potentially be used by a DaaS provider as a way of "scoping out" the impact of varying SKUs when making a purchasing decision.

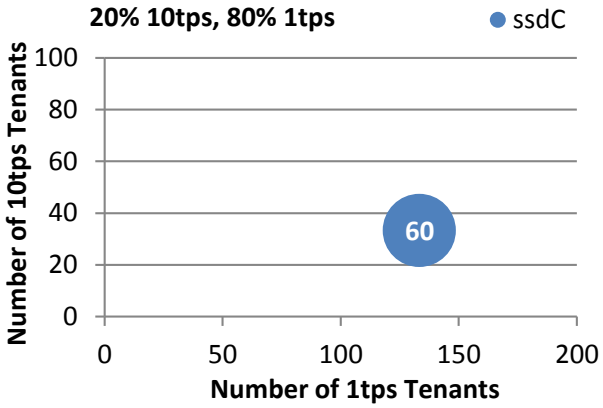
### 2.4.1 Solutions From The Framework

Hardware provisioning and tenant scheduling policies are depicted using bubble plots in a 2-dimensional space. Each bubble represents a single hardware SKU with a specific tenant schedule as determined by the coordinates of the center of the bubble. The size of the bubble denotes the number of servers provisioned from that SKU (i.e., $\alpha_i$ in Problem Definition 1). The position of the bubble corresponds to the the tenant scheduling policy represented by vector $\vec{b_i}$ in the problem definition. That is, the $y$ coordinate is the number of high-performance tenants scheduled on that SKU, and the $x$ coordinate is the number of low-performance tenants. Recall that Definition 2.3.4 allows a single hardware SKU to be used multiple ways with different tenant scheduling policies. Thus, even though we have only two types of servers, *ssdC* and *diskC*, a single plot may contain more than two bubbles.

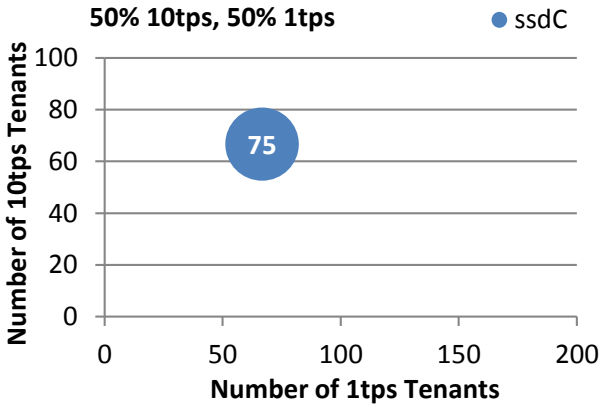
Next, we discuss the hardware provisioning and tenant scheduling policies obtained for each set of SLOs in turn.

**SLO Set 1 – 10tps and 1tps**

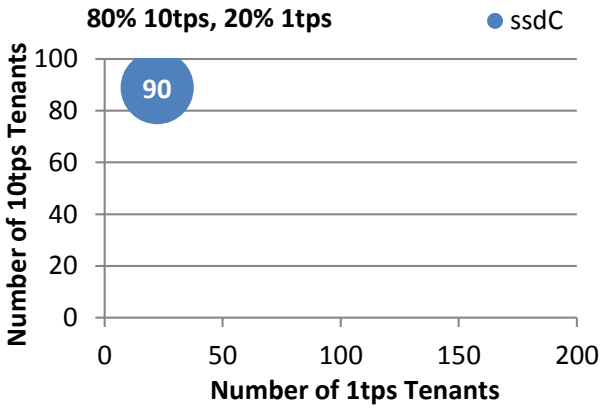
As shown in Figures 2.4(a) and (b), this set of SLOs results in linear SKU performance characterizing




(a)




(b)




(c)

Figure 2.8: Solutions for (a) SC1 - $7,500; (b) SC2 - $9,375; (c) SC3 - $11,250 (see Table 2.2 for details). Circle positions indicate tenant scheduling policy and circle size/annotation indicate hardware SKU provisioning policy.

functions for both the *ssdC* and *diskC* SKUs. Since the *ssdC* SKU can serve twice the number of 10tps *and* 1tps tenants as the *diskC* SKU, we expect the optimal hardware provisioning policy to favor *ssdC* servers, given that the price of an *ssdC* server is less than twice the price of a *diskC* server.

Figures 2.8(a)-(c) shows the optimal solutions for scenarios SC1, SC2 and SC3 (*diskC* costs 10% less than *ssdC*). As expected, the optimal provisioning policy uses only *ssdC* SKUs. Note that since exactly one SKU is used, the ratio of tenants scheduled on each server (determined by the $y$ and the $x$ coordinates) corresponds exactly to the total tenant ratio (20:80, 50:50 and 80:20 in Figures 2.8(a), (b) and (c) respectively). The total costs of the optimal solutions in each case are indicated at the bottom of figure. We can see that as the proportion of 10tps tenants increases from (a) to (c), more servers are required, which increases the total solution cost.

Next, we evaluated the scenarios SC4-SC6 (i.e., the *diskC* SKU is 30% less expensive than the *ssdC* SKU). The optimal policies obtained in this case are identical to those of scenarios SC1-SC3 shown in Figure 2.8 (and hence the figures are omitted). Since the solutions only used *ssdC* SKUs, the change in the *diskC* SKU cost does not affect the optimal solution cost. Again, this is expected given the much higher performance delivered by the *ssdC* servers for this set of SLOs.

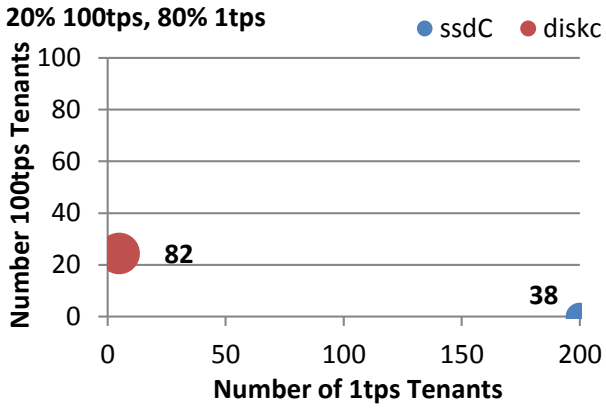
These results suggests that since the *diskC* SKU used in our evaluation delivers roughly half the performance of the *ssdC* SKU, it must cost less than half the *ssdC* SKU to be considered cost-effective. As this experiment (and recent studies [108]) show, the considerable performance benefits obtained by the SSDs may in some cases compensate for the price premium.

**SLO Set 2 – 100tps and 1tps**

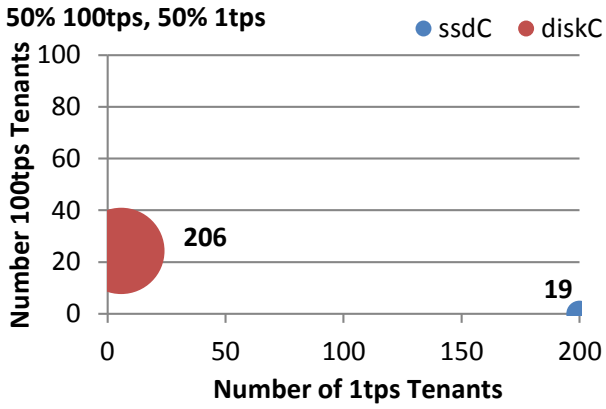
Figures 2.9(a)-(c) show the optimal hardware provisioning and the tenant scheduling policies for scenarios SC7, SC8, and SC9 respectively (*diskC* costs 10% less than *ssdC*). As expected, the less expensive *diskC* SKU plays a large role in the optimal solution. In fact, when the tenant mix contains a large proportion of 100tps tenants (Figure 2.9(c)), the *ssdC* SKU is not used at all! Furthermore, note that even when the *ssdC* servers are used (Figures 2.9(a) and (b)), only the 1tps tenants are scheduled on these servers. These results are somewhat counter-intuitive, since the high-end SKU is scheduled only with the low-end tenants.

In Figure 2.10(a)-(c), we show the optimal solutions for scenarios SC10, SC11 and SC12 (*diskC* costs 30% less than *ssdC*). Now, compared to the results shown in Figure 2.9, we observe that the hardware provisioning policy uses even fewer *ssdC* servers due to their higher relative cost.

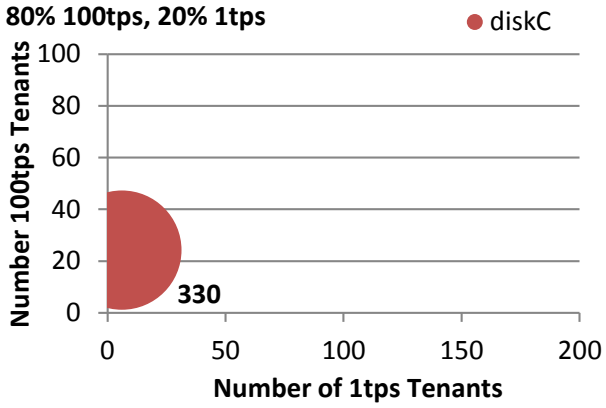
An interesting observation from these results is that in the recommended hardware provisioning pol-




(a)



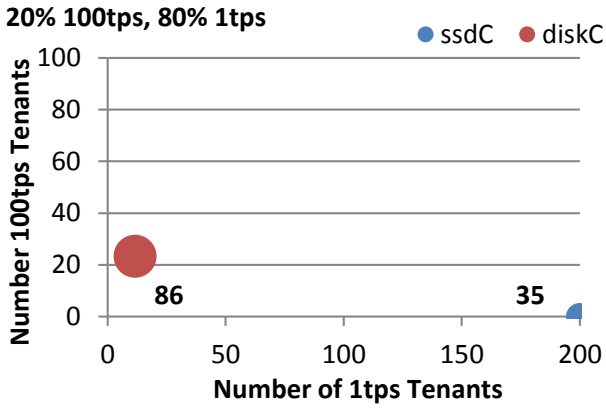

(b)



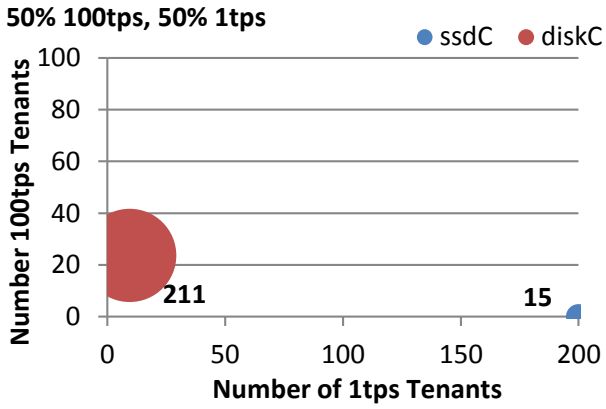

(c)

Figure 2.9: Solutions for (a) SC7 - $13,861; (b) SC8 - $25,264; (c) SC9 - $36,667 (see Table 2.2 for details). Circle positions indicate tenant scheduling policy and circle size/annotation indicate hardware SKU provisioning policy.

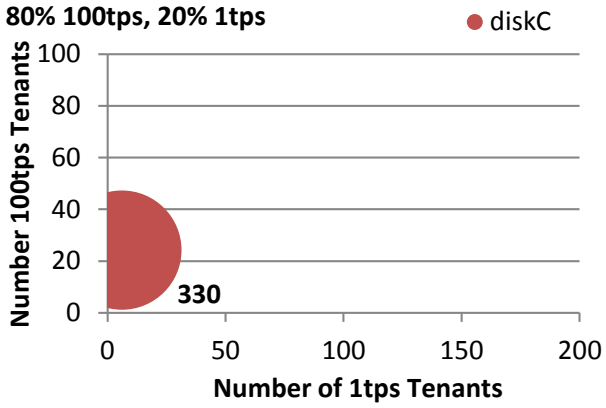
icy, the ratio of the number of servers of one SKU over the number of servers of the other SKU is very large. Examples of this can be found for SC8 and SC11, Figure 2.9(b) and Figure 2.10(b) respectively, where the number of *ssdC* servers is an order magnitude less than the number of *diskC* servers. An alternative (al-




(a)



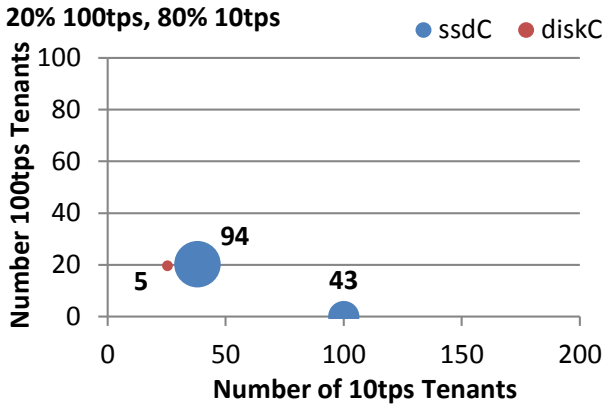

(b)



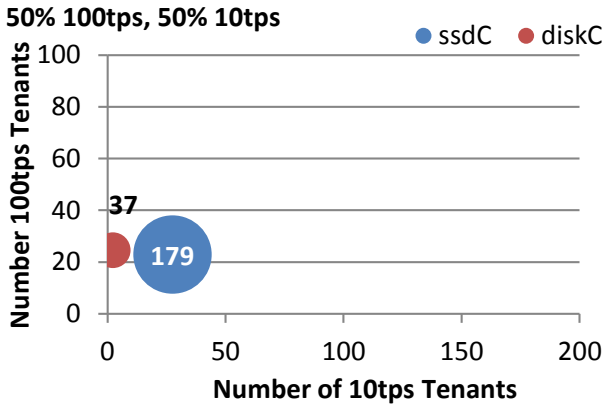

(c)

Figure 2.10: Solutions for (a) SC10 - $11,900; (b) SC11 - $20,338; (c) SC12 - $28,875 (see Table 2.2 for details). Circle positions indicate tenant scheduling policy and circle size/annotation indicate hardware SKU provisioning policy.

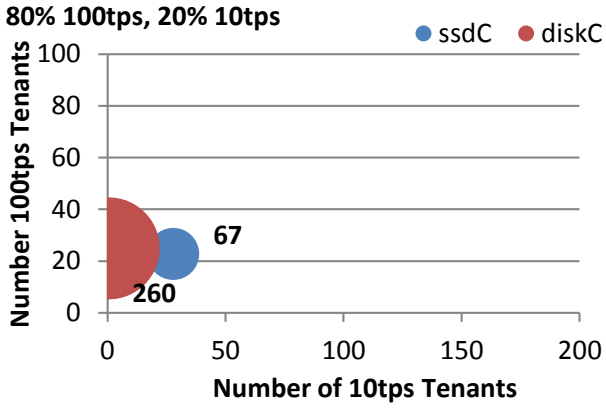
beit suboptimal) SKU provisioning strategy is to simply use only *diskC* servers, and ignore *ssdC* altogether (or vice versa). The advantage of this strategy is that it produces a homogeneous cluster that is easier to manage and administer. In Section 2.4.2, we discuss this and other suboptimal (from the initial hardware




(a)




(b)



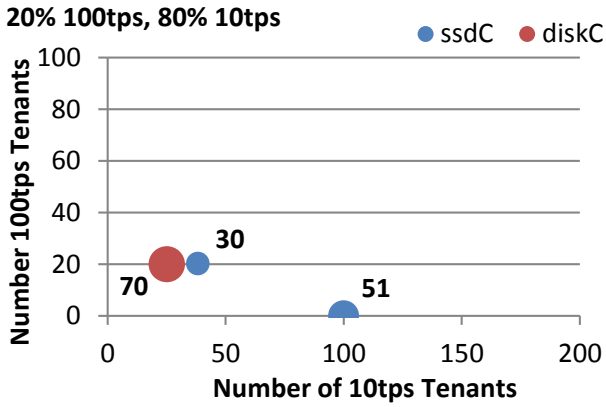

(c)

Figure 2.11: Solutions for (a) SC13 - $17,681; (b) SC14 - $26,486; (c) SC15 - $37,264 (see Table 2.2 for details). Circle positions indicate tenant scheduling policy and circle size/annotation indicate hardware SKU provisioning policy.

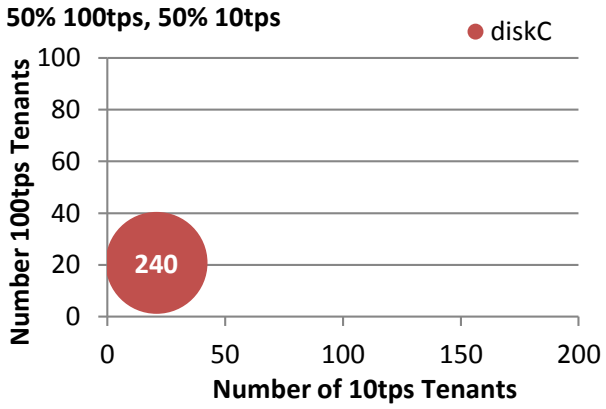
provisioning cost perspective) alternatives and their costs.

**SLO Set 3 – 100tps and 10tps**

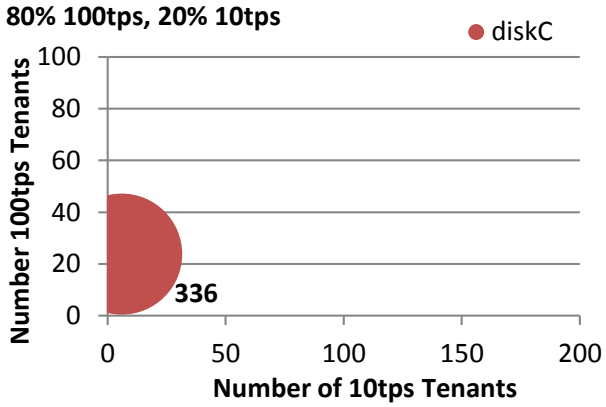
Here we consider SLO Set $S_3$ corresponding to scenarios SC13-15 and SC16-18 in Table 2.2. Fig-




(a)




(b)




(c)

Figure 2.12: Solutions for (a) SC16 - $16,250; (b) SC17 - $21,000; (c) SC18 - $29,400 (see Table 2.2 for details). Circle positions indicate tenant scheduling policy and circle size/annotation indicate hardware SKU provisioning policy.

ure 2.11 plots SC13-15 where the *diskC* SKU costs 10% less than the *ssdC* SKU, and Figure 2.12 plots SC16-18 for the case where the *diskC* SKU costs 30% less.

Interestingly, for this set of SLOs, in some scenarios, the optimal solution uses the *ssdC* SKU with

*two* different tenant scheduling policies. As seen in Figures 2.11(a) and 2.12(a), there are two blue bubbles representing *ssdC* servers – one bubble represents servers that are scheduled with only 10tps tenants and the other represents servers that are scheduled with a mix of tenants.

Since we have a 100tps SLO in $S_3$, the *diskC* servers provide better value because they can handle the same number of 100tps tenants at a lower price. This is why we predominantly see *diskC* servers in the solutions as the tenant ratio shifts toward the high-performance tenants. Similar to Figure 2.10, as we decrease the cost of the *diskC* SKU (Figure 2.12), or increase the number of 100tps tenants (SC15 in Figure 2.11 and SC18 in Figure 2.12), the optimal solution provisions mostly less expensive *diskC* servers.

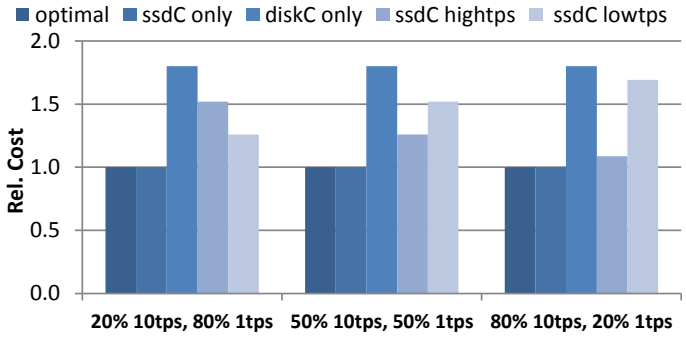
Note that in Figure 2.11(c), the *diskC* servers (red bubble) are scheduled with just one 10tps tenant per server. A simpler solution (with a possibly higher cost) might be to simply schedule no 10tps tenants on the *diskC* servers. Such solutions are discussed in the following section.

### 2.4.2 Suboptimal Solutions – Simplicity vs Cost

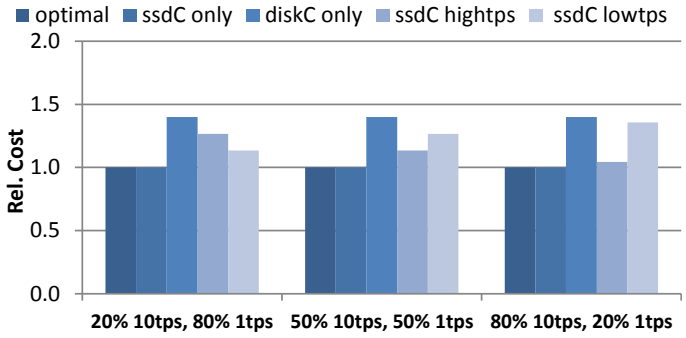
| **Methods** | **ssdC SKU** | **diskC SKU** |
|---|---|---|
| Optimal | heterogeneous SLOs | heterogeneous SLOs |
| ssdC-only | heterogeneous SLOs | – |
| diskC-only | – | heterogeneous SLOs |
| ssdC-hightps | homogeneous high-perf | homogeneous low-perf |
| ssdC-lowtps | homogeneous low-perf | homogeneous high-perf |

Table 2.3: Comparing heuristic tenant scheduling on two hardware SKUs.

In this section, we discuss issues related to the simplicity and manageability of the hardware provisioning and tenant scheduling policies dictated by our framework. At the outset, note that our notion of "total cost" is simplistic as it is only defined in terms of the costs of individual servers. In cloud deployments, issues such as cluster manageability also carry a cost and play an important role in provisioning decisions. In particular, heterogeneous clusters comprised of multiple SKUs can be harder to maintain, manage, and administer compared to homogeneous clusters comprised of a single SKU. A related issue is the complexity of scheduling policies. A straightforward scheduling policy (e.g., assign all tenants with SLO $s_1$ on SKU




(a)

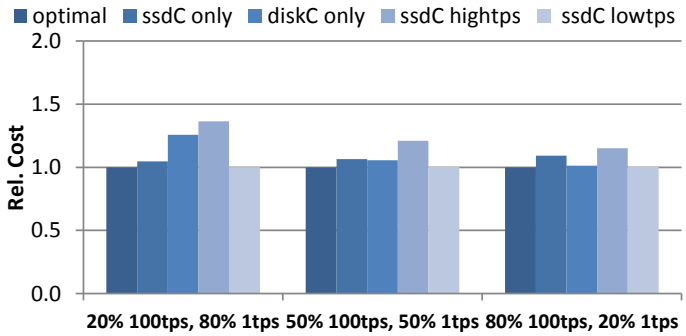



(b)

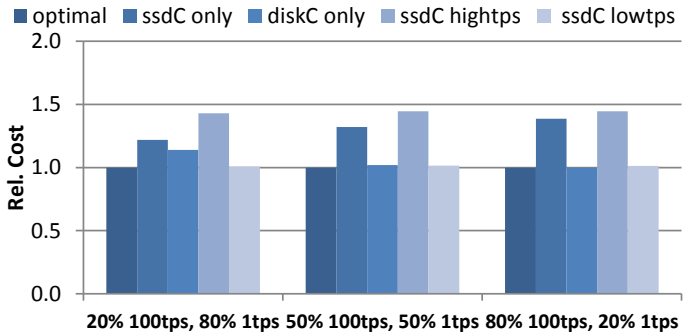
Figure 2.13: Relative costs corresponding to solutions for $\{10tps, 1tps\}$ Scenarios (a) SC1-3 and (b) SC4-6 (see Table 2.2) using our framework and 4 simple methods (see Table 2.3).

1, $s_2$ on SKU 2, etc.) may simplify hardware provisioning decisions as well as tenant pricing policies. For instance, if tenants of a given SLO class are tied to a certain SKU, then they can be charged at a rate determined by the price of that SKU. Here we do not attempt to quantify the notion of cluster "complexity", but leave that as part of future work. Nevertheless, the additional server costs imposed by simpler hardware provisioning and tenant scheduling policies can be determined.

Table 2.3 lists four alternative methods to our optimizing framework. In method *ssdC-only*, we use a homogeneous cluster comprised only of the *ssdC* SKU. Note that this method allows a heterogeneous mix of tenants with different SLOs on a server and also allows for different tenant scheduling policies on different *ssdC* servers. Method *diskC-only* is similar, but with *diskC* servers taking the place of the *ssdC* servers. In method *ssdC-hightps*, all of the high-end tenants are scheduled on the *ssdC* servers, and all of the low-end tenants on the *diskC* servers. In method *ssdC-lowtps*, this assignment is reversed. Thus, in the latter two policies, the SLOs are tied to SKUs. Note that another possible method is to provision a homogeneous




(a)



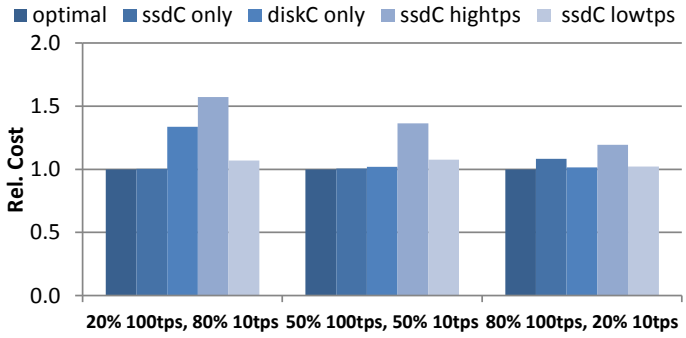

(b)

Figure 2.14: Relative costs corresponding to solutions for $\{100tps, 1tps\}$ Scenarios (a) SC7-9 and (b) SC10-12 (see Table 2.2) using our framework and 4 simple methods (see Table 2.3).

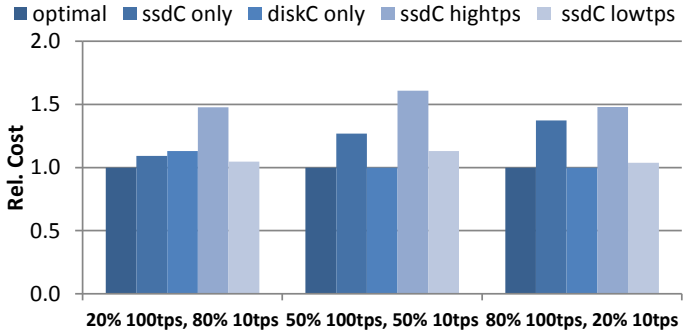
cluster *and* maintain a homogeneous tenant scheduling policy each server. We omit this method since it is subsumed by the *ssdC-only* and the *diskC-only* methods that allow for both homogeneous and heterogeneous tenant scheduling policies.

In Figures 2.13-2.15, we plot the total costs obtained by the five methods outlined in Table 2.3 for the 18 scenarios described Table 2.2. All solutions are plotted relative to the cost-optimal solution (shown as the left-most bar) discussed in Section 2.4.1. At a high-level, while in each case there are some solutions that are identical or very close to the optimal solution, *there is no single method that consistently gives a solution that is close to the optimal solution in all scenarios*. For example, while *ssdC-lowtps* seems to match optimal cost in the $S = \{100tps, 1tps\}$ cases, this is not the trend when $S = \{10tps, 1tps\}$. In another case, while the *ssdC-only* solution is optimal for the scenarios depicted in Figure 2.13, it is not optimal for the scenarios shown in Figure 2.14.

Let us examine a few solutions in more detail. In Figure 2.13(a) and (b), which correspond to the




(a)




(b)

Figure 2.15: Relative costs corresponding to solutions for $\{100tps, 10tps\}$ Scenarios (a) SC13-15 and (b) SC16-18 (see Table 2.2) using our framework and 4 simple methods (see Table 2.3).

$\{10tps, 1tps\}$ SLO scenarios SC1-3 (*diskC* SKU cost 10% less than the *ssdC* SKU) and SC4-6 (*diskC* SKU cost 30% less than the *ssdC* SKU) in Table 2.2 respectively, the *diskC-only* solution is significantly worse than the optimal solution while the *ssdC-only* solution is optimal. The *ssdC-hightps* method appears increasingly attractive as the proportion of 10tps tenants (high-perf. tenants) grows. The costliest solutions are the *diskC-only* and *ssdC-lowtps* methods. In Figure 2.14 (the $S = \{100tps, 1tps\}$ case), the *ssdC-only* and the *ssdC-hightps* methods are expensive solutions in Figures 2.14(b) and (a) respectively, since the *ssdC* and the *diskC* SKUs can both handle only 25 100 tps tenants, but the *ssdC* server is more expensive. Also, a homogeneous *diskC* cluster is generally more expensive when the tenants skew towards the 1tps SLO. This is because the ssdC SKU can schedule many per 1tps tenants than diskC SKU (Figure 2.3). The trends shown in Figure 2.15 (for $S = \{100tps, 10tps\}$) are similar to those of Figure 2.14 for the same reason.

This analysis shows that simpler provisioning methods may come close to the optimal solution provided by our framework, but no single method produces consistently good solutions. Moreover, these sim-

pler heuristics *still require SKU performance characterization* in order to schedule tenants while adhering to tenant SLOs. Our framework produces low-cost hardware provisioning and tenant scheduling policies for multi-tenant database clusters that are up to 33% less costly than simpler provisioning methods. Thus, the cost benefit of an optimal solution over a suboptimal solution must be weighed against cluster manageability and simplicity.

### 2.4.3 Suboptimal Solutions – Dynamic Tenants

Throughout our previous analysis and discussion, we have assumed that the DaaS provider only needs to optimize once for a static number of tenants. In this section, we will relax this assumption and consider what happens when the tenant population changes in size and/or SLO make-up (the ratio of high-performance tenants to low-performance tenants).

Specifically, we consider the following problem: first, the DaaS provider has an active set of tenants, $T_1$, that have been optimally scheduled to a set of provisioned servers according to our optimization framework. In addition, the provider has another set of tenants, $T_2$, that potentially has a different ratio of high-performance tenants to low-performance tenants. The set $T_2$ is intermittently active and is also optimally scheduled to provisioned servers using our framework. The problem we consider is: Given that the provider knows the $T_2$ tenant set will become active for time length $w$[4] , should the provider (i) keep the two tenant populations independently **locally-optimized**; or, (ii) **globally-optimize** all $T_1 \cup T_2$ tenants?

When considering a global optimization across $T_1 \cup T_2$, we must consider what happens when a $T_2$ tenant that was scheduled onto a particular server is re-scheduled onto a different server. In such a scenario, we must migrate the tenant's database from one server to another.[5] Since tenant migration consumes resources, this action is not free and so we associate a cost function $g(dest, source)$, in dollars, with migrating a tenant from server $source$ to server $dest$ (assuming a fixed size tenant).

Formally, we define an extension to Problem Statement 1 that compares the cost of keeping tenant

[4] If $w = \infty$, this is the case where the $T_2$ tenants will remain active indefinitely.

[5] In environments with high availability (HA) SLAs, tenant databases are replicated across multiple servers and we may only need to re-direct user requests to a replica. This "swap" approach is generally not as expensive as migrating tenant data. However, considering HA SLAs is beyond the scope of this study and is the focus of future work. For our discussion, we will simply consider a single replication environment, or one with a fixed primary copy, which in turn requires tenant data migration.

populations locally-optimized or globally-optimizing all the tenants:

**Problem Definition 2.** *We are given SLO set $S$, server SKU multiset $M$, and two tenant sets $T_1$ and $T_2$. Using our framework to solve Problem Statement 1 independently for $T_1$ and $T_2$, we get the* ***locally-optimized*** *solutions $(a_1, B_1)$ and $(a_2, B_2)$ respectively. Over a time length $w$, the amortized server cost of the $T_1$ solution $(a_1, B_1)$ will be $C_1^w$ and the amortized server cost of the $T_2$ solution $(a_2, B_2)$ will be $C_2^w$. Alternatively, the provider can* ***globally-optimize*** *the combined tenant set $T_1 \cup T_2$ using our framework and get a scheduling and provisioning solution $(a_{1\cup2}, B_{1\cup2})$ with a cost of $C_{1\cup2}^w$. However, during this global optimization, the set of tenants $\pi \subseteq T_1 \cup T_2$ need to be migrated with the cost $\Sigma_{\forall\tau_i\in\pi} g(to(\tau_i)\ , from(\tau_i)\ )$ where from$(\tau_i)$ and to$(\tau_i)$ provide the source and destination servers for migrating $\tau_i$ respectively. The provider should only globally-optimize all $T_1 \cup T_2$ tenants for the upcoming $w$ time span if:*

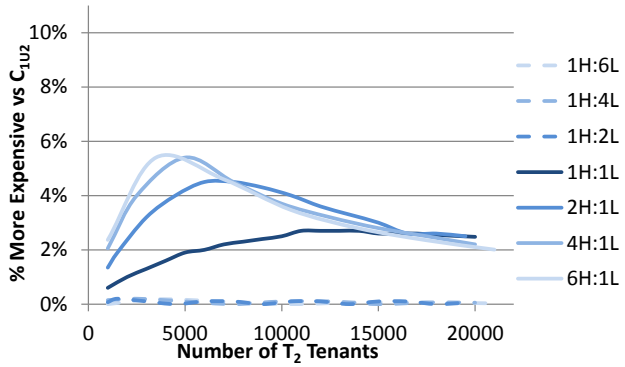
$$C_1^w + C_2^w > C_{1\cup2}^w + \Sigma_{\forall\tau_i\in\pi} g(to(\tau_i), from(\tau_i)) \tag{2.1}$$

In the following section, we will first analyze the difference between the sum of the costs of the **locally-optimized** solutions, $C_1^\infty + C_2^\infty$, and the cost of the **globally-optimized** solution, $C_{1\cup2}^\infty$. Following that, we will consider the cost of migration for globally optimizing the entire tenant population.

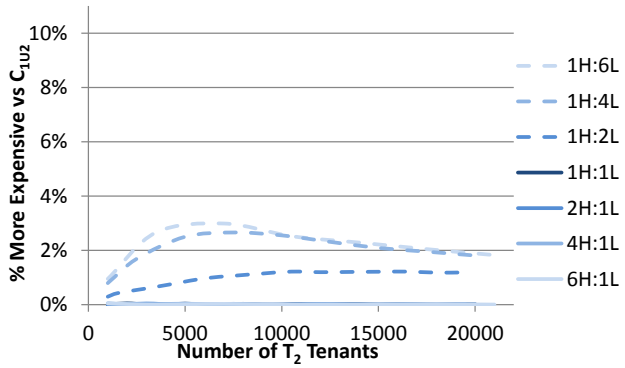
**The Cost of Locally-optimal solutions vs Globally-optimal solutions**

From Equation 2.1, it is intuitive to see that $C_1^\infty + C_2^\infty - C_{1\cup2}^\infty$ bounds the migration cost if the provider wishes to globally optimize $T_1 \cup T_2$. If the cost difference between the locally-optimal solutions and the globally-optimal solution is small, then it means that tenant migration must be done in a very cheap way to make global re-optimization viable. We now present some results that show how big the cost differences can be.

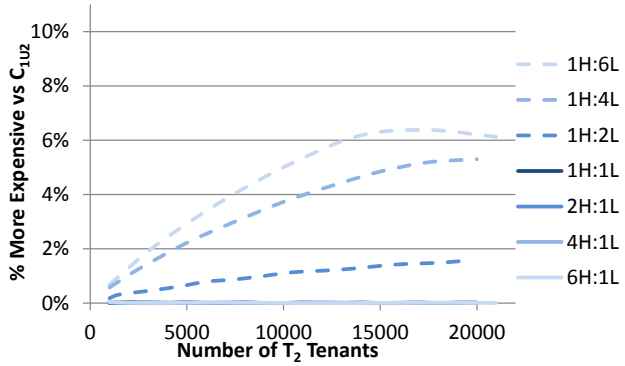
In Figure 2.16, we plot $(C_1^\infty + C_2^\infty)/C_{1\cup2}^\infty$ for various sets of $T_1$ and $T_2$ tenants where $S = \{H, L\}$, where $H$ corresponds to the 100tps SLO, and $L$ corresponds to the 10tps SLO. In all of the sub-figures, we considered a variety of $T_2$ populations from 1,000 to 20,000 (x-axis) and the new ratios that we considered are 1H:6L, 1H:4L, 1H:2L, 1H:1L, 2H:1L, 4H:1L, and 6H:1L (the various data series). The three sub-figures (a)-(c) correspond to 10,000 $T_1$ tenants with tenant ratios 1H:4L, 1H:1L, and 4H:1L respectively.




(a) $T_1$ tenants with 1H:4L SLO ratio




(b) $T_1$ tenants with 1H:1L SLO ratio




(c) $T_1$ tenants with 4H:1L SLO ratio

Figure 2.16: The relative cost difference between the two **locally-optimized** solutions, $(C_1 + C_2)$, and a single **globally-optimized** solution, $C_{1\cup2}$. The tenants have either "H" (100tps) or "L" (10tps) SLOs. There are 10,000 $T_1$ tenants. We varied the size of the $T_2$ tenant set and also the ratio of "H" and "L" tenants in each sub-figure. The diskC server SKU is 10% less expensive than the ssdC server SKU.

In Figure 2.16(a), we see that if the $T_2$ population has a SLO ratio skewed toward the 100tps – 'H' objective, the global re-optimization solution is significantly less expensive (over 5%). Since the $T_1$ tenants are skewed in a 1H:4L ratio, the 2H:1L, 4H:1L, and 6H:1L are oppositely skewed. Intuitively, separately optimizing for two oppositely skewed populations should result in a higher cost than a solution that has

optimized for the combined populations. Furthermore, we notice that as the $T_2$ population increases in size, dwarfing $T_1$, the cost differences begin to shrink since a dominating percentage of $(T_1 \cup T_2)$ is $T_2$ which was optimally scheduled onto an optimal server provisioning. We also notice that if the set $T_2$ has a similar tenant ratio (1H:2L, 1H:4L, 1H:6L – the dashed curves in Figure 2.16(a)) as $T_1$, then there is a negligible cost difference between the locally-optimized and globally-optimized solutions.

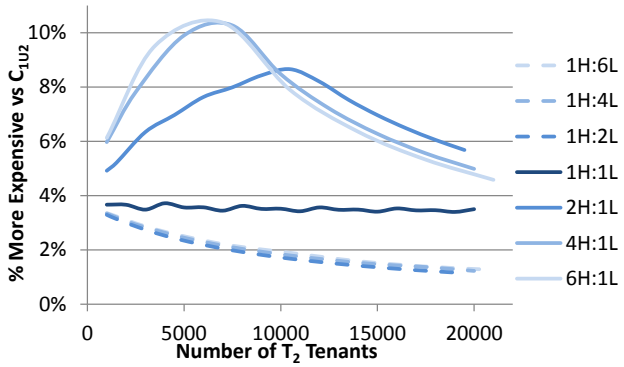
So from the results shown in Figure 2.16(a), we put forth a rule of thumb for dealing with Problem Definition 2: *The globally-optimized solution is preferred when the $T_1$ and $T_2$ tenant sets have opposing SLO ratios.* For example, in Figure 2.16(a), when $T_1$ has a 1H:4L SLO ratio and $T_2$ has a 4H:1L SLO ratio, optimizing the tenant sets separately is the most costly.

This rule of thumb is also validated in Figure 2.16(c), where the $T_1$ population has an SLO ratio of 4H:1L. The analysis shows that the 1H:2L, 1H:4L, and 1H:6L $T_2$ population curves have higher cost (over 6%). If $T_2$ has an H-oriented skew (the solid curves), we see that there is a negligible cost difference between the two solutions. In Figure 2.16(b), where the $T_1$ population is balanced at 1H:1L, we see that cost differences also arise when the $T_2$ tenants are skewed toward the 10tps (L) SLO. However, in this case, the cost difference between the locally-optimized and the global-optimized solutions is never over 3%.

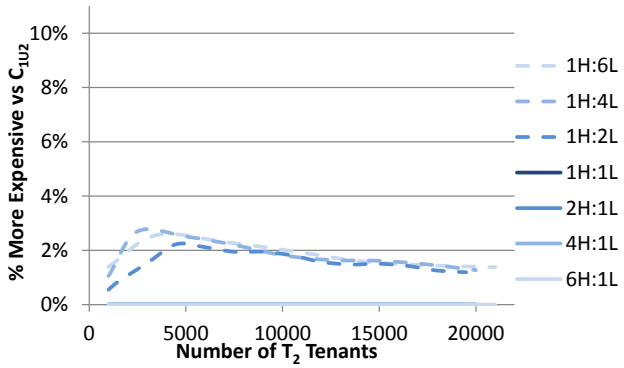
Next, let us consider the analysis above again, but with a lower price-point for the diskC server SKU (30% less than the ssdC SKU, see Section 2.4.1). We continue to use the 100tps, 10tps SLOs as we did in Figure 2.16. The results of our analysis for these three scenarios are shown in Figure 2.17. Again, in each sub-figure, we varied the $T_2$ population as we did in Figure 2.16; now, Figure 2.17(a)-(c) correspond to $T_1$ tenants with 1H:4L, 1H:1L, and 4H:1L SLO ratios respectively.

The rule of thumb we put forth in our prior discussion for Figure 2.16 continues to hold for the results shown in Figure 2.17. In Figure 2.17(a), the 10,000 $T_1$ tenants had an SLO ratio of 1H:4L. Again, we see that the characteristic of the tenant set $T_2$ that causes the largest cost difference between the locally-optimal and globally-optimal solutions is an SLO ratio skewed towards the high performance (100 tps) tenants (the solid curves). Since the diskC servers are now less expensive than in Figure 2.16(a), this results in cost differences in Figure 2.17(a) that are greater than in Figure 2.16(a).

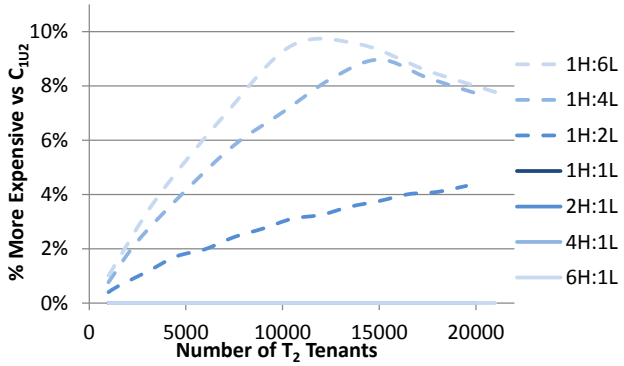
Similarly, in Figure 2.17(c) (where the $T_1$ tenants have an SLO mix of 4H:1L), we see that the cost




(a) $T_1$ tenants with 1H:4L SLO ratio




(b) $T_1$ tenants with 1H:1L SLO ratio




(c) $T_1$ tenants with 4H:1L SLO ratio

Figure 2.17: The relative cost difference between the two **locally-optimized** solutions, $(C_1 + C_2)$, and a single **globally-optimized** solution, $C_{1\cup2}$. The tenants have either "H" (100tps) or "L" (10tps) SLOs. There are 10,000 $T_1$ tenants. We varied the size of the $T_2$ tenant set and also the ratio of "H" and "L" tenants in each sub-figure. The diskC server SKU is 30% cheaper than the ssdC server SKU.

differences are also larger than the results shown in Figure 2.16(c). Here the 1H:2L, 1H:4L, and 1H:6L $T_2$ scenarios result in locally-optimal solutions that could be over 9% more expensive than the globally-optimal solution. As in Figure 2.16(b), the case where the $T_1$ population was evenly skewed (1H:1L), we see that the price difference is quite low and again is never above 3%.

**Bounding the Cost of Migration for Global-optimization**

Now that we have seen that there can be significant cost differences between having two locally-optimized solutions versus one globally-optimized solution, we can analyze the bounds on the migration costs.

Recall Equation 2.1 (the cost comparison model from Problem Statement 2): $C_1^w + C_2^w > C_{1\cup2}^w + \Sigma_{\forall\tau_i\in\pi} g(to(\tau_i), from(\tau_i))$, where $C_i^w$ is the cost of the optimized solution for tenant set $T_i$ over the time span $w$, and $\pi$ is the set of tenants that is migrated as a result of global-optimization. For this discussion, let us simplify our model so that all the tenant migrations cost the same amount $\gamma$, i.e. $g(to(\tau_i), from(\tau_i)) = \gamma, \forall\tau_i \in \pi$. Therefore, we can simplify our migration cost in Equation 2.1 to the number of migrations, $|\pi|$, multiplied by the migration cost, $\gamma$, and bound the re-optimization migration cost as follows:

$$(C_1^w + C_2^w - C_{1\cup2}^w > |\pi|\gamma) \tag{2.2}$$

We can apply this equation to our results in Figures 2.16 and 2.17. First, let us consider Figure 2.16(a), which corresponds to $T_1$ tenants from the scenario SC13 (10,000 tenants, 1H:4L, see Table 2.2). Now, consider the case when $|T_2| = 5000$ with a ratio of 4H:1L (in Figure 2.16(a)). In this case, the locally-optimized solutions are around 5% more expensive than the globally-optimized solution. If we consider $w = \infty$, then $C_1^w + C_2^w - C_{1\cup2}^w = 66528$.

