Reducing Cluster Energy Consumption through Workload Management

Sara Alspaugh

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2012-108
http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-108.html

May 11, 2012

Acknowledgement

Many thanks to my collaborators Yanpei Chen, Andrew Krioukov, Prashanth Mohan, and Laura Keys, as well as to the many individual student reviewers in the EECS department who provided helpful feedback on various versions of text, including Peter Alvaro, Ali Gh- odsi, Matei Zaharia, and Kay Ousterhout, among others. Also thanks to Druhba Borathkor and Stephen Dawson-Haggerty for their useful comments. Lastly, thanks to my advisor Professor Randy Katz and Professor David Culler for help initiating this line of research and providing much guidance along the way.

Reducing Cluster Energy Consumption through Workload Management

by Sara Alspaugh

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor R. Katz

Research Advisor

(Date)

* * * * * * *

Professor D. Culler

Second Reader

(Date)

Abstract

Energy consumption is a major and costly problem in data centers. For many workloads, a large fraction of energy goes to powering idle machines that are not doing any useful work. There are two causes of this inefficiency: low server utilization and a lack of power proportionality. We focus on addressing this problem for two workloads: (1) a traditional, front-end web server workload and (2) an emerging class of analytics workload containing both batch jobs and interactive queries. For the web server workload of the first study, we present NapSAC, a design for a power-proportional cluster consisting of a power-aware cluster manager and a set of heterogeneous machines. Our design makes use of currently available energy-efficient hardware, mechanisms for transitioning in and out of low-power sleep states, and dynamic provisioning and scheduling to continually adjust to workload and minimize power consumption. We build a prototype cluster which runs Wikipedia to demonstrate the use of our design in a real environment. For the MapReduce workload in the second study, we develop BEEMR (Berkeley Energy Efficient MapReduce), an energy efficient MapReduce workload manager motivated by empirical analysis of real-life traces at Facebook. The key insight is that although MIA clusters host huge data volumes, the interactive jobs operate on a small fraction of the data, and thus can be served by a small pool of dedicated machines; the less time-sensitive jobs can run on the rest of the cluster in a batch fashion. With our designs we are able to reduce energy consumption while maintaining acceptable response times. Our results for NapSAC show that we are able to achieve close to 90% of the savings of a theoretically optimal provisioning scheme would achieve. With BEEMR, we achieve 40-50% energy savings under tight design constraints, and represents a first step towards improving energy efficiency for an increasingly important class of datacenter workloads.

Contents

Chapter 1

Introduction

The energy consumption of data centers has received a great deal of attention over the past decade—from massive web services like Google and Amazon located in proximity with massive electrical generation, to enterprise data centers on urban grids, to building machine rooms. The EPA estimates that U.S. data centers consumed 61 billion kilowatt-hours in 2006 at a cost of about $4.5 billion. This constitutes about 2% of US electricity consumption and has been the fastest growing consumption sector, representing an alarming growth trend [70]. Much of the focus has been on the ratio of total data center consumption to that consumed by the computing equipment, called Processor Utilization Efficiency (PUE). Typical, large data centers operate at a PUE of 2 [32], [67] so the energy required to deliver power to and to remove the heat from the servers is equal to that used by the servers [7]. Tremendous design efforts focused on improved heat exchange, air flow, and power distribution has reduced this overhead and demonstrated PUE of 1.2-1.4 [57] [63]. PUE is simple to measure and track by metering the whole center and the PDU output, but it fails to recognize that real measure of effectiveness is not the power consumed by the servers, but the work accomplished. In fact, the utilization of the servers in data centers is typically

only about 25%. For example, in a study of over 5,000 of its servers, Google found that the average CPU utilization for most servers is between 10% and 50% of maximum utilization [8]. This should not be surprising, since reasonable capacity planning must provide headroom to accommodate transient bursts in demand and potential future growth.

Such underutilization is concerning because it results in great inefficiencies in energy use. This is because modern server platforms are very far from *power proportional* despite substantial improvements in power efficiency of the microprocessor, including Dynamic Voltage/Frequency Scaling (DVFS) and the introduction of a family of sophisticated power states. Even for specially engineered platforms [8], the power consumed when completely idle is over 50% of that when fully active, and idle consumption often over 80% of peak for commodity products [23]. Thus, the service power consumption is essentially proportional to provisioned capacity, not the request rate.

Approaches to increasing datacenter energy efficiency depend on the workload in question. One option is to increase machine utilization, i.e., increase the amount of work done per unit energy. This approach is favored by large web search companies such as Google, whose machines have persistently low utilization and waste considerable energy [6]. Clusters implementing this approach would service a mix of interactive and batch workloads [21], [49], [54], with the interactive services handling the external customer queries [47], and batch processing building the data structures that support the interactive services [24]. This strategy relies on predictable diurnal patterns in web query workloads, using latency-insensitive batch processing drawn from an "infinite queue of low-priority work" to smooth out diurnal variations, to keep machines at high utilization [6], [21], [54].

In this paper we consider an alternative approach for two types of workloads. The alternative approach focuses on decreasing total cluster energy consumption, rather than increasing total cluster utilization, since both result in increased efficiency. The first workload we

apply this approach to is a traditional, front-end web server workload. The second is an emerging class of batch analytic workloads run on MapReduce-like distributed computing frameworks, which we dub MapReduce with Interactive Analysis (MIA).

For web service workloads, the amount of work is primarily determined by the rate of user requests, which can vary drastically over time. Web service operators must at the very least provision for the observed peak, taking "...the busiest minute of the busiest hour of the busiest day and build[ing] capacity on that [45]". However, most clusters provision for far more than the observed peak in order to provide a safety margin against flash crowds. A typical rule of thumb is to provision for twice the maximum average load over a moderate window experienced in the last planning cycle. In a one-week request trace obtained from Wikipedia, we observed the peak demand over a minute that was 1.6 times the average rate, though peaks as large as 19 times the average have been observed [69]. A natural consequence of the gap between peak and average requests rates in workloads, amplified by over provisioning, is that much of the time many servers sit at low levels of utilization.

MIA workloads contain interactive services as well, but also contain traditional batch processing, and large-scale, latency-sensitive processing. The last component arises from human data analysts interactively exploring large data sets via ad-hoc queries, and subsequently