Now, to consider migration, we have to construct the set $\pi$. To compute the set $\pi$, we first identified server SKUs from the $T_1$ solution where more tenants of a given SLO class were scheduled onto the SKU than the $T_{1\cup2}$ solution. Then, we count the tenants that need to be migrated away from this server SKU so that its tenant scheduling policy matches the globally-optimized solution. For example, if the $T_1$ solution requires 94 ssdC servers with a (20H, 38L) scheduling policy while the $T_{1\cup2}$ solution requires 274 ssdC servers with a (22H, 33L) scheduling policy, then we calculate that $94 \times (38L - 33L) = 470L$ tenants need to be migrated. We only count the tenants that need to be migrated **away** from the server and ignore tenants that need to migrate **to** a server to avoid double counting. After we count the migrating tenants from the $T_1$ solution, we do the same for the $T_2$ solution.[6]

[6] We acknowledge this approach may not be optimal and is a complex problem in its own right. It is the focus of future work.

Following the step above, we find that $|\pi| = 8285$, thus $\gamma$ must be less than \$8.03 (since $66528 > 8285\gamma$). Thus, if the cost of migration is greater or equal to \$8.03, then it is **not** worthwhile to globally-optimize all the tenants. Since \$8.03 is the bounding cost of migrating our 1GB tenant, this suggests that it may be advantageous to re-optimize all the tenants if we are faced with this decision only once ($w = \infty$).

On the other hand, say the provider knows that $T_2$ is active for 12hrs and then inactive for the subsequent 12hrs every day (e.g., a diurnal pattern). Thus, $w = 12hrs$, and $C_1^w + C_2^w - C_{1\cup2}^w = 1.28$. Now, if $|\pi| = 8285$, then $\gamma < \$0.0001$ for migration to be cost effective ($1.28/8285 = 0.0001$) This means that globally-optimizing every 12hrs is only feasible when migration can be done at a very low cost.

As another example, consider Figure 2.17(a), where the $T_1$ tenants comes from the scenario SC16 (10,000 tenants, 1H:4L, diskC is 30% less expensive than ssdC). Again, consider the case where $|T_2| = 5000$ with a ratio of 4H:1L. In Figure 2.17(a), the cost gap is around 9%. If we consider $w = \infty$, $C_1^w + C_2^w - C_{1\cup2}^w = 100656$. Using the method described above, $|\pi| = 6012$, and $\gamma$ must be less than \$16.74 (since $100656 > 6012\gamma$). This is an even looser bound on the migration cost than our previous discussion, albeit for a lifetime time span of $w = \infty$. But, if we consider $w = 12hrs$, $C_1^w + C_2^w - C_{1\cup2}^w = 1.94$. and we find that $1.94/6012 > \gamma$ and so $\gamma$ must be lower than \$0.0003. This migration cost bound is 3 times greater than the previous example, which *may* help provide provide some slack to globally-optimize all the tenants every 12hrs.

To summarize, in this section, we have discussed how (i) tenant populations that vary in size, and (ii) tenant populations that vary in SLO make-up can change the way we apply our optimization framework. Intuitively, optimizing two independent sets of tenants results in a solution that is more costly than optimizing across the combined set of tenants. A globally optimal solution may be more cost effective (than the independent optimized solutions), but to dynamically switch to the global optimal solution required consider tenant migration costs. Using the method described in this section, we can incorporate the migration cost to determine if the data center should switch to the globally-optimal solution.

### 2.4.4 Discussion

While the focus of this chapter is on performance SLOs in a DaaS, we have not discussed the impact of tenant replication (a solution for currently prevalent uptime SLAs) on our performance models. While data replication may improve performance for read-mostly workloads, maintaining replica consistency under update-heavy OLTP workloads places additional demands on the resources of DaaS providers. A careful study of how to deal with replica consistency and availability while providing performance SLOs is beyond the scope of this study, but we sketch an initial method to deal with this issue.

For our framework to handle replica updates, we can modify the benchmarking method that is used to determine the SKU performance characterizing function (Section 2.3.2) to account for the extra work that is needed to maintain replica consistency. For example, instead of measuring tenant performance on a single server as we have done, we would measure the tps observed by a tenant whose replicas are placed on $r$ servers and maintained via eager or lazy updates. The functions obtained from such a benchmark could be used as constraints to the optimization problem defined in Section 2.3.3.

Using our framework, we can pose another interesting question: given a cluster with a specific composition of hardware SKUs, what (performance) SLOs can the DaaS provide agree to, so that it maximizes the number of tenants that can fit on this cluster? For this question, we need to formulate a new objective function that optimizes for $max(|T|)$ in Problem Definition 1 where $T$ is the set of all tenants. Our other constraints would remain the same as specified in Problem Definition 1.

We note that in calculating the amortized monthly costs, we have not accounted for run time energy costs or amortized infrastructure cost (e.g., for the building, networking equipment, and associated power and cooling equipments). However, these can be accommodated in our framework (provided there were an accurate model to compute these costs for each SKU) by simply adding these costs to the amortized monthly cost that we use in this study.

Finally, in this chapter we have shown an explicit benchmarking approach for understanding the effects of mixing SLO classes and tenants. However, our framework is modular in that it is possible to leverage other analytic approaches that predict the impact of mixing tenants with different workloads and

SLOs [30, 55].

## 2.5 Summary

This work presents the first study of a cost-optimization framework for multi-tenant performance SLOs in a DaaS environment. Our framework requires as input, a set of performance SLOs and the number of tenants in each of these SLOs classes, along with the server hardware SKUs that are available to the DaaS provider. With these inputs, we produce server characterizing models that can be used to provide constraints into an optimization module. By solving this optimization problem, the framework provides a hardware provisioning policy as well as a tenant scheduling policy for the selected server SKUs. We have evaluated our framework, and shown that in many cases a mixed hardware cluster is optimal. We have also explored the impact of simpler hardware provisioning and tenant scheduling policies. Additionally, we have also shown how our framework can be extended to deal with some dynamic changes in the workload mix.

To the best of our knowledge, this is the first study to formulate a new problem of performance-based SLOs for DaaS, presenting a framework for thinking about this problem, presenting an initial solution, and evaluating this initial solution to show its merits.

To limit the scope of our study, we have made some simplifying assumptions on aspects such as performance metrics, tenant workload, and multi-tenancy control mechanism. Relaxing these assumptions provides a rich direction for future work. One direction for future work is to include the impact of replication and load-balancing in our framework, perhaps building on the ideas presented in Chapter 6. Additionally, while our experimental evaluation uses average performance as an SLO metric, it could be extended to include variance as well (as implied by the use of random variables in Definition 2.3.2). Imbalanced load or flash-crowd effects could be modeled in our framework as additional tenant classes with high performance requirements – this would produce a hardware "over-provisioning" policy to deal with these effects. If workload spikes are detected in practice, tenants could be dynamically re-scheduled on these extra machines to maintain performance objectives. In addition, while the tenant classes used in this study have different memory and disk requirements, other workloads should be considered as well. Finally, in our framework we have taken an approach of explicitly benchmarking the tenant workload classes and mixes,

but our framework could be extended to take a more analytical approach that could predict the impact on performance of a different workload mixes, perhaps by using multi-query optimization-based approach to estimate the impact on performance.

# Chapter 3

## The Costs of Using Non-Traditional Server Hardware

Data center deployment is a big investment for any enterprise. Such data center costs are ultimately factored into the company bottom line through the monthly amortized costs. The largest proportions of this monthly total cost of ownership (TCO) are the server costs, power distribution and cooling, and the actual energy costs. Combined, they make up 88% of the total monthly TCO (3 year server, 10 year infrastructure amortization) [81]. The energy and energy related costs can account for a third of the monthly TCO, and several studies have shown that these costs are projected to increase as a proportion of the monthly TCO [23,34]. Consequently, there has been expanding interest in reducing data center power costs [21] (see Chapter 8).

In this chapter, we study the price/performance characteristics of parallel scale-out clusters where our notion of price includes server costs *and* direct energy costs thereby accounting for roughly two thirds of the monthly TCO.

To reduce the monthly TCO, a growing number of recent studies have focused on redesigning data center server clusters with low-cost, low-power "wimpy" nodes [13,145]. The argument for wimpy nodes is that they are relatively well-balanced [147,162]. With low-end CPUs and the use of low-power components, these nodes are claimed to be more energy-efficient. However, low-end nodes lag far behind traditional nodes in performance. Therefore, a small cluster of traditional nodes must be replaced by a larger cluster of low-end nodes.

These previous studies have generally come to the conclusion that a scale-out deployment of large clusters of "wimpy" nodes is the most effective solution. However, these published results focused on sim-

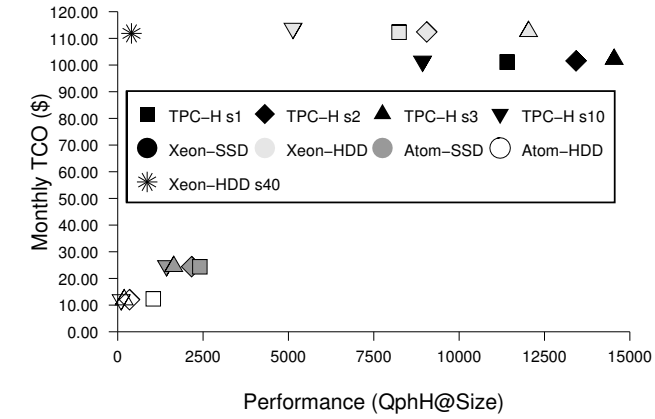
ple key-value workloads [13] and web-search environments [145], where near-perfect performance scaleup complements the poor performance of individual wimpy nodes. In [13], the authors admit that their approach targets *"data-intensive, computationally simple applications"*. Consequently, their system is described as a *datastore* instead of a *database* to emphasize that they do not provide transactional and relational interfaces. Such simple data lookup environments lack the complexities of more complex data processing workloads, and the focus of this chapter is to consider if the same conclusions apply for complex data processing workloads.

Recently, proponents of wimpy node clusters theorize that such architectures will also be able to handle more complex workloads such as sorting [173]. However, these arguments are based on single node, performance/Watt comparisons of Atom and Nehalem-based servers. Without the analysis of how performance is affected with increasing scale-out, which is the focus of this work, such arguments are largely based on theoretical ideal performance. The purpose of this work is to show that for complex database workloads, previously observed results show that parallel data processing overhead is significant enough to dilute the benefits of large wimpy node clusters. This work directly leads to Chapter 4 where we explore cluster designs that incorporate both low-power, "wimpy" nodes and traditional, "beefy" nodes into the cluster.

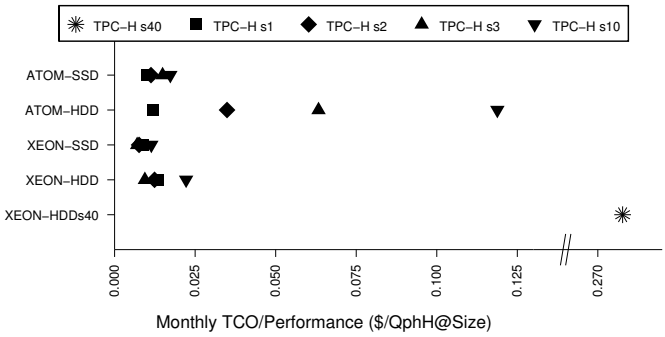
## 3.1 Motivating Illustrative Experiments

There are two factors that come into play when evaluating the efficiency of traditional versus low-end cluster deployments: (1) The individual node price/performance when processing the partitioned data; and (2) The effects of diminishing returns when undergoing *parallel scaleup* (constant response time when the data size and computing resources increase proportionally) due to startup-interference-skew factors (see Section 3.3 for further discussion).

Consider Figure 3.1(a), where our price and performance metrics are plotted for various types of nodes and different TPC-H workloads on a commercial DBMS. The amortized monthly TCO is calculated as the sum of the amortized node cost as well as the monthly energy costs for the node continuously running TPC-H (see Section 3.4). (Here, for the node cost, we have simply divided the purchase cost over 36




(a)




(b)

Figure 3.1: (a) Amortized Monthly TCO (includes energy costs) as a function of Performance over various hardware configurations and TPC-H scale factors. (b) Price/Performance metric of Amortized Monthly TCO ($) and Performance (QphH@Size) System details can be found in Section 3.4.1. QphH@Size is the unit for the TPC-H Power Test.

months.) Performance is provided by the TPC-H Power Test [168]. 'Atom' and 'Xeon' represent Atom (low-end node) and Xeon (traditional node) processors respectively. Both types of nodes were then outfitted with either SSD or mechanical HDD disks (see Section 3.4.1 for detailed node specs). Figure 3.1(a) plots costs versus performance for different node configurations and TPC-H scale factors. There are two interesting patterns to observe. First, SSDs provide better performance than regular disks. Second, the Atom nodes lag far behind the Xeon nodes in performance, which means that we must deploy significantly more Atom nodes in a cluster to equal the performance of a Xeon cluster.

We combined the two metrics in Figure 3.1(a) into a single price/performance metric as shown in

Figure 3.1(b). When we consider the more expensive SSD configurations of the Atom and Xeon, we notice that all their results for various TPC-H scale factors are tightly clustered together at the cheapest end of the price/performance spectrum. Even though the purchase cost for the systems increase with SSDs, the performance increase outpaces the cost increase and so we see better price/performance (similarly seen in [5]). This suggests that if we partition the TPC-H workload and use multiple Atom nodes, we can achieve similar performance as a Xeon node at a similar price point (assuming perfect parallel scaleup).

For example, assuming ideal scaleup, we could run a TPC-H scale 10 workload partitioned on 5 Atom-SSD nodes (i.e., 2GB per node) in the same time as a scale 2 workload on a single Atom-SSD node. This can also be done with even smaller partition sizes per Atom-SSD node, thereby changing the cluster size. If we used a 1GB partition size, we would need 10 Atom nodes. Since the measured performance for TPC-H scale 1 on the Atom is greater than 1000 QphH, ideal scaleup would infer a cluster performance greater than 10000 QphH. This would outperform a single Xeon-SSD running TPC-H scale 10 (9000 QphH).

## 3.2 Technical Contributions

The problem is that parallel database research has already shown two decades ago that the ideal proportional scaleup discussed in our motivating example, is far from guaranteed [53]. In this chapter, we will show published examples of such deviations. Essentially, replacing traditional clusters with increasing numbers of low-power nodes may result in diminishing returns in performance and as such, cost.

In this chapter, we will present real price/performance results for a traditional server versus a modern low-power wimpy node (detailed results in Section 3.4.3- 3.4.4). With these results, the interesting problem that we focus on in this chapter is how these results fit into various scaleup models. Since the parallel scaleup for a given workload is dependent on the parallel software system and node hardware configuration, we need to examine the effects of different scaleup models on a low-power cluster versus a traditional cluster. To this end, we will present several real scaleup profiles from various published parallelized data processing systems (see Section 3.3) and use our measured results to show that traditional server clusters may be more cost effective than a massively scaled-out wimpy node clusters.

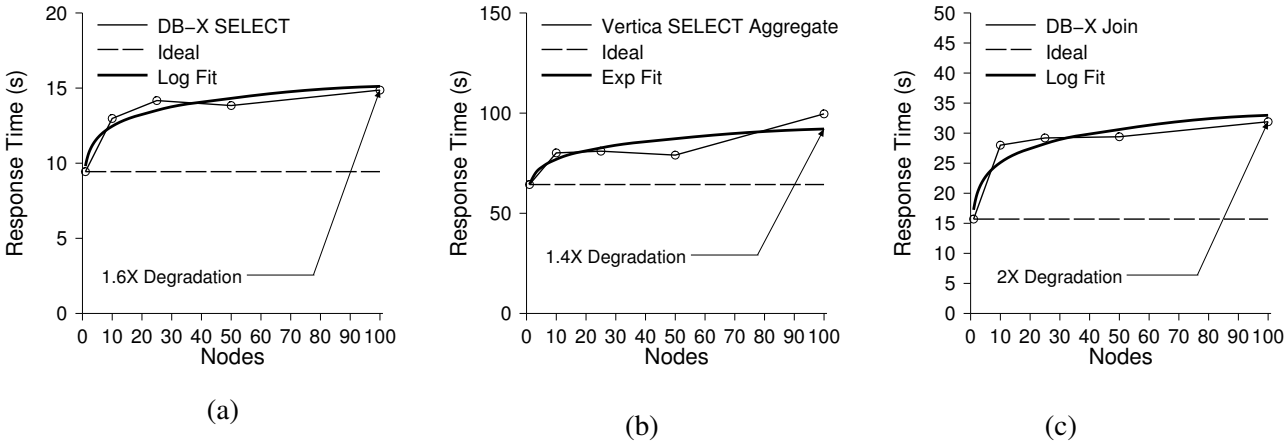



Figure 3.2: (a) DB-X running a 535MB/node SELECT query [135]. (b) Vertica running a 20GB/node SELECT aggregate query [135]. (c) DB-X running a join query (large table 20GB/node, small table 1GB/node) [135].

## 3.3 Background – Parallel Databases and Scaleup

This section recalls the lessons learned from more than two decades of parallel database research. Specifically, we discuss the factors that impact the ability to achieve ideal parallel scaleup when deploying larger clusters.

To start, we clarify the parallelism goals that often get misused with the overloaded term cluster "scale-out". Often, scale-out is used as a blanket term for increasing the size of a parallel data processing cluster to achieve ideal performance benefits. However, as defined in [53], this idea can be divided into two distinct goals: linear **speedup** and linear **scaleup**.

For example, given 100 machines processing 1TB of data in 1min, if the parallel system has the ideal linear *speedup* property, 400 machines could process the same 1TB in 15sec. On the other hand, given the same cluster nodes, a parallel system exhibiting ideal linear *scaleup* could process 10TB with 1000 machines in 1min. Scaleup can be further broken up into transactional or batch which essentially describe throughput or latency-based definitions of performance respectively.

DeWitt and Gray [53] identified three main threats to successful scaleup behavior of a parallel DBMS: startup, interference, and skew. **Startup** costs refer to overhead in time needed to start the parallel processing jobs. For example, if synchronization is required across hundreds of nodes for starting a short query, then

this cost can make up a significant fraction of the total response time. The impact of such costs often diminishes with long running queries. **Interference** costs are those caused by processes that need to share resources such as memory or disk. Finally, the last impediment of **skew** refers to the behavior that with increased parallelism and decreasing per node computation time, the the variance of node computation time can start to dominate the average runtime.

Consider Figure 3.2 where we have plotted the scaleup profiles of a simple selection query, another selection query but with an aggregate operation, and a third query with a join and an aggregation operation. These queries are taken from the recent paper by Pavlo et al. [135]. We present these real scaleup results, not for comparison purposes, but to illustrate the point that in practice, scaleup is often not ideally proportional for complex data processing workloads.

Figure 3.2(a) reports scaleup response time for a commercial parallel database, DB-X, running a simple SELECT scan where each node had 535MB partitions shows that there is significant response time degradation as the cluster size increases.

```
SELECT * FROM Data WHERE field LIKE '%XYZ%';
```

Table 'Data' has two rows 'key' and 'field' with sizes 10 and 90 characters respectively. Even for a workload as simple as a scan, the response time degradation at a 100X scaleup factor is 1.6 times worse than ideal.

Figure 3.2(b) shows an aggregate selection query with a scaleup from one node to 100 nodes. On a different commercial database, Vertica [175], the following query was run on a 233B wide table:

```
SELECT sourceIP, SUM(adRevenue)
FROM UserVisits GROUP BY SUBSTR(sourceIP,1,7);
```

Each node stored 20GB partitions of the table. This result showed that a different commercial parallel database also exhibits diminishing returns when the environment is scaled up; at 100X scaleup, there is a

1.4X performance degradation from the ideal performance.

Finally, Figure 3.2(c) shows a scaleup model for a complex join query on DB-X between the 'UserVisits' table in the above query, to a 108B table (1GB/node). The query also has an aggregation and date range predicate:

```
SELECT INTO Temp sourceIP, AVG(pageRank) as avgPageRank,
       SUM(adRevenue) as totalRevenue
FROM Rankings AS R, UserVisits AS UV
WHERE R.pageURL = UV.destURL
       AND UV.visitDate BETWEEN Date('2000-01-15')
       AND DATE('2000-01-22')
GROUP BY UV.sourceIP;
```

For DB-X, at 100X scaleup, the response time degradation is 2X worse than the ideal scaleup performance. Full details of the queries can be found in [135].

All three scaleup models show that there is a large startup cost between the one node 'cluster' and any multi-node cluster. As a result, we have fitted logarithmic models to the DB-X results and an exponential model to the Vertica result to account for the initial drop in performance. The chosen models provided the best correlation coefficient of various regression models we applied.

The core point of this discussion is to show that scaleup performance is not ideally constant for complex data processing workloads; in which case *wimpy node scale-out to save energy and purchasing costs may not be more cost effective than traditional servers if equivalent performance is sought*. Equivalently, in real (as opposed to ideal) scaleup environments, price/performance degrades as the scaleup factor is increased (i.e., it gets more expensive to achieve the same level of performance). Next, we will show experimental results incorporating our price/performance results in Figure 3.1 and analyze the cost effectiveness of traditional clusters versus low-power/low-cost clusters.

## 3.4 Experimental Evaluation

In this section, we discuss our experimental results which includes energy measurements of our nodes under different workloads using a commercial database system as well as scaleup experiments using published scaleup results. We start by describing the node characteristics and costs, followed by the measurement methods, and finally we present our results.

### 3.4.1 Server Costs and Specifications

In our tests, we use an Atom node (wimpy) and a typical high-end server-class Xeon node. Both ran the same commercial database system on Windows 7 Professional (Atom) and Windows Server 2008 (Xeon) which share the same kernel [149].

**Atom Node:** The Atom node had a dual core, hyper-threaded Intel D510 Atom processor with accompanying Intel motherboard ($80). The motherboard was filled with the maximum allowed 2x2GB GSkill DDR2 memory ($95). Our power supply was an 80plus certified Corsair VX450 ($65). We tried two different power supplies units (PSU); a cheap 120W PSU and a 450W Corsair. Even though the Corsair can provide almost 4X more power than the cheap PSU, we found that the power drawn by the system with the Corsair was almost half of that when using the cheap PSU. Given this, we chose the larger, but more efficient Corsair. We had two disk configurations: (1) the **SSD** configuration had an OCZ SATA2 64GB drive ($200) for the OS and DBMS applications and an Intel X-25E 32GB drive ($383) as the database data storage drive; (2) the mechanical **HDD** configuration used two WD Caviar Green SATA2 32MB Cache drives each with 500GB storage where both drives were used for database data storage ($120). The mechanical HDD configuration costs $360 while the SSD configuration costs $823.

**Xeon Node:** The Xeon node is an HP Proliant DL380GS with two quad core Xeon E5410 processors and 16GB of memory. The server has eight 146GB 10K RPM SAS drives with two in RAID1 for the OS and DBMS applications, another two in RAID1 for the DBMS log, and four drives for database data storage. This configuration cost approximately $3500. Each SAS drive can be purchased at $270 a drive. Our SSD configuration consists of removing the four data drives and replacing them with two Intel X25-E

SSDs (priced as above). The server cost now is approximately $3186. This drop in cost between the high capacity SAS configuration and SSD configuration is similar to [5].

To keep this study manageable, we only explored a limited number of IO hardware configurations. Tuning this IO system for price, performance and energy is an interesting topic for future research, and beyond the scope of this study.

### 3.4.2 Energy Measurement

AC current was measured at the wall outlet using a Fluke i200s AC current clamp (1.5% accuracy at 0.5A). The Fluke clamp was connected to an NI USB-6008 Multifunction DAQ and collected using NI's LabView, sampling at 1KHz. RMS current was calculated using a sliding window of 16 sample points (1 period) given an AC frequency of 50Hz. The RMS voltage was measured at 118V. Power was calculated as the product of the RMS current and the RMS voltage. Finally, energy consumption was calculated by summing the time discretized real power values over the length of the workload.

### 3.4.3 Single Node TPC-H

The raw energy and response time data that was used to create Figure 3.1 is shown in Figure 3.3. In this figure, we plotted the measured response time to complete the TPC-H Power Test against the energy consumed by the node during the run. There are a number of interesting features to note in Figure 3.3. First, for any given scale factor of TPC-H, the Atom node with the SSD configuration always consumes the least amount of energy. This is unsurprising given the low voltage of the Atom processor and SSDs. Second, the Xeon node with any storage configuration always finishes faster than the Atom-SSD node. This is also unsurprising given that the Xeon node has eight cores while the Atom only has two (four w/hyper-threading) and the Xeon node also has four times the memory of the Atom node.

Using these results, we calculated the amortized monthly total cost of ownership (TCO) using the node purchase prices (see Section 3.4.1) and a $0.07kWh data center energy cost [80]. Furthermore, we calculated the TPC-H Power@Size metric using the definition provided in [168]. Figure 3.1 shows our single node TPC-H results after transformation to a price/performance metric.

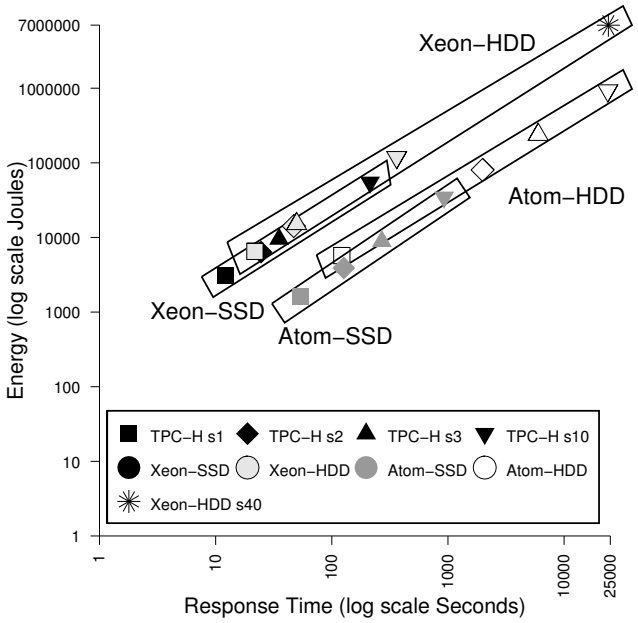



Figure 3.3: Raw measured response time and energy consumption of the TPC-H Power Test under various scale factors. TPC-H scale factors are represented by the shapes square, diamond, upward triangle, and downward triangle for scale factors 1, 2, 3, and 10 respectively. Shading of shapes correspond to different system configurations.

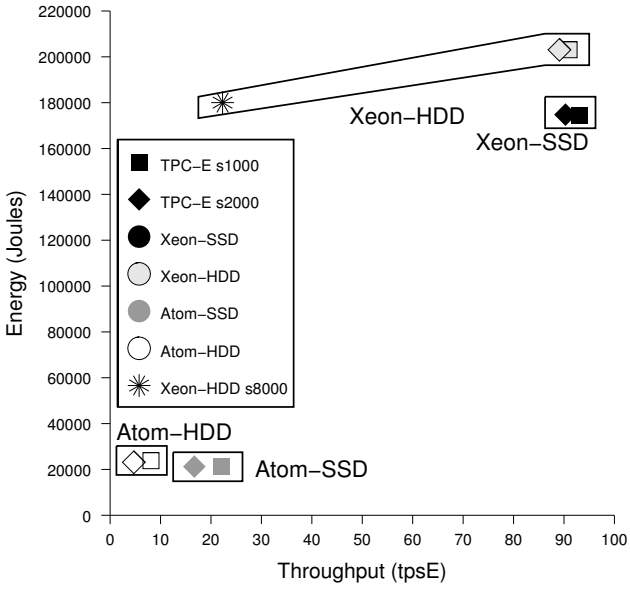



Figure 3.4: Raw measured throughput and energy consumption of a 10min window running TPC-E.

### 3.4.4 Single Node TPC-E

While TPC-H is a benchmark for decision support system (DSS), we also wanted to understand the performance and energy consumption profiles for OLTP workloads. Figure 3.4 shows a TPC-E throughput

versus energy consumption plot similar to Figure 3.3. Here we have measured the throughput (in transactions per second/E – tpsE [169]) over a 10min period. This measurement period was preceded by a 10min warm up time where both systems stabilized. The energy measurements represent the energy consumption over 10min. Here we notice that the Xeon nodes always provide very high tpsE for both 1000 and 2000 scale factors because the system is essentially CPU bound at scale 1000 and 2000. The Atom nodes have significantly lower throughput but also about an order magnitude lower energy consumption.

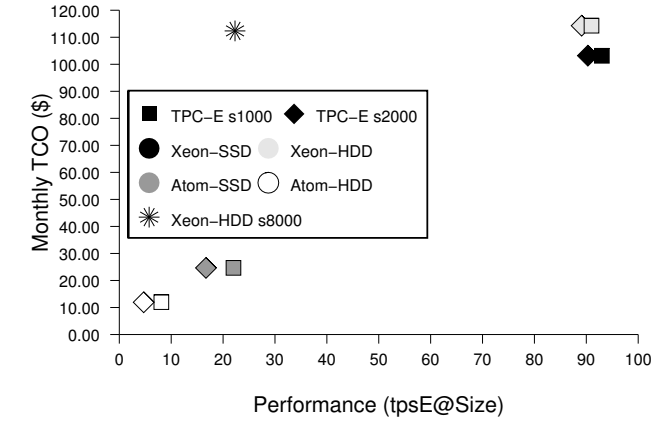
Figure 3.5 shows the similar price/performance plot as Figure 3.1, but now for TPC-E. We notice that while Figure 3.4 shows that there was massive differences in energy consumption between the Xeon nodes and the Atom nodes, Figure 3.5 shows that the actual price/ performance values for the nodes is only factors in difference and in the case of scale 1000, the Atom-SSD node has the same price/ performance as the Xeon nodes.

TPC-E is a workload that was designed to reflect a more realistic benchmark compared to TPC-C, and is well-known to not have the easily-partitionable characteristics of TPC-C. Thus the impediments to perfect scaleup (see Section 3.3) will likely affect the scale-out story for TPC-E like workloads. In this discussion, we will not consider TPC-E further, and focus only on the DSS TPC-H workload.

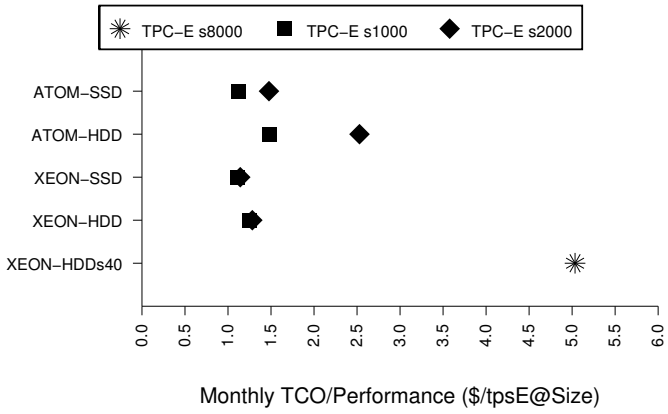
### 3.4.5 TPC-H Parallel Scaleup

Now, we examine the price and performance metrics for various scale-out clusters built from either traditional server nodes, or low-power Atom nodes. As we have seen in Figure 3.2(a-c), since scaleup characteristics are largely determined by the query workload, we apply these example scaleup models of Figure 3.2 to examine cluster price/performance. While these models are from various parallel systems and hardware, *the goal here is to show how the scale-out of our nodes would affect scaleup price/performance given various possible scaleup models*.

**Modeling Cluster Performance** Here we will describe the way we have applied the published scaleup models to our single node measurements in order to get cluster performance for a parallel data processing workload. First, the response time models we presented in Figure 3.2 are converted to give scaleup factors. Consider this example that illustrates how this is done: given that the DB-X Join response time model shows




(a)

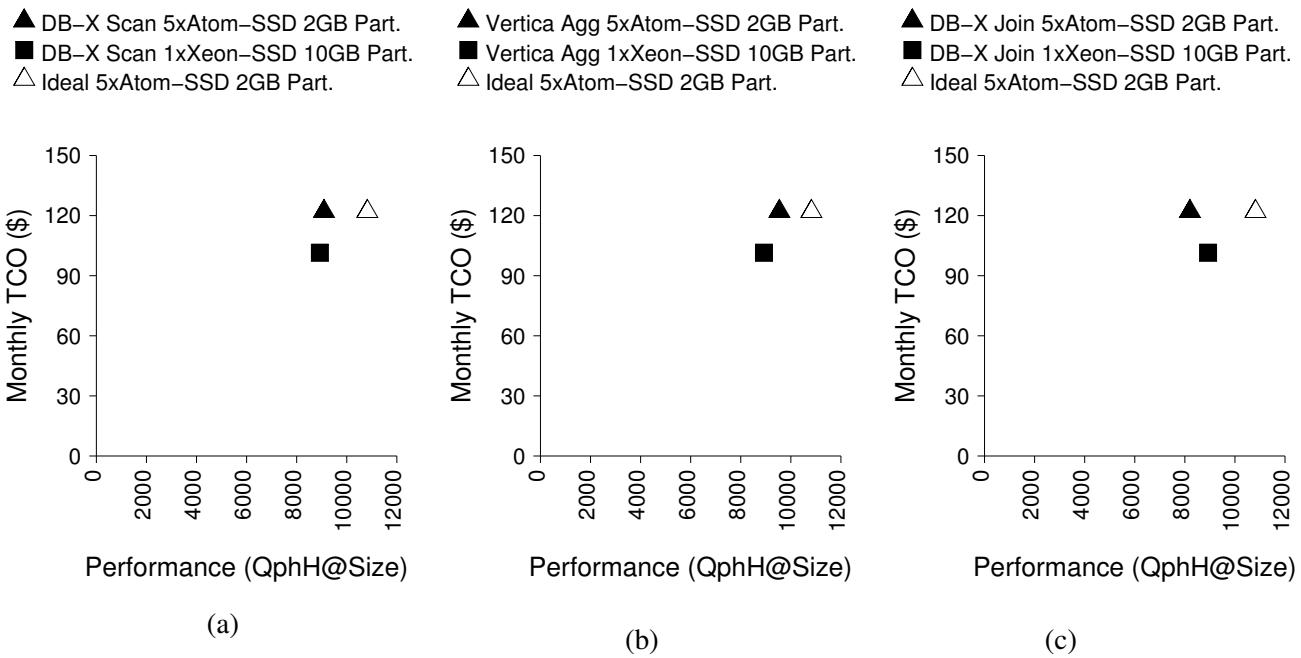



(b)

Figure 3.5: (a) Amortized Monthly TCO (includes energy costs) as a function of Performance over various TPC-E scale factors. (b) Price/Performance metric of Amortized Monthly TCO ($) and Performance (tpsE@Size)

that using two nodes the workload response time will be 1.14X the response of one node, then the scaleup factor will be $2 \times 1/1.14 = 1.75$. This scaleup factor can then be used with our measured performance data to provide cluster performance.

Performance for a $xy$ GB TPC-H dataset using $x$ nodes at $y$ GB partition each is calculated as $P(y)M(x)$ where $P(y)$ is the performance for the single node (Section 3.4.3) running TPC-H at scale $y$ and $M(x)$ is the modeled scaleup factor given a cluster of $x$ nodes (using the models in Figure 3.2). The ideal scaleup factor for $x$ nodes is $M(x) = x$. For example, given a single node 10GB performance value




(a) (b) (c)

Figure 3.6: Parallel scaleup of a 10GB TPC-H workload on Atom and Xeon clusters using different published scaleup models (5 Atom-SSD nodes and 1 Xeon-SSD node respectively). Atom nodes have 2GB partitions and the Xeon node has the entire 10GB dataset. For all figures, the effects of an ideal constant scaleup is shown. (a): DB-X Scan scaleup model (Figure 3.2(a)). (b): Vertica Aggregation scaleup model (Figure 3.2(b)). (c): DB-X Join scaleup model (Figure 3.2(c)).

of $P(10) = 9000QphH$, the ideal cluster performance for $M(2)$ nodes is $18000QphH$ while the modeled performance is $9000QphH \times 1.75 = 15840QphH$ for the Join model.

**Wimpy Clusters vs One Xeon**

Here we examine the effect of the startup costs, seen in Figure 3.2, on the cost of the Atom-SSD clusters as compared to a single Xeon-SSD node (which has no startup or any other parallel scale-out).

In Figure 3.6(a-c), we have applied each of the scaleup models of Figure 3.2 to the results shown in Figure 3.1. We have also included the data points for the ideal (constant) scaleup model. Since Figure 3.1 has shown us that outfitting nodes with SSDs typically decreases price/performance for both Atom and Xeon nodes, for our analysis here, we have used the SSD configurations of our nodes. To begin, consider Figure 3.6 which shows the results of scaleup when one Xeon node is compared to a cluster of Atom nodes.

For this discussion, we use the price and performance results for the Xeon-SSD node when running a TPC-H workload at scale 10. Based on the results shown in Figure 3.1(b), the Atom-SSD running TPC-H

scale 2 has a similar price/performance rating as the Xeon-SSD at scale 10. Figure 3.1(b) shows that the Atom-SSD at TPC-H scale 2 (diamond) and the Xeon-SSD at TPC-H scale 10 (downward triangle) both have a price/performance of $0.011.

To match the Xeon-SSD workload size, Atom-SSD cluster will be made up of 5 Atom-SSD nodes each with 2GB partitions. This means that the monthly TCO of the Atom cluster will be 5 times the single node monthly TCO (at TPC-H scale 2). If we had ideal scaleup, then this Atom-SSD cluster would also provide 5 times the performance of a single Atom-SSD node thereby retaining a price/performance of $0.011. However, Figure 3.2 shows that this ideal scaleup does not happen and we calculate the performance of such a cluster using the methods described above.

In Figure 3.6(a), we show the price as a function of performance for two setups when applying the DB-X Scan scaleup model of Figure 3.2(a). Since we are only using a single Xeon-SSD, there is no scaleup effects and the modeled price and performance is identical to the ideal. The Atom cluster price/performance is 18% worse than the Xeon node for a 10GB workload.

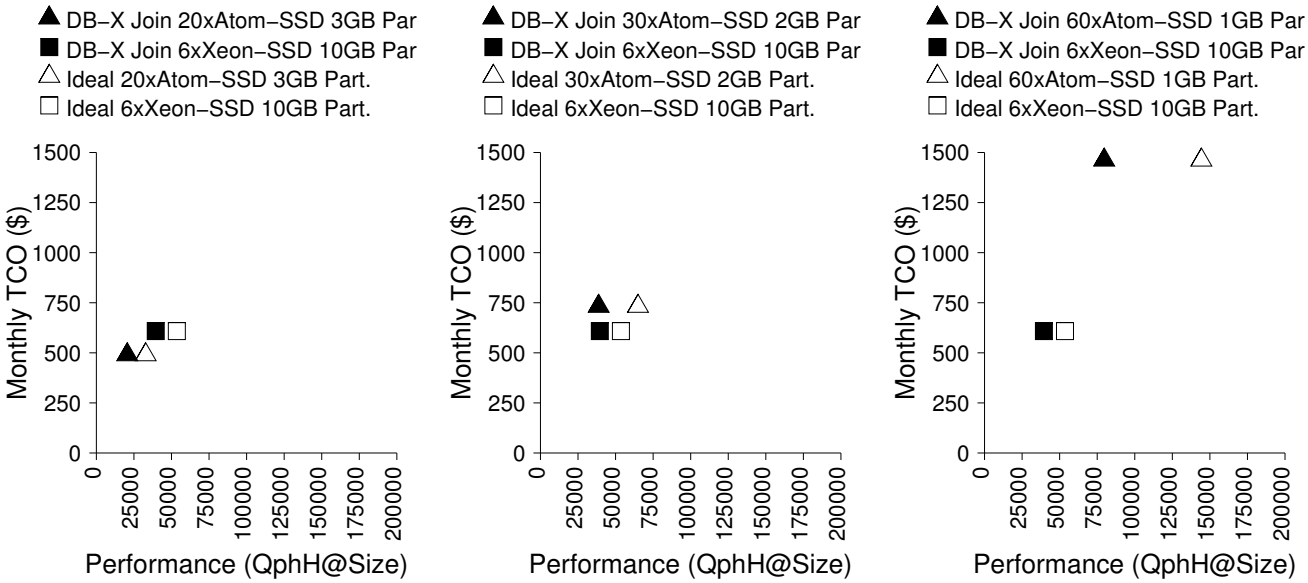
Similarly, for Figure 3.6(b) and (c), we have applied the Vertica Aggregate scaleup (Figure 3.2(b)) and DB-X Join scaleup (Figure 3.2(c)) respectively. The Atom cluster price/performance is 13% and 31% worse than the Xeon node for the Vertica Aggregate and DB-X Join models respectively. It is clear that if the scaleup behavior, such as those in Figure 3.2(a-c), has poor degradation, this will be reflected in the cluster performance.

The results in this section are for a single Xeon node and an Atom cluster. The next results are for multi-node Xeon clusters and larger Atom clusters.

**Wimpy Clusters vs Traditional Clusters**

The following results are for various levels of Atom-SSD scaleup for a 60GB TPC-H workload where the Xeon-SSD cluster is made of 6 nodes, each running 10GB partitions. We applied the DB-X Join scaleup model from Figure 3.2(c) as it represented the most complex workload of the three we discussed. Modeling was done as described previously above.

Figure 3.7(a-c) shows the price as a function of performance similar to Figure 3.6. In these figures, the amount of data (partition size) per Atom-SSD node decreases from 3GB (Figure 3.7(a)) to 1GB (Fig-




(a) (b) (c)

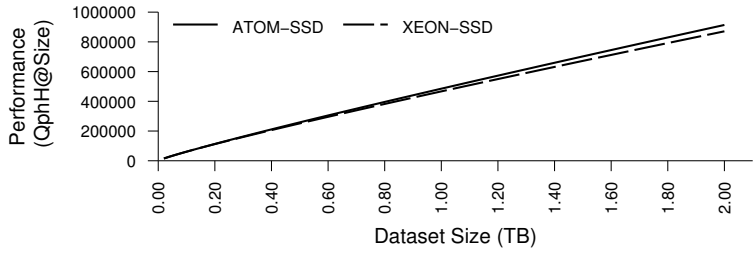
Figure 3.7: Parallel scaleup of a 60GB TPC-H workload on Atom (with different partition sizes) and Xeon (10GB partitions) clusters using the join scaleup model from Figure 3.2(c). (a) 3GB Atom-SSD partitions, (b) 2GB Atom-SSD partitions, (c) 1GB Atom-SSD partitions.

ure 3.7(c)).

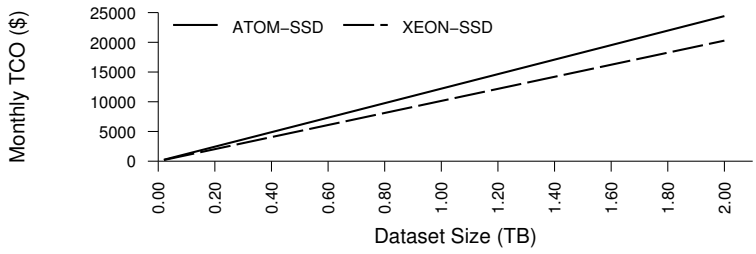
In the progression of decreasing partition size for the Atom cluster, as the partition size decreases, the size of the Atom cluster increases. First, we notice that the 20 node Atom cluster is cheaper than the Xeon cluster in Figure 3.7(a). However, its performance is about half of the Xeon-SSD cluster. If we consider the price/ performance, the 20 node Atom cluster is 55% higher than the 6 node Xeon cluster.

Next, in Figure 3.7(b), where the Atom cluster is 30 nodes large, both clusters have roughly equivalent performance. However, we notice that the cost of the Atom cluster is 20% higher than the Xeon cluster. In this case, the 30 node Atom cluster is 23% higher in price/performance than the 6 node Xeon.

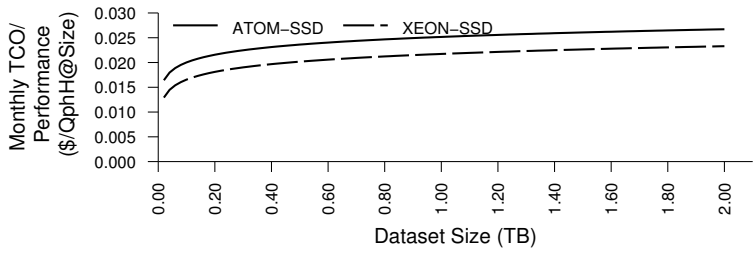
Finally, in Figure 3.7(c), we notice that as the Atom cluster increases with decreasing partition size, the performance increases. However, this is an effect of the increasing performance of the Atom-SSD as it works on a smaller, in memory dataset (at 1GB partition). It is quite clear in Figure 3.7(c), where the 60 node Atom-SSD cluster has higher performance than the 6 node Xeon-SSD cluster, that the cost to deploy such a wimpy node cluster is significantly higher than the Xeon-SSD cluster. While the Atom cluster's performance




(a)




(b)




(c)

Figure 3.8: Analysis of TPC-H workload: (a) Performance, (b) Price, and (c) Price/Performance for Atom-SSD and Xeon-SSD based clusters using the Join scaleup model (Figure 3.2(c)). Atom-SSD nodes have 2GB partitions and Xeon-SSD nodes have 10GB partitions.

is 2X better than the Xeon, it is 2.4X more costly. This translates to a 19% increase in price/performance over the Xeon cluster.

Since this analysis has held the Xeon-SSD cluster size constant while we varied the Atom-SSD cluster size, it is necessary to compare the cluster price/performance when both clusters vary in size.

Consider Figure 3.8, where we plot the (a) performance, (b) price, and (c) price/performance as a function of the dataset size for both clusters when the Join scaleup model is applied (Figure 3.2(c)). Here we partition the data so each Atom-SSD has 2GB partitions and each Xeon-SSD has 10GB partitions.

Figure 3.8(a) shows that as the clusters scaleup, the Atom-SSD cluster starts to exhibit higher performance than the Xeon-SSD cluster. This is because the Xeon cluster is growing and starts to become affected by the DB-X Join scaleup degradation. This performance difference in Figure 3.8(a) can be explained by the models in Figure 3.2 that show that parallel scaleup behavior flattens out as the cluster sizes increases. While the Atom cluster has slightly better performance with larger scaleup, Figure 3.8(b) shows that this is accompanied by higher cluster cost. Finally, Figure 3.8(c) shows the price/performance of both clusters under scaleup. It shows that the increase in performance and cost of the Atom-SSD cluster over the Xeon-SSD cluster is largely proportional.

**Discussion**

We have shown the effects of various scaleup models on two types of clusters running TPC-H workloads: an Atom-SSD cluster and a Xeon-SSD cluster. While these models are not directly drawn from a TPC-H workload, the purpose of using these models is to show how scale-out of different cluster architectures can be affected by different scaleup behavior. As such we have applied a variety of published scaleup models to the TPC-H measurements we collected.

With computationally simple workloads, such as those of [174], "wimpy" node clusters are claimed to be more effective than clusters made from traditional nodes. However, this analysis has shown that for data-intensive workloads (Figure 3.6, 3.7, and 3.8), large wimpy node clusters suffer from poor scaleup effects and are therefore potentially slower and a costlier solution than smaller Xeon clusters. The reason for this is because larger wimpy clusters are more affected by a diminishing return scaleup effect than a smaller traditional cluster.

Furthermore, small relative gains in performance or price/performance by a larger, low-power cluster over a smaller traditional cluster may not be worth the increase in mean-time-to-failure for the entire cluster [153, 154]. Providing fault tolerance over a large number of wimpy nodes requires replication and over-provisioning, thereby increasing the price/performance of such clusters.

Finally, increasing the cluster size by migrating over to Atom nodes requires substantially more network infrastructure. For example, if we assume that the wimpy node clusters will require 4X more nodes than the traditional cluster, then given a 16 node Xeon cluster that requires one 48 port switch, a wimpy 64

node cluster will require two 48 port switches. High performance network hardware cannot be sacrificed in wimpy node clusters and anecdotally we found this to be true in real deployments of Atom clusters. Optimistically assuming enterprise class 48 port 10 Gigabit switches cost $10,000 each, the additional amortized switch cost per wimpy node is $313 which doubles the cost of the spindle-based Atom node to $700 and increases the SSD-based Atom cost to $1150!

Our results suggest that there may be an interesting middle ground between wimpy Atom node clusters and traditional Xeon node clusters that lies with hybrid cluster deployments [43]. Such clusters made up of both wimpy and traditional nodes may be the most effective deployment. We pursue this avenue of research in the next chapter.

## 3.5 Summary

Our study presents evidence that for complex data processing workloads, a scale-out solution of a low-power low-end CPU-based cluster may not be as cost-effective (or produce equivalent performance) as a smaller scale-out cluster of traditional high-end server nodes. We have shown that depending on the scaleup characteristics of the query workload and the software system, poor scaleup behavior can occur when increasing the cluster size. Poor scaleup degrades the price/performance of a larger cluster. Thus, the parallel scaleup characteristics of the environment largely determines the feasibility of so-called “wimpy” node configuration for building clusters for such complex data processing workloads.

While our results suggest that wimpy node clusters are not suited for complex database workloads, it does open up the area of hybrid (heterogeneous) cluster deployment. Hybrid cluster deployment strategies, job scheduling, and scaleup analysis are studied in the following chapter.

# Chapter 4

## Energy-aware Parallel Data Processing I – Designing Energy Efficient Clusters

In recent years, energy efficiency has become an important database research topic since the cost of powering clusters is a big component of the total operational cost [67, 79, 97]. This trend will drive up the need for designing energy-efficient data processing clusters. The focus of this chapter is on designing such energy-efficient clusters for database analytic query processing.

As we presented in Chapter 3, an important problem regarding the energy efficiency of database clusters surrounds the classical problem of non-linear scalability in parallel data processing [53]. Recall, non-linear scalability refers to the inability of a parallel system to proportionally increase performance as the computational resources/nodes are increased. In that chapter, using prior work in the literature [135], we overlaid single node performance and energy measurements to extrapolate cluster price/performance ratings. In this chapter, we do an in-depth study on designing energy-efficient clusters with different node architectures.

### 4.1 Motivating Illustrative Experiments

First, we present two results that show how varying the cluster size and changing the cluster design can allow us to trade performance for reduced energy consumption for a single query workload.

In this chapter, we study the effect of this undesirable scalability phenomenon on the energy efficiency of parallel data processing clusters using a number of parallel DBMSs, including Vertica and HadoopDB. Our first result is shown in Figure 4.1(a) for TPC-H Q12 (at scale factor 1000) for Vertica. In this figure, we show the relative change, compared to a 16 node cluster, in the energy consumed by the cluster and the

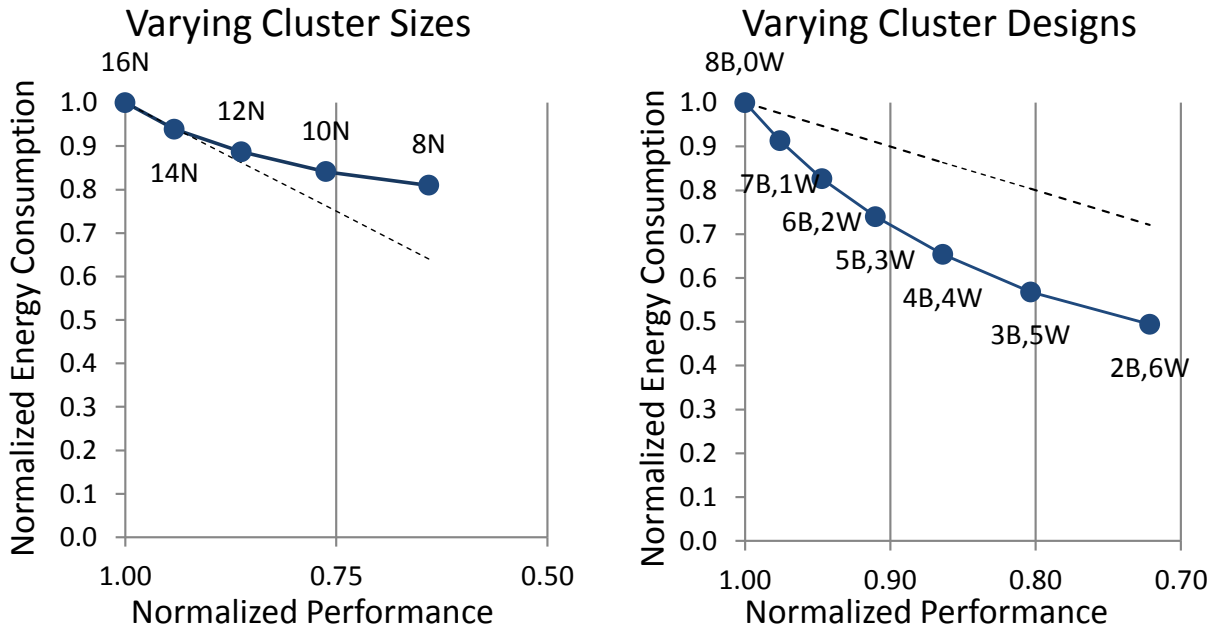



Figure 4.1: (a) Empirical energy consumption and performance results for Vertica running TPC-H Q12 (at scale factor of 1000). The dotted line indicates trading an X% decrease in performance for an X% decrease in energy consumed so that the Energy Delay Product (EDP= $energy \times delay$) metric is constant relative to the 16 node cluster. (b) Modeled performance and energy efficiency of an 8 node cluster made of various traditional Beefy nodes and low-power Wimpy nodes when performing a parallel hash join on our custom parallel execution engine P-store. Wimpy nodes only scan and filter the data before shuffling it to the Beefy nodes for further processing. A constant EDP relative to the all Beefy cluster is shown by the dotted line.

query performance[1] as we decrease the cluster size two nodes at a time (please see Section 4.3.1 for more details). Next, we discuss three key insights that can be drawn from Figure 4.1(a), which also shape the theme of this chapter.

First, this result shows the classic sub-linear parallel speedup phenomenon; namely, given a fixed problem size, increasing the computing resources by a factor of $X$ provides *less than* an $X$ times increase in performance [53]. Or, conversely, decreasing the resources to $1/X$, results in a relative performance *greater than* $1/X$. We can observe this phenomenon in Figure 4.1(a) because reducing the cluster size from 16 nodes (**16N**) to eight nodes (**8N**), results in a performance ratio greater than 50%. Note that since performance is the inverse of the response time, a performance ratio greater than 50% at 8N means that the response time at 16N is more than half the response time at 8N (i.e., sub-linear speedup).

[1] Here performance is the inverse of the query response time.

Second, as shown by the solid curve in Figure 4.1(a), the total energy required to execute the query decreases as we reduce the cluster size from 16N, even though it takes longer to run the query. Recall that energy is a product of the average power drawn by the cluster and the response time of the query. Due to the sub-linear speedup, going from 16N to 8N reduces the performance by only 36%, but the average power drops by roughly half (since we have half the number of nodes). Consequently, the energy consumption ratio (relative to the 16N case) for fewer than 16 nodes is *less than 1.0*. This is an encouraging energy efficiency gain, albeit at the cost of increased query response time.

Lastly, in Figure 4.1(a), the dotted line shows the line where the *Energy Delay Product* (EDP) is constant, relative to the 16N case. EDP is defined as $energy \times delay$ (measured in Joules seconds), and is commonly used in the architecture community as a way of studying designs that trade-off energy for performance[2] . Here "energy" refers to the query energy consumption and "delay" refers to the query response time. A constant EDP means that we have traded $x\%$ of performance for an $x\%$ drop in energy consumption. Preferably, it would be nice to identify design points that lie below the EDP curve, as such points represent trading proportionally less performance for greater energy savings.

In Figure 4.1(a), all the actual data/design points (on the solid line) are above the EDP curve. In other words, as we reduce the cluster size from 16 nodes to 8 nodes, we are giving up proportionately more performance than we are gaining in energy efficiency. For example, the 10 node configuration (**10N**) pays a 24% penalty in performance for a 16% decrease in energy consumption over the 16N case. Such trade-offs may or may not be reasonable depending on the tolerance for performance penalties, but the EDP curve makes it clear in which directions the trade-offs are skewed. This observation motivates the key question that is addressed in this chapter: *What are the key factors to consider when we design an energy-efficient DBMS cluster so that we can favorably trade less performance for more energy savings (i.e., lie below the EDP curve)?*

To understand the reasons why our observed data points lie above the EDP curve in Figure 4.1(a), and to carefully study the conditions that can produce design points that lie below the EDP curve, we built and

[2] With the growing viewpoint of considering an entire cluster as a single computer [22], EDP is also a useful way of thinking about the interactions between energy consumption and performance when designing data centers [103, 181].

modeled a custom parallel data execution kernel called P-store. The data that we collected from real parallel DBMSs was used to validate the performance and energy model of P-store (see Section 4.4 for details). Then, using P-store, we systematically explored both the query parameters and the cluster design space parameters to examine their impact on performance and energy efficiency. We also developed an analytical model that allows us to further explore the design space.

One of the interesting insights we found using both P-store and our analytical model is that there are query scenarios where certain design points lie below the EDP curve. One such result is shown in Figure 4.1(b). Here, we use our analytical model to show the energy versus performance trade-off for various eight node cluster designs, when performing a join between the TPC-H LINEITEM and the ORDERS tables (see Section 4.5 for details). In this figure, similar to Figure 4.1(a), we plot the relative energy consumed by the system and the response time against a reference point of an eight node Xeon-based ("**B**eefy") cluster. We then gradually replaced these nodes with mobile Intel i7 based laptops ("**W**impy")[3] nodes. The wimpy nodes do not have enough memory to build in-memory hash tables for the join, and so they only scan and filter the table data before shuffling them off to the beefy nodes (where the actual join is performed). As opposed to Figure 4.1(a), which is an experiment done with homogeneous beefy nodes, Figure 4.1(b) shows data points below the EDP curve. This result is interesting as it shows that it is possible to achieve a relatively greater energy savings than response time penalty (to lower the EDP) when considering alternative cluster designs.

Energy-efficient cluster design with potentially heterogeneous cluster nodes needs to be considered since non-traditional heterogeneous clusters are now showing up as database appliances, such as Oracle's Exadata Database Machine [131]. Thus, to a certain extent, commercial systems designers have already started down the road to heterogeneous clusters and appliances. Such considerations may also become important as future hardware (e.g., processor and/or memory subsystems) allows systems to dynamically control their power/performance trade-offs. This chapter provides a systematic study of both the software and the hardware design space parameters in an effort to draw conclusions about important design decisions when designing energy-efficient database clusters.

[3] We use the term "wimpy" as in [174], i.e. "slower but [energy] efficient".

## 4.2 Technical Contributions

The key contributions of this work are: first, we empirically examine the interactions between the scalability of parallel DBMSs and energy efficiency. Using three DBMSs: Vertica, HadoopDB, and a custom-built parallel data processing kernel, we explore the trade-offs in performance versus energy efficiency when performing speedup experiments on TPC-H queries (e.g., Figure 4.1(a)). From these results, we identify distinct bottlenecks for performance and energy efficiency that should be avoided for energy-efficient cluster design.

Second, non-traditional/wimpy low-power server hardware has been evaluated for its performance/energy-efficiency trade-offs, and we leverage these insights along with our own energy-efficiency micro-benchmarks to explore the design space for parallel DBMS clusters. Using P-store, we provide a model that takes into account these different server configurations and the parallel data processing bottlenecks, and predicts data processing performance and energy efficiency for various node configurations of a database cluster.

Third, using our model we study the design space for parallel DBMS clusters, and illustrate interesting cluster design points, under varying query parameters for a typical hash join (e.g., Figure 4.1(b)).

Finally, we organize the insights from this study as (initial) guiding principles for energy-efficient data processing cluster design.

| | **cluster-V** | **cluster-H** |
|---|---|---|
| DBMS | Vertica | HadoopDB |
| # nodes | 16 | 12 |
| TPC-H size | 1TB (scale 1000) | 120GB (scale 120) |
| CPU | Intel X5550 2 sockets | Intel X3370 1 socket |
| RAM | 48GB | 8 GB |
| Disks | 8x300GB | 4x750GB |
| Network | 1Gb/s | 1Gb/s |
| SysPower | $130.03C^{0.2369}$<br>C = CPU utilization | $13.81ln(C) + 111.46$<br>C = CPU utilization |

Table 4.1: Cluster Configuration

## 4.3 Parallel Database Behavior

In this section we examine the performance behavior and speedup in shared-nothing DBMS clusters, and examine its effect on energy efficiency. We see that bottlenecks that degrade performance, like network bottlenecks, decrease the energy efficiency of a cluster design. We experiment with two off-the-shelf, column-oriented, parallel DBMSs, Vertica [175] and HadoopDB [7] (with Vectorwise [92]). Vertica was deployed on *cluster-V* (see Table 4.1), and we used queries from the TPC-H benchmark at scale factor 1000. HadoopDB was deployed on *cluster-H* (see Table 4.1) and also used queries from the TPC-H benchmark, but at scale factor 120. These configurations were chosen based on a combination of factors, including finding the largest clusters that we could get dedicated access to, and with the appropriate software licenses.

### 4.3.1 Vertica

We used Vertica v.4.0.12-0 64 bit RHEL5 running on 16 HP ProLiant DL360G6 servers (the cluster configuration is described in Table 4.1), varying the cluster size between 8 and 16 nodes, in 2 node increments. We only present results with a warm buffer pool. Given the use of column-store in Vertica, the working sets for all the queries fit in main memory (even in the 8 node case).

We did not have physical access to our clusters so we used real power readings from the iLO2 remote management interface [89] to develop server power models based on CPU utilization, following an approach that has been used before [165]. Using a single cluster-V node, we used a custom parallel hash-join program (see Section 4.4) to generate CPU load, and iLO2 measured the reported power drawn by the node usage. We varied the number of concurrent joins to control the utilization on the nodes. iLO2 reports measurements averaged over a 5 minute window, and we ran the experiments for three 5 minute windows for each CPU utilization measurement. The power readings at each CPU utilization level were stable, and we use the average of the three readings to create our server power models. (Our measurements of physically accessible servers, not using iLO2, in Section 4.5 produced similar models.) Figure 4.2 shows the measured data and our fitted power model and is listed in Table 4.1 as "SysPower". (In this study, we explored exponential, power, and logarithmic regression models, and picked the one with the best $R^2$ value.)

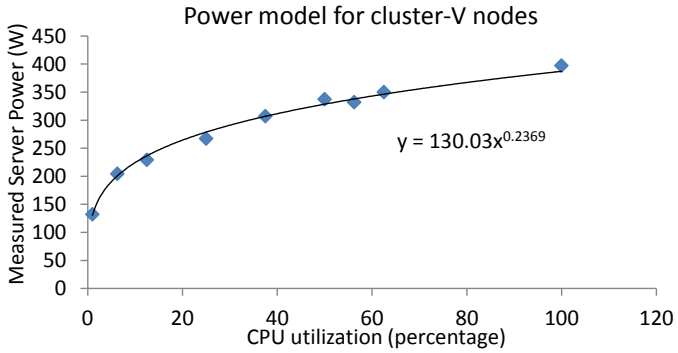
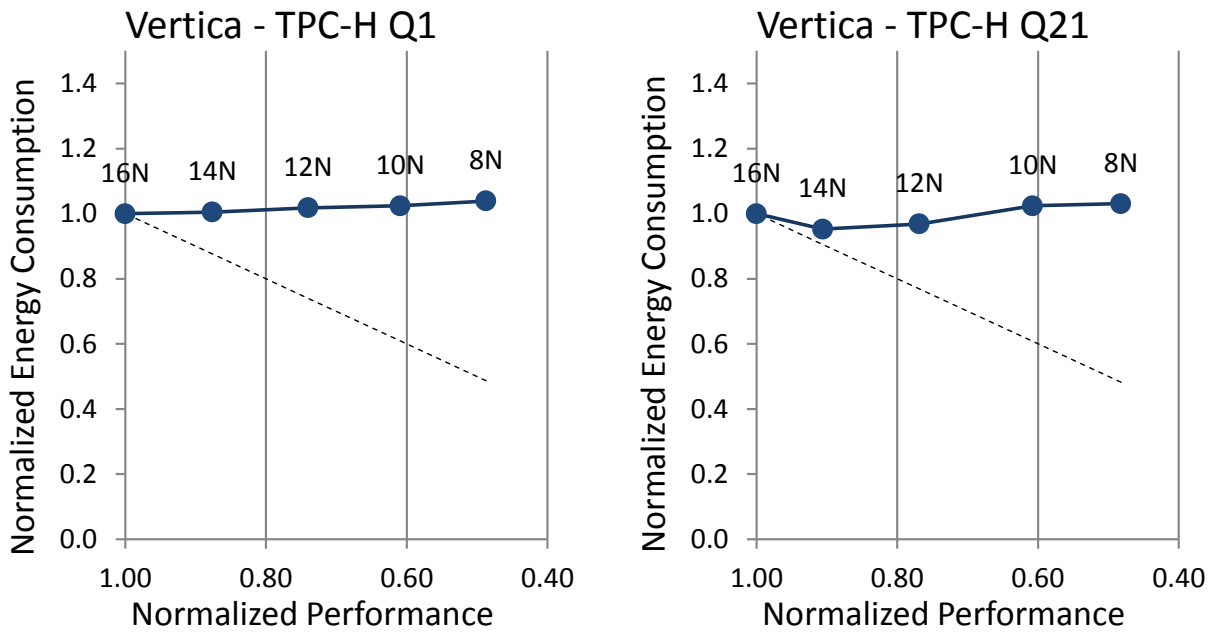



Figure 4.2: Modeling a cluster-V node's power as a function of its CPU utilization by using a our custom parallel hash-join program to generate CPU load and measuring the server power through iLO2.

We employed Vertica's hash segmentation capability which hash partitions a table across the cluster nodes on a user-defined attribute. We partitioned the LINEITEM, ORDERS, and CUSTOMER tables using the hash segmentation, while all the remaining TPC-H tables were replicated on each node. The ORDERS and the CUSTOMER tables were hashed on the O_CUSTKEY and C_CUSTKEY attributes respectively, so that a join between these two tables does not require any shuffling/partitioning of the input tables on-the-fly. The LINEITEM table was hashed on the L_ORDERKEY attribute.

We ran a number of TPC-H queries and present a selection of our results in this section. In Figure 4.3(a) we show the energy consumed and performance (i.e., the inverse of the query response time) for various cluster sizes running the TPC-H Query 1. This query does not involve any joins and only does simple aggregations on the LINEITEM table. The data points are shown relative to the largest cluster size of 16 nodes, and the "break-even" EDP line is also plotted, as was also done earlier in Figure 4.1(a). Recall from the beginning of this chapter, that this dotted line represents data points that trade energy savings for a performance penalty such that the EDP remains constant.

In Figure 4.3(a), we observe that Vertica's performance scales linearly, as the 8 node cluster has a performance ratio of about 0.5 compared to the 16 node case. Consequently, the energy consumption ratio is fairly constant since the 50% performance degradation is offset by a 50% drop in average cluster power. This result is important because it shows that a partitionable analytics workload (like TPC-H Query 1), exhibits ideal speedup as we allocate more nodes to the cluster. Thus the energy consumption will remain




(a) (b)

Figure 4.3: Vertica TPC-H (a) Q1, (b) Q21 (scale 1000) empirical parallel speedup results and its effect on energy efficiency under various cluster sizes. "Cluster-V" details can be found in Table 4.1. The dotted line indicates trading a proportional decrease in performance for a decrease in energy consumed such that the EDP ($= energy \times delay$) metric is constant.

roughly constant as we change the cluster size. In other words, one interesting energy-efficient design point is to simply provision as many nodes as possible for this type of query (as there is no change in energy consumption, but there is a performance penalty).

Let us consider a more complex query. Figure 4.3(b) shows the results for TPC-H Query 21, which is a query that involves a join across four tables: SUPPLIER, LINEITEM, ORDERS, and NATION. The SUPPLIER and NATION tables were replicated across all the nodes, so only the join between the LINEITEM and the ORDERS tables on the ORDERKEY attribute required repartitioning (of the ORDERS table on O_ORDERKEY). Besides the four table join, Query 21 also contains SELECT subqueries on the LINEITEM table within the WHERE clause.

Surprisingly, the results for the more complex TPC-H Query 21 results, shown in Figure 4.3(b), is similar to that of the simpler TPC-H Query 1, which is shown in Figure 4.3(a). Since both queries scale well, the energy consumption is fairly flat in both cases. It is interesting to consider why the more complex TPC-H Query 21 scales well, even though it requires a repartitioning of the ORDERS table during query processing.

The reason for this behavior is that the bulk of this query (94.5% of the total query time for eight nodes – **8N**) is spent doing node local execution. Only 5.5% of the total query time is spent repartitioning for the LINEITEM and ORDERS join. Since the bulk of the query is processed locally on each node, this means that increasing the cluster size increases performance nearly linearly. Thus, for a query with few bottlenecks, Vertica exhibits nearly ideal speedup, and so the energy-efficient cluster design is to simply use as many nodes as possible.

However, we have already seen a complex query where Vertica *does not* exhibit ideal speedup. Compare the TPC-H Query 21 result to Query 12 (shown above in Figure 4.1(a)), which is a much simpler two table join between the ORDERS and the LINEITEM tables, and performs the same repartitioning as Query 21. Compared to the 5.5% time spent network bottlenecked during repartitioning in Query 21, Query 12 spends 48% of the query time network bottlenecked during repartitioning with the eight node cluster. Since the proportional amount of total query time spent doing node local processing is now dramatically reduced, we see in Figure 4.1(a), that increasing the cluster size does not result in a proportional increase in performance. As such, the energy efficiency suffers dramatically as we increase the cluster size from 8 to 16 nodes.

**Summary:** Our Vertica results show that for queries that do not involve significant time repartitioning (i.e., most of the query execution time is spent on local computation at each node), the energy consumption vs. performance curves are roughly flat. This implies that an interesting point for energy-efficient cluster design is to pick a cluster that is as large as possible, as there is no energy savings when using fewer nodes, but there is an increase in query response time. However, for queries that are bottlenecked, as we saw with TPC-H Q12's network repartitioning, non-linear speedup means that a potential energy-efficient design decision is to reduce the cluster size up to the point where the lower performance is acceptable[4] . Of course, one simple way to mitigate repartitioning bottlenecks is to devise energy-aware repartitioning or *replication* strategies. An analysis and comparison to such strategies is beyond the scope of this study, but an interesting target of future work.

[4] Cluster systems often have implicit or explicit minimum performance targets for many workloads. We recognize that determining when such limits are acceptable is a broad and emerging research topic.

### 4.3.2 HadoopDB

HadoopDB is an open-source, shared-nothing parallel DBMS that uses Hadoop to coordinate independent nodes, with each node running Ingres/VectorWise [7, 92]. We used Hadoop ver. 0.19.2 and Ingres VectorWise ver. 1.0.0-112. Setup scripts for HadoopDB and the TPC-H queries we ran were provided by the authors of [7].

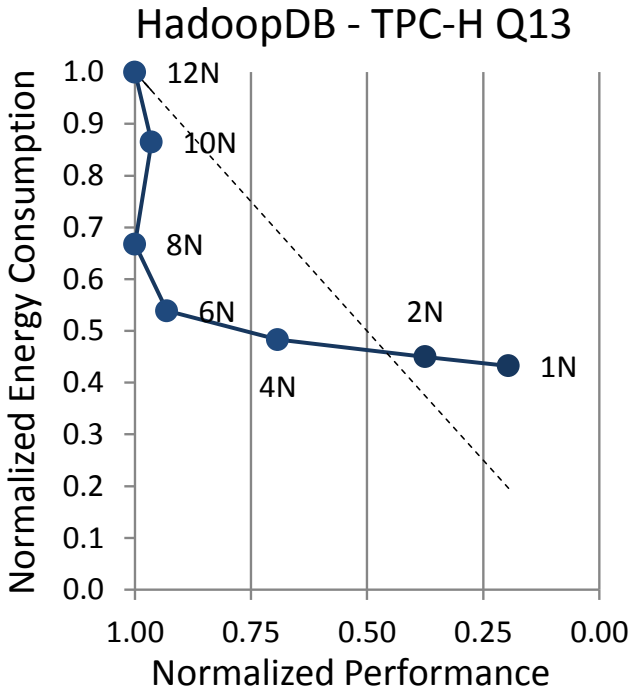



Figure 4.4: HadoopDB TPC-H Q13 (scale 120) empirical parallel speedup results and its effect on energy efficiency under various cluster sizes. "Cluster-H" details can be found in Table 4.1. The dotted line indicates trading a proportional decrease in performance for a decrease in energy consumed such that the EDP ($= energy \times delay$) metric is constant.

We ran TPC-H with a scale factor of 120, and partitioned the 120GB dataset across each cluster configuration that we tested. For each TPC-H query that we ran, we varied the number of cluster nodes between 2 and 12 in increments of 2. In addition, we also used a cluster size of 1. We used cluster-H as shown in Table 4.1. As with the Vertica setup (see Section 4.3.1) all results reported here are warm numbers. However, depending on the TPC-H query run and the cluster size, some queries still required disk access. The result we present below was such a case where cluster sizes smaller than 6 nodes, required disk access.

Due to the early version of HadoopDB we used, we only had a limited number of queries that could be run. Here, we present one query that shows two interesting results: (i) the best performing cluster is not the highest energy-efficient (lowest energy consuming) design point; and (ii) there can exist local optima in the energy consumption versus response time curves.

In Figure 4.4, we show the results for TPC-H Query 13 which is a join query between the ORDERS and the CUSTOMER tables. Both the tables were partitioned on the CUSTKEY attributes, so this is a **partition-compatible** join.

Figure 4.4 shows that eight nodes can achieve the same performance as 12 nodes, but reduces the energy consumption by more than 30%. Furthermore, if we drop down to 6 nodes, we can save just under 50% of the energy consumption (relative to the 12N case) without losing more than 7% in performance. This is perhaps the best operating point because at 4 nodes, we trade 44% in extra running time for over 50% in energy savings. Since the performance plateaus after 6 nodes, this means that any further increase in cluster size simply increases the amount of energy that is consumed when processing this query, but without any other benefits. The performance plateau was caused by the Hadoop communication overhead. Following the cluster size increase from 4 to 6 nodes, VectorWise never needed to access disk and query performance was simply gated by the overhead of Hadoop. Cluster sizes smaller than 6 nodes were bottlenecked in part by disk access and so we see proportional decreases in performance with cluster sizes from 4 to 1 node.

This result is different from those of Vertica (Figures 4.1(a) and 4.3(a) and (b)) because we see that our HadoopDB result curve crosses the dotted EDP line. This means that for all the cluster configurations between four and ten nodes, we proportionally save *more* energy than what we give up in performance, compared to the full 12 node cluster. After four nodes, the remaining two nodes and single node results are above the dotted line and the disproportionate performance decrease has caused an increase in EDP.

### 4.3.3 Discussion

From our results with Vertica, we have found that there are queries where the highest performing cluster configuration is *not* the most energy efficient (TPC-H Q12, Figure 4.1(a)) due to a network bottleneck. With a commercial DBMS, simpler queries that do not require any inter-node communication scale fairly

linearly with increased cluster size. We saw this with Query 1 and Query 21 (Figures 4.3(a,b)). However, with communication heavy queries such as Q12 (Figure 4.1(a)), increasing the cluster size simply adds extra network overhead that dampens the performance increase and reduces our energy savings. Similarly, with HadoopDB in Figure 4.4, we saw the poor scalability of a TPC-H query means that the best energy-efficient cluster design point was not the one with the most allocated resources. *Thus, we conclude that optimizing for parallel DBMS performance does not always result in the lowest energy consumed per query and this is the opposite conclusion from prior work which dealt with single server DBMS environments [171].*

With our results, we hypothesize that queries with bottlenecks, such as the network or disk, cause node underutilization and thus, the bottlenecks decrease the energy efficiency as the cluster size increases. Since Vertica and HadoopDB are black-box systems, to further explore this hypothesis of performance bottlenecks affecting affecting energy efficiency, we built a custom parallel query execution engine called P-store. Using P-store we empirically and analytically studied various join operations that require varying degrees of network repartitioning.

## 4.4 Energy Efficiency of a Parallel Data Processing Engine

From our empirical, off-the-shelf, parallel DBMS observations, we concluded that the scalability of the system running a given query plays an important role in influencing the energy efficiency of the cluster design points. Typically, poor scalability is a consequence of bottlenecks in the system. We now describe some of these bottlenecks, and then present our custom parallel query execution engine P-store that allows us to study these bottlenecks in more detail.

### 4.4.1 Bottlenecks

Since there can be a multitude of implementation-specific issues that can affect DBMS scalability, in this work we are interested only in fundamental bottlenecks that are inherent to the core engine. Since our focus is on analytic SQL queries, we examine core parallel database operators, such as scans, joins, and exchanges. For these kinds of queries and operators we identify three categories of bottlenecks that can lead to underutilized hardware components in a cluster environment.

**Hardware bottleneck (network and disk):** Decision support queries often require repartitioning during query execution. Replication of tables on different partition keys can alleviate the need for repartitioning in some cases, but this option is limited to only small tables as it is often too expensive to replicate large tables.

The repartitioning step is often gated by the speed of the network interconnect, causing the processor and other resources of a node to idle while the node waits for data from the network. Additionally, an increase in network traffic on the cluster switches causes interference and further delays in communication. Future advances in networking technology are expected to be accompanied by advances in CPU capabilities, making this performance gap a persistent problem [133].

The balance between the network and disk subsystems can be easily disturbed with varying predicate selectivities which diminish the rate at which the storage subsystem can deliver qualified tuples to the network. As a result, underutilized CPU cores waiting to process operators at the higher levels of a query plan waste energy.

**Algorithmic bottleneck (broadcast):** For certain joins, the cheapest execution plan may be to broadcast a copy of the inner table (once all local predicates have been applied) to all the nodes so that the join is performed without re-partitioning the (potentially larger) outer table. Such a broadcast generally takes the same time to complete regardless of the number of participating nodes (e.g., for $m$GB of qualifying tuples and say 16 nodes, each node needs to receive $(15m/16)$GB of the data, while for 32 nodes this changes by a small amount to $(31m/32)$GB). As a result, scaling out to more nodes does not speed up this first phase of a join and it reduces the cluster energy efficiency.

**Data Skew:** Although partitioning tools try to avoid data skew, even a small skew, can cause an imbalance in the utilization of the cluster nodes, especially as the system scales. Thus, data skew can easily create cluster and server imbalances even in highly tuned configurations. While we recognize data skew as a bottleneck, we leave an investigation of this critical issue as a part of future work.

### 4.4.2 P-store: A Custom-built Parallel Engine

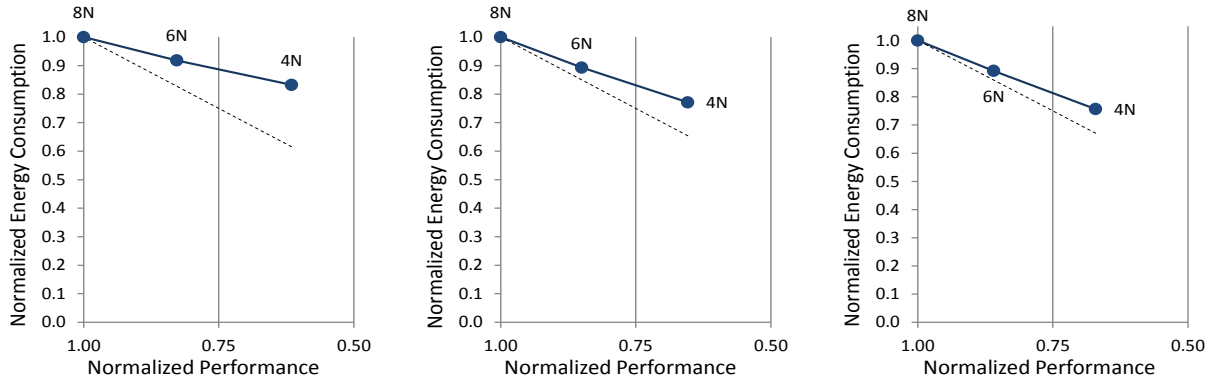
P-store is built on top of a block-iterator, tuple-scan module and a storage engine [82], that has scan, project, and select operators. To this engine, we added network exchange and hash join operators. Since energy consumption is a function of average power and time, our goal was to make our engine perform at levels comparable to commercial systems. Therefore, it was imperative that our exchange operator is able to transfer over the network at rates near the limits of the physical hardware, and our operators never materialize tuples while maximizing utilization through multi-threaded concurrency.

### 4.4.3 Experiments

The purpose of these P-store experiments is to stress our "work-horse" exchange operator when performing partition-incompatible hash joins. By stressing the exchange operator, we wanted to see how the hash join operation behaves at different points in the cluster design space with respect to performance and energy consumption.

Our hash join query is between the LINEITEM and the ORDERS tables of TPC-H at a scale factor of 1000, similar to the Vertica experiments (see Section 4.3.1). We used eight of the *cluster-V* nodes described in Table 4.1. By measuring the CPU utilization, the average cluster power was found using our empirically derived model from the cluster-V column of Table 4.1, Section 4.3.

To explore the bottleneck of *partition-incompatible* joins, we hash partitioned the LINEITEM and ORDERS tables on their L_SHIPDATE and O_CUSTKEY attributes respectively, and examined the join between these two tables that is necessary for TPC-H Query 3. The LINEITEM table is projected to only the L_ORDERKEY, L_EXTENDEDPRICE, L_DISCOUNT, and L_SHIPDATE columns, while the ORDERS table is projected to the O_ORDERKEY, O_ORDERDATE, O_SHIPPRIORITY, and O_CUSTKEY columns. To simulate the benefit of a columnar storage manager, for both tables, these four column projections (20B) were stored as tuples in memory for the scan operator to read. We applied a 5% selectivity predicate on both the tables using a predicate on the O_CUSTKEY attribute for ORDERS and a predicate on the L_SHIPDATE attribute for LINEITEM, as is similarly done in TPC-H Query 3.




(a) One query (b) Two concurrent queries (c) Four concurrent queries

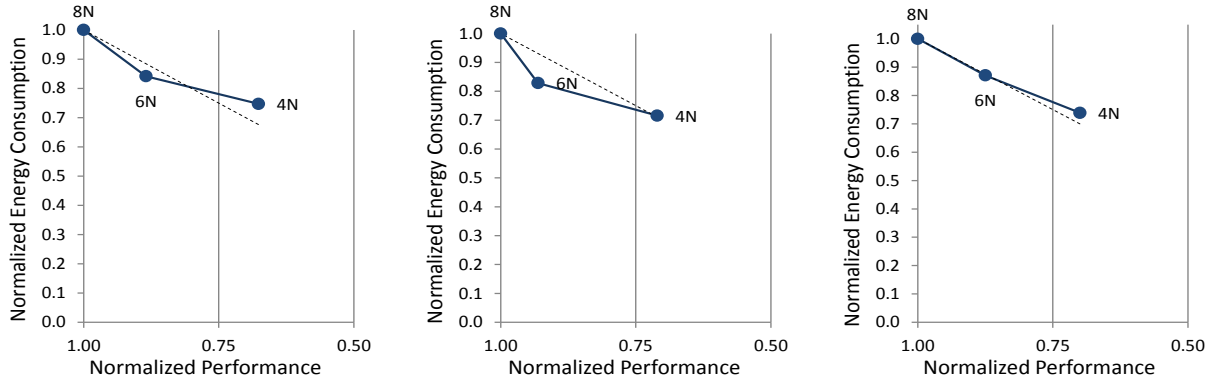
Figure 4.5: A partition incompatible TPC-H Q3 dual shuffle (exchange) hash join between LINEITEM and ORDERS (scale 1000) in P-store. Each subfigure (a-c) shows the energy consumption and performance ratios relative to the 8 node cluster reference point. The dotted line indicates the points where the EDP metric is constant.

To perform the partition-incompatible join in this TPC-H query (Query 3), the hash join operator needs to build a hash table on the ORDERS table and then probe using the LINEITEM table. There are two ways to do this: (i) repartition both tables on the ORDERKEY attribute – a **dual shuffle**, and (ii) **broadcast** the qualifying ORDERS tuples to all nodes. With these two join methods, we show the effects of network and disk bottlenecks as well as algorithmic bottlenecks that affect energy efficiency.

#### 4.4.3.1 Dual Shuffle

By doing a repartitioning of both tables, our P-store results shows behavior that is similar to that of Vertica running TPC-H Query 12 (Figure 4.1(a)). Using P-store, first the ORDERS table is repartitioned and the hash table is built on this table on-the-fly (no disk materialization) as tuples arrive over the network. After all the nodes have built their hash tables, the LINEITEM table is repartitioned and its tuples probe the hash tables on-the-fly.

In Figures 4.5(a)–(c), we see that poor performance scalability due to network bottlenecks can allow us to save energy by using fewer nodes. These figures show the comparison of relative energy consumption to relative performance of the hash join with 1, 2, and 4 independent concurrent joins being performed respectively on 4 to 8 node clusters. We increased the degree of concurrency to see how multiple simultaneous




(a) One query (b) Two concurrent queries (c) Four concurrent queries

Figure 4.6: A partition incompatible TPC-H Q3 broadcast hash join between LINEITEM and ORDERS (scale 1000) in P-store. Each subfigure (a-c) shows the energy consumption and performance ratios relative to the 8 node cluster reference point. The dotted line indicates the points where the EDP metric is constant.

requests for the network resource affected the network bottleneck.

Our results show that 4 nodes (**4N**) always consumes less energy than 8 (**8N**). Also, as the concurrency level increases, the degree of energy savings also increases (i.e., the results move closer to the dashed EDP line). In Figure 4.5(a), halving the cluster size only results in a 38% decrease in performance and translates to almost 20% savings in energy consumption. At a concurrency level of two (Figure 4.5 (b)), the 4-node cluster has a 23% increase in energy savings over the 8-node cluster with a 35% penalty in performance. With 4 concurrent hash joins running on the cluster (Figure 4.5 (c)), the 4-node cluster saves 24% of the energy consumed by the 8-node cluster while losing 33% in performance.

The reason we see greater energy savings with more concurrent queries is because the CPU utilization does not proportionally increase with the increasing concurrency level. This behavior is due to the network being the bottleneck, and so the CPU stalls and idles as other hash joins also require the network resource.

To summarize, these results show that reducing the cluster size can save energy, but we pay for it with a proportionally greater loss in performance – i.e., these data points lie above the dashed EDP line. As we have mentioned previously, ideally we would prefer results that lie on or below this dotted curve. The next section shows that for broadcast joins, the trade-off is much more attractive.

#### 4.4.3.2 Broadcast

The broadcast join method scans and filters the ORDERS table, and then sends the qualifying tuples to all the other nodes. Therefore, the full ORDERS hash table is built by each node and the LINEITEM table does not need to be repartitioned. To keep the full ORDERS hash table in memory, we increased the ORDERS table selectivity from 5% to 1% but held the LINEITEM table selectivity at 5%.

The results for this experiment are shown in Figure 4.6(a)–(c). From these figures, we observe that the data points now lie on the EDP line, indicating that we proportionally trade an X% decrease in performance for an X% decrease in energy consumption compared to the 8 node case (**8N**). Similar to the dual shuffle cases, here we also used various concurrency levels – 1, 2, and 4.

In Figure 4.6(a), with one join running, the performance only decreases by 32% when we halve the cluster size from 8 to 4 nodes. With 2 and 4 concurrent joins, the performance decreases by around 30% in Figures 4.6(b,c) when halving the cluster size. This drop in performance causes the 4 node cluster to always save between 25% to 30% in energy consumption compared to the full 8 node cluster.

Compared to the dual shuffle join (see Figure 4.5), the broadcast join saves more energy when using 4 nodes rather than 8 nodes (data points lie closer to the EDP line in Figure 4.6). This means that the broadcast join suffers a higher degree of non-linear scalability than the dual shuffle join. This is because broadcasting the ORDERS table does not get faster with 8 nodes since every node must receive roughly the entire table $(7/8)$ over the network (see Section 4.4.1).

### 4.4.4 Discussion

Figure 4.7 summarizes our findings in this section. The energy consumption of the system, when running a 2-way join under different query execution plans, can vary significantly under different cluster configurations. Due to sub-linear performance speedup, when the join is not partition compatible ("shuffle both tables" and "broadcast small table"), halving the cluster size does not result in doubling the response time, and this is one reason why the energy consumption at the "Half Cluster" configuration is lower than at "Full Cluster" point. Furthermore, we can see that when we use half of the cluster nodes instead of the

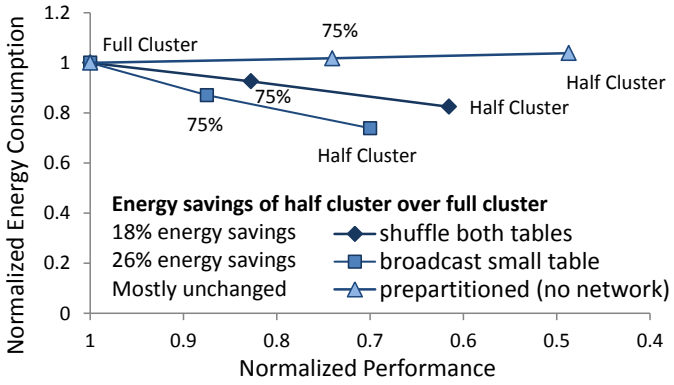



Figure 4.7: Network and algorithmic bottlenecks cause non-linear scalability in the partition-incompatible joins which mean smaller cluster designs consume less energy than larger designs. If the query is perfectly partitioned with ideal scalability (Vertica TPC-H Q1, Figure 4.3(a)), changing the number of nodes only affects performance and not energy consumption.

full cluster, the broadcast join method saves more energy and suffers less performance penalty than the dual shuffle join. This is because, for the broadcast join approach, the hash join build phase does not get faster with more nodes and so 4 nodes is much more efficient than 8 nodes. We can put this result in perspective by comparing these results to the partitioned TPC-H Q1 result from Vertica, where the energy consumption is flat regardless of the cluster size.

## 4.5 On Cluster Design

Our empirical results using two off-the-shelf DBMSs and a custom parallel query execution kernel have shown that a number of key bottlenecks cause sub-linear speedup. These diminishing returns in performance, when additional cluster nodes are added, are a key reason why the energy consumption typically drops when we reduce the cluster size. Observing this reoccurring pattern of "smaller clusters can save energy" suggests that the presence of performance bottlenecks is a key factor for energy efficiency.

However, we have thus far ignored the other factor for total system energy consumption: namely, the power characteristics of a single node. As we demonstrate in this section, the hardware configuration (and performance capability) of each individual node also plays a significant role in improving the energy efficiency of parallel DBMSs, and needs to be considered for energy-efficient cluster design. By comparing some representative hardware configurations, we found one of our "Wimpy" laptop configurations yielded

| System | CPU (cores/threads) | RAM | Idle Power (W) |
|---|---|---|---|
| Workstation A | i7 920 (4/8) | 12GB | 93W |
| Workstation B | Xeon (4/4) | 24GB | 69W |
| Desktop | Atom (2/4) | 4GB | 28W |
| Laptop A | Core 2 Duo (2/2) | 4GB | 12W (screen off) |
| Laptop B | i7 620m (2/4) | 8GB | 11W (screen off) |

Table 4.2: Hardware configuration of different systems.

the lowest energy consumption in single node experiments. As such, we ask the question: *Given what we now know about performance bottlenecks and energy efficiency, what if we introduced these so-called "Wimpy" nodes into a parallel database cluster?*

In the remainder of this chapter, we explore how query parameters, performance bottlenecks, and cluster design are the key factors that we should consider if we were to construct an energy-efficient cluster from scratch. We first show that wimpy nodes consume less energy per query than "Beefy" nodes (Section 4.5.1). Then, using P-store we ran parallel hash joins on heterogeneous cluster designs (Section 4.5.2), and used these results to generate and validate a performance/energy consumption model (Sections 4.5.3). Finally, using this model, we explore the effects of some important query and cluster parameters (Section 4.5.4) to produce insights about the characteristics of the energy-efficient cluster design space (Section 4.5.5).

### 4.5.1 Energy Efficiency of Individual Nodes

In this section we demonstrate that non-server, low-power nodes consume significantly lower amounts of energy to perform the same task as a traditional "Beefy" server nodes. For the rest of the results in this study, we had physical access to the nodes/servers so we measured power directly from the outlet using a WattsUp Pro meter. The meter provided a 1Hz sampling frequency with a measurement accuracy of +/- 1.5%. [5]

We studied five systems with different power and performance characteristics, ranging from low-power, ultra-mobile laptops to high-end workstations. Laptops are optimized for power consumption and

[5] We observed that the server CPU utilization/power models from these measurements validate our iLO2-based approach in Table 4.1.

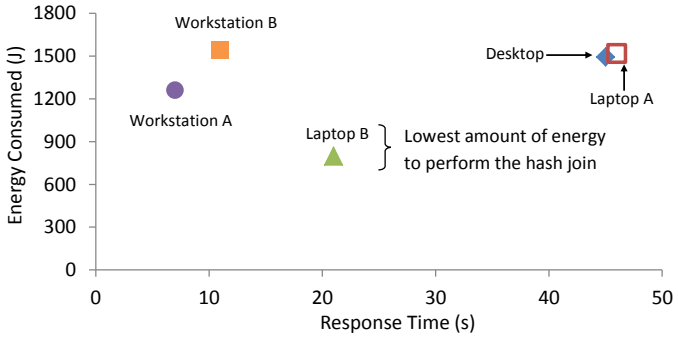



Figure 4.8: A hashjoin: 0.1M and 20M tuple (100B) tables.

are typically equipped with mobile CPUs and SSDs, whereas high-end workstations are optimized for performance. The configuration and power consumption details are shown in Table 4.2.

To assess the energy efficiency of these systems, we used an in-memory database workload. This workload executes a basic hash join operation, and is designed to stress modern CPUs (the hash join code is cache-conscious and multi-threaded). Our hash join is between a 10MB table (100K cardinality, and 100 byte tuples) and a 2GB table (20M cardinality, and 100 byte tuples).

Figure 4.8 plots the energy consumed versus the hash join response time for the different configurations. From this figure, we observe that the Laptop B system consumes the lowest energy for processing this in-memory hash join. As expected, the high-end workstations exhibit the best performance (lowest response time). However, the workstations are not the best when we consider the energy consumption. The energy consumption of Laptop B is 800 Joules while the energy consumption of Workstation A is near 1300 Joules, even though it takes significantly longer to perform the in-memory join on the laptop. As this experiment suggests, low-power systems can reduce the average power that they draw more than they reduce performance, thereby reducing the energy consumption of running a database query. Since "Laptop B" consumed the least energy, we use it to represent a "Wimpy" server node when considering architectural designs of heterogeneous clusters.

Combining these results with the scalability and bottleneck observations (from Section 4.4.2), next we explore energy-efficient cluster design points, starting with experimental evidence of potential design opportunities.

### 4.5.2 Heterogeneous Clusters Design

We prototyped two different four node clusters and measured the energy efficiency for dual shuffle hash joins (see Section 4.4.3), using P-store. Each cluster node has a 1Gbs network card and is connected through a 10/100/1000 SMCGS5 switch. Node power was measured using the WattsUp power meters as described in Section 4.5.1. The cluster specifications are as follows.

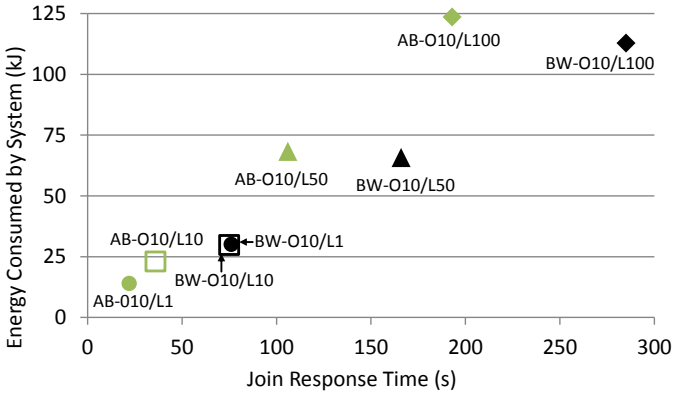
**Beefy Cluster:** This cluster has four HP ProLiant SE326M1R2 servers with dual low-power quad-core Nehalem-class Xeon L5630 processors. Each node also has 32GB of memory and dual Crucial C300 256GB SSDs (only one was used for data storage). During our experiments, the average node power was 154W.

**2 Beefy/2 Wimpy:** This cluster has two "beefy" nodes from the Beefy cluster above, and two Laptop Bs (from Section 4.5.1), with i7 620m CPUs, 8GB of memory, and a Crucial C300 256GB SSD. During our experiments, the average laptop power was 37W.

Before we continue with the experimental results, there are two important notes to make. First, P-store does not support out-of-memory joins (2-pass joins), and therefore we either run in-memory joins across all the nodes (*homogeneous execution*), or we only perform the join on the nodes that have enough memory while the other nodes simply scan and filter data (*heterogeneous execution*). Since energy is a function of time and average power, in-memory join processing dramatically reduces the response time (as well as maintains high CPU utilization) and hence provides a substantial decrease in energy consumption per query. Second, our network has a peak capacity of 1Gbps, and therefore is typically the bottleneck for non-selective (where a high percentage of input tuples satisfy the selection predicates) queries. While a faster network could be used, as we discussed in Section 4.4.1, the network-CPU performance gap is likely to persist into the near future.

Using P-store, we ran the same hash join necessary for TPC-H Q3 between the LINEITEM and the ORDERS tables (as was done in the experiment described in Section 4.4.3), but with a scale factor of 400. The working sets (after projection) for the LINEITEM and the ORDERS tables are 48GB and 12GB respectively. We warmed our memory cache with as much of the working set as possible. The hash join is partition-


(a) all nodes build hash tables




(b) beefy nodes build hash tables, wimpy nodes scan/filter data

Figure 4.9: P-store dual-shuffle hash join between LINEITEM (L) and ORDERS (O) tables at various selectivities (e.g., O10 means 10% selectivity on the ORDERS table). Empirical hash join energy efficiency of the all Beefy (AB) versus the 2 Beefy/2 Wimpy clusters (BW).

incompatible on both the tables just as in Section 4.4.3 (i.e., dual shuffle is needed). We varied the selectivity on the LINEITEM table to 1%, 10%, 50%, and 100%. The ORDERS table had a selectivity of 1% and 10%. This gave us 8 different hash join workloads.

The Beefy cluster can run all of the 8 hash join workloads by first repartitioning ORDERS on the join key O_ORDERKEY, and building the hash table on each node as tuples arrive over the network. The LINEITEM table is then repartitioned on the L_ORDERKEY attribute, and each node probes their hash table on-the-fly as tuples arrive over the network. The beefy nodes have enough memory to cache the working set and build the in-memory hash table for the ORDERS table for both the 1% and 10% ORDERS selectivity values.

The hash join parameters cause the need for the 2 Beefy/2 Wimpy cluster to alternate between homo-

geneous and heterogeneous execution. The wimpy nodes only have 8GB of memory, and hence can only cache the 3GB ORDERS table partition and some of the 12GB LINEITEM table partition. Thus, for the 2 Beefy/2 Wimpy cluster, a 1% selectivity on ORDERS allows the hash table to fit in the laptop memory, and we can execute the join in the same way as the Beefy cluster – *homogeneous execution*. However, if the selectivity on ORDERS is low (i.e., the predicate matches $\geq$ 10% of the input tuples), then we can only leverage the wimpy nodes to scan and filter their partitions, and have them send all the qualifying data to the beefy nodes where the actual hash tables are built – *heterogeneous execution*. Similarly, the wimpy nodes scan and filter the LINEITEM (probe) tuples and repartition the data to the beefy nodes.

*Therefore, in our study, we consider two "wimpy" aspects of our mobile nodes: (1) their lower power and performance due to low-end CPUs, and (2) their lower memory capacity which constrains the execution strategies for evaluating a query.*

**Small Hash Tables – Homogeneous Execution**

Homogeneous parallel execution of a hash join requires a highly selective predicate that produces a small ORDERS hash table that can be stored in memory on all the nodes. We measured the response time (seconds) and the energy consumed (Joules) of both our cluster designs when running the hash join where the selectivity of the predicate on the ORDERS table was 1%. Figure 4.9 (a) shows the comparison of the all Beefy (**AB**) cluster to the 2 Beefy/2 Wimpy (**BW**) cluster when we vary the LINEITEM table selectivity (**L1,L10,L50,L100**) for a 1% selectivity on the ORDERS table (**O1**). We notice that for the 1% and 10% LINEITEM selectivity cases (the circle and square markers respectively), the **AB** cluster consumed less energy than the **BW** cluster. However, for the 50% LINEITEM selectivity case (triangle), the **BW** cluster saves 43% of the energy consumed over the **AB** cluster. When the LINEITEM table has no predicate (diamond), the **BW** cluster saves 56% of the energy consumed by the **AB** cluster.

Here, we illustrate one of the bottlenecks we discussed above in Section 4.4: namely the network/disk. With 100% LINEITEM selectivity, the bottleneck is network bandwidth: the bottleneck response time of the **AB** cluster is 155s, while the response time of the **BW** cluster is 168s. With 1% LINEITEM selectivity, we are bound by the scan/filter limits of the wimpy, mobile nodes: the **AB** cluster finishes executing this join in 8s while the **BW** cluster takes 50s.

**Large Hash Tables – Heterogeneous Execution**

As mentioned above, in the cases where the hash tables are larger than the available memory in the wimpy, mobile nodes, the wimpy nodes of the mixed node cluster simply scan and filter the data for the beefy nodes. Figure 4.9 (b) shows the comparison of the **AB** cluster to the **BW** cluster in terms of energy efficiency, similar to Figure 4.9 (a). Like our previous result for the 1% ORDERS selectivity case, the **BW** cluster consumes less energy than the **AB** cluster at a low LINEITEM selectivity (50% and 100%). The **BW** cluster consumes 7% and 13% less energy than the **AB** cluster at 50% and 100% LINEITEM selectivity respectively.

Thus, we see that in both situations, a heterogeneous cluster can offer improved energy efficiency at reduced performance. To further explore the full spectrum of available cluster and workload parameters that are important for cluster design, we built a model of P-store's performance and energy consumption. Our model is validated against these results in Figure 4.9. This model enables us to freely explore the cluster and query parameters that are important for cluster design.

### 4.5.3 Modeling P-store and Bottlenecks

Our model of P-store's performance and energy consumption behavior is aimed at understanding the nature of query parameters and scalability bottlenecks affecting performance and system energy consumption. The model predicts the performance and energy consumption of various different ways to execute a hash join. The input parameters that we consider are listed in Table 4.3.

From Table 4.3, note that our model makes a few simplifying assumptions. First, the disk configuration for both the wimpy nodes and the beefy nodes are the same and have the same bandwidth. Second, the same uniformity assumption has been made about the network capability of both node types. These assumptions matched our hardware setup, but we can easily extend our model to account for separate wimpy and beefy I/O bandwidths.

First, let us look at when the wimpy nodes can build hash tables because they have enough memory to hold the hash tables (i.e., we do not have to run a 2-pass hash join). This is the case when $H$ is true (see Table 4.3).

| | | | |
|---|---|---|---|
| $T_{bld}$ | Build phase time (s) | $T_{prb}$ | Probe phase time (s) |
| $E_{bld}$ | Build phase energy (J) | $E_{prb}$ | Probe phase energy (J) |
| $N_B$ | # Beefy nodes | $N_W$ | # Wimpy nodes |
| $M_B$ | Beefy memory size (MB) | $M_W$ | Wimpy memory size (MB) |
| $I$ | Disk bandwidth (MB/s) | $L$ | Network bandwidth (MB/s) |
| $Bld$ | Hash join build table size (MB) | $Prb$ | Hash join probe table size (MB) |
| $S_{bld}$ | Build table predicate selectivity | $S_{prb}$ | Probe table predicate selectivity |

| | |
|---|---|
| $R_{Wbld}$ | Rate at which a Wimpy node builds its hash table (MB/s) |
| $R_{Bbld}$ | Rate at which a Beefy node builds its hash table (MB/s) |
| $U_{Wbld}$ | Wimpy node CPU bandwidth during the build phase |
| $U_{Bbld}$ | Beefy node CPU bandwidth during the build phase |
| $R_{Wprb}$ | Rate at which the Wimpy node probes its hash table (MB/s) |
| $R_{Bprb}$ | Rate at which the Beefy node probes its hash table (MB/s) |
| $U_{Wprb}$ | Wimpy node CPU bandwidth during the probe phase |
| $U_{Bprb}$ | Beefy node CPU bandwidth during the probe phase |
| $C_B = 5037$ | Maximum CPU bandwidth of a Beefy node (MB/s) |
| $C_W = 1129$ | Maximum CPU bandwidth of a Wimpy node (MB/s) |
| $G_B = 0.25$ | Beefy CPU utilization constants for P-store |
| $G_W = 0.13$ | Wimpy CPU utilization constants for P-store |

| | |
|---|---|
| $f_B(c) = 130.03 \times (100c)^{0.2369}$ (c=CPU util.) | Beefy node power model |
| $f_W(c) = 10.994 \times (100c)^{0.2875}$ (c=CPU util.) | Wimpy node power model |
| $H = M_W \geq (Bld * Bld_{sel})/(N_B + N_W)$ | Wimpy can build the hash table |

Table 4.3: List of Model Variables

**Homogeneous Execution:** In this case, all the nodes execute the same operator tree. We can divide the hash join into the build phase and the probe phase. During the build phase, we are either bound by (1) the effects of the disk bandwidth and the selectivity on the build table ; or (2) the network bandwidth:

$$R_{Bbld} = R_{Wbld} = \begin{cases} IS_{bld} & \text{if } IS_{bld} < L \\ \frac{(N_B + N_W)L}{(N_B + N_W - 1)} & \text{otherwise} \end{cases}$$

These two variables $R_{Bbld}$ and $R_{Wbld}$ give us the rates at which the build phase is proceeding (in MB/s). We also need to model the build phase CPU utilization to determine the power drawn by each node type. This is done by determining the amount of data that the beefy CPU and the wimpy CPU is processing per second, $U_{Bbld}$ and $U_{Wbld}$, and then dividing each of these values by the maximum measured rates for the CPU, $C_B$ and $C_W$ respectively. Finally, we need to add CPU constants that are inherent to P-store, which are $E_B$ and $E_W$ for the beefy and wimpy nodes respectively. This CPU utilization is the input for our server

power functions, $f_B$ and $f_W$, for the beefy and wimpy nodes respectively (see Table 4.3).

$$U_{Bbld} = U_{Wbld} = \begin{cases} I & \text{if } IS_{bld} < L \\ \frac{(N_B + N_W)L}{(N_B + N_W - 1)} \div S_{bld} & \text{otherwise} \end{cases}$$

Therefore, we can now calculate the build phase response time and the build phase cluster energy consumption.

$$T_{bld} = \frac{Bld \times S_{bld}}{(N_B R_{Bbld}) + (N_W R_{Wbld})}$$

$$E_{bld} = T_{bld} \times (N_B f_B(G_B + \frac{U_{Bbld}}{C_B}) + N_W f_W(G_W + \frac{U_{Wbld}}{C_W}))$$

Since this is homogeneous execution, the probe phase can be modeled in the same way as the build phase but now using the probe table variables.

$$R_{Bprb} = R_{Wprb} = \begin{cases} IS_{prb} & \text{if } IS_{prb} < L \\ \frac{(N_B + N_W)L}{(N_B + N_W - 1)} & \text{otherwise} \end{cases}$$

And similarly, the CPU bandwidth during the probe phase is:

$$U_{Bprb} = U_{Wprb} = \begin{cases} I & \text{if } IS_{prb} < L \\ \frac{(N_B + N_W)L}{(N_B + N_W - 1)} \div S_{prb} & \text{otherwise} \end{cases}$$

Like the build phase, we can now calculate the probe phase response time and the probe phase cluster energy consumption.

$$T_{prb} = \frac{Prb \times S_{prb}}{(N_B R_{Bprb}) + (N_W R_{Wprb})}$$

$$E_{prb} = T_{prb} \times (N_B f_B(G_B + \frac{U_{Bprb}}{C_B}) + N_W f_W(G_W + \frac{U_{Wprb}}{C_W}))$$

With these two phases modeled, the total response time is simply $T_{bld} + T_{prb}$, and the total energy consumed is $E_{bld} + E_{prb}$.

**Heterogeneous Execution:** When the wimpy nodes can not store the hash join hash table in memory ($H$ is false), P-store uses the wimpy nodes as scan and filter nodes and only the beefy nodes build the hash tables. Our model accounts for this by calculating the rates at which predicate-passing tuples are delivered to the beefy nodes. In the same spirit as the above model for the homogeneous execution, the key factors

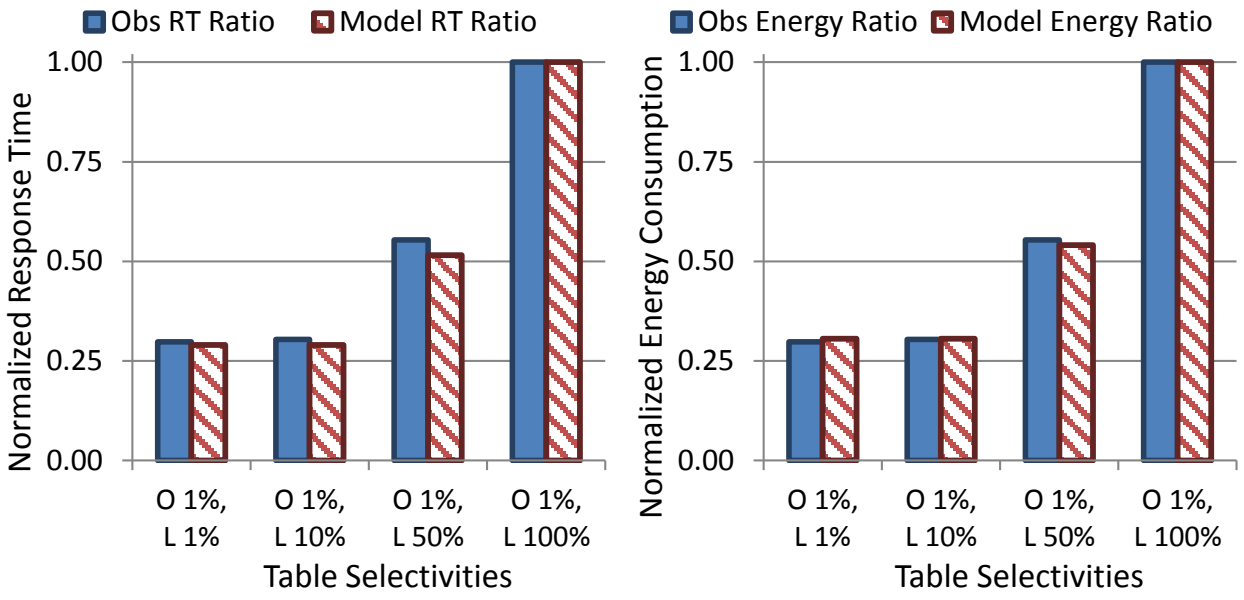
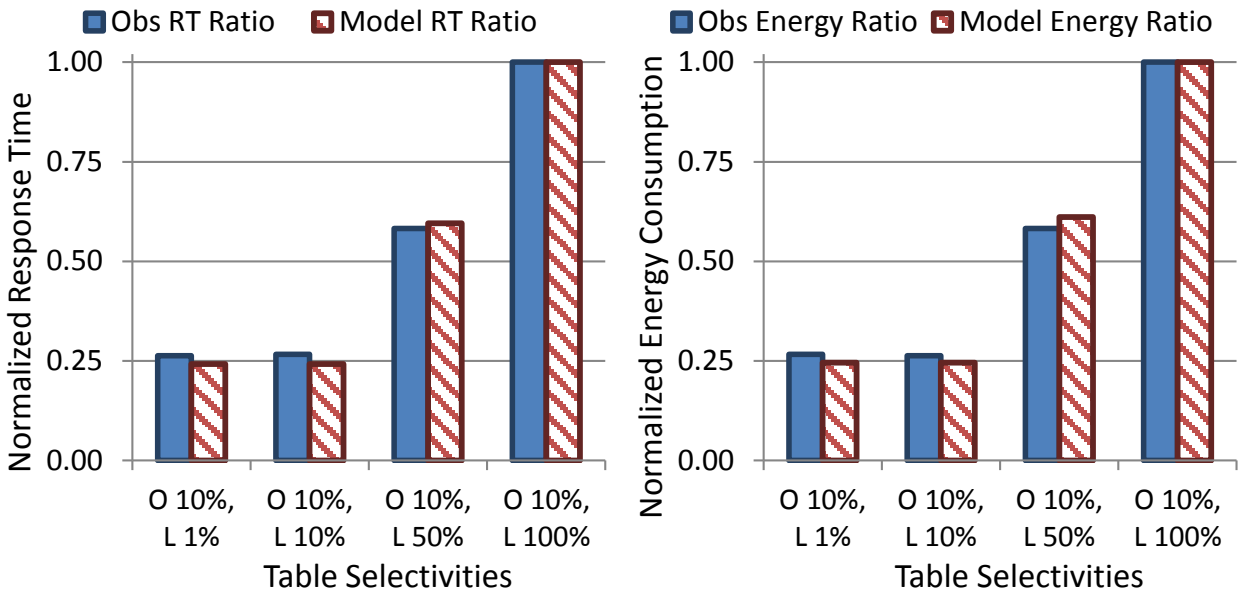



Figure 4.10: Performance and Energy model validation for the 2 Beefy/2 Wimpy case with 1% ORDERS selectivity and varying LINEITEM selectivity against observed data from Figure 4.9(a)

are whether or not we are disk bound or network bound. However, since a smaller set of beefy nodes need to receive the data from the entire cluster, in addition to out-bound network limitations from nodes sending data, there is an ingestion network limitation at the beefy nodes, which becomes a performance bottleneck first. That is, the beefy nodes that are building the hash tables can only receive data at the network's capacity even though there may be many wimpy nodes trying to send data to them at a higher rate. This back-pressure slows the rate at which all the nodes will scan and filter their locally stored data. Consequently, this reduces the CPU utilization and so the server power is reduced as well. For instance, if we are beefy-node, network-ingestion bound during the probe phase ($(N_B + N_W - 1)IS_{prb} > L$), then the maximum rate at which the wimpy nodes can send qualifying tuples outbound is $(L/(N_B + N_W - 1))$ which means the maximum wimpy-node, disk-scan rate is $(L/(N_B + N_W - 1) \div S_{prb})$.

**Model Validation**

Next, we present validation of our P-store model using the real observed data of the 2 Beefy/2 Wimpy cluster in Section 4.5.2. Since the results in Section 4.5.2 were from warm-cache (some of the table data resides in memory and disk I/O may be necessary) hash joins, we changed the input parameters of our




(a) (b)

Figure 4.11: Performance and Energy model validation for the 2 Beefy/2 Wimpy case with 10% ORDERS selectivity and varying LINEITEM selectivity against observed data from Figure 4.9(b)

model to account for such behavior. For example, we changed the scan rate of the build phase to that of the maximum CPU bandwidth $C_W$ and $C_B$. In this way, the time it takes to finish the build phase is equal to the time it takes the CPU to process the build table at maximum speed plus the time to send the filter qualifying tuples over the network.

For our model, we used the following hardware parameter settings: $M_B = 31000$, $M_W = 7000$, $N_B = N_W = 2$, $I = 270$, $L = 95$. Since our experiments used a different beefy node (based on 2 L5630 Xeon CPUs) than the cluster-V nodes, the $f_B$ function for node power is given by $79.006 \times (100 * u)^{0.2451}$ and $C_B = 4034$.

Our validation results of our model against the 2 Beefy/2 Wimpy cluster results from Section 4.5.2 are shown in Figures 4.10 and 4.11. In Figure 4.10(a) and (b), we validate the response time and energy consumption results of our model against the 1% ORDERS table selectivity joins presented in Figure 4.9(a). The execution plans for the 1% ORDERS selectivity were homogeneous across all the nodes. In Figure 4.10, our model provided relative response time behavior and relative energy consumption behavior to the 100% LINEITEM selectivity result within 5% error compared to the observed data. Similarly, in Figure 4.11(a) and (b), we validated the relative response time and energy consumption results from our model for the 10%

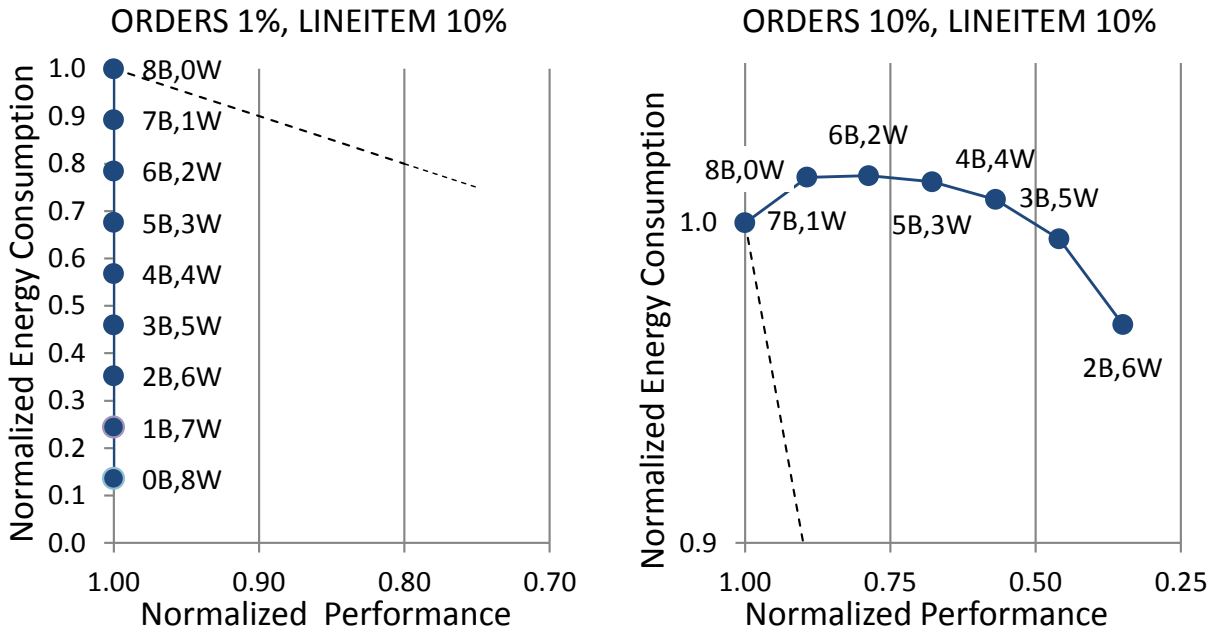



Figure 4.12: Modeled P-store dual-shuffle hash join performance and energy efficiency of an 8 node cluster made of various Beefy and Wimpy nodes. (a) Parallel execution is homogeneous: the Wimpy and Beefy nodes can build in-memory hash tables. (b) Parallel execution is heterogeneous: Wimpy nodes scan and filter their local data before shuffling it to the Beefy nodes for further processing. A constant EDP relative to the all Beefy cluster is shown by the dotted line.

ORDERS selectivity joins from Figure 4.9(b). Here the error rate was within 10% compared to the observed data. The execution plans here were heterogeneous, with the wimpy nodes only scanning and filtering data for the beefy nodes.

With our model validated, we now explore a wider range of cluster designs beyond four nodes while also varying the query parameters, such as predicate selectivity. Our results reveal interesting opportunities for energy-efficient cluster design.

### 4.5.4 Exploring Query and Cluster Parameters

In this section, we explore what happens to query performance and energy consumption as we change the cluster design (the ratio of beefy to wimpy nodes) and query characteristics (the selectivity of build and probe tables), while executing the hash join query on P-store. This hash join is between a 700GB TPC-H ORDERS table and 2.8TB TPC-H LINEITEM table. We join these tables on their join attribute ORDERKEY. For

our model, we used the following hardware parameter settings: $M_B = 47000$, $M_W = 7000$, $I = 1200$, $L = 100$. The memory settings correspond to those of our cluster-V nodes (Table 4.1, Section 4.3) and Laptop B (Table 4.2). We model the IO subsystem of our nodes as if they each had four of the Crucial 256GB SSDs that we used in Section 4.5.2, as well as the 1Gbps network interconnect of that experiment. We kept the remaining CPU parameters the same as those listed in Table 4.3.

We have already presented one of these results at the beginning of this chapter. In Figure 4.1(b), we showed that increasing the ratio of wimpy to beefy nodes results in more energy-efficient configurations compared to the homogeneous cluster design consisting of only beefy nodes. In that figure, we used the model that we described in Section 4.5.3 to compare the energy consumption vs. performance trade-off for the hash join between the TPC-H ORDERS table and the LINEITEM table at 10% and 1% selectivity respectively.

In Figure 4.12(a), we show that the best cluster design point, when executing a hash join between an ORDERS table with 1% selectivity and a LINEITEM table with 10% selectivity, is to use all wimpy nodes. In this case, since a 1% selectivity on the hash join build table means that each node only needs at least 875MB of memory to build in-memory hash tables, the parallel execution across all nodes is homogeneous. Since we have modeled the wimpy and the beefy nodes to have the same IO and network capabilities, we should not expect any performance degradation as we replace beefy nodes with wimpy nodes (i.e., the network and disk bottlenecks mask the performance limitations of the wimpy nodes). This is shown in Figure 4.12(a) as the performance ratio remains 1.0 throughout all the configurations. Consequently, since performance does not degrade, we see that the energy consumed by the hash join drops by almost 90% because a wimpy node power footprint is almost 10% of the beefy node power footprint. Due to similar results to Figure 4.12(a), we omit the figure for 1% ORDERS and 1% LINEITEM selectivity.

In Figure 4.12(b), the best cluster configuration for a hash join when the ORDERS table has the same 10% selectivity as in Figure 4.1(b) and the LINEITEM table also has 10% selectivity is to use all beefy nodes. The wimpy nodes do not have the 8.8GB of main memory that is needed to build the in-memory hash table, so we model them to scan and filter data for the beefy nodes; i.e., we have a heterogeneous parallel execution. We varied the cluster design from the all beefy to 2 beefy and 6 wimpy cases, after which, the aggregate beefy memory cannot store the in-memory hash table. Here the results stand in stark contrast to

the results shown in Figure 4.12(a), because there is not a significant energy savings when we use wimpy nodes in the cluster. The reason is because with a 10% selectivity predicate on the tables, the server's IO subsystem has enough bandwidth to saturate the network interconnect, and the network becomes the bottleneck. Specifically, as we decrease the number of beefy nodes – they are responsible for building and probing the hash tables – each beefy node becomes more network bottlenecked ingesting data from all the other nodes in the cluster. Consequently, we see that the performance degrades severely, while the energy consumption does not drop below 95% of the **8B,0W** cluster reference point.

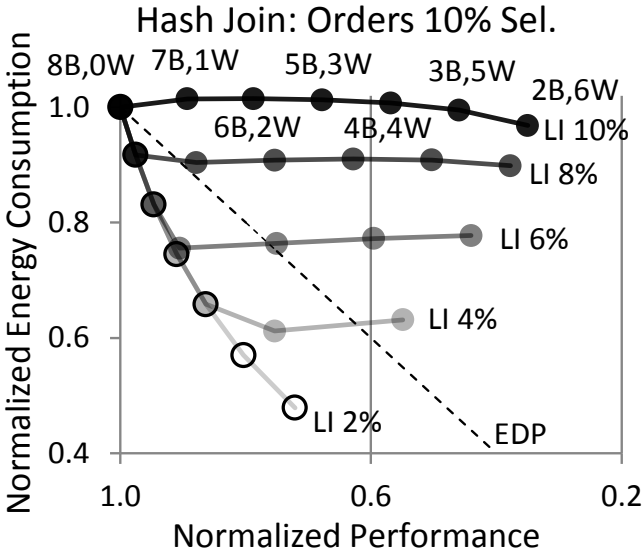



Figure 4.13: Modeled P-store dual-shuffle hash join performance and energy efficiency of an 8 node cluster made of Beefy and Wimpy nodes. The join is between the TPC-H ORDERS table (10% sel.) and LINEITEM table (2-10% sel.). Parallel execution is heterogeneous across cluster nodes. A constant EDP relative to the all Beefy cluster is shown by the straight dotted line.

This last result is interesting because we saw that for another heterogeneous execution plan in Figure 4.1(b), a 1% LINEITEM selectivity, saw significant wins in trading performance for energy savings. Naturally, we wanted to understand why we saw these different results when we only change the selectivities of the predicate on the LINEITEM table.

In Figure 4.13, we show that as we increase the selectivities of the predicate on the LINEITEM table from 10% to 2% (in 2% increments), given a 10% selectivity predicate on the ORDERS table, we begin to trade less performance for greater energy savings. Each curve represents a different LINEITEM selectivity, and

as the curve moves to the right, each dot indicates more Wimpy nodes in the 8 node cluster. Again, we do not use fewer than 2 beefy nodes because 1 beefy node cannot build the entire hash table in memory. The dashed line indicates the constant EDP metric.

We notice that as we gradually decrease the number of LINEITEM tuples passing the selection filter (i.e., increase the LINEITEM table selectivity from 10% to 2%), the results start to trend downward below the EDP line. More interestingly, we notice that the results start to trend downward in a way such that the knee in the curves moves lower towards the cluster designs with more wimpy nodes (right ends of the result curves). To the right of the knee, *the heterogeneous parallel plans saturate the beefy node network ingestion during the probe phase*. To the left of the knee, the nodes delivering data to the beefy nodes are sending data as fast as their IO subsystem (and table selectivity) can sustain. As the amount of probe (LINEITEM) data passing the selection filter decreases (the gradually lighter-shaded curves in Figure 4.13), the number of wimpy nodes that are needed to saturate the inbound network port at the beefy nodes increases, so the knee moves to the right end with more wimpy nodes.

### 4.5.5 Summary

In this section, we have shown that there is an interesting interplay between the most energy-efficient cluster design (the ratio of wimpy to beefy nodes) and the query parameters such as predicate selectivity (Figure 4.13) for a simple parallel hash join query. Furthermore, heterogeneous cluster designs can trade proportionally less performance for greater energy savings (i.e., they lie below the EDP line) compared to a homogeneous cluster design.

## 4.6 Cluster Design Principles

In this section we summarize guiding principles for building energy-efficient data processing clusters. Figure 4.14 outlines the summary of our findings, which we discuss in this section.

First, consider a query that is being run on a parallel data processing system. Using initial hardware calibration data and query optimizer information, consider the case when the query is deemed to be highly scalable. That is, the energy/performance model for the query looks like Figure 4.14(a). (We have seen

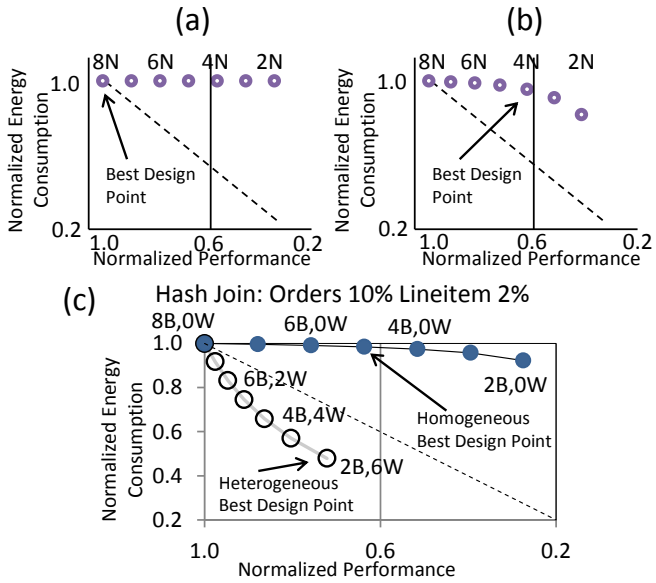



Figure 4.14: Query scalability characterizations and energy-efficient design points when the target query performance is 40% of an eight Beefy node configuration, i.e., the target normalized performance is 0.6: (a) A highly scalable parallel workload: Use all available nodes, since the highest performing design point is also the most energy efficient. (b) A parallel workload with some scalability bottleneck (e.g., network I/O): Use the fewest nodes such that performance limits are still met. (c) Using fewer nodes reduces energy consumption on bottlenecked hash join but adding **W**impy nodes to supplement **B**eefy nodes can provide a more energy-efficient design point. The dashed EDP curve provides a reference for equal performance vs energy trade-offs.

empirical results like this, in Figures 4.3(a) and (b).) For such queries, the best cluster design point is to use the most resources (nodes) to finish the query as soon as possible. So, in Figure 4.14(a), we find that the largest cluster is the best design point.

However, if the query is not scalable (potentially due to bottlenecks), then the highest performing cluster design may not be the most energy-efficient design. (The results shown in Figures 4.1(a), 4.5, and 4.6 are examples of this case.) For such queries, reducing the cluster size decreases the query energy consumption, although at the cost of performance. In such situations, one should reduce the performance to meet any required target (e.g., performance targets specified in SLAs in cloud environments). Then, the system can reduce the server resource allocation accordingly. Thus, as we show in Figure 4.14(b), if the acceptable performance loss is 40%, then using 4 nodes is the best cluster design point for this query.

Finally, these previous design principles assume a homogeneous cluster. For queries that are not

scalable, a heterogeneous cluster may provide an interesting design point. As before, in this case, to pick a good cluster design point, we start with an acceptable target performance. Then, heterogeneous cluster configurations may provide a better cluster design point compared to the best homogeneous cluster configuration. As an example, consider the P-store dual-shuffle hash join query in Figure 4.14(c), and an acceptable performance loss of 40% relative to an eight "Beefy" node homogeneous cluster design (labeled as 8B in the Figure). Here, the 5B configuration is the best homogeneous cluster design point. But if we substitute some of the beefy nodes with lower-powered "Wimpy" nodes then with two beefy nodes and six wimpy nodes we consume less energy than the 5B case and also have better query performance. Notice that the heterogeneous design points are below the EDP curve, which means that in these designs one proportionally saved more energy than the proportional performance loss (compared to the 8B case). Thus, for non-scalable queries, a heterogeneous cluster configuration may provide a better design point, both from the energy efficiency and performance perspective, compared to homogeneous cluster designs.

## 4.7 Summary

In this chapter, we have studied the trade-offs between the performance and the energy consumption characteristics of analytical queries, for various cluster designs. We have found that the query scalability properties have a key impact in determining the interaction between the query performance and its energy consumption characteristics. We have summarized our findings (in Section 4.6) as initial guiding principles for building energy-efficient data processing clusters.

There are a number of directions for future work, including expanding this work to consider entire workloads, exploring heterogeneous execution plans to take advantage of heterogeneous clusters, examining the impact of data skew, and investigating the impact of dynamically varying multi-user workloads.

# Chapter 5

## Energy-aware Parallel Data Processing II – Managing MapReduce Clusters

If we consider that commodity server prices are consistently falling, then it is estimated that this year, the three year cost of electricity per server will exceed the initial cost of the server itself [34]. Trends show that processor performance doubles (in number of cores) every 18 months while the performance per Watt only doubles every two years [23]. Thus, it should be no surprise that an early EPA study estimates that servers will make up 3% of the total energy consumption in the U.S. in 2011 [4].

One reason contributing to the high server energy costs is that server nodes in cluster environments are typically only 20-30% utilized, and energy efficiency in this range is under 50% [21]. This suggests that 42% of the total monthly operating cost stemming from power [80] can be reduced if we increase energy efficiency of the cluster nodes during low utilization periods.

### 5.1 Technical Contributions

This chapter discusses how to improve the energy efficiency of MapReduce (MR) clusters to exploit low utilization periods. Our methods can easily be generalized to other cluster management strategies, such as for Dryad [93], but to keep the discussion focused and better connected to prior work, we cast the discussion within the MapReduce framework.

Recently, the Covering Set (CS) method was proposed for cluster energy management [112]. The CS strategy exploits the replication that is provided by a distributed file systems (DFS), which keeps multiple copies of each data block spread across nodes in the cluster. The CS strategy designates some nodes in the system as special nodes, called the CS nodes, and keeps at least one copy of each unique data block on these

nodes. Thus, by altering the data placement policy of the underlying DFS, during periods of low utilization, some or all of the non-CS nodes can now be powered down to save energy. For example, if 33% of the nodes are CS nodes, then at 33% utilization only the CS nodes are online. CS is general and allows the CS nodes to be an arbitrary fraction of the total nodes, and also allows for part of the non-CS nodes to stay online.

With CS, the workloads take longer to run when the cluster is partially powered down, as fewer nodes are available to run the workload. Other downsides to CS include significant over-provisioning of space on the CS nodes as well as requiring code modifications in the underlying DFS software (see Section 5.4.5).

In this work, we propose an alternative cluster energy management strategy, called the All-In Strategy (AIS). In AIS, rather than increasing the response time of a workload as in the CS strategy, we run the workload (or a batch of workloads) on all the nodes in the cluster. Then, when we are in a low utilization period and the cluster is idle, the cluster is transitioned to a low power state. Thus, in AIS, rather than selectively powering down the nodes as in CS, the cluster essentially wakes up, runs as fast as it can, and then powers down again. (This is similar to the race-to-idle energy-efficiency argument [49].) One advantage of AIS is that there is a very predictable degradation in the workload response time. This degradation is based on the time it takes for the hardware and the OS to power up and down nodes from deep power saving modes, and efforts such as [120] are pushing to reduce this cost dramatically.

In fact, both AIS and CS can be thought of as two ends in a range of solutions for selectively powering down/up MR nodes to deal with low utilization periods. In this work, we present a framework for this general mechanism, and explore the effect of such a mechanism on the workload response time and overall cluster energy consumption.

Using our framework we expose two key parameters that contribute to the effectiveness of the MR energy management solutions. These parameters are: a) The response time degradation of a workload when running with fewer resources, and b) The (relative) time it takes to transition servers to and from deep power saving modes, compared to the time it takes to run the workload.

Our experimental results show that in many cases, AIS results in lower energy consumption than CS. For example, running TeraSort on a relatively small 77GB dataset on a 24 node cluster at 33% utilization is always 11% more energy efficient with AIS than with CS, and 60% more efficient than an unmanaged

cluster. Perhaps more importantly, CS also incurs a 3.6X increase in response time while AIS suffers only a 12% response time degradation. With larger complex workloads, we show these energy gains of AIS over CS improve rapidly, and factors of 2X improvement in energy or more over CS are easily possible.

## 5.2 Energy Management Framework

This section describes a framework for cluster energy management. This framework targets techniques that turn off nodes to reduce the energy consumption when the overall system utilization drops (and vice versa). The framework considers the impact of workload characteristics, hardware characteristics, and performance targets (e.g., response time goals) to bring out the interactions between these factors and the cluster energy consumption.

We present a mathematical model for the energy consumption of a MapReduce cluster during a specified time window $\upsilon$, when running a workload $\omega$ using a cluster with hardware characteristics $\eta$. For simplicity, the workload characteristics ($\omega$) and hardware characteristics ($\eta$) are considered as abstract meta-models in our model. More detailed models for capturing these can be plugged into our model. The workload characteristics model describes the job characteristics, such as an expected resource consumption, performance goals, computational complexity, etc. The hardware characteristics describes aspects such as the average power consumption of the hardware when running the workload, time and energy required to power up/down nodes, etc.

When a job arrives, the cluster is in some state, which potentially includes having some nodes already in a powered down mode. The energy management technique may choose to power up or down some nodes (based on the energy management policy and workload characteristics) to execute this workload. After the workload is done, if there is still idle time left in the window $\upsilon$, it may power down more nodes. To allow for iterative application of our model, the end state of the cluster in terms of the nodes that are online, is the same as the starting state. (Extensions to relax this assumption are straight-forward.)

Thus, using the variables in Table 5.1, the total energy consumption, denoted as $E(\omega, \upsilon, \eta)$, is:

$$E(\omega, \upsilon, \eta) = (P_{tr}T_{tr}) + (P_w^n + P_w^{\bar{n}})T_w + (P_{idle}^m + P_{idle}^{\bar{m}})T_{idle} \tag{5.1}$$

| | | | |
|---|---|---|---|
| $N$ | total # nodes in the cluster | $T_{tr}$ | total transitioning time in $\upsilon$ |
| $n$ | # online nodes running the job | $T_w$ | workload runtime |
| $\bar{n}$ | $N$-$n$, # offline nodes during job processing | $P_{tr}$ | average transitioning power |
| $m$ | # online nodes in the idle period | $P_w^{[n,\bar{n}]}$ | on/off-line workload power |
| $\bar{m}$ | $N$-$m$, # offline nodes during the idle period | $T_{idle}$ | idle time |
| | | $P_{idle}^{[n,\bar{n}]}$ | on/off-line idle power |

Table 5.1: List of Variables in our Framework

The time components for $E(\omega, \upsilon, \eta)$ must sum to $\upsilon$, so:

$$\upsilon \quad = \quad T_{tr} + T_w + T_{idle} \tag{5.2}$$

Finally, the workload characteristics may require that the job be run within some time limit, $\tau$. The cluster energy management problem can then be cast as:

$$min(E(\omega, \upsilon, \eta)) \quad | \quad T_w \leq \tau \tag{5.3}$$

Based on this model, we can see that there are several approaches to reduce the cluster energy consumption. From Equation 5.1, we see that one can reduce the energy consumption by reducing the idle energy consumption, $(P_{idle}^{m} + P_{idle}^{\bar{m}})T_{idle}$, by powering down part of the cluster. But powering down part of the cluster implies that the job has fewer nodes/resources to run, which potentially impacts the execution time of the workload ($T_w$). From Equation 5.1 this means that the energy cost to run the workload could rise as $T_w$ increases. The rate of increase in the workload energy consumption with fewer nodes will depend on the workload characteristics, and primarily the computational complexity of the workload.

From Equation 5.1, we also observe that the time to transition nodes between powered up and down states ($T_{tr}$) can have a significant impact on the energy consumption, especially when the workload energy component in Equation 5.1 is small; i.e., when the workload execution time is small, schemes that require powering up and down often will consume significant energy in transitions.

Finally, from Equation 5.1, we can see that reducing the power drawn by online idle nodes $P_{idle}^{m}$ can have a big impact on energy management schemes.

## 5.3 Energy Management Strategies

In this section, we use the framework developed in Section 5.2 to consider two cluster energy management strategies. These two strategies are: a) Covering Set (CS) – a recently proposed data placement and power down strategy to reduce the energy consumption of MapReduce clusters, and b) All-In Strategy (AIS) – a technique that we propose.

First, we present an overview of CS and AIS (Section 5.3.1), followed by various extensions to CS that are needed to make it a practical solution (Section 5.3.2), followed by a discussion of the AIS strategy (Section 5.3.3). We compare both techniques in Section 5.4.

### 5.3.1 Overview of CS and AIS

The CS strategy powers down nodes to reduce the idle energy consumption in Equation 5.1. In an ideal case, CS knows the workload perfectly ahead of time, and can power down just the right number of nodes at the start of the workload execution to reduce idle energy consumption to zero. However, as discussed in Section 5.2, such powering down of nodes can increase the response time of the workload, which in turn can increase the energy consumed during the workload execution. Thus, CS can only power down nodes such that it still adheres to performance constraints (Equation 5.3). CS also changes the data placement policy of the DFS so that one copy of the data is always online. The original CS work does not describe a strategy for powering down nodes. In Section 5.3.2 we discuss various node power down strategies for CS.

The AIS strategy is to trade idle energy consumption for transitioning energy consumption in Equation 5.1. It takes an extreme view and toggles the entire cluster between "all-nodes-on" and "all-nodes-off" modes. It uses all the nodes in the system to run the workload as fast as it can (i.e., minimizes $T_w$ Equation 5.1), and then at the end of the workload execution, powers down all the nodes to reduce the idle energy cost. The price AIS pays is a high transitioning energy cost. The scale of this increase in transitioning cost is determined by the power ($P_{tr}$) and length of time ($T_{tr}$) of the cluster transition. While this transitioning power ($P_{tr}$) may be similar to the power when the cluster is fully on and running a workload, $T_{tr}$ is solely

defined by the capabilities of the hardware and the operating system.

### 5.3.2 Covering Set (CS)

The Covering Set (CS) strategy, proposed recently by Leverich and Kozyrakis [112] aims to reduce the energy consumption of clusters by changing the data placement policy in a DFS. The main idea is that in a DFS, such as GFS [50, 65] and HDFS [33], every data block is replicated three times. The cluster energy consumption can be reduced if the server powers down some nodes. But, powering down some nodes can make some data unavailable. To avoid this case, CS changes the data placement policy so that one copy of every data block is kept on a set of nodes. These nodes constitute the Covering Set nodes and are never powered down. To reduce energy consumption, non-CS nodes can be powered down. The CS nodes can be any arbitrary fraction of the total nodes in the system. For example, the CS nodes could be 25% of the total nodes in the system, which implies that up to 75% of the nodes could be powered down when running a workload.

The CS method proposed in [112] does not include any strategy to determine which nodes to power down when the overall system utilization drops. To use CS practically, one needs such a method. A CS power down strategy's main goal is to be simple and help maintain predictable response time degradation. Also, to use CS in practice one also needs a power up method, which is the reverse of the power down method, and omitted in the interest of space. Below, we discuss and compare a number of power-down strategies.

**Random Power Down**

Consider powering down a cluster of $N$ nodes where a dataset is triply replicated and a MapReduce workload is run. Suppose utilization drops and the system responds by powering down the cluster one node at a time. (Extension to power down by more than one node in each step is straight-forward.) In this case, the work at each remaining online node goes up at the rate of $N/(N-i)$, in an $N$ node system for $i$ nodes that are powered down. The actual response time will also go up at this rate, if the computational complexity of the workload is linear.

However, randomly selecting non-CS nodes for powering down can result in suboptimal performance,

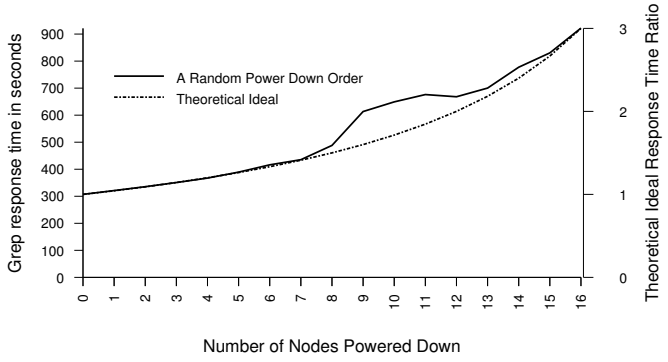



Figure 5.1: An example of load imbalance with a bad power down order on a MapReduce cluster using a CS data placement. A Grep workload was run on a 24 node cluster with a CS of 8 nodes. Ideally as $i$ nodes are powered down in an $N$ node cluster, the amount of work at each node increases by a factor of $N/(N-i)$.

as explained below. To begin this discussion, consider a distributed Grep workload on a 24 node Hadoop cluster. (More details about the system setup and workload can be found in Section 5.4.1.) In this case, the system has three racks and each rack had 8 nodes. The CS set was set to the nodes in the third rack. (Similar issues as those described below happen, if the CS nodes are spread across the racks.)

Now, consider selecting nodes at random for powering down from the two non-CS racks. Figure 5.1 shows the effect on response time for the Grep workload as nodes are powered down. Also plotted in this figure is the theoretical ideal response time curve ($N/(N-i)$) for Grep. As can be seen from this figure, there is a significant degradation in response time when the 9th node is powered down. The reason for this degradation is as follows: first, recall that for each data block, HDFS keeps one replica on a node on the same rack, and another replica on a node on another rack. Second, because of the HDFS replication policy, a natural way to produce a CS node set is to allocate an entire rack to the CS nodes (in our 3 rack case). Third, Hadoop tries to schedule Map and Reduce jobs so that they work on the data that is local (called "data local" tasks), but the Hadoop scheduler will assign tasks to work on remote blocks if some nodes have no additional unprocessed local blocks. These remote tasks incur additional overhead as they interfere with the disk activity at the remote node (which is presumably running a data local task), and incurs additional delays because of the network activity. Fourth, as non-CS nodes are powered down, the probability that nodes in the CS rack have the only copy of the data increases. Finally, if by chance there is a disproportionate number of

nodes in one non-CS rack that are turned off, then the chance that some node in the CS rack will end up with a disproportionately larger number of single replica blocks increases. This node will then be the bottleneck as some blocks on that node will probably have to be fetched remotely by other nodes for processing. In fact, this is precisely what happens in Figure 5.1 when the 9th node is powered down and the fraction of non-data local nodes increases rapidly over the previous case when the 8th node was turned off. Consequently, as can be seen in Figure 5.1, the response time degrades rapidly when the 9th node is taken down. Thus, simple random powering down has the drawback of resulting in surprising jumps in response time.

**Load Balanced (LB) Power Down**

The drawback of selecting a random node for powering down, can be addressed by keeping a precise track of the load increase that will result from powering down a node. The DFS file system keeps metadata about the placement of each block and replica, and this central metadata can be augmented to keep track of the nodes that are being powered down.

Then, when the energy management module needs to power down a node, it looks at the metadata and calculates the expected data local node load for each node. For instance, if nodes A, B, and C store the same data block $b$, then the expected node load for all three nodes because of block $b$ is $1/3$. (In other words there is a 3 in 1 chance for each node to be asked to process this block.). If node C is powered down, then block $b$ contributes a node load of $1/2$ at each node A and node B. If A, B, and C store two blocks (instead of one above), and C is powered down, then A and B have a node load $1$ each.

The "load balanced" power down strategy is simple: In response to a request to select a node for powering down, it iterates through each node, $d$, in the system and computes for each remaining node, $u$, the expected node load on the node $u$ if node $d$ is powered down. A priority queue is maintained on the *maximum expected* node load measure, and the next node to power down is the node that has the smallest maximum expected node load increase.

This load balanced power down method has some drawbacks, as the computation of the load increase can be expensive, especially for large clusters. Next, we present a simpler algorithm that also produces balanced load, but requires less storage and computation.

**Round – Robin Random (RRR) Power Down**

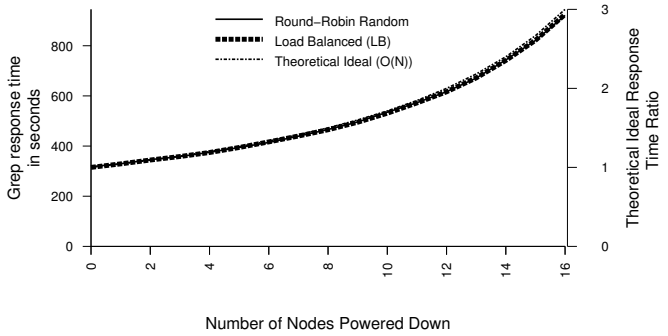



Figure 5.2: $77GB$ Grep workload response time on a $24$ node cluster as $i$ nodes are powered down. A comparison of the effects of the RRR and Greedy power down strategy and the ideal $O(N)$ degradation.

This scheme simply goes through the non-CS racks in a round robin fashion and selects a random node (that is not powered down) in each rack as the next "victim". Thus, in the case above, where we have two non-CS racks this strategy will first select, at random, a node from the first non-CS rack for powering down. In the next iteration, it will select a victim from the second non-CS rack, and in a subsequent iteration it will return back to the original first non-CS rack for victim selection, and so on. The difference between this strategy and a purely random strategy is that we do not allow any two physical racks to have their number of powered down nodes to differ by more than one. In this way, we minimize the number of single replica blocks that are created by each node power down.

This scheme is simple and requires minimal overhead to operate. We only need to keep track of the round robin sequence of the racks, and which rack needs to be examined next.

**Comparing the Power Down Schemes**

Figure 5.2 shows the corresponding behavior of the load balanced and the round-robin random schemes compared to the ideal behavior (as was shown in Figure 5.1). Figure 5.2 shows two important points. First, the response time of both the Round-Robin Random (RRR) and the Load Balanced (LB) schemes match the theoretical ideal case. Second, in this case both the RRR and LB methods have nearly identical response time. The simplicity of the RRR method, implies that it is a better method for use with the Covering Set technique.

We have analyzed these three schemes for a 24 node Terasort workload. Unlike Grep, the Terasort

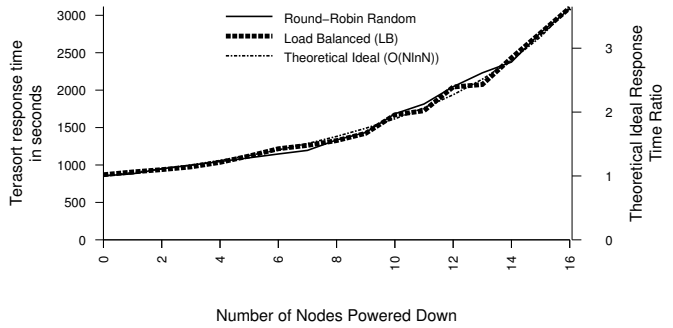



Figure 5.3: $77GB$ Terasort workload response time on a $24$ node cluster as $i$ nodes are powered down. A comparison of the effects of the RRR and Greedy power down strategy and the ideal $O(NlnN)$ degradation.

workload has an $O(NlnN)$ computational complexity. Figure 5.3 shows similar Terasort results as in Figure 5.2. That is, RRR and LB provide similar response time degradation and very closely follow the ideal $O(NlnN)$.

As we have seen here, the computational complexity models are good for modeling CS response time degradation. Further, we will need these response time models for modeling CS energy consumption. This is discussed in Section 5.4.4.

We have developed and examined three power down strategies for CS, namely: Random, Load Balanced, and Round-Robin Random. A random power down strategy for CS simply powers down the non-CS nodes randomly. It does not take into account any other cluster configuration (such as physical rack topology) besides the dichotomy of CS and non-CS, and is thus vulnerable to load imbalances on the remaining online nodes. The Load Balanced strategy we examined tries to minimize the maximum expected node load on each MR node in the cluster given a possible node power down. This method requires a good metric to determine the maximum expected node load to avoid load imbalances. Finally, the Round-Robin Random method, which we found to be both simple and balanced, iteratively powers down one random node per rack as the non-CS nodes are powered down.

Since we found the Round-Robin Random method to be both effective and efficient, for the rest of this chapter, CS discussion and results imply using this power down/up strategy with CS.

### 5.3.3 All-In Strategy (AIS)

The CS technique described in the previous section (Section 5.3.2), has a few drawbacks. First, the CS strategy requires modifying the DFS code to alter the data placement strategy. As a result, it is not a broad generic solution and it is tied to the specific replication strategies used by the DFS. (It also makes it harder to use in cases where a single system may have different data sets with varying replication factors.) Second, CS does not explicitly consider the impact on response time. As we will see for workloads like distributed Grep that have linear computation complexity, this is manageable, but for workloads that have worse than linear complexity, this is problematic as running on fewer nodes can result in rapid response time degradation, which may not be acceptable. This means that to use CS, one would need a detailed workload run time estimation technique for all the cluster configurations that CS might transition to. Finally, CS requires good workload prediction as the system has to determine how many nodes to power down, and for how long. Compared to CS, AIS does not require modifying the data placement strategy or a detailed node power down strategy, and can trivially calculate the workload response time degradation.

The strength of CS is that it maintains data availability (though not in the presence of updates/appends, as in the general case such operations require that all the nodes in the system to be online). However, one needs to make an important distinction between data availability because of node failures (which is why we have replication), and data unavailability caused by powering down nodes. The latter can simply be reversed by powering up the offline nodes. Thus, one can think of a relaxed, or "eventual data availability" in which data becomes available eventually when the node with the copy of the data is powered up (from a low power state).

We exploit this idea of eventual data availability and develop a new scheme for cluster energy management that addresses the shortcomings of CS outlined above. This new strategy is called the All-In Strategy (AIS). The AIS mechanism is simply to run the MapReduce job on all the nodes in the system and power down the entire system when there is no work. (Here, we could have a mechanism to transition the entire system which could include everything – the compute nodes, rack power supply, other power supplies, routers, etc., to and from a low power state. Or, we could have a mechanism to transition selected parts

| "Down" means idle state ($114W$) transitioning to offline state, "Up" is the reverse. | | | | | |
|---|---|---|---|---|---|
| State | Down Time(s) | Down Cost(J) | Up Time(s) | Up Cost(J) | State Cost(W) |
| Stopgrant | 1 | 114 | 1 | 114 | 112 |
| Hibernate | 11 | 1300 | 100 | 12900 | 10 |
| Off | 27 | 3200 | 156 | 20000 | 10 |

Table 5.2: Costs for different types of offline states available on our MR nodes. Hibernate and Shutdown draw 10W because the motherboard/NIC is still powered on (for IPMI).

of the entire system, e.g., only the compute nodes. The benefits are larger if more and more power hungry components provide mechanisms for quick transitions to and from power savings mode. AIS provides one more argument for building data centers out of fast transitioning hardware components.)

In cases where there is a consistent low utilization period, AIS would batch the MR jobs in a queue, and periodically power up the entire system and run the entire batch of jobs on the cluster (and then power down). For instance, for the default FIFO queue scheduler in Hadoop or the add-on Fair scheduler, AIS could batch intermittently arriving jobs to then submit all the jobs in the batch simultaneously. This idea mirrors the QED idea of energy efficient batching of database queries at a single node [103], and requires techniques for making the decision of how long to batch the jobs. These decisions could be guided by the delays that the job can tolerate (see Equation 5.3), and other workload characteristics. Workload prediction models, such as [26], would be used to guide the energy management framework. We leave such complex workload management as part of future work. (Note that CS would need such techniques too, so there is a broader set of research agendas on developing the decision making algorithms for system transitions.)

A crucial aspect for AIS is the cost to transition between low power states ($T_{tr}$, in Equation 5.1, Section 5.2) and the energy consumed in the idle state. There are a number of choices for these parameters that are offered by modern hardware. Consider Table 5.2 where we present *all* the available power up and down characteristics of one of our cluster nodes (details are presented in Section 5.4.1). In this table, it is clear that the hibernate state is the ideal energy efficient state to use; it is faster than full shutdown and consumes much less in its "off" state cost than the stopgrant state. Technology on the horizon, such as phase change memory [130], and systems research, such as automatically transitioning hardware [120], will help

reduce the transitioning costs further.

The All-In Strategy is quite simple to fit into our framework. The data placement module need not alter the respective systems data partition placement rules since the cluster operates in an all-or-nothing manner which is not affected by data unavailability. The runtime cluster node management simply keeps the entire cluster powered up when data availability is needed and powers down the cluster otherwise.

## 5.4 Evaluation

In this section, we compare CS and AIS using actual end-to-end response time and high resolution energy measurements taken on a Hadoop cluster. We used sort and scan jobs as was used in [112, 135]. This section largely focusing on *single-user latency-sensitive* environments. This type of environment can be found using Hadoop-On-Demand by Yahoo! that partitions user specific virtual cluster partitions of the physical cluster [52]. An evaluation of *multi-user throughput-sensitive* environments is found in Section 5.4.3.

### 5.4.1 Experimental Background

For our evaluation we used a cluster with 24 nodes, each with a 2.4 GHz Intel Core 2 Duo processor running 64-bit RHEL5 with Linux kernel 2.6.18, 4GB of memory, and two 250GB SATA-I hard disks. The cluster nodes were connected with Cisco Catalyst 3750E-48TD switches with gigabit Ethernet ports for each node and an internal switching fabric of 128Gbps. Switches were connected to 50 nodes and linked together with Cisco StackWise Plus giving a 64Gbps ring between the switches.

Energy measurements were taken using the following setup: The cluster is composed of 3 racks of 8 nodes each, and each rack was plugged into an APC Switched Rack PDU. AC current was measured at the APC PDU using Fluke i200s AC current clamps. Three Fluke clamps were connected to a National Instruments USB-6009 Multifunction DAQ and collected using National Instruments LabView sampling at 1KHz. RMS current was calculated using a sliding window of 20 sample points (1 period) given an AC frequency of 50Hz. The RMS voltage was measured at 116V. Our observed power factor was 0.96 and we used this to calculate the real power from the apparent power (RMS current x RMS voltage). Finally,

energy consumption was calculated by summing the time discretized real power values over the length of the workload.

On our cluster we ran Hadoop version 0.20.0 and Java version 1.6.0. We used the standard 64MB block size and set the sort buffer size to 768MB. The amount of memory given to the task tracker child process was 1024MB. The sort spill percentage was set to 0.95. The data was triple replicated. Rack awareness was enabled in Hadoop and re-replication due to under-replication was disabled. The master node that hosts the Namenode and the Jobtracker was run on a separate server (but on the same network). Finally, we ran one mapper and one reducer per node.

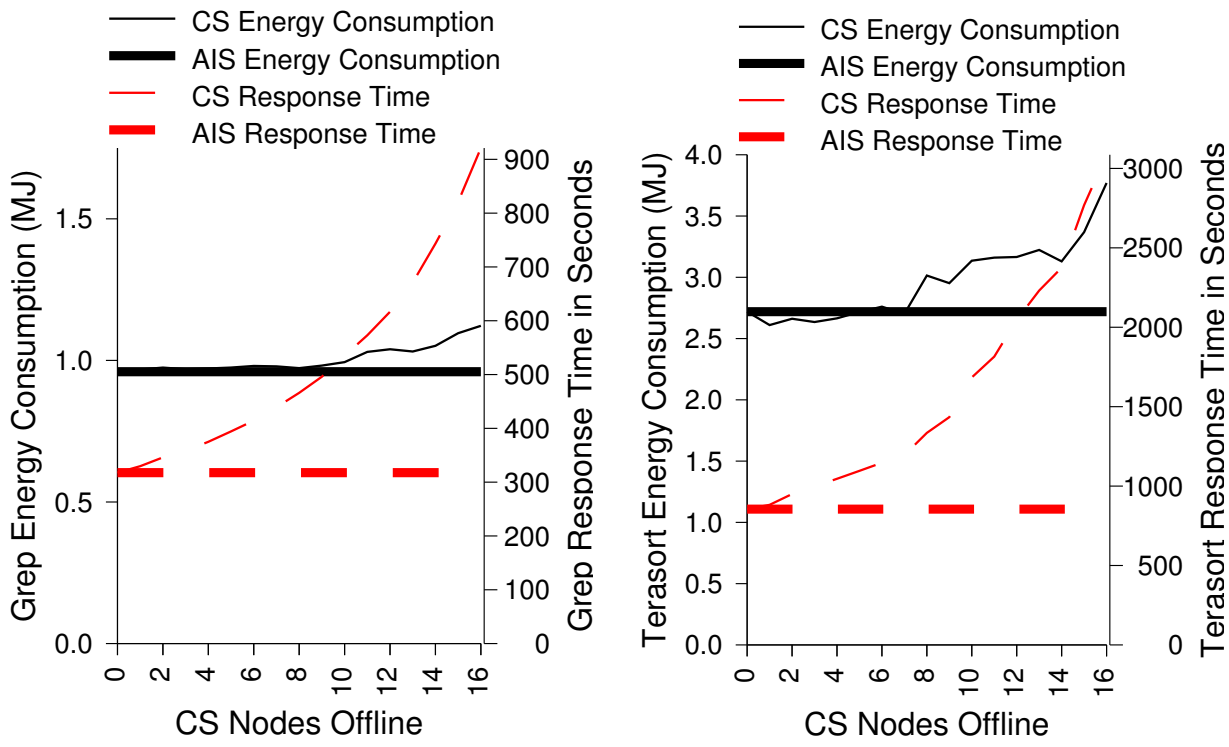
We used the distributed Grep and the Terasort workload. We chose these two workloads because of their striking differences and also because other studies [112, 135] have relied on these workloads. The distributed Grep workload is a map-only file scan job with variable selectivity and requires little additional space on disk. For the Grep workload, we ran a three character query against the Teragen dataset as was done in [135]. The Terasort workload stresses both map and reduce components of the MR framework. It needs to read and write significant amounts of intermediate data and the size of the output is equal to the size of input.

The dataset we used for these two workloads was a 77GB Terasort dataset generated using the default Hadoop Teragen application. With this size, each node stores on average 150 blocks (with triple replication). For CS, one entire rack was designated as the Covering Set. Thus, with CS we allowed powering down up to 66% of the nodes in the entire clusters.

We also ran the workloads on a 96 node cluster with a 4X larger data set and measured the response time (though not the power as we did not have enough instruments to accurately measure power for 96 nodes). The response time behavior on 96 nodes is similar to the 24 node case, and both matched the analytical model (found in Section 5.4.4).

### 5.4.2 Workload-Only Evaluation

Now, consider a best-case scenario for both CS and AIS in which the MR cluster already exists in the state that the strategy needs (see Section 5.2). In other words, for CS, the system has already powered down




(a) Grep

(b) Terasort

Figure 5.4: 77GB Grep and Terasort workload (no transitioning/idle) response time and energy consumption on a 24 node cluster using CS and AIS.

the desired number of nodes. For AIS, the cluster is fully powered up. Since our framework expects the cluster to be returned to the state in which it originated, the best-case scenario means that no transitioning costs are needed by either method and there is no idle time (Equation 5.1 in Section 5.2). In other words, we only measured the actual time and energy consumed when running the actual workload (we relax this assumption in the next experiment).

Figures 5.4 (a) and (b) show our actual measured CS and AIS results for the Grep and Terasort workloads respectively. In each figure, the workload energy consumption is plotted on the left y-axis and the response time on the right y-axis.

CS is very dependent on the workload complexity when it comes to its response time degradation. This response time degradation typically translates to increased energy consumption during workload evaluation since the linear decrease in online nodes results in a non-linear increase in response time for non-linear

jobs. We can see this result in the response time curves of Figures 5.4 (a) and (b). Grep in Figure 5.4 (a) follows a response time degradation exactly proportional to $M = N/(N - i)$ for an $N$ node cluster with $i$ nodes powered down. Similarly, Terasort in Figure 5.4 (b) shows a response time degradation proportional to $MlnM$ consistent with sort complexity (see analysis in Section 5.4.4).

Our measured energy results show that CS steadily consumes *more* energy to run the same workload with fewer online nodes. Figure 5.4 (b) shows that this increase for Terasort is about 39% between *performance mode* (all nodes being powered up) and when all the non-CS nodes are powered down. Since in this scenario, AIS consumes the same amount of energy as performance mode, AIS is up to 39% more energy efficient than CS for Terasort. For linear Grep, AIS is 17% more energy efficient than CS when all the non-CS nodes are powered down (Figure 5.4 (a)). This is because CS' offline nodes still draw 10W (Table 6.5).

The main point is that AIS consumes less energy than CS in this experiment. *Further, if the workload is super-linear in complexity, CS degrades very poorly in both runtime and energy cost.* While these are best-case scenarios that assumes that both strategies do not make any transitions, the next section presents similar results in a more detailed setting that includes both transition and idle costs.

### 5.4.3 Workloads with Idle Periods

Next we evaluate CS and AIS with full idle and transitioning costs factored in. We will present results for both latency-sensitive and (briefly) throughput-sensitive workloads.

**Latency-sensitive Workloads**

Now let us consider a scenario in which CS and AIS need to transition nodes to minimize idle energy costs (114W/node), which would happen when the cluster is underutilized. Consider a 1032 second window, which is the time it takes for CS to run the Grep workload without idle cost, including powering down all non-CS nodes (11s), execute the workload (921s with 8 online nodes), and then powering up 16 nodes to return to performance mode (100s). If we can not power down 16 nodes (due to performance limitations as discussed in Section 5.2), the workload will finish and the cluster will consume idle energy. In performance mode, CS runs Grep in 318s and then it will have to idle 24 nodes for the rest of the period which consumes

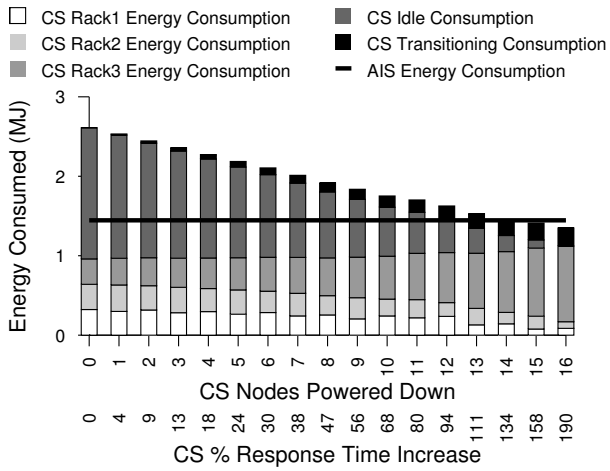



Figure 5.5: Cluster energy consumption (transitioning and idle) of Grep over a 1032s time window.

tremendous amounts of energy as shown in Figure 5.5 – see the bar corresponding to zero nodes powered down. If CS powers down nodes prior to running Grep, Grep will take longer to return and the idle costs diminish but transition costs increase (shown in Figure 5.5 where the black bars increase as more nodes are powered down). When 16 nodes are offline, there is no idle energy cost.

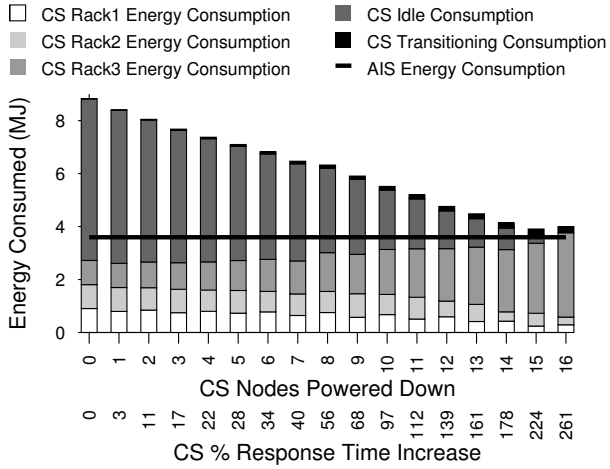



Figure 5.6: Cluster energy consumption (transitioning and idle) of Terasort over a 3197s time window.

Similarly, in Figure 5.6 we present the same analysis for Terasort but over a 3197 second window where CS can run Terasort on 8 nodes (3086s) and perform all round-trip transitions (111s) to performance

mode. Similarly, the energy consumption over the window decreases as we lengthen the workload running time to fill 3197 seconds and erase all idle time cost.

In Figures 5.5 and 5.6, the energy consumption of AIS during the respective time windows includes the cost to power up the 24 node cluster, run the workload, power it back down, and draw 10W per node (see Table 6.5) while they are powered down for the rest of the time period. For Grep, AIS consumes less energy during the 1032s window than CS most of the time until CS powers down 13 or more nodes. Since AIS has the overhead of transitioning, this cost makes AIS less desirable for this short workload. However, for the Terasort workload, we notice that AIS *always* consumes less energy during the 3197s window. Due to the complexity of the workload, AIS' overhead costs are less than the energy consumption increases of CS. Consequently, AIS saves 10% in energy over CS even at CS' more efficient operating state.

Consider an example of a scenario where the response time performance requirements (Equation 5.3) cause CS to consume more energy than AIS because it cannot power down sufficient nodes to eliminate idle cost. Let the tolerable level of Grep response time degradation be 50% ($\tau = 450s$). Figure 5.5 shows that during this underutilized period, CS will consume 33% *more* than AIS because CS can only power down $8$ nodes and draws idle power.

This problem is even worse with a super-linear complexity job such as Terasort. If acceptable response time $\tau = 1300s$ (in Equation 5.3) is 1.5X the performance mode response time, then CS can only power down 7 nodes (less than for Grep). Then Figure 5.6 shows that CS consumes 80% *more* than AIS!

Thus, the response time degradation with CS can make it untenable in many operating environments, even with moderately acceptable response time degradation.

**Throughput-sensitive Workloads** We have also evaluated AIS and CS on throughput-sensitive workloads, and observed that AIS can save significantly more energy than CS given a fixed level of throughput degradation. We ran a heterogeneous workload of Grep and Terasort jobs (similar to [112]) and used CS and AIS to manage the cluster energy consumption. In our results we found that when AIS batches jobs, it consumes 26% less energy than CS given a acceptable throughput degradation (3%). These results are not surprising, and follow the same intuition from the evaluations in Section 5.4.3; CS results in rapid response time degradation which impacts both the energy consumption and throughput.

For AIS, the method that we employ is the batching method described in Section 5.3.3. AIS keeps the MR cluster powered down while jobs are batching [103,163]. When enough MR jobs are collected, AIS powers up the cluster and submits all the jobs. The job collection, or batching delay, effectively degrades throughput.

In contrast, CS runs the jobs as they arrive but can process them with some MR cluster nodes powered down. However, as fewer nodes are available for the job, this also lowers throughput.

Our throughput workload mimics that of [112] whereby sort and scan jobs are injected into the MR cluster. We use the same Terasort and Grep jobs of Section 5.4 whereby each job runs on a 77GB dataset.

We evaluate a heterogeneous job workload consisting of four sort and four scan jobs, randomly ordered and individually submitted to the 24 node cluster in 850 second intervals. Recall from Section 5.4.2, that the Grep job can run in about 300 seconds and the Terasort job can run in 850 seconds.

Given this normal operating environment, the cluster throughput is essentially eight jobs in 6800 seconds with an energy cost of 20.5MJ. Now, suppose we have a tolerable throughput degradation of 3% which means we can accept eight jobs in 7000 seconds.

Using CS, we can power down nodes, degrading throughput, and potentially saving energy. We found that if CS powers down 3/24 nodes, then its throughput degrades to 6984 seconds. Given this throughput, CS with 21 nodes powered up, consumes 18.8MJ of energy for this eight job workload.

Now using AIS, if we delay jobs such that we can then submit two jobs in a batch to the system at a time, then our measured throughput for eight jobs is 6962 seconds (which includes all batching time). With this throughput, AIS will power down all the nodes while batching jobs and power the entire cluster up to execute the batch. The measured energy consumption for AIS is 13.9MJ (which includes all transitioning costs). Therefore, for this heterogeneous job workload, AIS saves 26% of CS' energy consumption when both have equal throughput rates.

### 5.4.4 Effects of Workload and Hardware

In this section, we analytically model the effects of workload and hardware characteristics on CS and AIS to fully explore the pros and cons of these methods in diverse workload and hardware settings.

The response time of any AIS job is simply the performance mode (all nodes online) response time plus transitioning time. Similarly the energy cost for AIS is simply the performance mode cost plus the transitioning costs. For CS, we have shown that the workload complexity determines the response time when nodes are powered down (Section 5.4.2). Energy modeling for CS similarly requires incorporation of the workload complexity and also the transitioning costs.

**Modeling Performance and Energy Consumption**

Modeling AIS is simple. AIS only has two different operating modes: *performance mode* in which the entire cluster is always powered up and *energy savings mode* in which the cluster is powered off until it needs to be powered up to fulfill a job request, and then powered back down. Thus, the response time modeling of the energy savings mode simply requires adding the times associated with each of these components. Furthermore, energy consumption modeling can be similarly defined to be the sum of the performance mode energy consumption, and the cost to power up and down the cluster nodes.

CS on the other hand, is more complex in its modeling of response time and energy consumption. We have already presented results showing that the computational complexity of the workloads can be used to accurately model the response time degradation of CS as more nodes are powered down (Section 5.3.2).

If we recall Equation 5.1 from Section 5.2, the workload energy consumption is $E_w = (P_w^n + P_w^{\bar{n}})T_w$, which considers the power drawn by both the online and offline nodes during the actual workload execution. The effect of CS powering down nodes is that the idle energy shrinks, and eventually reduces to zero. As CS powers down more nodes, ($P_w^n$) decreases while ($P_w^{\bar{n}}$) increases. Using Equation 5.1, we can model the energy consumption of the workload under CS when we substitute $T_w$ with the workload complexity models just described. Any transitioning and idle costs are straight-forward to include as we just add them to the workload cost. For simplicity, we do not include them in our modeled analysis of CS.

Figure 5.7 shows a comparison of the observed and modeled energy consumption of the Grep and Terasort workloads using the workload energy consumption $(P_w^n + P_w^{\bar{n}})T_w$ from Equation 5.1. For this figure, we used the power data shown in Table 6.5, along with using O(N) and O(NlnN) complexities to model the response times of Grep and Terasort respectively. The results shown in Figure 5.7 demonstrate that the models are quite accurate in predicting the energy consumption of CS: an average error of 1% and

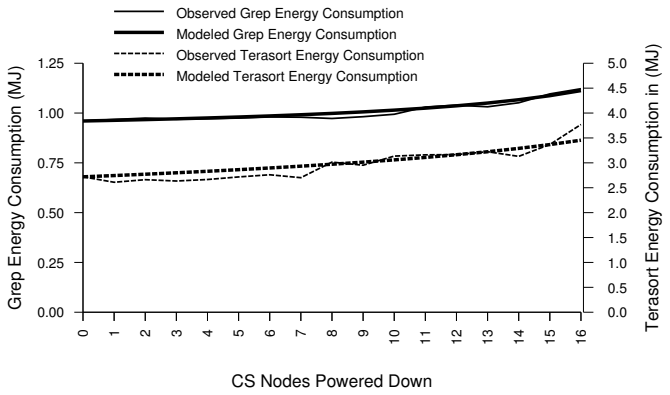



Figure 5.7: Comparing the observed and modeled energy consumption of the Grep and Terasort workloads for CS. The average error is 1% and 4% for the Grep and Terasort models respectively.

4% for Grep and Terasort jobs respectively.

**Workload Characteristics and Hardware Capabilities Matter the Most**

Given the results of Section 5.4.3, we have shown that the main factor that affects AIS is the transitioning costs that it has to incur. But the question is how does this transitioning cost affect AIS' potential advantage over CS?

To explore this question, let us model the differences between CS and AIS when we have powered down 50% of a 2000 node cluster. Furthermore, we increase the amount of data that needs to be processed given constant transitioning parameters such as the node power up time. We assume that each node draws 150W when running a job (an average from our empirical results) and the cluster nodes have the hibernate transitioning characteristics as in Table 6.5.

For simplicity, in the following analysis, for CS, we assume that the cluster is already powered down appropriately, and do not add any transition costs for CS. However, for AIS, we include the full power up and power down that is required. For both, we do not include any idle time cost.

Figure 5.8 shows the energy consumption and response time of CS and AIS for Terasort on this 2000 node cluster as we increase the amount of data to be sorted. As the results in Figure 5.8 show, the relative rather than the absolute transition time is the important factor since as we increase the workload length, AIS' transitioning penalties will be overcome. Thus, the relative transitioning time is a significant

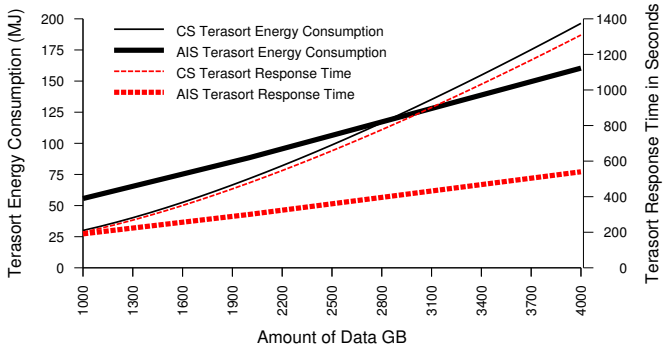



Figure 5.8: Analytic comparison between CS and AIS response time and energy cost for Terasort. CS uses 50% of the 2000 node cluster. Hibernate parameters are found in Table 6.5. Average operating power is 150W for each online node.

factor in determining the feasibility of AIS. When running a 1TB sort job with half the nodes in the cluster, CS provides better energy efficiency and response time characteristics than AIS. However, at this point, $T_{tr} = 111s$ is about half of the performance mode response time ($200s$). Now as the data size increases, the workload response time increases while the $T_{tr}$ factor remains constant. As this happens, the advantage of AIS becomes apparent – beyond a 2.8TB data set (1.4GB/node), AIS is both faster and more energy efficient than CS.

The results above show that the absolute value of $T_{tr}$ is not important, but rather the important measure is the ratio between $T_{tr}$ and workload response time when run in the performance mode. Thus we call this measure the *relative* $T_{tr}$.

Figures 5.9 and 5.10 compares the response time and energy consumption characteristics respectively, of both methods for workloads with varying computational complexity, as we increase the proportion of the cluster that is powered down by CS. In these figures, we show four different cases for AIS, with *relative* $T_{tr}$ values of $1\%$, $5\%$, $10\%$, and $20\%$. Since our observed transitioning power ($P_{tr}$ in Equation 5.1) is approximately equal to workload power ($P_w^n$), the rel. $T_{tr}$ also translates to the relative increase in AIS energy consumption.

In these figures, we have presented the proportional increase in response time and energy consumption for both CS and AIS over an "ideal" case in which the hardware is perfectly energy-proportional, for three

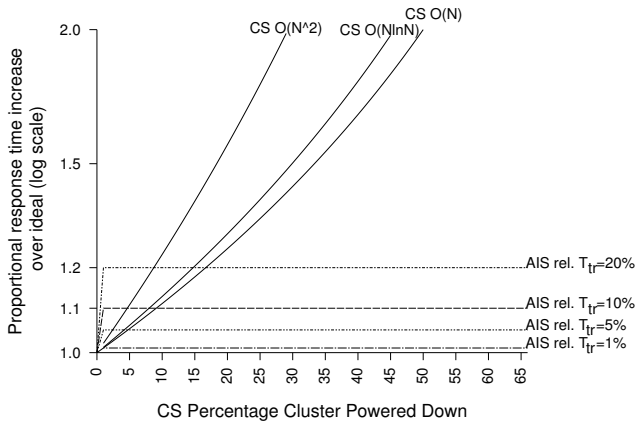



Figure 5.9: An analysis on the effect of workload complexity and relative $T_{tr}$ on CS and AIS workload response time.

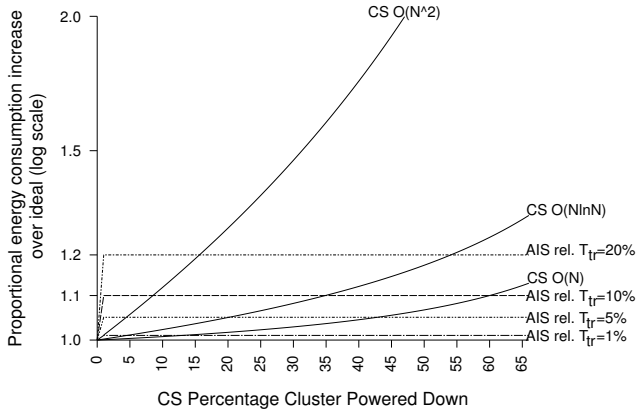



Figure 5.10: An analysis on the effect of workload complexity and relative $T_{tr}$ on CS and AIS workload energy consumption.

different classes of jobs with linear, sort, and quadratic computation costs.

From Figure 5.9, we observe that across all workloads, even with the largest $20\%$ relative $T_{tr}$, AIS generally has a better workload response time than CS. This is not surprising as AIS runs the workload in *performance mode*, and its response time degradation is based only on the transitioning overhead. AIS has worse response time than CS only when the relative $T_{tr}$ is very large.

Looking at the energy consumption in Figure 5.10, we notice that if the relative $T_{tr}$ is large (e.g., $20\%$), then AIS will consume too much energy in transitioning, and will only be more effective with

| Relative $T_{tr}$ | $O(N)$ | $O(NlnN)$ | $O(N^2)$ |
|---|---|---|---|
| 1% | AIS | AIS | AIS |
| 5% | CS/AIS | AIS | AIS |
| 10% | CS | CS/AIS | AIS |
| 20% | CS | CS | AIS |

Table 5.3: Summary of the two main factors that discriminate CS from AIS: Workload Complexity and Relative Transitioning Cost. This summary is based on the workload energy consumption since the response time performance of AIS is better than that for CS in the vast majority of cases.

large complex workloads, where the computational complexity of the workload is high (e.g., polynomial or worse). AIS' energy consumption drops rapidly as the relative $T_{tr}$ decreases. For example, with a relative $T_{tr}$ of 1%, AIS is the preferred strategy for all three workloads. In this environment, for a sort workload, if the cluster is powered down by 66% by CS, then AIS saves more than 30% in energy consumption over CS.

These results also shows that if the transitioning cost is high (relative $T_{tr} \geq 10\%$), then generally CS provides better efficiency because the AIS energy cost of powering up the entire cluster overshadows any inefficiencies from operating CS with a smaller commitment of resources.

Table 5.3 presents a two dimensional summary of the factors that affect the energy efficiency of AIS and CS. These factors are the computational complexity of the workload and efficiency of node transitioning ($T_{tr}$). Table 5.3 shows that AIS is favoured when the workload computational complexity is high. Furthermore, when AIS and CS provide about the same benefits (linear-rel. $T_{tr} = 5\%$ and sort-rel. $T_{tr} = 10\%$), AIS is preferred when the fraction of idle time is high and CS needs to power down a large proportion of the nodes. Finally, when the $T_{tr}$ factor is small, AIS is preferred even when the computational complexity is linear. Of course, this summary is caveated with the assumption that the CS response time degradation is acceptable. As shown in Section 5.4.3, if the CS response time is unacceptable, AIS is the preferred method.

### 5.4.5 Discussion

In this section, we discuss various implications on implementing and running AIS and CS. We also discuss other cluster energy management methods that fit in our framework.

**Drawbacks of CS**

There are three important drawbacks of CS which need to be considered when deploying CS.

**Storage Over-provisioning** – CS requires *significant* over-provisioning of storage for the Covering Set nodes. Consider a large five terabyte dataset on 100 nodes. With DFS triple replication, the nodes must collectively store 15TB of data. In addition, the output of Terasort takes another 15TB (assuming it is also triple replicated). This means that in performance mode, each node must have 300GB of storage for this workload. But when CS powers down all 66 non-Covering Set nodes, each Covering Set node must now have 600GB of storage. Essentially, the online nodes must be over-provisioned in storage, consuming even *more* energy. (This is why our real workload results ran relatively small Terasort jobs.)

**Response Time Degradation** – As discussed in Section 5.4.3, Figure 5.5 and 5.6 shows that CS can only save energy when it commits exactly the right amount of resources such that all the idle time in a given time window is erased (Equation 5.1). However, this requires that the workload is willing to tolerate a potentially large response time penalty (constraint $\tau$ in Equation 5.3). If this response time penalty is not acceptable and CS must commit more resources and incur more idle energy, Figure 5.5 and 5.6 shows that AIS will consume less energy than CS for the majority of the cases. Section 5.4.3 shows that with an acceptable 50% increase in response time, AIS can save up to 80% of CS' consumption.

**DFS modification** – The last drawback of CS is that it requires modifying the data placement code in the DFS. These changes can be complicated if one has to deal with creating new data when the cluster is in power savings mode, and when the cluster has heterogeneous nodes.

**Hybrid Approaches**

The effectiveness of AIS for energy management improves as the relative $T_{tr}$ value drops, and the effectiveness of CS improves as the workload computational complexity decreases. As a result, each method may have its sweet spot for a given hardware and workload characteristics. However, it is possible to combine AIS and CS to build a hybrid solution.

For example, if CS runs with a combination of CS and some non-CS nodes up (e.g., to cap the response time degradation), then after running the workload, the non-CS nodes could be powered down, or all the nodes in the cluster could be powered down.

## 5.5 Summary

In this chapter we have presented a general framework for designing and evaluating methods to reduce the energy consumption of MR clusters. We have also investigated the class of techniques that power down (and power up) MR nodes to save energy in periods of low utilization. Using this framework, we closely examined two broad strategies for MR energy management – a recently proposed strategy called CS, and a new strategy called AIS that we propose in this work. We also compared these two techniques within the context of MR systems. Our results show that there are two crucial factors that affect the effectiveness of these two methods (and generally any energy management method that fits in our framework). These factors are the computational complexity of the workload, and the time taken to transition nodes to and from a low power (deep hibernation) state to a high performance state. We evaluated both CS and AIS on an actual cluster, and also developed an accurate and detailed analytical model for both methods. Our evaluation shows that CS is more effective than AIS only when the computational complexity of the workload is low (e.g., linear), and the time it takes for the hardware to transition a node to and from a low power state is a relatively large fraction of the overall workload time (i.e., the workload execution time is small). In all other cases, which tend to be the common cases for MR systems, the benefits of AIS over CS are significant – both in terms of energy savings and response time performance.

The crux of the CS approach is to leverage data replication to allow us to selectivity use more or less cluster resources. Given that we found potential load balancing problems when exploiting data replication. In the next chapter we will study the relationship between energy management, data replication, and load balancing in more detail.

# Chapter 6

## Energy-aware Parallel Data Processing III – Exploiting Data Replication

As we have seen in Chapter 5, clusters that are typically underutilized [21], can be made more energy efficient by carefully exploiting data replication to power down or de-allocate cluster nodes. However, care must be taken as it was shown that load imbalances may occur when we remove cluster nodes. These load imbalances come about from naïve power-down sequences and/or replication schemes that are ill-suited for our purposes.

### 6.1 Motivating Example

To see this point, consider a system that uses the common replication strategy of mirroring partitioned data. To make this example more concrete, suppose that there are four nodes using mirrored replication. In addition, suppose that the data set is split into two partitions, $P_0$ with mirror $R_0$, and $P_1$ with mirror $R_1$. Assume that node $n_0$, $n_1$, $n_2$, and $n_3$ store $P_0$, $P_1$, $R_0$, and $R_1$ respectively. Furthermore, assume that queries can be sent to either the primary copy or the replica for load balancing. If the overall system utilization is at or below 50% of the provisioned utilization, then nodes $n_2$ and $n_3$ could be turned off to save energy, while nodes $n_0$ and $n_1$ would then operate at 100% utilization. This is an ideal scenario and may be sufficient for certain systems. However, we wish to explore powering down nodes when utilization is between $50 - 100\%$ for a finer grained energy management scheme.

Now, consider another scenario in which the four nodes each initially see a load of 75%. The system has the capacity to run this workload on only three processors. Furthermore, by exploiting replication, we can certainly turn off one processor and still maintain access to all data.

Unfortunately, if we turn off node $n_3$, then nodes $n_0$ and $n_2$ will continue to operate at 75% utilization, but now both node $n_1$ and node $n_3$'s original load will be directed at node $n_1$, so the presented load there will be 150%, and the system will likely fail to meet its performance requirement. Such large load imbalances may be acceptable in certain environments, but the performance degradations are usually unacceptable (see Sections 6.3.1 and 6.4.1 for more details).

Given this example, our goal is to investigate the interaction between replication and power down schemes to provide the foundation for energy management approaches that gracefully adapt to overall system utilization. This should be done in such a way as to maximize energy efficiency by powering down some nodes while ensuring that the utilization of the remaining nodes does not exceed a targeted peak utilization.

The database and distributed systems communities have a rich history of designing various replication schemes for reliability [3, 14, 28, 32, 90, 134]. This raises the question of whether or not there is a replication scheme that can be exploited to better meet our goals than the commonly used mirroring strategy adopted in our example above. As we will demonstrate, the surprising answer is yes — one of the earliest proposed parallel database data replication schemes, the "Chained Declustering" technique [90], when coupled with careful choices of which nodes to power down, can be exploited to achieve the above goal.

## 6.2 Technical Contributions

In this chapter, we explore node power down sequences that leverage Chained Declustering to mitigate the load imbalances created by other replication and power down sequences. We present two node power down techniques, called "Dissolving Chains" and "Blinking Chains", that view the nodes in the cluster as a "chain" and then specify which nodes are powered down as load drops (the power up sequence in response to an increasing load follows a reverse strategy, as discussed in Section 6.5). In Dissolving Chains, as system utilization decreases, it simply powers down more nodes (the chain dissolves). Blinking Chains differs in its power down transition because it may first power up some nodes before powering down the desired number of nodes (the chain blinks) in order to reduce load imbalances.

This is the first energy-efficiency study that explores this interaction between power down sequences and replication strategies while controlling load imbalances.

In addition, we also evaluate these techniques using an extensive experimental methodology, which includes using an actual commercial DBMS, and show that: (1) given an input parameter, namely, the percentage load imbalance the power management scheme is allowed to introduce, our method guarantees that it will not introduce any additional load imbalances beyond that percentage. This percentage refers to the tolerable load imbalance that the system is allowed to take. As we will see, our methods produce low imbalances (none at some points), and this measure can be used by the system to determine if a certain power down transition is acceptable. (2) Our methods have the potential to produce significant energy savings (of $40\%$ or more) over a wide variety of system loads while maintaining data availability and a well-balanced system; and (3) our methods provide a trade off between mitigating load imbalance and ease of transitioning between operating states.

## 6.3 Background and Problem Specification

Before we proceed, we define a few terms that we use throughout this chapter. We use the term *load* on a node to refer to the work that is being carried out on a node. In a system with a number of concurrent queries, each with the same processing cost, the load can simply mean the number of queries per node.

The term *utilization* of a server node refers to the resource consumption on the node. Typically utilization of a system in cluster environments is measured simply as the CPU utilization [21, 62], which is a simplistic measure as it ignores other resources such as memory, disk, and network, but often works well in practice. The term *overall system utilization* refers to the average utilization across all the server nodes in the system. *Maximum node utilization* refers to the maximum utilization across all the server nodes.

Often cluster systems are designed to handle a certain provisioned *peak* load. We will often refer to the utilization using a value expressed as a percentage. Within this context, a utilization of 100% simply refers to operating at an initial designated "peak load" (which could be lower than the system's peak load at which it is stable). Lower utilization values, e.g., 50%, imply a corresponding reduction in the load (and an increase in server idle time).

The energy management schemes that we describe in this chapter, work by taking some nodes *offline*, which refers to a node being powered down to save energy. Nodes that are available to run queries are

*online*. An offline node becomes available when it is powered up, in which case it then comes online. (In the more traditional case of replication for failure management, offline refers to the node being unavailable due to some component failure.)

Finally, an operational state for the entire system is defined as:

**Definition 6.3.1.** *The **operating state** of the entire system, $s(m)$, is a state where $m$ of the $N$ total nodes in the system are offline.*

### 6.3.1 Server Load vs. Energy Consumed

In this section we discuss the interaction between the load on a server and the energy consumed by the server. The main point here is that this relationship is not linear — even at zero load, a server consumes an unfortunately high fraction of its fully loaded energy requirement, mirroring the observation in [21], but for DBMSs.

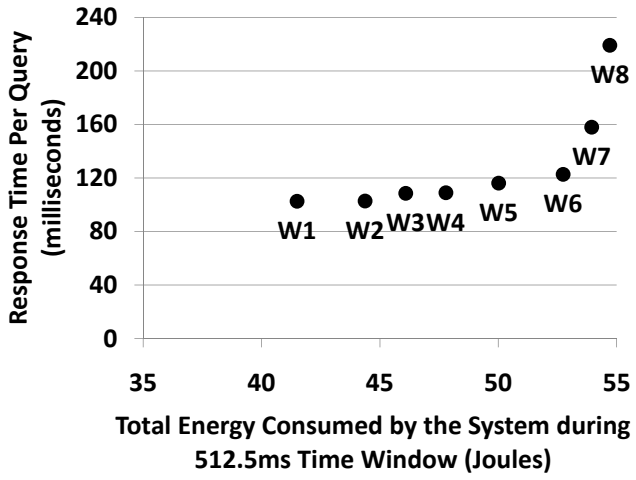



Figure 6.1: Energy Consumption and Response Time Profile

As an example, consider Figure 6.1, which shows the characteristics of a 1% clustered index query workload running on a commercial DBMS. (Each point is actually an average over a thousand runs; more details about this workload are presented in Section 6.6.2.) In this graph, the point W1 corresponds to a server workload in which one instance of the query takes X ms to run followed by the server being idle for

4X ms. One can view this workload as a series of time windows, each of size $5X$ ms, where $X$ is the time to run the query. For workload W1, only one query is run in each window.

Other points in this graph correspond to higher server utilizations, which we achieved by randomly adding more queries in the time window (of length 5X ms), thereby reducing the idle component. Specifically, a point W$i$ corresponds to injecting $i$ queries, with random arrival times, into each $5X$ ms time window. Figure 6.1 shows for each workload the average execution time per query and the energy consumed by the server to run the workload.

Now, consider the point W1 in Figure 6.1. In this case, the server consumes about 41.5 Joules and provides a query response time of 102.5 ms. Most of this energy, specifically 74%, is consumed while the server is idle. As we add more queries to the workload, i.e., go beyond W1, the idle time decreases and a larger fraction of the energy consumed by the server is spent actually running the queries. At W5, since each query takes X ms to run, we are running at some provisioned "peak" utilization of $100\%$. Notice how performance rapidly degrades beyond W6. Operating at such points (W7 and beyond) merely to save power may be unacceptable as this region likely represents an unstable operating range.

If efficiency is defined as the energy consumed by the server per query, of the five workloads W1 to W5, W5 has the highest efficiency. Notice, however, the response time per query is slightly worse at W5 than at the other four points, since at the other points there is less contention for resources across different queries.

Thus we have two possibly conflicting optimization goals. The first is the traditional one — we could simply optimize for response time, which means running the system at point W1. However, typically in data center environments, the performance constraint to meet is not "as fast as possible;" but rather, something more like "no worse than $t$ seconds per query for this workload." When agreeing to such Service Level Agreements (SLAs), data center service providers tend to be conservative and agree to performance that they can generally guarantee under the heaviest provisioned load, rather than performance they can meet in the best case. Consequently, the second optimization goal, and the one that we focus on in this work, is to reduce the energy consumption while staying below a response time target.

### 6.3.2 Problem Statement

We want an energy management scheme that starts with an operating state $s(m)$ for a system with maximum node utilization of $u$ ($u < M$). Here $M$ refers to some maximum tolerable system utilization (perhaps defined by an SLA). We want the system to move to a new operating state $s(m')$ with maximum node utilization $u'$ such that $u' < M$ and $m \leq m'$, and at least one copy of each data item is available on the remaining servers that are still powered up.

Note that $M$ is defined relative to the initial designated peak load (see discussion at the beginning of Section 6.3). Consequently, $M$ can be greater than $100\%$; e.g., if the maximum tolerable response time is $120ms$ in Figure 6.1, then $M$ is $120\%$ (at W6).

Notice that the problem statement also allows setting M to $100\%$, in which case no node operates over the designated peak capacity.

In addition, in our problem formulation we require "data availability" – i.e., the power down sequence does not deliberately make any data item unavailable on the live servers that are powered up. We make this assumption since the time it takes to bring up a powered down server can be very high (e.g., booting up from system-off or from hibernation – see Section 6.6.5), and any queries against data that is made unavailable by a power down scheme will incur this latency. This high latency/delay may be unacceptable if the queries are short (as opposed to longer data-processing tasks such as MapReduce jobs – see Chapter 5), and also makes it harder to maintain the fault-tolerance property of replication in the presence of updates (See Section 6.5.3).

The schemes that we present differ in the "variance" in the load across the different nodes. In other words, some schemes result in larger variation in the loads across the nodes (see Section 6.6.4, Figure 6.6). While load variance (imbalances) are inevitable, and minor load imbalances do not create a problem, artificially creating major load imbalances can result in the system failing to meet its targeted performance (e.g., W7 and W8 in Figure 6.1). Accordingly, we require that the energy management techniques bound the load imbalances ($M$) that they introduce.

The parameter $M$ can be set based on what the system administrator feels is a comfortable upper bound for that system (e.g., W6 in Figure 6.1). Note such a bound is important as it provides a guarantee

that the energy management method will not introduce unbounded load imbalances. We expect that there might be other sources of imbalances that the system might face, such as flash crowds. In such situations, the system can be pulled out of energy-savings mode. Now the situation is the same as what happens today when systems are faced with sudden load changes. The system can then execute whatever method it is currently using to deal with load fluctuations. It is an interesting direction of future work to see if we can improve upon this scheme to more deeply integrate flash crowd load management and prediction with energy management techniques that are proposed here, and/or to pick $M$ automatically based on other system operational settings.

Finally, for certain system states, the nodes can be "perfectly balanced" – which means that each online node has the same node load. In Section 6.5.2, we discuss these perfectly balanced states.

## 6.4 Candidate Replication Strategies

In parallel and distributed data processing systems, replication allows continued access to data when some nodes fail. Here we want to exploit replication for a related but different purpose: namely, allowing continued data access not when nodes fail, but when they are deliberately powered down to save energy, while controlling the resulting load imbalance.

When we look at the commonly used techniques: RAID [134], Mirrored Disk [28,32], and Interleaved Declustering [3], we find that they all produce undesired load imbalances as nodes become inoperable or do not allow us to turn off multiple nodes. For instance, Interleaved Declustering retains load balance when one node fails but loses data availability if any additional nodes are lost.

RAID storage uses an array of disks controlled either by hardware or software to act as a single unit. Different RAID levels define different storage properties such as parallel data access, data redundancy, and data recoverability. However, RAID suffers from load imbalances when operating in failure mode. For example, in RAID 1, if a disk fails, the redundant copy disk must now handle all the requests that were shared across the two disks. Recent methods for a Power-Aware RAID [178] attempt to solve this problem with distinct energy saving operating states. However, these methods require pre-determining all the operating states and are generally not adaptable to changes in data size.

Our goal is to leverage a replication scheme to safely and easily power down any number of nodes for energy efficiency, and exploit the load balancing and failover properties of replication.

Note that we are powering down nodes to save energy, but the node has not failed. In other words, our schemes don't change the fault-tolerance property of replication (updates require special care as discussed in Section 6.5.3).

### 6.4.1 Mirroring Replication

The basic principle used in mirroring [28, 32] is to make a second copy of the data and store it on a different storage device. Mirroring can be implemented in a variety of different ways. One mechanism is to have disk pairs (RAID 1), with one disk storing the primary copy and the other storing the mirrored copy [32]. Access to the disks could have redundancy (e.g., there could be dual ports) so that if a controller fails the disk can be accessed from a different port (but this adds hardware costs). Mirroring can also increase parallelism by allowing queries to use either copy. When one disk fails, the mirrored copy takes over the work of its pair. However, this technique doubles the load on the disk that is still up.

There are a variety of different ways of mirroring data. However, in most schemes, when some disk fails, the load on the remaining copies goes up dramatically. For example, if we use a 2X replication scheme, in which we have a primary copy and one additional replica, then when a disk with either of these copies fails, all the load from the failed disk is transferred to the remaining disk. Thus, if we say that the cluster system can only operate as fast as its bottleneck, when a node is taken offline, the system operates at 2X load. In other words, if one decides to take one node offline in a mirrored scheme, from a load perspective, one might as well take half of the system offline. This means mirroring essentially has only two operating states, $100\%$ online nodes or $50\%$ online nodes. If a 2X increase in load is unacceptable (results in a maximum node utilization beyond an acceptable threshold), then with this scheme, there is no energy savings if the system load is between 50 and 100%. Our goal is to design schemes that will let us power down an appropriate number of nodes at *any* utilization between $50 - 100\%$ without creating unacceptable overloads, given a 2X replication scheme.

Fortunately, Chained Declustering [90] seems to have the properties that allow this exploration.

| Nodes: | $n_0$ | $n_1$ | $n_2$ | $n_3$ | $n_4$ | $n_5$ | $n_6$ | $n_7$ |
|---|---|---|---|---|---|---|---|---|
| Primary: | $R_0$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ | $R_7$ |
| Backup: | $r_1$ | $r_2$ | $r_3$ | $r_4$ | $r_5$ | $r_6$ | $r_7$ | $r_0$ |
| Load: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Table 6.1: An 8 node Chained Declustered ring without failure.

Chained Declustering can lose multiple nodes in the cluster and maintain data availability. For this reason, in the rest of this chapter, we consider techniques built upon Chained Declustering.

### 6.4.2 Chained Declustering (CD)

Chained Declustering [90] is a replication scheme that stripes the partitions of a data set two times across the nodes of the system, thereby doubling the amount of required disk space. The main hallmark of this scheme is its tolerance to multiple faults along the chain, if those faults do not occur on adjacent nodes. Furthermore, along with high availability, the arrangement of the replicas along the chain allows for balanced workload distribution when some nodes are offline. If one thinks of all the nodes in the system as being arranged in a ring or chain, then Chained Declustering (CD) places a partition and its replica in adjacent nodes in the chain.

As an example of CD, consider a data set $R$, spread over $8$ nodes in Table 6.1. Here the primary copies of the data set are $R_0$ ... $R_7$. The corresponding replicas are shown as $r_0$ ... $r_7$. The nodes $n_0$ ... $n_7$ are conceptually organized in a ring. Primary copy $R_i$ is placed on node $i$ and its replica $r_i$ is placed on the "previous" node. During normal operation, if the access to all the partitions is uniform, then the queries simply access the primary partitions while updates in CD go to both partitions.

Now consider what happens when a node is taken offline by our energy management methods. Table 6.2 shows what happens when node $n_0$ is offline. Since node $n_0$ holds the partition $R_0$, all queries against this partition must now be serviced by node $n_7$, which holds the only other copy of this partition. But simply redirecting the queries against partition 0 to node $n_7$ could double the load on node $n_7$. CD

| Nodes: | $n_0$ | $n_1$ | $n_2$ | $n_3$ | $n_4$ | $n_5$ | $n_6$ | $n_7$ |
|---|---|---|---|---|---|---|---|---|
| Primary: | — | $R_1(1)$ | $R_2(\frac{6}{7})$ | $R_3(\frac{5}{7})$ | $R_4(\frac{4}{7})$ | $R_5(\frac{3}{7})$ | $R_6(\frac{2}{7})$ | $R_7(\frac{1}{7})$ |
| Backup: | — | $r_2(\frac{1}{7})$ | $r_3(\frac{2}{7})$ | $r_4(\frac{3}{7})$ | $r_5(\frac{4}{7})$ | $r_6(\frac{5}{7})$ | $r_7(\frac{6}{7})$ | $r_0(1)$ |
| Load: | 0 | $\frac{8}{7}$ | $\frac{8}{7}$ | $\frac{8}{7}$ | $\frac{8}{7}$ | $\frac{8}{7}$ | $\frac{8}{7}$ | $\frac{8}{7}$ |

Table 6.2: An 8 node Chained Declustered ring with 1 failure.

solves this problem by redistributing the queries against partition 7 across both copies of that partition's data, namely $R_7$ and $r_7$. It does this for all the partitions, and ends up with a system in which each node is serving the same number of queries (i.e., the system is balanced after the node failure.) As shown in Table 6.2, for partition 7, $6/7^{th}$ of the queries are directed to node $n_6$ and $1/7^{th}$ of the queries are directed to node $n_7$. The distribution of the queries for the other partitions are shown in brackets in Table 6.2. The load on each node is balanced and is $8/7$ times the load on the node when all nodes were online.

While Table 6.2 shows what happens when one node is offline, CD can also tolerate additional nodes going offline. In fact, it can allow all failures in which two consecutive nodes are not both offline. One can easily see why this is the case in Table 6.1 and 6.2. If adjacent nodes fail, an entire partition will be lost since CD places backup partitions in adjacent nodes, which leads to the definition below.

**Definition 6.4.1.** *A node in the Chained Declustering scheme is **essential** if removing it makes the data stored in the Chained Declustering scheme unavailable.*

In fact, CD can allow up to $N/2$ alternating nodes to go offline, where $N$ is the number of nodes in the system. These $N/2$ offline nodes result in the uniform doubling of load across the remaining nodes in the system (provided $N \mod 2 = 0$). We exploit this property of CD to develop various energy management schemes.

## 6.5 Exploiting Replication for Energy Management

We can now design schemes to exploit CD to manage the energy consumption of a cluster system when the overall system utilization is less than the peak utilization (i.e., 100% utilization, using the termi-

**Algorithm 1** Dissolving Chain

**INPUT:** $s(m)$, $m'(m' \leq N/2$ for an N node system)
$Q \Leftarrow$ Seg$(s(m))$ //Extract segments in the current state.
$Q \Leftarrow$ Sort$(Q)$ //Sort in descending order of segment lengths.
$curr \Leftarrow$ NumOfflineNodes$(s(m))$
**while** $curr \neq m'$ **do**
  $seg = Q.pop$
  **if** $\mid seg \mid > 2$ AND $1 < \lceil(\mid seg \mid)/2\rceil < \mid seg \mid$ **then**
    $Q.push(seg_{1...\lceil(|seg|)/2\rceil-1})$
    $Q.push(seg_{\lceil(|seg|)/2\rceil+1...|seg|})$
    $seg_{\lceil(|seg|)/2\rceil}$.**turnOff** //turn off this node
    $curr \Leftarrow curr + 1$
  **end if**
**end while**

nology described in Section 6.3). Recall that from the discussion in Section 6.3 we want to control load imbalances such that we obey the constraint of the utilization parameter $M$.

While CD can tolerate a variety of configurations with nodes/servers being offline, as we show below, some of these configurations lead to system load imbalances. The protocol that is used to take nodes offline directly determines the uniformity and balance of the load on the remaining online nodes.

For the discussion below, we introduce a few additional terms: a *ring* refers to the logical ordered arrangement of all the nodes in a CD scheme. When a node in a ring goes offline, the ring is *broken* and produces a *segment*. Additional node failures partition segments into other segments. Each segment has two *end nodes*.

Now, consider the following proposition:

**Proposition 6.5.1.** *If the **ring** or a **segment** of a Chained Declustered set of nodes is broken because a node goes offline, then the two new end nodes of the resulting segment(s) are **essential**.*

The proof follows directly from the properties of CD and Definition 6.4.1.

From Proposition 6.5.1 it follows that to take nodes offline any scheme must select additional nodes from the remaining online nodes that are not end points of the remaining segments.

Now we present two protocols for selecting which nodes to take offline. The Dissolving Chain scheme walks along segments of the ring, taking nodes offline at the halfway point of the given segment. In contrast,

**Algorithm 2** Transition Controller

**INPUT:** $N$, $s(m)$, $m'$, $U$, $M$
$L \Leftarrow$ **maxlen**$(T, N, m')$
**if** $m' \leq N/2$ and $U(L+1)/L \leq M$ **then** CALL DissolvingChain($s(m)$, $m'$)

the Blinking Chain scheme spaces offline nodes on the ring evenly to achieve better load balancing.

### 6.5.1 Dissolving Chain (DC)

The Dissolving Chain (DC) protocol sequentially withdraws nodes using Proposition 6.5.1 so that data is always available. A simple generic algorithm to implement this scheme is shown in Algorithm 1. The input parameters to this Algorithm are: the current state ($s(m)$), and the number of offline nodes ($m'$) in the target state. At the end of running Algorithm 1, the system will be in the new state $s(m')$.

From the properties of CD, nodes in a longer segment of a CD ring have lower loads compared to a node in a shorter segment. By choosing to power down the middle node in a segment, we are both minimizing load imbalance as well as the load increase on the remaining nodes, as a result of the transition.

Continuing the example from Table 6.2, the second node that can be taken offline is node $n4$. The resulting system is balanced (uniform node load) as shown in Table 6.3.

Now that we have taken two nodes offline, let us consider taking another node offline to reduce the energy consumption in response to a lowering of the overall system utilization. Notice in Table 6.3 we have two segments of length 3. To take the next node offline, we can pick one of these two segments and cut it into two equal parts. However the load on the system now becomes imbalanced as illustrated in Table 6.4. Essentially, DC can only reach a balanced state when the number of offline nodes is $2^i$ and $2^i$ divides $N$,

| Nodes: | $n_0$ | $n_1$ | $n_2$ | $n_3$ | $n_4$ | $n_5$ | $n_6$ | $n_7$ |
|---|---|---|---|---|---|---|---|---|
| Primary: | — | $R_1(1)$ | $R_2(\frac{2}{3})$ | $R_3(\frac{1}{3})$ | — | $R_5(1)$ | $R_6(\frac{2}{3})$ | $R_7(\frac{1}{3})$ |
| Backup: | — | $r_2(\frac{1}{3})$ | $r_3(\frac{2}{3})$ | $r_4(1)$ | — | $r_6(\frac{1}{3})$ | $r_7(\frac{2}{3})$ | $r_0(1)$ |
| Load: | 0 | $\frac{4}{3}$ | $\frac{4}{3}$ | $\frac{4}{3}$ | 0 | $\frac{4}{3}$ | $\frac{4}{3}$ | $\frac{4}{3}$ |

Table 6.3: Dissolving Chains at $s(2)$

| Nodes: | $n_0$ | $n_1$ | $n_2$ | $n_3$ | $n_4$ | $n_5$ | $n_6$ | $n_7$ |
|---|---|---|---|---|---|---|---|---|
| Primary: | — | $R_1(1)$ | — | $R_3(1)$ | — | $R_5(1)$ | $R_6(\frac{2}{3})$ | $R_7(\frac{1}{3})$ |
| Backup: | — | $r_2(1)$ | — | $r_4(1)$ | — | $r_6(\frac{1}{3})$ | $r_7(\frac{2}{3})$ | $r_0(1)$ |
| Load: | 0 | 2 | 0 | 2 | 0 | $\frac{4}{3}$ | $\frac{4}{3}$ | $\frac{4}{3}$ |

Table 6.4: Dissolving Chains at $s(3)$

where $N$ is the number of nodes in the system.

The DC algorithm can solve the problem defined in Section 6.3.2 if it is implemented within a wrapper controller algorithm. This is because Algorithm 1 is not aware of the the maximum $M$ utilization requirement. The controller algorithm is given in Algorithm 2. This algorithm takes as input: the size of the system $N$, current operating state $s(m)$ desired number of offline nodes $m'$, the current system utilization $U$, and the utilization requirement $M$. Function **maxlen** calculates the maximum segment length that would be produced using $T()$ if this transition occurred. For DC, **maxlen** first puts value $N-1$ into an empty queue. It then pops the top value in the queue ($y$) and pushes $\lceil (y-1)/2 \rceil$ and $\lfloor (y-1)/2 \rfloor$ into the queue. This is done $m'-1$ times and then the maximum value in the queue is returned. If the number of offline nodes is valid and the maximum node utilization constraint ($M$) is not violated, the controller calls the transitioning algorithm.

A companion algorithm is also required to bring nodes online when utilization increases. In a simple implementation, this companion algorithm simply reverses the transitions made by Algorithm 1.

### 6.5.2 Blinking Chain (BC)

The general intuition behind the Blinking Chain (BC) methods is to allow more general cuts than the simple binary cuts used by DC to: a) to reduce the variation in the load across the nodes that are still up, and b) produce states where the load across the nodes is "balanced" (see Definition 6.5.2).

For example, for a system where $N = 40$ nodes, for a DC system at $s(9)$, there will be segments of length $4, 2, 1$ with 28 nodes at $(5/4)$ load, 2 nodes at $(3/2)$ load, and 1 node at double load. A better way to

**Algorithm 3** Blinking Chain

**INPUT:** $N$,$m'$ ($m' \leq N/2$),$root$
$curr \Leftarrow root$; $s \Leftarrow 0$
**if** $m' > 0$ **then** $curr$.**turnOff**
$curr \Leftarrow root.next$; $tgtlen \Leftarrow \lceil (N - m')/m' \rceil$; $ctr \Leftarrow 0$
**while** $curr \neq root$ **do**
  **if** $ctr \neq tgtlen$ **and** $curr$.**isOff then** $curr$.**turnOn**
  **if** $ctr = tgtlen$ **and** $curr$.**isOn then** $curr$.**turnOff**
  $ctr \Leftarrow (ctr + 1)\%(tgtlen + 1)$
  $curr \Leftarrow curr.next$; $s \Leftarrow s + 1$
  **if** $s = N \bmod m'$ **then** $tgtlen \Leftarrow \lfloor (N - m')/m' \rfloor$
**end while**

cut the $N = 40$ ring results in 4 segments of length 4 and 5 segments of length 3. This results in minimal load variation across the remaining online nodes (the benefits of this are shown in Section 6.6.4). We now discuss how to create these segments.

**Segments and Transitions**

The general algorithm for transitions from a balanced state with $m$ nodes offline to a target state with $m'$ nodes offline is shown in Algorithm 3. The goal of this algorithm is to power down nodes such that the remaining online segments have lengths as uniform as possible. In this algorithm, one node in the ring is always deemed the *root* node. The algorithm iterates through every node in the ring starting from the root and changes the state of the node if appropriate. Notice that this algorithm can be used to both transition down (i.e., $m < m'$) or transition up (i.e., $m > m'$). After running this algorithm, the system moves to the state $s(m')$.

In Algorithm 3, when $m'|N$ (| is the "divides" operator), all the resulting segments will be of equal length. However if $N$ is not divisible by $m'$, ($N$ mod $m'$) segments will have lengths $\lceil tgtlen \rceil$ and the remaining $m' - (N \bmod m')$ segments of length $\lfloor tgtlen \rfloor$.

BC can also adhere to the maximum node utilization constraint $M$ defined in Section 6.3.2 by using a controller algorithm similar to Algorithm 2 except that a) we call Algorithm 3 (Blinking Chain) instead of Dissolving Chain in the last line and b) **maxlen** calculates $\lceil tgtlen \rceil$ as described above.

Now, consider transitioning from a state with $m$ nodes offline to $m'$ nodes offline. A method to implement this transition is to bring all but the root node back online and then turn $m' - 1$ of them off, but

this results in a high transitioning cost as each transition requires making $m + m' - 2$ node state changes (i.e., changing the state of a node from offline to online, or vice versa). These state changes can consume a significant amount of energy (see Section 6.6.5), and we would also *like to minimize the energy spent in making these transitions*. An interesting property of BC is that when transitioning from state $s(m)$ to $s(m')$, there may be offline nodes in the $s(m)$ configuration that can remain offline in the $s(m')$ configuration. By not changing the status of these nodes, the transitions can be made more energy efficient, as discussed next.

**Optimizing the Transitions**

First consider finding states that provide the most "efficient" transitions, which implies making the least number of node state changes in the transition. In BC, the most efficient transition between two states $s(m)$ and $s(m')$ is such that only $|m - m'|$ nodes undergo transition. This efficient transition is defined formally as:

**Definition 6.5.1.** *The **Optimal Blinking Chain Transition** $s(m)$ to $s(m')$ only requires $|m - m'|$ nodes to undergo transition.*

This optimal transition can be implemented by using Algorithm 3. We now give Proposition 6.5.2 which highlights a key relationship between divisible states ($s(m)$, $s(m')$ such that $m|m'$ or $m'|m$) and the Optimal Blinking Chain Transition.

**Proposition 6.5.2.** *$s(m)$ to $s(m')$ is an optimal Blinking Chain transition iff $(m|m'$ OR $m'|m)$*

*Proof.* First, if $s(m)$ to $s(m')$ is an optimal transition, then only $|m - m'|$ nodes have changed state. Given any $s(m)$, we know there are $(N \bmod m)$ segments of length $\lceil (N - m)/m \rceil$ and $[m - (N \bmod m)]$ of length $\lfloor (N - m)/m \rfloor$. Also, we know that there are $m$ total segments in $s(m)$. Without loss of generality, assume $m' > m$, which means exactly $x = (m' - m)$ nodes have powered down. We can prove $m|x$ by contradiction. Assume that $x$ is not a multiple of $m$, then segments in $s(m)$ are not all cut the same number of times. This means that the maximum difference between two segments in $s(m')$ cannot be 1, which is a contradiction of the above. Next, we need to prove that given $m|m'$, then the segments in $s(m)$ can be cut into the segments in $s(m')$ with $m' - m$ offline nodes. Since $m|m'$, then each segment

in $s(m)$ will receive $c = (m' - m)/m$ cuts. So the lengths of the segments in $s(m')$ will be $(\lceil (N - m)/m \rceil - c)/(1 + c) = \lceil (N - m')/m' \rceil$ and $(\lfloor (N - m)/m \rfloor - c)/(1 + c) = \lfloor (N - m')/m' \rfloor$. Lastly, $x\lceil (N - m')/m' \rceil + y\lfloor (N - m')/m' \rfloor = N - m'$ and certainly $(x = N \bmod m')$ and $(y = m' - (N \bmod m'))$ hold. □

Proposition 6.5.2 tells us that in a given operating state, $s(m)$, for $N$ CD nodes, we can transition to another $s(m')$ with maximum efficiency if and only if $m'$ is a multiple or factor of $m$. While the Optimal Blinking Chain Transition has interesting properties, it does not handle all possible state transitions. Specifically, it does not cover transitions between any states $s(m)$ and $s(m')$ when $m$ and $m'$ do not divide each other. For example, if $N = 42$, we cannot execute $s(6)$ to $s(15)$, since the optimal transition is not defined in this case.

To handle transitions between any two arbitrary states, we need a **General Blinking Chain Transition**. This transition is implemented as a composition of two Optimal Blinking Chain Transitions: $s(m)$ to $s(GCD(m, m'))$ to $s(m')$, which maximizes the number of offline nodes that are untouched during the transition.

Using our previous example, if $N = 42$ and we wish to transition from $s(6)$ to $s(15)$, then using the General Blinking Chain Transition, we can save $4$ node transitions by doing two optimal transitions: one from $s(6)$ to $s(3)$ and the second from $s(3)$ to $s(15)$. Finally, we note that since the Optimal Blinking Chain Transition can be implemented with Algorithm 3, the General Blinking Chain Transition can simply be implemented using two iterations of Algorithm 3.

Notice that BC transitions are "optimal", when only $|m - m'|$ nodes transition. Recall this is *always* the case for DC transitions. The implication of this property is discussed in Section 6.6.5.

**Number of Balanced States**

Let us now consider the special states $s(m)$ where $m|N$. In these states, all the nodes have identical loads and we deem this "balanced" as defined in Definition 6.5.2.

**Definition 6.5.2.** *If all segments of a Chained Declustered ring are of equal length ($m|N$) in a given operating state $s(m)$, then we deem this a **balanced** operating state, $\bar{s}(m)$ and all nodes have the same load.*

We can calculate the total number of possible balanced operating states for a Chained Declustered system of $N$ nodes as follows: consider the system configured with $N = p_1^{N_1} p_2^{N_2} ... p_j^{N_j}$ where $p_i$ is the $i^{th}$ prime; by simple combinatorics, the total number of unique factors of $N$ is $\Pi_{1 \leq i \leq j}(N_i + 1)$, which is also the number of balanced states for this system since $\bar{s}(0)$ replaces factor $N$.

### 6.5.3 Updates

Updates while operating in energy saving modes can be handled (without sacrificing the fault-tolerance properties of replication) as follows: if an update needs to be applied to a partition replicate that is offline, then the "next left node" can store the updates applied to a partition that has a replicate powered down.

For example, consider the node segment "A–B–C–D" in a CD ring with C has been powered down. In this case, node B can store the updates that have been applied to node D, and node A can store the updates that have been applied to node B. Updates for node A go to node B as usual. When node C comes back online, the update logs stored on A and B will be applied to the partitions on node C. Note that the original fault tolerance property of replication is maintained for CD even when we are operating in energy efficient modes, as the system always keeps two copies of each update.

Section 6.6.6 presents results on the cost of log replay with respect to the amount of data updated and node power up costs.

## 6.6 Evaluation

In this section we present results evaluating the effectiveness of our energy management methods. At a high level our methodology was the following: we took an actual server and ran two prototypical workloads on the server. We then took actual measurements for both energy and response time on this server, as we varied the load on the server (i.e., changed the server utilization). We then produced a model for a single node in a system. This model was then plugged into a larger model for the entire distributed system. Using this method, we were able to explore a range of system configurations.

In all results presented below, we consider a system with 1000 nodes (i.e., $N = 1000$).

### 6.6.1 Experimental Setup

Our system under test (SUT) consisted of an ASUS P5Q3 Deluxe WIFI-AP motherboard with an Intel Core2Duo E8500, 2GB Kingston DDR3 memory, an ASUS GeForce 8400GS 256M graphics card, and a Western Digital Caviar SE16 320G SATA disk. The power supply unit was a Corsair VX450W PSU. System energy draw was measured using a Yokogawa WT210 unit as suggested by the SPEC power benchmarks [159]. The WT210 measurements were collected by a separate system through the RS232 interface and the provided Yokogawa software.

We used both a DBMS index query workload and a table scan workload (described below). The DBMS workload was run on a commercial DBMS. Our database consisted of the Wisconsin Benchmark (WB) tables [54]. Client applications accessing the database were written in Java 1.6 using the JDBC connection drivers for the commercial DBMS.

All empirical results were the average of the middle three results of five runs. The offline mode used was the hibernation (ACPI S4) state. Alternative offline modes are discussed in Section 6.6.5.

### 6.6.2 Workload

We model two different types of workloads. The first workload uses WB Query 3. This query is a 1% selection query using a clustered index. The target table for this query is a table with $20M$ tuples (approx. 4GB table size). The actual workload consists of 1000 such queries with randomly selected ranges. This workload is used to model simple lookup queries. Our second workload is a file scan on a WB table (of varying sizes) that has no indices. This workload mimics queries that require scanning tables in a DSS environment. These workloads are described in more detail below.

**Index Queries Workload**

To simulate varying node underutilization with the indexed range query, we defined various workloads for the indexed query by varying idle times (this is the same setup as described in Section 6.3.1). First, we ran this query and measured the query runtime. Lets call this X seconds. Then, we defined a 20% utilization workload as one in which the query runs for X seconds followed by an idle time of 4X seconds. In this

setup, the server is presented with a series of these 5X time windows. An actual run consists of 1000 such windows, with random arrival time for the query in each window. We average the results over each run. Workloads with higher utilization are generated by injecting additional queries in this 5X window. For example a workload with 40% utilization has two queries in each 5X window, and a workload with 100% utilization has 5 queries in each 5X window.

To determine the value of X above, we ran 10000 random 1% selection queries and measured the average response time at 102.5 ms, with a standard deviation of 0.46 ms.

**Database Scan Workload**

We modeled utilization of the system running scan workloads slightly differently to mimic a scenario in which a single scan runs across all the nodes in the system. In this case, when nodes are taken offline, the remaining online nodes have to scan larger portions of the data. In this model, let the time it takes a node to scan a $20M$ tuple WB table be $56.49$ seconds. This node is operating at $100\%$ utilization, scanning as much as possible. For $75\%$ utilization, we ask the node to scan a $15M$ tuple table every $56.49s$. Thus, over time, it is doing $75\%$ of the work that it would do in the $100\%$ case. Similarly, for $50\%$ utilization, we ask it to scan a $10M$ tuple WB table every $56.49s$. Energy consumption is measured for the entire $56.49s$ window. With increased utilization, the increase in response time increases (nearly) linearly. All scans are "cold" and there is no caching between successive scans.

### 6.6.3 Modeling Energy and Response Time

In this section we present the measured energy consumption and response time results for each workload. We then use these results to develop a model for the behavior of a node in the system. All models were picked by trying a number of different linear and polynomial regression models, and picking the one with the lowest coefficient of determination, $R^2$. All presented models had $R^2 > 0.94$.

**Indexed Query Workload**

The response time and energy measurement results for the index workload are presented in Figure 6.1 (in Section 6.3.1). Figure 6.2 plots this data with utilization on the x-axis, system energy consumption (for a 5X window) on the primary y-axis, and the query response time in milliseconds on the secondary y-axis.

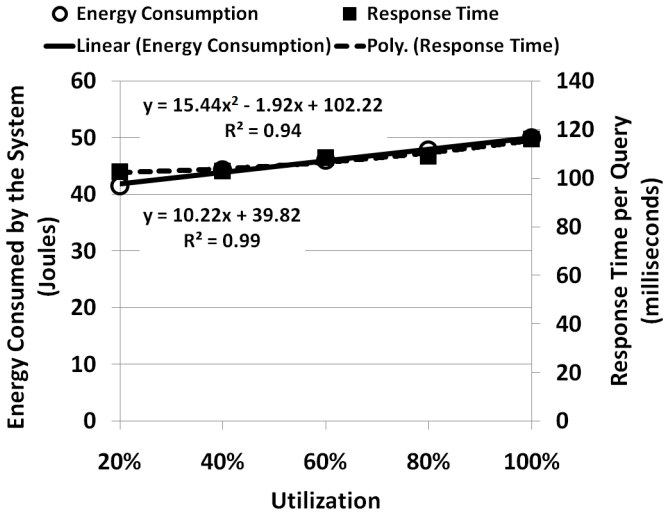



Figure 6.2: Index query regression model

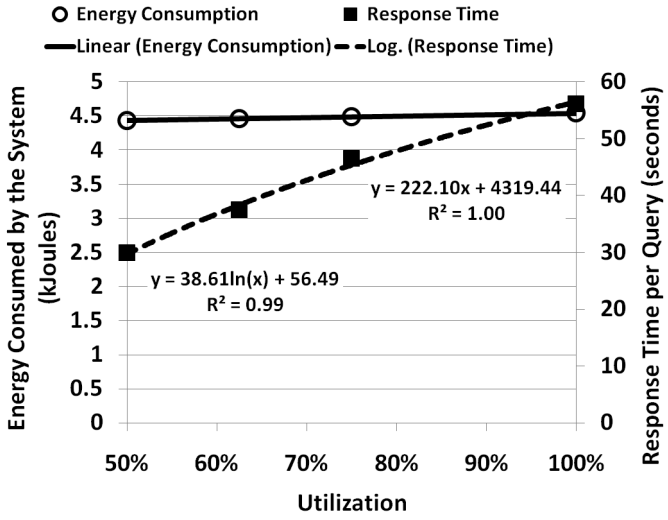
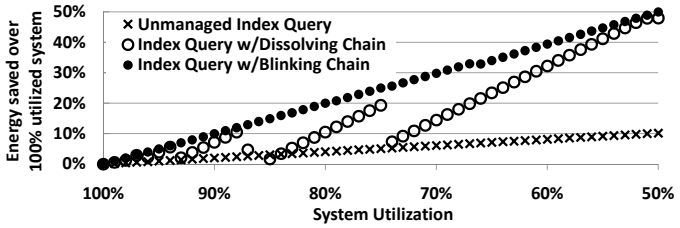



Figure 6.3: Database scan regression model

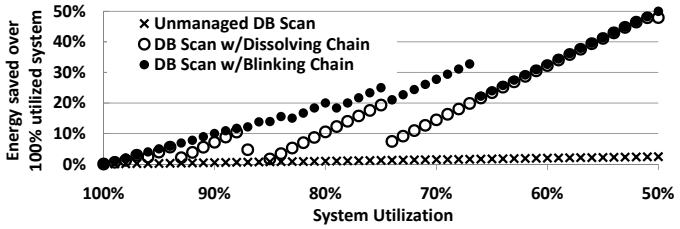
Figure 6.2 also show the derived regression models for the average energy consumed by our SUT and the average query response time as a function of utilization. The energy consumption model is linear while the response time model is quadratic.

**Database Scan Workload**

For the scan workload, increased utilization corresponds to increasing the length of time that an instance runs (to mimic what would happen if we turned nodes offline for such workloads). The results for




(a) Index Queries, N=1000




(b) Scan Queries, N=1000

Figure 6.4: Energy savings under varying system utilization.

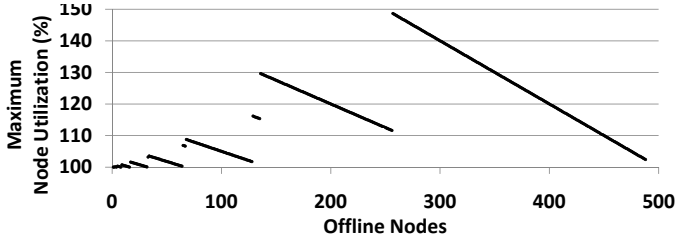
this workload are presented in Figure 6.3. Again, the energy model is linear, but for scan the response time model is logarithmic. The average response time curve is sublinear as the pre-fetching used by the DBMS decreases the per-record response time as we increase the amount of data that is read. While the energy consumption curves in Figures 6.2 and 6.3 are both linear, as utilization increases, energy consumption grows faster with the CPU-bound index workload.

### 6.6.4 Effect of Decreasing Utilization

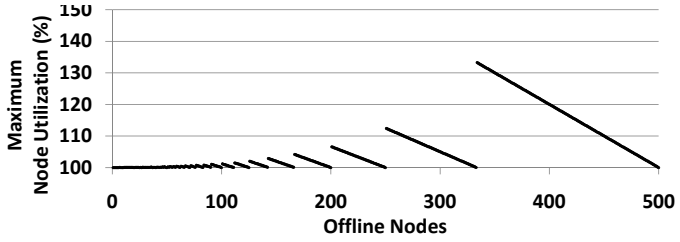
Using the models described in the previous section, we now apply the workload models to a $N = 1000$ system configuration under varied system utilization.

We then analyze the workload energy consumption of the overall system as the overall system utilization decreases from 100%. In addition to comparing differences between our methods, we also compare against the *Unmanaged* system, where all nodes are always online regardless of the overall system utilization.

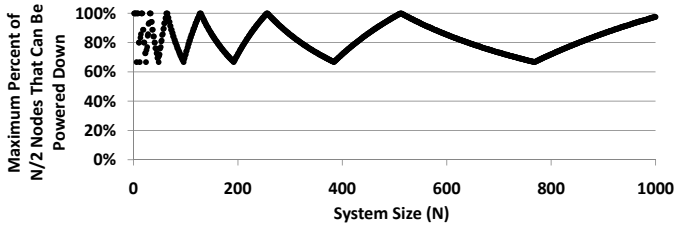
These results are shown in Figures 6.4 (a) and (b). In these figures, we vary the system utilization from 100% to 50% as shown on the x-axis (going from 100% on the left to 50% on the right). So going left to right, corresponds to decreasing the overall system utilization from the fully loaded (100%) system.




(a) Dissolving Chains




(b) Blinking Chains




(c) Dissolving Chain maximum number off offline nodes

Figure 6.5: (a),(b) Maximum node utilization as we iteratively take nodes offline. (c) Ability of Dissolving Chains to power down half of the nodes.

For each point in these figures, we apply our empirically derived models from Section 6.6.3 to calculate the energy consumption. Using this calculated energy consumption, we plotted, on the y-axis, the energy saved by the entire system compared to the energy consumption at the 100% point.

We notice that an unmanaged cluster saves at most $10\%$ in energy consumption (for the Index query workload Figure 6.4 (a)) at $50\%$ utilization. For the Scan workload (Figure 6.4 (b)), the unmanaged cluster only saves 3% of energy at $50\%$ utilization! However, using DC and BC, we can save 48% and 50% of the energy consumption at $50\%$ utilization respectively. Notice, because of DC's inability to power down $500$ nodes for $N = 1000$, its savings is slightly lower than BC.

Another striking observation from Figure 6.4 is that the curves for both BC and DC have big swings/spikes. These spikes can be seen for both methods clearly in Figure 6.4 (b). This behavior is because both methods

introduce load imbalances at certain operating states. Notice that the swings for BC are more gradual compared to DC – this is because BC maintains optimal load balance on the online nodes at any given operating state, which makes its energy swings are more subtle compared to DC.

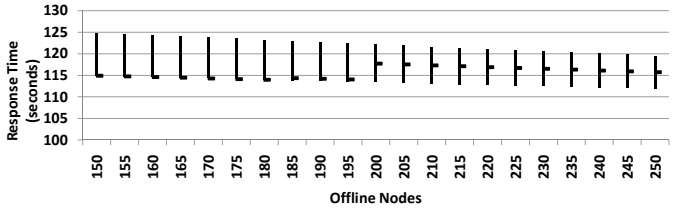
Let us explore these swings in greater detail. Consider Figures 6.5 (a) and (b), where we power down $m$ nodes when the system utilization is $(1000 - m)/1000$ for a $1000$ node system. As the system utilization drops, consider taking nodes offline one by one (incrementing $m$ by 1), up to a maximum of $500$ nodes, using both DC and BC methods. Note that not all states will be balanced.

Figures 6.5 (a) and (b) show the *maximum node utilization* for both methods, i.e., the maximum relative increase (compared to $m = 0$) that any system node will see. Note the maximum node utilization is a crude way to determine the imbalance of the system. (This type of analysis can be used to avoid load spikes seen in Figure 6.1.) Comparing these two figures, we see that BC is more graceful in its worst-case node utilization in imbalanced states (where maximum node utilization is greater than $100\%$) compared to DC.

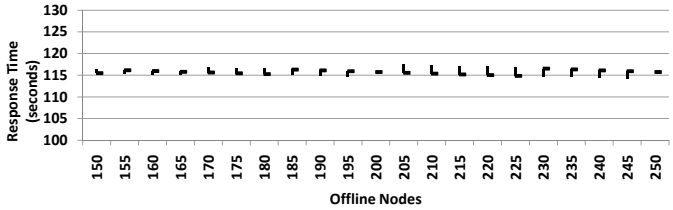
In addition, from Sections 6.5.1 and 6.5.2 we know that BC has $16$ balanced states (see Section 6.5.2) for $N = 1000$ while DC only has $4$. (These correspond to a 100% maximum node utilization in Figures 6.5 (a) and (b).) Furthermore, even when both methods are imbalanced, BC has a better worst-case behavior than DC, as is evidenced by the lower height (node utilization) of the operating points in Figures 6.5 (a) and (b). For example, with respect to our problem statement in Section 6.3.2, if $M = 120\%$, then BC has $67$ states where the maximum node utilization violates this constraint while DC has $209$ states. This is simply a count of all possible operating states with a maximum nodes utilization greater than $M$.

Lastly, we notice from in Figure 6.5 (a) that DC cannot reach $\bar{s}(500)$ for $N = 1000$. This is because as it systematically traverses the ring, cutting segments in half, it may create irreducible segments of length 2. Thus, it cannot reach the optimal number of offline nodes. This effect can be seen in Figure 6.4, where near $50\%$ utilization, DC is slightly lower in energy savings than BC.

An analysis of this phenomenon over varying system sizes ($N$) is shown in Figure 6.5 (c). Here we show how close DC can come to powering down $N/2$ nodes for $1 \leq N \leq 1000$. What we notice is that there are dramatic swings, but more importantly, we notice that DC can transition to $\bar{s}(N/2)$ only when $N = 2^i$.




(a) Dissolving Chains Response Time




(b) Blinking Chains Response Time

Figure 6.6: Comparing imbalanced operating points using the Index Query workload. The vertical lines in represent the range between the minimum and the maximum response times and the horizontal bar is the median response time.

Ultimately, the reason this occurs is because DC heuristically takes nodes down and will never self-correct by bringing them back online as utilization monotonically decreases. The upside to this heuristic is a low (constant) transitioning energy cost that is discussed in Section 6.6.5.

For a detailed look at further effects of BC optimal load balancing to DC heuristic balancing, we zoom in on a smaller set of operating states. We use the models of Figures 6.2 and 6.3 and compare how energy consumption and response time are affected by these imbalanced states. Figure 6.6 examines the imbalanced operating points for the range of $150$ to $250$ offline nodes, in $5$ node increments, while executing the Index query workload (Figure 6.2). (The results for the scan query workloads are similar and omitted here.) Figures 6.6 (a) and (b) compare the variance in node response time between the operating states for DC and BC, respectively. The response time variance is clearly far smaller with BC.

To summarize, BC transitioning results in more balanced node loads than DC. With its lower maximum node utilizations, BC offers greater opportunities to power down nodes and stay within the threshold $M$ in our problem statement (see Section 6.3.2).

"Down" means going from the idle online state ($75.2W$) to an offline state, and "Up" is the reverse. The ASUS offline state is a proprietary idle state provided by the motherboard software.

| | Down time (s) | Down cost (J) | Up time (s) | Up cost (J) | State cost (W) |
|---|---|---|---|---|---|
| ASUS | 0.8 | 83.5 | 0.7 | 68.4 | 72.3 |
| Standby | 12.1 | 1033.2 | 14.3 | 1299.8 | 11.6 |
| Hibernate | 12.2 | 1107.6 | 37.3 | 3531.6 | 0 |
| Shutdown/Off | 8.7 | 700.2 | 177.6 | 9655.9 | 0 |

Table 6.5: Costs for different types of offline states.

### 6.6.5 Effect of Transitioning Costs

So far we have not included any energy or latency costs associated with making transitions from one state to the next. There are a number of possible offline "power states" for a node. For our test system (SUT), Table 6.5 shows the different offline and online transitions, along with the time it takes to make the transition and the energy consumed in making the transition. To put the energy costs into perspective, our SUT has an idle power consumption of $75.2W$. While the "ASUS" state has the fastest transitioning time, it consumes $95\%$ of the idle cost, making it of limited use. The standby mode is more efficient, consuming 11.6W while keeping keeping memory and system state online. The hibernate state has no sustained cost in the offline mode, and provides faster transitions than turning the machine off. Here we use hibernation as our power down mechanism (note that machines can be powered up/down using IPMI which is fairly ubiquitous on modern servers).

From Section 6.6.4, we know that BC is optimal in balancing the load across the nodes, but the cost of this optimality is a complex transitioning mechanism (see Section 6.5.2). In contrast, DC always powers up/down the minimal number of nodes required to reach the target operating state. (As discussed in Section 6.6.4, not all BC operating states are available to DC. To facilitate direct comparison, here we compare operating states that are accessible to both DC and BC). From the perspectives of energy consumed

Table 6.6: Example transitioning sequence, energy costs, and $\tau$

| Util (%) | BC (kJ) | DC (kJ) | $\tau$ (secs) | Util (%) | BC (kJ) | DC (kJ) | $\tau$ (secs) |
|---|---|---|---|---|---|---|---|
| 100 to 90 | 111 | 111 | 0 | 50 to 60 | 1,745 | 353 | DNE |
| 90 to 80 | 111 | 111 | 0 | 60 to 70 | 1,281 | 353 | 168.9 |
| 80 to 70 | 575 | 111 | 84.4 | 70 to 80 | 817 | 353 | 141.6 |
| 70 to 60 | 1,039 | 111 | DNE | 80 to 90 | 353 | 353 | 0 |
| 60 to 50 | 1,503 | 111 | 321.9 | 90 to 100 | 353 | 353 | 0 |

during the actual transitions, DC is clearly more efficient. Now we answer the question: *How much worse is the transition cost of BC?*

Let $O_s(N, y, z, t)$ be the energy cost of running a workload on an $N$ node system with $y$ offline nodes at $z\%$ utilization for $t$ seconds using scheme $s$ (e.g., $O_D$ and $O_B$ for Dissolving and Blinking Chains respectively).

$$\Delta O(N, a, b, z, t) = O_D(N, a, z, t) - O_B(N, b, z, t) \tag{6.1}$$

$$\forall y, z, t : O_B(N, y, z, t) \leq O_D(N, y, z, t) \tag{6.2}$$

Given Equation 6.1 and Equation 6.2 (the load balancing of DC is lower bounded by BC), $\Delta O(N, y, y, z, t) \geq 0$. Given this, if $a \leq b$, the function $\Delta O(N, a, b, z, t)$ grows monotonically as $t$ increases.

$$\Delta O(N, y, y, z, \tau) > \gamma \tag{6.3}$$

Equation 6.3 introduces the notion of $\tau$, which is the length of time that the two systems, one using DC and the other using BC, must operate for for the BC system to overcome its extra (with respect to DC) transitioning cost penalty ($\gamma$) with its efficient load balancing (energy efficiency).

For example, given $N = 1000$ and the Index Query workload, if we want to transition $s(200)$ to $s(300)$, BC pays $\gamma = 464kJ$ in extra transitioning cost over DC. If we have two systems where both DC

and BC making this transition, then the BC system must stay at $s(300)$ for at least $\tau = 36.4sec$ to overcome this penalty, otherwise DC is a more cost (energy) effective solution.

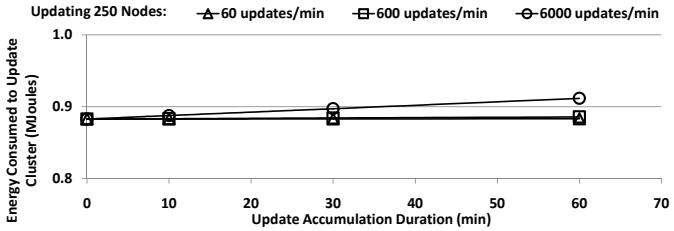
In Table 6.6, we provide two example transitioning sequences for two $1000$ node systems: one using the DC scheme and the other using the BC scheme. We show a transitioning sequence (in columns 1-4) where the utilization starts at $100\%$ and falls by $10\%$ increments until $50\%$. We also show a transitioning sequence (in columns 5-8) where utilization increases from $50 - 100\%$ in $10\%$ increments. In both these scenarios, we assume that there is no time spent for the change in utilization, the decision to transition, and the actual power down sequences. That is, if there is a change in utilization, the decision to transition and the execution of the transition occur instantly.

The rows show the transitioning energy cost and the time it takes ($\tau$) for a more balanced BC derived operating state to overcome its heavy transitioning cost. We notice a number of key points. First, there are a number of transitions where $\tau = 0$ such as transitioning from 100% to 90% (i.e. $s(1000)$ to $s(900)$). In this case, both Blinking and Dissolving Chains perform optimal $|m - m'|$ node transitions and so the node transition costs are the same. Second, if both Blinking and Dissolving Chains result in states with identical load imbalance, then Blinking Chains can never overcome its disadvantage in transition cost and $\tau$ does not exist (DNE). This is seen in transitions $50\%$ to $60\%$ utilization and $60\%$ to $70\%$.

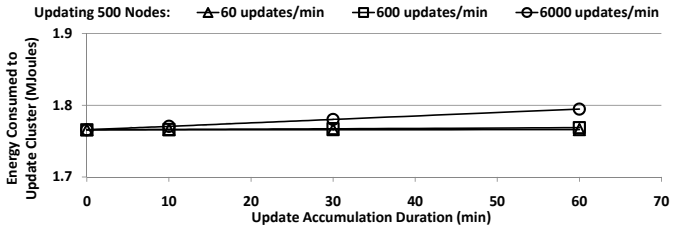
### 6.6.6 Update Costs

The idea of transitioning costs is extended when we consider the costs of updates. Recall from Section 6.5.3 that when both replicas are online the updates are applied to both replicas, but if one of the replica is offline, then the update is applied to the online copy and a log of the update is stored in the "left" node. This log is then applied when the node with the (stale) replica is powered up (in a single update transaction). In this section, we provide an analysis of the energy cost of powering up a cluster and applying the update logs that have accumulated over varying amounts of time.

Consider a 1000 node cluster powered down to 75% and 50%. Let us assume 3 different update rates of 1, 10, 100 updates per second and different periods of time 10, 30, and 60 minutes during which the updates are accumulating. So the powered down nodes have been down for this time and updates are




(a) Powering up 250 nodes.




(b) Powering up 500 nodes.

Figure 6.7: Energy cost of powering up nodes and updating the data partitions, given different rates of incoming updates and duration for which the nodes were powered down.

accumulating, while a second log of the updates is stored on the next left node for fault tolerance (see Section 6.5.3). Further, let us assume that updates are uniformly distributed across the file and thus the nodes in the cluster. This means that the total number of updates that must be replayed is amortized over the number of nodes that are brought up. (We have run different variants of this setup by varying the # nodes, start and end states, update rates, and down time, and the results are similar to the ones presented here.)

Using our 20 million tuple Wisconsin Benchmark table with a non-clustered index, our SUT can update individual tuples at 933 updates per second at a cost of 0.08 Joules/update. In Figures 6.7(a) and (b), we show the energy cost of powering up 250 nodes and 500 nodes with the varying update rates and accumulation lengths. The cost in energy is primarily dominated by the energy spent in bringing nodes out of hibernation. We notice that in both cases, updates cause at most a 2% increase in the transitioning energy.

Finally, the time spent bringing nodes online is also largely dominated by the time spent powering up from hibernation. In the case of Figure 6.7(a) where 6000 queries per minute accumulates for 60 minutes, the entire update process takes 38.79 seconds, of which only 1.5 seconds is the actual time to run the update transaction while the remaining 37.29 seconds is used to bring the node out of hibernation (Table 6.5).

| Methods | Properties: Load Balancing | Properties: Transitioning Overhead |
|---|---|---|
| Blinking Chains | Good | High |
| Dissolving Chains | Fair | Low |
| Mirroring | Poor | Low |

Figure 6.8: Comparison of energy management methods

### 6.6.7 Discussion

Now we discuss some of the practical implications of our work. In a setting where load balance is not as important, as we discussed in Section 6.4.1, simple mirroring can be used. The power down scheme is simple (turn off one of the two replicate nodes, causing a 2X load increase on the remaining node) and it affords the $100\%$ and $50\%$ online balanced states. However, in cases where the huge 2X load imbalances must be avoided (in most cases involving SLAs), we suggest the Dissolving Chain (DC) and the Blinking Chain (BC) methods.

The differences between DC and BC are summarized in Figure 6.8. If avoiding load imbalances and the variation in loads across the nodes is important, then BC offers excellent load balancing in energy saving states. However, BC requires significant state transitioning overhead that would be amplified when system utilization is highly variable. Thus, if one knows the system utilization will be highly variable, DC offers low transitioning cost but incurs slight but predictable load imbalances and offers fewer state transitions.

Finally, notice that since both schemes leverage Chained Declustering, the usage of one over the other is not exclusive; if utilization fluctuates, we can switch to DC, and if there is little fluctuation, we can switch to BC.

## 6.7 Summary

In this chapter we have presented energy management methods that can be used in distributed data processing environments to reduce energy consumption. We leverage the properties of replication schemes and design techniques that can take nodes offline to conserve energy when the system utilization is low. Our

results show that by simply choosing an appropriate replication scheme and power down strategy, significant energy savings ($35\%$ or more in some cases) can be gained over unmanaged systems without extra hardware or data migration. Further, our methods trade off load balancing against energy efficient state transitioning, allowing the user to choose a suitable strategy.

# Chapter 7

## Energy-aware Database Management Systems

As we have discussed in previous chapters, with the rising energy costs and energy-inefficient server deployments, it is clear that there is a need for DBMSs to consider energy efficiency as a first-class operational goal. In fact, driven by requests from its customers, the Transaction Processing Performance Council (TPC) has moved in this direction, and all TPC benchmarks now have a component for reporting the energy consumed when running the benchmarks [167]. While the first version of this benchmark has resulted in a compromise that makes this energy reporting optional, the organization clearly expects that "*Competitive demands will encourage test sponsors to include energy metrics as soon as possible.*" Consequently, database and hardware vendors that wish to report TPC energy metrics will have a keen interest in minimizing their "power/performance" results. While the previous work discussed in this dissertation as largely focused on distributed parallel data processing, here we will focus on single-server DBMS systems that are likely to be the first to report energy benchmarks.

Trading performance for lower energy consumption can take place because a DBMS server may have opportunities to execute the query slower, if the additional delay is acceptable. For example, this situation may occur if the Service Level Agreement (SLA) permits the additional response time penalty. Since typical SLAs are written to meet peak or near peak demand, SLAs often have some slack in the frequent low utilization periods, which could be exploited by the DBMS.

## 7.1 Motivating Illustrative Experiments

There are opportunities to carefully increase the energy efficiency in a traditional DBMS environment with acceptable performance loss. As we showed in Figure 1.3, Chapter 1, there are a variety of potential hardware power/performance mechanisms we can exploit to trade performance for lower energy consumption. Here we will present an example of how DBMSs will have to re-consider how to optimize database query plans if energy is to become a first-class optimization goal.

In this chapter, we focus on exploiting upcoming memory technology that will expose power/performance mechanisms to software [19, 48, 121, 164]. We know memory draws 18% of a commercial DBMS server's power [140]. If we extrapolate this percentage as DBMS servers migrate to SSD storage technology, this proportion should increase to 28%. Also, as a consequence of trends toward in-memory data storage, this proportion will grow. In such large-memory systems, 'hot' power-down (parking) of physical memory DIMMs may save upwards of 40% of system energy consumption [19, 164]. It has been shown that the power of a 4GB DIMM of memory can be reduced from an average of 5.5W to 0.5W [48].

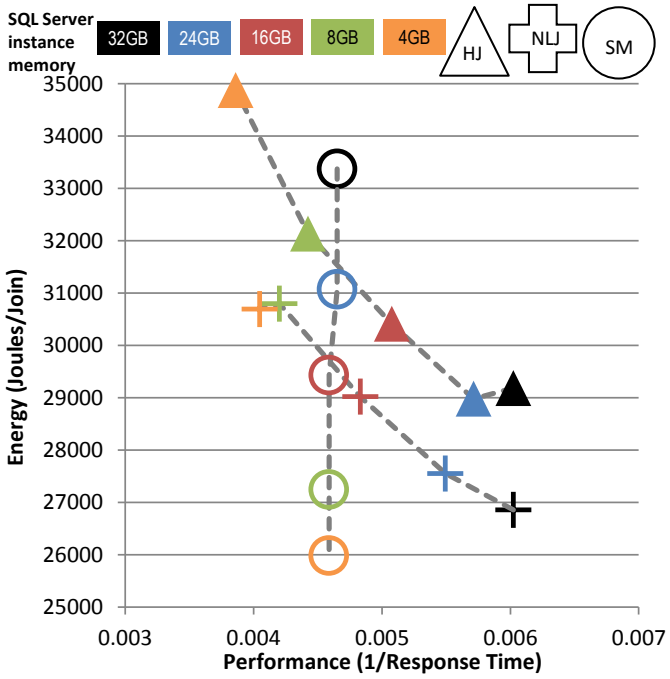



Figure 7.1: Power/performance trade-offs when exploring different join algorithms and memory allocations in SQL Server

Consider Figure 7.1, where we plot the actual energy consumed (for the entire server) and performance of a commercial DBMS server running a TPC-H join. For this result, we ran a join between the PART (8GB) and PARTSUPP (37GB) tables of the TPC-H benchmark at scale factor 300. (Clustered indices are built on the join key PARTKEY.) The server has dual Intel Nehalem L5630 processors (hyper-threading turned off) with 32GB DDR3 memory and 8 Crucial 256GB SSDs for data storage. The commercial DBMS was SQL Server 2008 R2. For this experiment, we reduced the amount of memory available to SQL Server from 32GB to 4GB. When we reduced the amount of memory available to SQL Server, we calculated the amount of energy savings we should expect if we "parked" the unused memory using the results from [48]. In addition to hardware manipulation, we also changed the hash join algorithms used to perform the join – hash join, nested loops join, and sort merge. The results are averaged over three cold runs[1] .

Here there are a couple of interesting things to note from these results. First, both hash join and nested loops join have nearly identical performance when given the full 32GB of system memory, but significantly different energy consumption characteristics. Nexted loops join is consumes 9% less energy than hash join due to lower CPU and memory stress when running the algorithm. Unfortunately, in this case, hash join is chosen by the optimizer. Thus, there is an immediate opportunity to save query energy by considering the algorithms' energy efficiency. Secondly, if we look at the sort merge algorithm's results, they barely change in performance as we reduce the amount of memory given to SQL Server (simulating if we had used memory "parking"). Since the two tables are sorted on the join key using the clustered index, the sort merge algorithm theoretically only needs one page of memory for each table during the merge step. If the user is willing to tolerate a 30% drop in join performance, we are then able to consider the sort merge algorithm and all of its data points shown in Figure 7.1. So, if we were to park all the memory down to 4GB available to SQL Server (orange circle), we could save 13% in system energy consumption by switching from hash join (full memory – black triangle), to 4GB sort merge. This energy reduction was due to *changing the algorithm and exploiting potential hardware power/performance mechanisms*.

With this motivating example, we argue that the traditional DBMS needs to change in two ways

[1] Due to the implementation of SQL Server, full parallel CPU utilization was not observed when using 8 cores to scan 8 clustered index files across 8 SSDs

if we are to make it energy aware: (1) the DBMS needs to have control of modern and future hardware power/performance mechanisms so it can trade performance for lower energy consumption; and (2) the DBMS needs to be able to model query performance for SLA performance limits as well as query energy consumption so that the query optimizer can account for both performance and energy consumption.

## 7.2 Technical Contributions

This chapter proposes a new way of thinking about processing and optimizing database queries. In our framework, we assume that queries have some response time goal, potentially driven by an SLA. The query optimization problem now becomes: *Find and execute the most energy-efficient plan that meets the SLA*. This crucial aspect of our work that distinguishes it from related work [171, 181] is our focus on the slack available in performance between the optimal performance and the SLA goal, and leveraging this slack to reduce the energy consumption (and thereby overall operating cost).

To enable this framework, we propose extending existing query optimizers with an energy consumption model, in addition to the traditional response time model. With these two models, the enhanced query optimizer can now generate what we call an *Energy Response Time Profile (ERP)*. The ERP is a structure that details the energy and response time cost of executing each query plan at every possible power/performance setting under consideration. The ERP of a query can then be used to select the appropriate "energy-enhanced" strategy to execute the query. Figure 7.1 is an example of an ERP.

It should be noted that the framework proposed in this chapter is not limited to single node database environments, as it can be applied to optimizers for parallel DBMSs as well. Such parallel DBMSs also have system settings that include cluster configuration as well as server configuration. In a sense, the framework described here can encompass and drive all of the techniques described in Chapters 4, 5, and 6.

## 7.3 Framework

In this section we present a general query processing framework that uses both energy and response time as the optimization criteria. The questions we tackle are: (1) How to redefine the job of the query optimizer in light of its additional responsibility to optimize for energy consumption? (2) Given this new

role, how do we design a query optimizer? We discuss answers to these questions below.

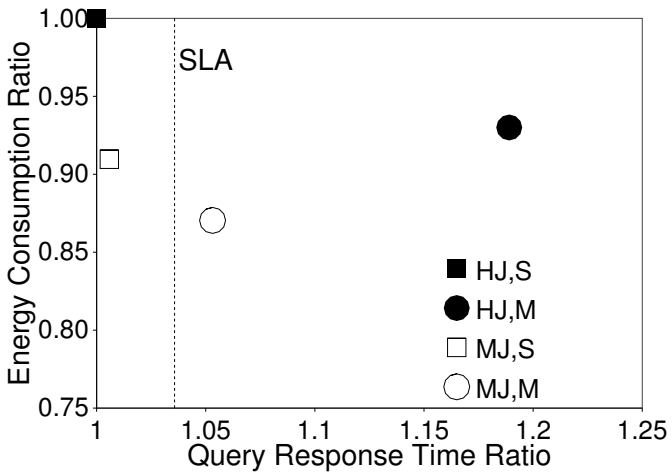



Figure 7.2: Energy Response Time Profile (ERP) for an equijoin query on two 50M tuple (5GB) Wisconsin Benchmark relations, on the attribute $four$, and a 0.01% selection on both relations. The energy and response time values are scaled relative to the stock settings (HJ, S) that is currently used by DBMSs.

### 7.3.1 New Role of the Query Optimizer

The traditional query optimizer is primarily concerned with maximizing the query performance. The optimizer's new goal now is to find query plans that have *acceptable* performance, but consume *as little energy as possible*. As shown in Figure 7.2, we want the query optimizer to return an energy-enhanced query plan that is to the left of the SLA-dictated performance requirement, and as low as possible along the y-axis. (We note that performance SLAs are often not rigid and violating SLAs could be compensated by other mechanisms – e.g., some financial compensation. There are potential business decisions to be made about when violating SLAs are okay, but these considerations are beyond the scope of this study.)

The main task here is to generate ERP plots like that shown in Figure 7.2 (and Figure 7.1). In Figure 7.2 we show an another ERP for a single equijoin query on two Wisconsin Benchmark relations [54]. The plot shows the system energy consumption and response time measurements for executing the query using two different query plans – hash join (HJ) and sort-merge join (MJ) at two different system settings (labeled 'S', 'M'), on SQL Server. System setting 'S' corresponds to the "stock" (high power/performance) system settings. System setting 'M' is the power/performance setting that can save energy by reducing the

memory capacity as we simulated in Figure 7.1. (Details about the experimental setup and systems setting for *this* experiment is presented in more detail in Section 7.5.) The energy measurement is the actual energy that is drawn by the entire server box – i.e., we measure the energy drawn from the wall socket by the entire system. In this figure, we have plotted all the other data points proportionally relative to (HJ, S), for both the energy consumption (on the y-axis), and the response time (on the x-axis). A response time SLA is represented by a dashed vertical line. To generate these plots, the query optimizer needs to *quickly* and *accurately* estimate both the response time and the energy consumption for each query plan. Query optimizers today are fairly good at predicting the response time, but doing so while accounting for varying hardware configuration is a new challenge. Of course, this challenge of predicting the energy consumption of a given query plan for a specific system setting also requires that the query optimizer understands the power/performance settings that the hardware offers.

### 7.3.2 System Settings, Optimization, and ERP

The "energy-enhanced" query optimizer must be provided with a set of system operating settings, where each system operating state is a combination of different individual hardware component operating settings. The optimizer then uses an energy prediction model along with its current response time model to produce an ERP for that query, like the example shown in Figure 7.2.

Given a query $Q$ with a set of possible plans $P = \{P_1, ..., P_n\}$ that is to be executed on a machine with system operating settings $H = \{H_1, ..., H_m\}$, the ERP contains one point for every plan for every system operating setting (and hence an ERP has $n \times m$ points). Note that heuristics could be developed to prune the logical plans $P$ so that only a small subset of the $n$ plans are explored for specific queries – e.g., plans for where the 'stock' setting does not meet the performance requirement may be assumed to not meet the requirement in all other system settings. (An interesting direction for future work is to explore this option.)

Several methods can be envisioned to predict the energy cost of a query plan. One simple method is to assume some constant power drawn by the system for any specific query such that Energy $\alpha$ Response Time. But, we have found in our analysis that such simple models are not very accurate. Note that an

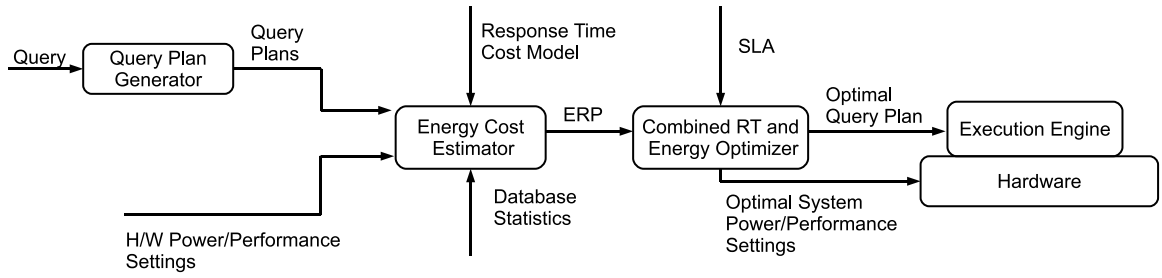



Figure 7.3: An overview of the framework that optimizes for both energy and response time. The energy cost estimator is described in more detail in Figure 7.4.

accurate energy estimation model is essential so that the optimizer does not make a wrong choice – such as choosing query plan *HJ* and system setting *M* in Figure 7.2, which incurs penalties in response time that lie beyond the SLA and does not save enough energy to make this choice attractive.

To enable this new query optimizing framework, we develop an accurate analytical model to estimate the energy consumption cost for evaluating a query plan. We discuss this model in Section 7.4.

### 7.3.3 Our Framework

Figure 7.3 gives an overview of our framework. The query is supplied as input to the query plan generator in SQL Server, which is then requested to list all the promising query plans that the optimizer has identified. These query plans along with the information about available system settings is provided as input to the Energy Cost Estimator, which generates the ERP using an analytical model for predicting the energy consumption (see Section 7.4). The generated ERP is then used by the combined energy-enhanced query optimizer to choose the most energy-efficient plan that meets SLA constraints. Then, a command to switch to the chosen system operating state is sent to the hardware, followed by sending the optimal query plan to the execution engine.

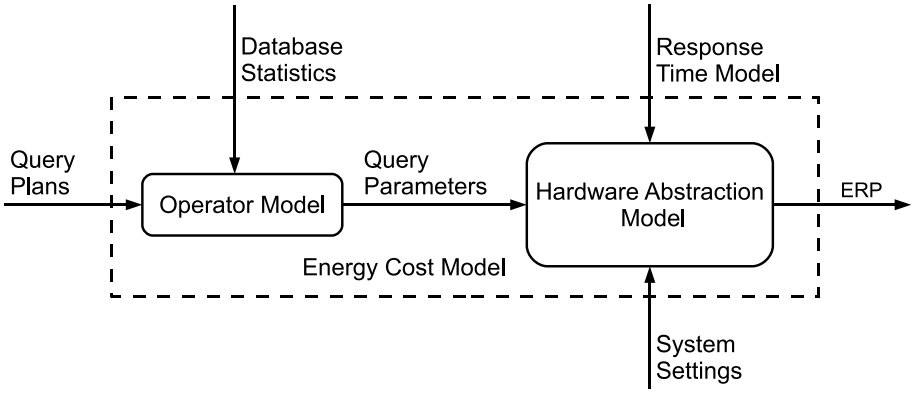



Figure 7.4: An overview of the energy cost model. The operator model estimates the query parameters required by the hardware abstraction model for each query plan from the database statistics available. The hardware abstraction model uses the query parameters estimated and the system parameters learnt to accurately estimate the energy cost.

## 7.4 Energy Cost Model

In this section, we briefly describe an analytical model that estimates the energy cost of executing a query at a particular power/ performance system setting. Our model abstracts away the energy cost of a query in terms of system parameters that can be learnt through a "training" procedure, and query parameters (CPU instructions, memory fetches, etc.) that can be estimated from available database statistics.

**Model Overview**

We want to develop a simple, portable, practical, and accurate method to estimate the energy cost of a query plan. Unfortunately, prior techniques used to estimate energy consumption fail to satisfy one or more of these goals. For example, a circuit-level model of hardware components [1, 78, 177] can accurately predict the energy consumption, but these models also have a high computational overhead which make them impractical for query optimization. On the other hand, a higher level model that treats the entire system as a black box, though simple and portable, is not very accurate.

In our approach, we use an analytical model that offers a balance between these two extremes. In our models, the power drawn by different hardware components are abstracted into learn-able system parameters that are combined with a simple operator model. Figure 7.4 gives an overview of the energy cost model that we have designed.

The operator model takes as input the query plan and uses the database statistics to estimate the query parameters that is required by the hardware abstraction model to estimate the energy cost. The hardware abstraction model that we describe below required estimations of four query parameters from the operator model: the number of CPU instructions, the number of memory accesses, the number of disk read and write requests anticipated during query execution. Our operator model provided estimates for these four query parameters for three basic operations: selection, projection, and joins. The hardware abstraction model then uses these four query parameters and the response time model (since response time is dependent on system settings) to estimate the energy cost of evaluating a query using a particular query plan at a particular power/performance system setting, essentially computing the ERP.

While we have considered various hardware abstraction models, we will only describe the model that we found to be the most accurate. Equation 7.1 shows the model for average system power during a query as defined by three types of variables:

(1) time ($T$)

(2) query parameters for query $Q$: CPU instructions - $I_Q$, disk page reads - $R_Q$, disk page writes - $W_Q$, and memory page accesses - $M_Q$

(3) train-able system parameters: CPU - $C_{cpu}$, disk read - $C_R$, disk write - $C_W$, memory access - $C_{mm}$, remaining system - $C_{other}$). These parameters quantify the component power properties.

The intuition behind this power model is to sum of the average CPU power ($C_{cpu} * \frac{I_Q}{T}$), read power ($C_R * \frac{R_Q}{T}$), write power ($C_W * \frac{W_Q}{T}$), memory power ($C_{mm} * \frac{M_Q}{T}$), and remaining system power ($C_{other}$) during the duration of the query.

$$P_{av} \quad = \quad C_{cpu} * \frac{I_Q}{T} + C_R * \frac{R_Q}{T} + C_W * \frac{W_Q}{T} + C_{mm} * \frac{M_Q}{T} + C_{other} \tag{7.1}$$

Using the work of [118, 155], we can model these query parameters to provide accurate component power draw.

The key features of our energy cost model are:

- **Simplicity:** The models require no additional database statistics other than those used in traditional query optimizers for response time cost estimation. Also, minimal overhead is incurred by the query optimizer in calculating the most energy efficient power/performance operating setting and query plan. The computational complexity is $O(|H| * |P|)$ where $H$ is the set of valid power/performance system settings and $P$ is the set of query plans for the query.
- **Portability:** The model makes few assumptions about the underlying hardware or the database engine internals and can be ported across many DBMSs and machines.
- **Practical:** Detailed system simulators like DRAMSim [177] and Sim-Panalyzer [1] model hardware components at the circuit level to estimate power consumption. This process though accurate is computationally very expensive, and is hence not practical for use in a query optimizer. In our model we abstract the power drawn by different components into learn-able system parameters.
- **Accuracy:** In our tests, our models have an average error rate of around 3% and a peak error rate of 8%.

## 7.5 Evaluation and Discussion

In this section we will present results demonstrating the potential benefit of our optimization framework on a SQL Server using end-to-end measurements.

### 7.5.1 System Under Test

The system that we use in this analysis has the following main components: ASUS P5Q3 Deluxe Wifi-AP motherboard (which has inbuilt mechanisms for component-level power measurements), Intel Core2 Duo E8500, 4x1GB Kingston DDR3 main memory, ASUS GeForce 8400GS 256M, and a 32G Intel X25-E SSD. The power supply unit (PSU) is Corsair VX450W. System power draw was measured by a Yokogawa WT210 unit (as suggested by SPEC and TPC energy benchmarks) connected to a client measuring system. Energy consumption was measured by tapping into the line that supplied power to the entire box, so our measurements and results below are real end-to-end gains. The operating system used was Microsoft Windows Server 2008, and we used SQL Server 2008 R2. Note that we use an SSD-based IO system as many

| Query | Predicate | Size of R, S |
|---|---|---|
| A | `R.unique2 < 0.1*\|\|R\|\| AND R.unique1 = S.unique2` | 1GB, 1GB |
| B | `R.unique2 = S.unique2` | 5GB, 5GB |

Table 7.1: Both queries have the template (SELECT * FROM R, S WHERE <predicate>). These queries are used for the ERP plotted in Figure 7.5. All relations are modified (100 byte tuples) Wisconsin Benchmark relations.

studies have shown that SSD-based configurations are more energy efficient [171] and also provide a better total cost of ownership because of their higher performance-per-$ [5].

### 7.5.2 End-to-End Results: ERP Effectiveness

We now present end-to-end results using the techniques that we have proposed in this chaper. We use the two system settings described in the beginning of this chaper; namely, (1) **S** – the default stock settings of our system, and (2) **M** – a reduced memory setting where the memory is reduced from 4GB to 2GB.

Here we present results with the two queries shown in Table 7.1. Both queries are join queries on two modified Wisconsin Benchmark [54] tables, where the tuple length has been reduced to 100 bytes. Query A is a 10% selectivity join on two small 1GB tables while Query B is a full join on the sorted keys of two 5GB tables.

First, let us examine the scenario when switching to a lower power/performance state has little effect on the response time. With only 2GB of memory, we expect that a query whose peak main memory requirement is less than 2GB will take approximately the same amount of time to execute, and hence will provide significant energy savings. Figure 7.5 (a) shows the ERP of query A in Table 7.1. As we can see, retaining the same hash join plan but using system setting 'M' *reduces the energy consumption by 18% but increases the query response time by only 1%.* In comparison, changing the join algorithm to sort-merge incurs significantly higher response time penalties.

Query B in Table 7.1, is an equijoin on two tables that are clustered on the join attribute 'unique2'. The two 5GB tables are relatively large compared to the amount of available main memory, but since the tables are already sorted on the join attribute the inputs do not need to be sorted before joining using a

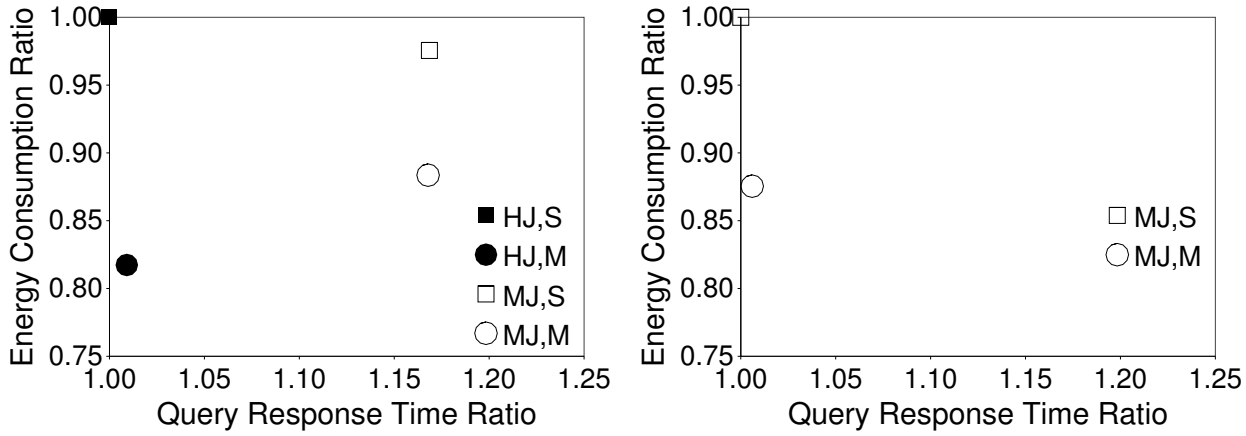



Figure 7.5: ERPs of two equijoin query classes: (a) Low memory requirement (b) Low memory and I/O heavy. Two join algorithms are used: hash join (HJ) and sort-merge join (MJ); along with 'stock' (S) and low memory (M).

merge join operation. Query B is I/O-intensive, requires minimal computation, and has a low peak memory requirement. For this query, as shown in Figure 7.5 (b), using the sort-merge join along with reducing the memory requirement produces a win of 12% energy savings for less than 1% performance penalty. The response times for the hash join plans are far greater than sort-merge for Query A, and are therefore left out of Figure 7.5 (b).

### 7.5.3 Summary

Our preliminary results described above shows that significant (>10% for the queries above) energy savings can be attained by careful optimization of both query plans and system settings in SQL Server based on end-to-end measurements. We now summarize key takeaways messages from our study.

**Energy-Aware Query Optimization:** Current DBMS query optimizers can be made energy aware using our modular optimization framework. By introducing a Hardware Abstraction Model (Figure 7.4) in addition to the traditional Operator Model, we are able to create Energy Response Time Profiles – ERPs (Figure 7.2). The Hardware Abstraction Model is a train-able model for estimating both the query response time *and* the query energy consumption. ERPs allow us to maximize the query's energy efficiency while adhering to performance SLAs.

**SLA-Driven Resource Allocation:** The main concept behind the ERP is that by analyzing the relationship between different query plans and different hardware power/ performance settings, we can trade decreased query performance for increased energy efficiency in the presence of slack in SLAs. In some cases, this trade-off is disproportionate, and in some cases, this trade-off can be highly favorable for energy efficiency. By plotting the SLA performance limit on the ERP, we can easily discover the most energy-efficient combination of query execution plan and hardware system settings while still adhering to the SLA limit. Figures 7.5 (a,b) show preliminary results where only two plans with two system settings can provide very interesting execution options.

**Exploring New Hardware Mechanisms:** Traditionally, CPU has been the first target of studies in server energy management [103]. In these results, we have shown that emerging hardware mechanisms such as memory DIMM parking [19, 164] can provide significant energy savings for DBMSs. Other hardware parts such as the networking components [8,77] are also becoming energy-aware, which will further increase the number of system settings that comprise the ERP in cluster database environments. As was mentioned above, in distributed parallel DBMS settings, other parameters such as cluster size and cluster heterogeneity (Chapters 3 6may also impact the sizd of the ERP space that must be considered.

**Complex Queries and Concurrent Queries:** The preliminary results presented here point to interesting opportunities for energy savings with simple queries, and it would be interesting to extend this study to more complex queries, and query workloads (e.g., concurrent queries).

## 7.6 Summary

This chapter presents a new framework for energy-aware query processing. The framework augments query plans produced by traditional query optimizers with an energy consumption prediction to produce an Energy Response Time Profile (ERP) for a query. These ERPs can then be used by the DBMS in various interesting ways, including finding the most energy-efficient query plan that meets certain performance constraints (dictated by SLAs). To enable the framework, a DBMS needs an energy consumption model for queries, and we have developed a simple, portable, practical, and accurate model for an important subset of database operations and algorithms. We have used our framework to augment a commercial DBMS using

actual energy measurements, and demonstrated that significant energy savings are possible in some cases.

# Chapter 8

# Related Work

This chapter will discuss data-processing related work specific to the work presented in the prior chapters of this dissertation. We will also discuss other trending research for increasing the efficiency of data centers hosting cloud services. Since data centers typically contain the largest clusters and draw the most power, significant research from many fields have begun to tackle this issue.

## 8.1 Data Processing, Data Centers, and Cost

The modern data center can hold tens of thousands of servers and costs hundreds of millions of dollars in investment and continual upkeep [75, 80]. The amortized cost breakdown of a modern data center (40K-50K servers) points to servers, power draw, and power-related infrastructure costs making up between 80-88% of the monthly total cost of ownership (TCO) [75, 80, 81]. Traditionally, operational staffing costs have made up the leading costs in enterprise; however, in the data center environment, the staff to server ratio is 1:100, making human costs small compared to the remaining cost components [75]. Server costs still make up around 45-50% of the amortized monthly TCO. Unfortunately, servers are typically over-provisioned due to a number of reasons like uneven application fit, uncertain demand, risk management, and virtualization inefficiencies [40, 75]. For cloud services such as Database-as-a-Service, managing infrastructure is an active area of research.

## 8.2 Management of Database-as-a-Service Infrastructure

DBMSs have traditionally been engineered for a single-tenant "on-premises" environment. However, emerging trends indicate that DBMS workloads are moving towards the cloud. In recent literature [9, 143], several systems for providing databases in the cloud have been proposed and discussed.

In [25], issues such as performance, scalability, security, availability and maintenance must be re-considered in a multi-tenant cloud environment. The costs of providing these services to customers must be kept low so that the DaaS provider can keep their prices low [64]. With our goal of cost management, in such an environment, maximizing server utilization via tenant consolidation helps us minimize wasted resources [35, 38, 146].

As discussed in Chapter 2, Section 2.3.1, there are several methods to consolidate multiple tenants on a single server [16,17,24,42,47,146,179]. In particular, methods based on the use of Virtual Machines (VMs) have been studied in [6]. However, the performance overhead caused by VMs (paging [87], contention [122], OS redundancy [47]) may be too expensive for the more data-intensive workloads that we consider. Thus, a number of frameworks for building native multi-tenant applications have also been proposed [18, 36, 152].

Being able to accurately and effectively determine the necessary server provisioning for cloud services, we must model the system's performance under a realistic multi-tenant cloud workload. To this end, recent work has focused on formulating and evaluating performance benchmarks in a cloud environment [46, 98, 156]. Complicating factors such as unpredictable load spikes [29], interference between tenants [55, 100] have also been analyzed. Load balancing may require tenant migration [57] or alternatively, reassignment of a tenant's "master" replica. Other work has studied how to benchmark production systems and train performance and resource utilization models without breaking performance SLOs [20, 30]. In [180], the authors present a middleware architecture to optimize CPU and memory utilization in a cluster while allocating resources fairly.

Costs generally increase with the quality of services (guarantees) provided by the provider [40]. SLAs for cloud-based services are usually formulated in terms of uptime/availability guarantees [12] and require extra hardware for failover. Other work in this field has considered allowing tenants to choose between SLAs

that guarantee different levels of consistency [99] and guaranteeing response times in in-memory column databases [151].

## 8.3 Data Center Energy Efficiency

The problem of increasing energy consumption in large-scale data processing environments necessary for public and private clouds, has received considerable attention over since the beginning of this decade. Intense interest can be found especially in the context of data center (DC) construction and operation. The increasing attention is driven, in part, by the minimization of the total cost of ownership (TCO) for DCs [81, 132, 141]. But, additional reasons include the inevitable architectural [58] and operational [31, 96] power limits and constraints of computing and potential legislative requirements to work within energy budgets [27].

It has been observed that data centers have huge (energy) inefficiencies in the electrical distribution and mechanical components. Various best practices have been proposed, and rapidly adopted over the past few years, that dramatically reduce these energy losses. Examples of these methods include reducing the number of power conversions, bringing in higher voltage closer to the rack, using more efficient power supply parts such as high-efficiency and/or rotary UPSs, raising data center temperatures, shorter control of airflow (e.g., avoiding pumping cool air from a cooling source that is far way from the target), using cooling tower rather than A/C, using lower-end equipment, etc. [73, 76, 79, 80, 113, 128, 136]. All these efforts have resulted in dramatic improvements in the energy efficiency of data centers. For example, consider the Power Usage Efficiency (PUE) metric [74, 117], which is simply defined as the ratio of the total power going into a DC over the total power delivered to the actual IT equipment. Just a few years ago, it was estimated that the PUE of 85% of DCs was greater than 3.0 [95]. In other words for every Watt of power that was delivered to an actual computing unit, 2 additional Watts were expended by the electrical and mechanical systems that supported the computing infrastructure.

Other studies have indicated rosier pictures of efficiency with DCs having average PUEs of 2.0 [76, 170]. Further, the EPA has predicted that by 2011, state-of-the-art DCs should have PUE's of 1.4 [4]. This estimate may be overly pessimistic as certain well designed DCs have already reported PUEs of 1.2 [68, 124].

But where are the bottlenecks in DC efficiency? The sources of distribution efficiency losses in DCs is typically the uninterruptible power supply (UPS), 88-94% efficient in the best case [69,166]. Further, long low-voltage (110V/220V)lines to the racks typically can lose another 1-3% of power. Ultimately though, cooling this unending collection of computing equipment rounds out the balance of the inefficiency. Efficient distribution of cool air over long distances between the Computer Room Air Conditioner (CRAC) and the equipment is an open problem [86, 126]. This includes keeping the air cool, preventing warm and cool air mixing, and maximizing the CRAC efficiency itself [2, 15, 56].

Thus, most of the gains in DC energy efficiency (measured by PUE) will come from advances in the IT infrastructure energy efficiency [72, 116]. Some of these advancements, such as shipping container-based DCs, may have already arrived [70, 125] while others may be an increased focus on DC level power management and provisioning [38,40,63] or networking efficiencies [10]. However, PUE only accounts for the efficiency of getting the energy to the equipment, not the efficiency within the equipment. Efficiency within the equipment can be quantified by the server PUE (SPUE). This is defined as the ratio of total server input power to the amount of useful power; that is, the power directly involved in computation. Current computing technology has SPUE ratios of 1.6-1.8 while state-of-the-art SPUE should be under 1.2 [45]. So combining even state-of-the-art PUEs and best case SPUEs would yield in about 70% of energy delivered to a DC to be actually used for computation [22]. Therefore, while we can count on the infrastructure to be increasingly more efficient, part of the burden will fall on the SPUE side of DC energy efficiency. Our efforts looked to bring down the SPUE through power-aware engineering of software systems.

One concept that would ultimately yield in ideal SPUEs would be energy proportional computing. That is, an X% utilized server should consume X% of the power it would when it is at 100% utilization (peak power). One of the hurdles to this is the problem that idle machines typically consume an significant amount (50%) of its peak power [21]. Poor energy proportionality is caused by all the major components of the server. In the past, the CPU has been the biggest culprit, but with CPU manufacturers optimizing CPU utilization and power consumption, this component is now the most energy proportional of the system [22]. However, the remaining components such as memory, disks, and networking have much poorer energy/utilization proportionality. In many ways, this problem must be solved at the component level such

as disk power consumption [37, 150, 161]; however, software systems must also be aware of hardware capabilities or adapt its usage of hardware to also achieve energy proportional computing. One example is the Tickless kernel project which aims to change the way OS kernels operate at idle [157]. The systems community has since begun to develop energy based metrics of efficiency that place power optimization as a first-order goal [148, 160].

Another important factor in computing energy efficiency is the issue of dealing with failures [153, 154]. This problem is largely solved with replication of services on multiple hardware systems. However, replicating services on servers increases the energy consumption of the DC, especially under low utilization periods. Recent studies have looked at powering down systems in low utilization environments to reduce idle energy consumption and increase efficiency [120, 138, 142]. However, these have been studies on clusters of nodes running web based services where computation migration to another replicated server feasible. If we begin to consider data intensive services where nodes are storing and processing large amounts of data (such as a database service), powering down nodes will lead to data unavailability. Leveraging data replication is one way to sidestep this problem (Chapter 6).

## 8.4 "Wimpy" Nodes and Modern Low-Power Hardware

Recent studies on the energy consumption of large clusters [4, 21, 62] have shone a light on existing [23, 126, 142, 144] and future problems faced by data center operators. The database community has begun to seriously consider the energy costs of database management systems [84, 123, 140, 147]. It was shown that over the past decade, published TPC-C results used systems that have increased their power needs six-fold [140], and studies began revealing the true energy consumption costs of running database workloads [123]. Hardware advances, such as low-power non-volatile Phase-Change Memory [130] and traditional solid state/flash memory may be able to decrease the large proportion of a DBMS systems power by eliminating the traditional rotating disk drive in storage subsystems.

While these studies have examined the energy consumption profiles of single nodes running a database, another direction is to treat a cluster as a holistic entity for energy optimization. This cluster-level approach to energy management offers many different directions for research.

Both the systems and database communities have produced a number of studies on reducing the energy consumption of data processing clusters [13, 80, 83, 111, 145, 172, 174] by stepping out of the box and considering novel hardware configurations for cluster nodes. Some of these have focused on rethinking cluster node architecture and using new low-power, low-cost hardware [13, 145, 172, 174]. In work from CMU [13, 172, 174], low-power, "Wimpy" nodes were used for key-value, "get-put" workloads. In complementary work from MSR [145], wimpy nodes were used for clusters that served web workloads. Similarly, both of these studies looked at workloads that were generally partitionable and thus required little cross-node communication.

Custom energy-efficient hardware has also been presented, which targets the needs of data center operators [80, 120, 162]. In [80, 162], new designs are proposed for server hardware that increase the energy efficiency and decrease the amount of idle power drawn. In [120], a new system is proposed that allows a server to dramatically improve its sleep and wake-up transitioning time. This would allow underutilized systems to reduce their base-line power dramatically while minimizing any impacts on performance.

Alternatively, servers can be augmented with custom, low-power circuitry, like Field-Programmable Gate Arrays [91, 127]. In this way, as opposed to heterogeneous clusters where we supplement traditional, high-performance servers with low-power, low-performance nodes (Chapter 4), we move the "wimpy" computing directly into the traditional server itself (or into its components).

## 8.5 Energy Efficiency in Databases

Early database research studies speculated that the database software has an important role to play in improving the energy efficiency, and argued for the redesign of several key components such as the query optimizer, the workload manager, the scheduler and the physical database design [71, 84]. Many of these suggestions assumed that, like cars, computer systems have different optimal performance and energy efficiency points. However, only preliminary experimental data was provided to support these claims.

Subsequent efforts have studied how alternate database designs and configurations can improve the energy efficiency of a DBMS [102, 171, 181].

Meza et al. [123], have studied the energy efficiency of a scaleup enterprise decision support system,

and showed that the most energy-efficient configuration was not always the fastest depending on the degree of declustering across disks.

Xu et al. modified the query optimizer to incorporate the energy costs of query execution plans [181]. Through a real implementation inside PostgreSQL they were able to demonstrate 11% to 22% power savings in TPC workloads. However, their study did not consider the baseline system power costs, thereby potentially over-estimating the improvements in energy efficiency.

Tsirogiannis et al. [171] explored the energy efficiency of a typical scale-out database server by varying numerous hardware, software, and database parameters. They found that because of the high start-up power draw of traditional servers, the most energy efficient configuration was also always the fastest. This study, however, did not consider multi-node database configurations or low-power hardware.

Neither of these two prior works [171, 181] target energy efficiency along with SLA constraints. Our work shows that when considering SLAs, energy optimal and performance optimal are not always the same operating points.

Besides our methods to optimize the efficiency of a DBMS, other local-level techniques have been explored. Other work that directly target DBMS energy efficiency include energy efficient execution engines [88]. While that work increases the energy efficiency of the DBMS, making changes to the execution engine is a complex and platform specific task that is not portable.

The problem of modifying query optimizers to optimize for energy has largely been ignored by the database community. Alonso et al. [11] discuss a simple energy model which can be used to find the most energy efficient query plan to execute the query. However, their methods (studied over a decade ago) do not consider fluctuating power draw from the system and such an assumption incurs high error.

In Chapter 7, we discussed how the DBMS should be able to directly control any exposed power/performance mechanisms provided by modern energy-aware components. There have been a number of related research studies in other communities that have considered the effectiveness and impact of these mechanisms. Most of these efforts focus on achieving energy savings by switching memory modules [19, 48, 51, 60, 164] and/or CPU [61, 115] into lower power states during periods of low workload.

## 8.6 Power-proportional, Energy-efficient Clusters

As mentioned above, one of the biggest reasons that data center clusters are energy inefficient is because the clusters are not power proportional [22]. To alleviate this, one approach is to increase the utilization levels of the cluster nodes by reducing the cluster size. Studies into shutting down online web servers were discussed in [138, 142]. "Dynamic right-sizing of clusters" has also been recently studied in mail-serving cluster environments [114]. Other related methods [137, 139] for powering down storage disks, either rely on learning request skew, specialized hardware, and/or data migration to increase cluster utilization. These two studies also did not consider what load imbalances may occur when disks are power down. Weddle et al. [178] described a RAID-based system to turn off disks to save power when utilization is low. Their system requires pre-setting well-defined "gears", one for each operating point for the system with some disks offline. This scheme can produce up to k-replicas of some data items for k-different operating points. If gears are not setup, then the system requires on-the-fly replication as disks are taken offline, which increases the costs of taking disks offline.

In Chapter 5, we discussed the approach proposed by Leverich et al. [112] in which they reduce the size of the MapReduce cluster to increase its energy efficiency. Chen et al. [39], considered heterogeneous usage of MapReduce servers through a software workload manager. In addition, some of the authors in that work also recently presented a study on MapReduce operating variables that affect energy efficiency [41]. In [66], the authors present a way to schedule MapReduce jobs so that they take advantage of renewable energy sources such as solar power.

Increasing utilization can be done by consolidation using a virtual machine (VM) solution [44, 144, 165,176]. However, using VMs when running data intensive services, is challenging for a number of reasons such as performance penalties from VM overhead [47, 87, 122], homogeneous performance from heterogeneous hardware [129], and costs of VM migration and over-provisioning [57]. Weddle et al. [178] described a RAID-based system to turn off disks to save energy.

As we have shown in Chapter 5, sometimes, reducing the cluster size has an adverse effect on the energy efficiency of running a data processing task. In these cases, the most efficient course of action

is to “race to idle”, or, finish the task as fast as possible and let the system transition to a lower energy state [120]. Other related research has also come to this conclusion in certain scenarios [49]. In that work, they found that in a single-system environment, they were unable to increase energy efficiency without losing performance. This dissertation has shown that if either of these assumptions is taken away, single-system environment and/or maximal performance retention, we have many opportunities to make our systems more energy efficient.

# Chapter 9

# Conclusions and Future Work

This dissertation has explored how the growing energy consumption and hardware costs of data processing systems in cloud clusters can be managed. The work presented has focused on four broad areas defined by two different dimensions: **hardware costs versus energy costs**, and **studying distributed cluster environments versus looking within a single server**. The broad challenge has been to balance potential trade-offs of performance for lower costs in a *controllable, predictable, and significant* manner.

## 9.1 Key Contributions

The thesis of this work is that we can re-architect our hardware and software systems so that we can decrease and control our hardware and energy costs by trading-off excess data processing performance. This dissertation has demonstrated this through the following major contributions.

### 9.1.1 A Framework for Managing Costs in the Cloud with Performance Goals

Chapter 2 presented a hardware-cost optimization framework for provisioning servers and scheduling Database-as-a-Service users in the cloud while adhering to the users' performance objectives. The framework requires a server characterization function that describes the scheduling capacity of the server under different user performance objective combinations. We described an empirical methodology to formulate this server characterizing function. By determining the characterizing functions for a number of different candidate servers, we then perform a non-linear programming optimization to determine the number of servers of each type to provision as well as how to schedule users onto them. *With this optimization frame-*

*work, it is possible to manage the server provisioning costs while maintaining cloud user performance – the key thesis of this dissertation.*

### 9.1.2 The Impact of Scalability on Cost and Energy Efficiency

Chapters 3 and 4 discuss potential pitfalls in using low-power, low-performance server hardware when attempting to reduce hardware costs. The classic problem of non-linear scalability when running complex data analytics tasks cause a diminishing return in performance as we increase the number of nodes in a cluster. This is particularly a problem when using so-called "wimpy" nodes in non-scalable situations as they require larger numbers to store and process data at the same performance levels of smaller clusters made of high-performance servers. The work in Chapter 3 showed that care must be taken in using wimpy clusters because evidence was shown that they may end up being costlier than expected.

However, in Chapter 4, we examined the impact of wimpy nodes by supplementing traditional high-power servers in a distributed parallel environment. By providing commercial system empirical evidence and building/modeling our own custom kernel engine, we found scenarios where wimpy nodes are able to increase the energy efficiency of the entire system when they replace some of the "beefy" nodes. *By strategically assigning suitable data processing tasks to wimpy, low-power nodes, in certain cases, they are able to help trade proportionally less performance for more energy savings.*

### 9.1.3 Cluster Energy Management, Data Replication, and Load Balancing

Chapters 5 and 6 deal with energy-saving approaches that leverage data replication to allow a system to power down some of its cluster nodes without losing data availability. Since clusters in large data processing environments are typically underutilized (and thus energy inefficient), it is intuitive that reducing the number of nodes should increase node utilization and thus increase energy efficiency. In data-intensive environments, powering down nodes may lead to data being unavailable unless the data is replicated. *Here, the take-away is that powering down cluster nodes may not always be the most energy-efficient approach and, even when it is, naïvely exploiting data replication to increase energy efficiency can result in unwanted load imbalances in the cluster.*

In Chapter 5, we study how the MapReduce framework, which leverages a distributed file system that typically replicates its data partitions, may or may not benefit from powering down cluster nodes. Specifically, we find that two important factors impact whether or not we should reduce the number of cluster nodes or simply "race to idle" and use as many nodes as possible. The first is hardware capability, specifically, the speed at which the cluster nodes can transition from offline, sleep modes to an online state. If this transitioning time is fast, relative to the job lengths, then we showed that it is more energy efficient to batch up jobs with the entire cluster offline, and power it up to "race to idle". The second is the complexity of the job being performed. We showed that for higher complexity workloads, reducing the cluster size can super-linearly increase the response time and thus energy savings are diminished.

In Chapter 6, we showed that if we choose a suitable data replication strategy, such as Chained Declustering, then we can formulate ways to power down the cluster nodes in a way that load across the cluster remains fairly balanced. In this chapter, as well as Chapter 5, it is shown that naïve power-down strategies can cause severe load imbalances. We discuss two power-down approaches that trade optimal load balance against simplicity and speed of transitioning between different cluster states (different number of nodes powered down).

### 9.1.4 A Framework for Energy-aware Data Processing

Finally, in Chapter 7, a general optimization framework is presented which augments the traditional performance-based database query optimizer (or distributed parallel query optimizer) so that it is energy-aware. The concept of the Energy-Response Time Profile (ERP) is presented and we argue that it is a necessary abstraction for an energy-aware DBMS (or parallel data processing system). An ERP allows an energy-aware optimizer to trade performance for lower energy consumption by exploiting energy-aware hardware mechanisms, cluster manipulations, or lower-performance/lower-power database operators. Additionally, we argue and show that an energy-aware optimizer needs to understand the concept of a performance SLA (or acceptable level of performance) so that it knows what energy-saving trade-offs are acceptable. *At a broad level, this framework encompasses all of the work presented from Chapters 3-6 as the framework determines when and how we should employ energy-saving measures.*

## 9.2 Future Work

Managing the costs of providing Databases-as-a-Service (DaaS) is of utmost importance as this is key to keeping service prices low, which is a dominating reason for customers to use cloud-based services. The work in Chapter 2 discusses server provisioning in cloud, DaaS environments but we worked under a simplifying assumption of non-replicated customer databases. In this way, future work that expands on the work in Chapter 2 will need to relax this assumption to consider real-world environments that are typically replicated three times or more. As such, the work in Chapters 5 and 6 that deal with data replication and load balancing tie in closely with this direction of future work. Replicating customer databases in the cloud will increase the costs to the DaaS provider; however, data replication across the cluster may also provide an interesting means to provide performance SLAs since the user load can now be served from more than one server source.

Specifically, work will first be needed to better understand the characteristics of deployed DaaS clusters (e.g., SQL Azure) to characterize the operational efficiency of the existing system. A theoretical model of operational efficiency will be needed and used to measure any potential gap between the existing operational scenario and a theoretical ideal. Once we have identified the potential "gap", we can plan to systematically explore methods for hardware provisioning, cluster configuration, workload scheduling, dynamic load balancing, and the interaction of the above with replication schemes. The overall goal is to examine the entire life cycle of a data center deployed for database services. This starts from initial hardware provisioning, day-to-day operational methods, and hardware aging (whereby new hardware is introduced to selectively replace aging hardware) to produce an end-to-end solution that optimizes the cloud deployment cost, while meeting the objectives specified in customer-facing Service Level Agreements.

## Bibliography


[1] The SimpleScalar-Arm Power Modeling Project. http://www.eecs.umich.edu/~panalyzer/.

[2] Thermal Considerations in Cooling Large Scale High Compute Density Datacenters. Technical report, Flometrics. http://www.flomerics.com/flotherm/technical_papers/t299.pdf.

[3] DBC/1012 Database Computer System Manual Release 2.0. Technical report, Teradata, 1985.

[4] Report To Congress on Server and Data Center Energy Efficiency. Technical report, U.S. Environmental Protection Agency, 2007.

[5] A Comparison of SSD, ioDrives, and SAS rotational drives using TPC-H Benchmark. Technical report, HP Corporation, 2010.

[6] A. Aboulnaga, K. Salem, A. A. Soror, U. F. Minhas, P. Kokosielis, and S. Kamath. Deploying Database Appliances in the Cloud. IEEE Data Eng. Bull., 32(1):13–20, 2009.

[7] A. Abouzeid, K. Bajda-Pawlikowski, D. J. Abadi, A. Rasin, and A. Silberschatz. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. In VLDB, pages 922–933, 2009.

[8] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu. Energy proportional datacenter networks. In ISCA, pages 338–347, 2010.

[9] D. Agrawal, A. E. Abbadi, S. Antony, and S. Das. Data Management Challenges in Cloud Computing Infrastructures. In DNIS, pages 1–10, 2010.

[10] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In SIGCOMM, pages 63–74, 2008.

[11] R. Alonso and S. Ganguly. Energy Efficient Query Optimization. Technical report, Matsushita Info Tech Lab, 1992.

[12] Amazon Incorporated. http://aws.amazon.com/ec2-sla/.

[13] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. Fawn: a fast array of wimpy nodes. In SOSP, pages 1–14, 2009.

[14] R. H. Arpaci-Dusseau, E. Anderson, N. Treuhaft, D. E. Culler, H. M. Hellerstein, D. A. Patterson, and K. Yelick. Cluster I/O with River: Making the Fast Case Common. In IOPADS, pages 10–22, 1999.

[15] D. Atwood and J. G. Miner. Reducing Datacenter Costs With An Air Economizer. Technical report, Intel Corporation, 2008.

[16] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger. Multi-Tenant Databases for Software as a Service: Schema-Mapping Techniques. In SIGMOD, pages 1195–1206, 2008.

[17] S. Aulbach, D. Jacobs, A. Kemper, and M. Seibold. A Comparison of Flexible Schemas for Software as a Service. In SIGMOD, pages 881–888, 2009.

[18] S. Aulbach, M. Seibold, D. Jacobs, and A. Kemper. Extensibility and Data Sharing in Evolving Multi-Tenant Databases. In ICDE, pages 99–110, 2011.

[19] R. Ayoub, K. R. Indukuri, and T. S. Rosing. Energy Efficient Proactive Thermal Management in Memory Subsystem. In ISLPED, pages 195–200, 2010.

[20] S. Babu, N. Borisov, S. Duan, H. Herodotou, and V. Thummala. Automated Experiment-Driven Management of Database Systems. In HotOS, pages 19–19, 2009.

[21] L. A. Barroso and U. Hölzle. The Case for Energy-Proportional Computing. IEEE Computer, 40(12):33–37, 2007.

[22] L. A. Barroso and U. Hölzle. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2009.

[23] C. Belady. In the Data Center, Power and Cooling Costs More than the IT Equipment it Supports. Electronics Cooling, 23(1), 2007.

[24] P. Bernstein, I. Cseri, N. Dani, N. Ellis, G. Kakivaya, A. Kalhan, D. Lomet, R. Manne, L. Novik, and T. Talius. Adapting Microsoft SQL Server for Cloud Computing. In ICDE, pages 1255 – 1263, 2011.

[25] C.-P. Bezemer and A. Zaidman. Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare? In IWPSE-EVOL, pages 88–92, 2010.

[26] R. Bianchini and R. Rajamony. Power and Energy Management for Server Systems. IEEE Computer, 37(11):68–74, 2004.

[27] J. Bingaman. Energy Savings and Industrial Competitiveness Act. 2011. Senate Committee on Energy and Natural Resources, 112th Congress Senate Report 112-071.

[28] D. Bitton and J. Gray. Disk Shadowing. In VLDB, pages 331–338, 1988.

[29] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson. Characterizing, Modeling, and Generating Workload Spikes for Stateful Services. In SoCC, pages 241–252, 2010.

[30] P. Bodik, R. Griffith, C. Sutton, A. Fox, M. I. Jordan, and D. A. Patterson. Automatic Exploration of Datacenter Performance Regimes. In ACDC, pages 1–6, 2009.

[31] S. Borkar and A. A. Chien. The Future of Microprocessors. CACM, 54(5):67–77, 2011.

[32] A. Borr. Transaction Monitoring in Encompass: Reliable Distributed Transaction Processing. In VLDB, pages 155–165, 1981.

[33] D. Borthakur. The Hadoop Distributed File System: Architecture and Design. 2007. hadoop.apache.org.

[34] K. G. Brill. Data Center Energy Efficiency and Productivity. Technical report, The Uptime Institute, 2007.

[35] H. Cai, B. Reinwald, N. Wang, and C. J. Guo. SaaS Multi-Tenancy: Framework, Technology, and Case Study. IJCAC, 1(1), 2011.

[36] Y. Cao, C. Chen, F. Guo, D. Jiang, Y. Lin, B. C. Ooi, H. T. Vo, S. Wu, and Q. Xu. ES2: A Cloud Data Storage System for Supporting Both OLTP and OLAP. In ICDE, pages 291–302, 2011.

[37] E. V. Carrera, E. Pinheiro, and R. Bianchini. Conserving Disk Energy in Network Servers. In ICS, pages 86–97, 2003.

[38] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing Energy and Server Resources in Hosting Centers. In SOSP, pages 103–116, 2001.

[39] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz. Energy efficiency for large-scale mapreduce workloads with significant interactive analysis. In EuroSys, pages 45–56, 2012.

[40] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing Server Energy and Operational Costs in Hosting Centers. In SIGMETRICS, pages 303–314, 2005.

[41] Y. Chen, L. Keys, and R. Katz. Towards Energy Efficient Hadoop. In UC Berkeley Technical Report, number UCB/EECS-2009-109, 2009.

[42] F. Chong, G. Carraro, and R. Wolter. Multi-Tenant Data Architecture. 2006. http://msdn.microsoft.com/en-us/library/aa749086.aspx.

[43] B.-G. Chun, G. Iannaccone, G. Iannaconne, R. Katz, G. Lee, and L. Niccolini. An Energy Case for Hybrid Datacenters. In HotPower, pages 76–80, 2009.

[44] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In NSDI, pages 273–286, 2005.

[45] Climate Savers Computing Initiative. Climate Savers Computing Efficiency Specs. http://www.climatesaverscomputing.org/about/tech-specs.

[46] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking Cloud Serving Systems with YCSB. In SoCC, pages 143–154, 2010.

[47] C. Curino, E. P. C. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational Cloud: A Database-as-a-Service for the Cloud. In CIDR, pages 235–240, 2011.

[48] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu. Memory Power Management via Dynamic Voltage/Frequency Scaling. In ICAC, pages 31–40, 2011.

[49] S. Dawson-Haggerty, A. Krioukov, and D. Culler. Power Optimization – a Reality Check. Technical report, Berkeley, 2009.

[50] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In OSDI, pages 137–150, 2004.

[51] V. Delaluz, A. Sivasubramaniam, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Scheduler-based dram energy management. In DAC, pages 697–702, 2002.

[52] S. Delap. Yahoo's Doug Cutting on MapReduce and the Future of Hadoop. 2007. http://www.infoq.com/articles/hadoop-interview.

[53] D. DeWitt and J. Gray. Parallel Database Systems: The Future of High Performance Database Processing. CACM, 35(6):85–98, 1992.

[54] D. J. DeWitt. The Wisconsin Benchmark: Past, Present, and Future. In J. Gray, editor, The Benchmark Handbook for Database and Transaction Systems (2nd Edition). Morgan Kaufmann, 1993.

[55] J. Duggan, U. Cetintemel, O. Papaemmanouil, and E. Upfal. Performance prediction for concurrent database workloads. In SIGMOD, pages 337–348, 2011.

[56] D. Dyer. Current Trends/Challenges in Datacenter Thermal Management - a Facilities Perspective. In ITHERM, 2006.

[57] A. J. Elmore, S. Das, D. Agrawal, and A. E. Abbadi. Zephyr: Live Migration in Shared Nothing Databases for Elastic Cloud Platforms. In SIGMOD, pages 301–312, 2011.

[58] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burgur. Dark Silicon and the End of Multicore Scaling. In ISCA, pages 365–376, 2011.

[59] Facebook. Open Compute Project. 2011. http://opencompute.org.

[60] X. Fan, C. Ellis, and A. Loebeck. Memory controller policies for dram power management. In ISLPED, pages 129–134, 2001.

[61] X. Fan, C. S. Ellis, and A. R. Lebeck. The synergy between power-aware memory systems and processor voltage scaling. In In Workshop on Power-Aware Computing Systems, pages 164–179, 2003.

[62] X. Fan, W.-D. Weber, and L. A. Barroso. Power Provisioning for a Warehouse-sized Computer. In ISCA, pages 13–23, 2007.

[63] M. E. Femal and V. W. Freeh. Boosting Datacenter Performance Through Non-Uniform Power Allocation. In ICAC, pages 250 – 261, 2005.

[64] A. Floratou, J. M. Patel, W. Lang, and A. Halverson. When Free is Not Really Free: What Does It Cost to Run a Database Workload in the Cloud? In TPCTC, 2011.

[65] S. Ghemawat, H. Bobioff, and S.-T. Leung. The Google File System. In SOSP, pages 29–43, 2003.

[66] I. n. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. Greenhadoop: leveraging green energy in data-processing frameworks. In EuroSys, pages 57–70, 2012.

[67] Google Incorporated. Efficiency measurements - Google Datacenters. http://www.google.com/corporate/datacenter/efficiency-measurements.html.

[68] Google Incorporated. Efficient Computing - Step 2: Efficient Datacenters. http://www.google.com/corporate/green/datacenters/step2.html.

[69] Google Incorporated. Efficient Data Center, Part 1. http://www.youtube.com/watch?v=Ho1GEyftpmQ.

[70] Google Incorporated. Efficient Data Center Summit, 2009. http://www.google.com/corporate/green/datacenters/summit.html.

[71] G. Graefe. Database Servers Tailored to Improve Energy Efficiency. In Software Engineering for Tailor-made Data Management, pages 24–28, 2008.

[72] Green Grid. Quantitative Analysis of Power Distribution Configurations for Datacenters. http://www.thegreengrid.org/gg_content.

[73] Green Grid. Seven Strategies to Improve Datacenter Cooling Efficiency. http://www.thegreengrid.org/gg_content.

[74] Green Grid. The Green Grid Datacenter Power Efficiency Metrics: PUE and DCiE. http://www.thegreengrid.org/sitecore/content/Global/Content/white-papers/The-Green-Grid-Data-Center-Power-Efficiency-Metrics-PUE-and-DCiE.aspx.

[75] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The Cost of a Cloud: Research Problems in Data Center Networks. SIGCOMM Computer Communication Review, pages 68–73, 2009.

[76] S. Greenberg, E. Mills, and B. Tschudi. Best Practices for Datacenters: Lessons Learned from Benchmarking 22 Datacenters. 2006. http://eetd.lbl.gov/EA/mills/emills/PUBS/PDF/ACEEE-datacenters.pdf.

[77] C. Gunaratne, K. Christensen, B. Nordman, and S. Suen. Reducing the Energy Consumption of Ethernet with Adaptive Link Rate. IEEE Transactions on Computers, 57(4):448 – 461, 2008.

[78] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. Jane, I. N. Vijaykrishnan, M. Kandemir, T. Li, and L. K. John. Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach. pages 141–150, 2001.

[79] J. Hamilton. Where Does Power Go In DCs and How To Get It Back? 2008. http://mvdirona.com/jrh/TalksAndPapers/JamesRH_DCPowerSavingsFooCamp08.ppt.

[80] J. Hamilton. Cooperative Expendable Micro-slice Servers (CEMS): Low Cost, Low Power Servers for Internet-Scale Services. In CIDR, 2009.

[81] J. Hamilton. Internet Scale Storage. In SIGMOD, 2011. Keynote.

[82] S. Harizopoulos, V. Liang, D. J. Abadi, and S. Madden. Performance tradeoffs in read-optimized databases. In VLDB, pages 487–498, 2006.

[83] S. Harizopoulos and S. Papdimitriou. A Case for Micro-cellstores: Energy-Efficient Data Management on Recycled Smartphones. In DaMoN, pages 50–55, 2011.

[84] S. Harizopoulos, M. A. Shah, J. Meza, and P. Ranganathan. Energy Efficiency: The New Holy Grail of Database Management Systems Research. In CIDR, 2009.

[85] D. Hastorun, M. Jampani, G. Kakulapati, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazons Highly Available Key-Value Store. In SOSP, pages 205–220, 2007.

[86] T. Heath, A. P. Centeno, P. George, L. Ramos, Y. Jaluria, and R. Bianchini. Mercury and Freon: Temperature Emulation and Management for Server Systems. In ASPLOS, pages 106 – 116, 2006.

[87] G. Hoang, C. Bae, J. Lange, L. Zhang, P. Dinda, and R. Joseph. A Case for Alternative Nested Paging Models for Virtualized Systems. Computer Architecture Letters, 9(1):17 – 20, 2010.

[88] H. Hopfner and C. Bunse. Towards an energy aware dbms - energy consumptions of sorting and join algorithms. In Grundlagen von Datenbanken, pages 69–73, 2009.

[89] HP Corporation. `http://h18013.www1.hp.com/products/servers/management/remotemgmt.html`.

[90] H.-I. Hsiao and D. J. DeWitt. Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines. In ICDE, pages 456–465, 1990.

[91] IBM Corporation. `http://www.netezza.com`.

[92] Ingres Corporation. `http://www.ingres.com/vectorwise`.

[93] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed Data-Parallel Programs From Sequential Building Blocks. In EuroSys, pages 59–72, 2007.

[94] E. P. C. Jones, D. J. Abadi, and S. Madden. Low Overhead Concurrency Control for Partitioned Main Memory Databases. In SIGMOD, pages 603–614, 2010.

[95] M. Kalyanakrishnam, Z. Kalbarczyk, and R. Iyer. Failure Data Analysis of a LAN of Windows NT Based Computers. IEEE Symposium on Reliable Distributed Systems, pages 178 – 187, 1999.

[96] J. Kessler. Facebook Joins The Arctic Crowd, 2011. `http://money.cnn.com/2011/10/28/technology/facebook_arctic_data_center/index.htm`.

[97] J. G. Koomey. Growth in Data Center Electricity Use 2005 to 2010, 2011. `http://www.mediafire.com/file/zzqna34282frr2f/koomeydatacenterelectuse2011finalversion.pdf`.

[98] D. Kossmann, T. Kraska, and S. Loesing. An Evaluation of Alternative Architectures for Transaction Processing in the Cloud. In SIGMOD, pages 579–590, 2010.

[99] T. Kraska, M. Hentschel, G. Alonso, and D. Kossmann. Consistency Rationing in the Cloud: Pay only when it matters. In VLDB, pages 253–264, 2009.

[100] T. Kwok and A. Mohindra. Resource Calculations with Constraints, and Placement of Tenants and Instances for Multi-tenant SaaS Applications. In ICSOC, pages 633 – 648, 2008.

[101] W. Lang, S. Harizopoulos, J. M. Patel, M. A. Shah, and D. Tsirogiannis. Towards Energy-Efficient Database Cluster Design. In VLDB, 2012.

[102] W. Lang, R. Kandhan, and J. M. Patel. Rethinking Query Processing for Energy Efficiency: Slowing Down to Win the Race. IEEE Data Eng. Bull., 34(1):12–23, 2011.

[103] W. Lang and J. M. Patel. Towards Eco-friendly Database Management Systems. In CIDR, 2009.

[104] W. Lang and J. M. Patel. Energy Management for MapReduce Clusters. In VLDB, pages 129–139, 2010.

[105] W. Lang and J. M. Patel. Energy-Conscious Data Management Systems: The Need for a Closer Hardware and Software Synergy. In New England Database Summit, 2012.

[106] W. Lang, J. M. Patel, and J. F. Naughton. On Energy Management, Load Balancing and Replication. SIGMOD Record, 38(4):35–42, 2009.

[107] W. Lang, J. M. Patel, and J. F. Naughton. On Energy Management, Load Balancing and Replication. Technical report, Wisconsin-Madison, 2009.

[108] W. Lang, J. M. Patel, and S. Shankar. Wimpy Node Clusters: What About Non-Wimpy Workloads? In DaMoN, pages 47–55, 2010.

[109] W. Lang, S. Shankar, J. M. Patel, and A. Kalhan. Towards Multi-Tenant Performance SLOs. In ICDE, 2012.

[110] W. Lang, S. Shankar, J. M. Patel, and A. Kalhan. Towards Multi-Tenant Performance SLOs. In TKDE (invited, Best Papers of ICDE Issue), 2012.

[111] J. Larkby-Lahet, G. Santhanakrishnan, A. Amer, and P. K. Chrysanthis. Step: Self-tuning energy-safe predictors. In MDM, pages 125–133, 2005.

[112] J. Leverich and C. Kozyrakis. On the Energy (In)efficiency of Hadoop Clusters. In HotPower, pages 61–65, 2009.

[113] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments. ISCA, pages 315–326, 2008.

[114] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska. Dynamic right-sizing for poewr-proportional data centers. In INFOCOM, 2011.

[115] J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with pace. In SIGMETRICS, pages 50–61, 2001.

[116] M Isard. Autopilot: Automatic Datacenter Management. SIGOPS Operating Systems Review, 41(2):60 – 67, 2007.

[117] C. Malone and C. Belady. Metrics to Characterize Datacenter and IT Equipment Energy Use. In Digital Power Forum, 2006.

[118] S. Manegold, P. A. Boncz, and M. L. Kersten. Generic Database Cost Models for Hierarchical Memory Systems. In VLDB, pages 191 – 202, 2002.

[119] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. Big data: The next frontier for innovation, competition, and productivity. McKinsey Global Institute, 2011.

[120] D. Meisner, B. T. Gold, and T. F. Wenisch. PowerNap: Eliminating Server Idle Power. In ASPLOS, pages 205–216, 2009.

[121] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. Power management of online data-intensive services. In ISCA, pages 319–330, 2011.

[122] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel. Diagnosing Performance Overheads in the Xen Virtual Machine Environment. In VEE, pages 13–23, 2005.

[123] J. Meza, M. A. Shah, P. Ranganathan, M. Fitzner, and J. Veazay. Tracking the power in an enterprise decision support system. In ISLPED, pages 261–266, 2009.

[124] Microsoft Corporation. http://blogs.technet.com/msdatacenters/archive/2009/06/29/microsoft-brings-two-more-mega-data-centers-online-in-july.aspx.

[125] Microsoft Corporation. Microsoft's Top 10 Business Practices for Environmentally Sustainable Data Center. http://www.microsoft.com/environment/our_commitment/articles/datacenter_bp.aspx.

[126] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making Scheduling 'cool': Temperature-aware Workload Placement in Datacenters. In USENIX, page 5, 2005.

[127] R. Mueller, J. Teubner, and G. Alonso. Data Processing on FPGAs. In VLDB, pages 910–921, 2009.

[128] D. Nelson, M. Ryan, S. DeVito, K. V. Ramesh, P. Vlasaty, B. Rucker, and B. Nelson. The Role of Modularity in Datacenter Design. http://www.sun.com/storagetek/docs/EED.pdf.

[129] A. Noll, A. Gal, and M. Franz. CellVM: A Homogeneous Virtual Machine Runtime System for a Heterogeneous Single-Chip Multiprocessor. In Workshop on Cell Systems and Applications, 2008.

[130] Numonyx Memory Solutions. www.numonyx.com/Documents/WhitePapers/PCM_Basics_WP.pdf.

[131] Oracle Corporation. http://www.oracle.com/us/products/database/exadata-database-machine/overview/index.html.

[132] C. D. Patel and A. J. Shah. Cost Model for Planning, Development and Operation of a Datacenter. Technical report, HP Labs. http://www.hpl.hp.com/techreports/2005/HPL-2005-107R1.pdf.

[133] D. A. Patterson. Latency Lags Bandwidth. CACM, 47(10):71–75, 2004.

[134] D. A. Patterson, G. Gibson, and R. H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In SIGMOD, pages 109 –116, 1988.

[135] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A Comparison of Approaches to Large-Scale Data Analysis. In SIGMOD, pages 165–178, 2009.

[136] PG&E. High Performance Datacenters. Technical report, Lawrence Berkeley National Labs. http://hightech.lbl.gov/documents/DATA_CENTERS/06_DataCenters-PGE.pdf.

[137] E. Pinheiro and R. Bianchini. Energy Conservation Techniques for Disk Array-Based Servers. In ICS, pages 68 – 78, 2004.

[138] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems. In Workshop on Compilers and Operating Systems for Low Power, 2001.

[139] E. Pinheiro, R. Bianchini, and C. Dubnicki. Exploiting Redundancy to Conserve Energy in Storage Systems. In SIGMETRICS, pages 15 – 26, 2006.

[140] M. Poess and R. O. Nambiar. Energy Cost, The Key Challenge of Today's Data Centers: A Power Consumption Analysis of TPC-C Results. In VLDB, pages 1229–1240, 2008.

[141] M. Poess and R. O. Nambiar. Tuning Servers, Storage and Database for Energy Efficient Data Warehouses. In ICDE, pages 1006–1017, 2010.

[142] K. Rajamani and C. Lefurgy. On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters. In IEEE ISPASS, pages 111–122, 2003.

[143] R. Ramakrishnan, B. Cooper, A. Silberstein, and U. Srivastava. Data Serving in the Cloud. In LADIS, 2009.

[144] P. Ranganathan, P. Leech, D. Irwin, and J. Chase. Ensemble-level Power Management for Dense Blade Servers. In ISCA, pages 66 – 77, 2006.

[145] V. J. Reddi, B. Lee, T. Chilimbi, and K. Vaid. Web Search Using Small Cores: Quantifying the Price of Efficiency. Technical Report MSR-TR-2009-105, Microsoft Research, 2009.

[146] B. Reinwald. Multitenancy. 2010. `http://www.cs.washington.edu/mssi/2010/BertholdReinwald.pdf`.

[147] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis. JouleSort: a balanced energy-efficiency benchmark. In SIGMOD, pages 365–376, 2007.

[148] S. Rivoire, M. A. Shah, P. Ranganathan, C. Kozyrakis, and J. Meza. Models and Metrics to Enable Energy-Efficiency Optimizations. Computer, 40(12):39–48, 2007.

[149] M. Russinovich. Windows 7 and Windows Server 2008 R2 Kernel Changes, 2009.

[150] S. Sankar, S. Gurumurthi, and M. R. Stan. Intra-disk Parallelism: An Idea Whose Time Has Come. In ISCA, pages 303–314, 2008.

[151] J. Schaffner, B. Eckart, D. Jacobs, C. Schwarz, H. Plattner, and A. Zeier. Predicting In-Memory Database Performance for Automating Cluster Management Tasks. In ICDE, pages 1264–1275, 2011.

[152] O. Schiller, B. Schiller, A. Brodt, and B. Mitschang. Native Support of Multi-tenancy in RDBMS for Software as a Service. In EDBT, pages 117–128, 2011.

[153] B. Schroeder and G. A. Gibson. Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You. In USENIX Conference on File and Storage Technologies, 2007.

[154] B. Schroeder, E. Pinheiro, and W. D. Weber. DRAM Errors in the Wild: a Large-Scale Field Study. In SIGMETRICS, pages 193–204, 2009.

[155] L. D. Shapiro. Join processing in database systems with large main memories. ACM Trans. Database Syst., 11(3):239–264, 1986.

[156] P. Shivam, V. Marupadi, J. Chase, T. Subramaniam, and S. Babu. Cutting Corners: Workbench Automation for Server Benchmarking. In ATC USENIX, 2008.

[157] S. Siddha, V. Pallipadi, and A. V. D. Ven. Getting Maximum Mileage Out of Tickless. In Linux Symposium, 2007.

[158] S. Srikantaiah, A. Kansal, and F. Zhao. Energy-Aware Consolidation for Cloud Computing. In HotPower, 2009.

[159] Standard Performance Evaluation Corporation. http://www.spec.org/power_ssj2008/docs/SPECpower_ssj2008-Hardware_Setup_Guide.pdf.

[160] Standard Performance Evaluation Corporation. SPEC Power. http://www.spec.org/power_ssj2008.

[161] Storage Networking Industry Association. SNIA Green Storage Initiative. http://www.snia.org/forums/green.

[162] A. S. Szalay, G. Bell, H. H. Huang, and A. White. Low-Power Amdahl-Balanced Blades for Data Intensive Computing. In HotPower, 2009.

[163] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. Concurrency - Practice and Experience, pages 323 – 356, 2005.

[164] M. E. Tolentino, J. Turner, and K. W. Cameron. Memory MISER: Improving Main Memory Energy Efficiency in Servers. IEEE Transactions on Computers, 58(3):336–350, 2009.

[165] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu. Delivering Energy Proportionality with Non Energy-Proportional Systems - Optimizing the Ensemble. In HotPower, 2008.

[166] M. Ton and B. Fortenbury. High Performance Buildings: Datacenters - Server Power Supplies. Technical report, Lawrence Berkeley National Laboratories and EPRI, 2005.

[167] Transaction Processing Council. http://www.tpc.org/tpc_energy/default.asp.

[168] Transaction Processing Council. http://www.tpc.org/tpch/.

[169] Transaction Processing Council. http://www.tpc.org/tpce.

[170] W. F. Tschudi, T. T. Xu, D. A. Sartor, and J. Stein. High Performance Datacenters: a Research Roadmap. Lawrence Berkeley National Laboratories, 2003.

[171] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the energy efficiency of a database server. In SIGMOD, pages 231–242, 2010.

[172] V. Vasudevan, D. Andersen, M. Kaminsky, L. Tan, J. Franklin, and I. Moraru. Energy-efficient cluster computing with fawn: workloads and implications. In e-Energy, 2010.

[173] V. Vasudevan, D. G. Andersen, M. Kaminsky, L. Tan, J. Franklin, and I. Moraru. Energy-efficient cluster computing with FAWN: Workloads and implications. Apr. 2010. (invited paper).

[174] V. Vasudevan, J. Franklin, D. Andersen, A. Phanishayee, L. Tan, M. Kaminsky, and I. Moraru. FAWNdamentally Power-efficient Clusters. In HotOS, 2009.

[175] Vertica Systems. http://www.vertica.com.

[176] VMware Incorporated. VMware Infrastructure Architecture Overview White Paper. http://www.vmware.com/pdf/vi_architecture_wp.pdf.

[177] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob. Dramsim: a memory system simulator. SIGARCH Comput. Archit. News, 33(4):100–107, 2005.

[178] C. Weddle, M. Oldham, J. Qian, A. Wang, P. Reiher, and G. Kuenning. PARAID: A gear-shifting power-aware RAID. Transactions on Storage, 3(3), 2007.

[179] C. D. Weissman and S. Bobrowski. The Design of the force.com Multitenant Internet Application Development Platform. In SIGMOD, pages 889–896, 2009.

[180] F. Wuhib, R. Stadler, and M. Spreitzer. Gossip-based Resource Management for Cloud Environments. In CNSM, pages 1–8, 2010.

[181] Z. Xu, Y.-C. Tu, and X. Wang. Exploring Power-Performance Tradeoffs in Database Systems. In ICDE, pages 485–496, 2010.

[182] L. Zhou and W. D. Grover. A Theory for Setting the "Safety Margin" on Availability Guarantees in an SLA. In DRCN, 2005.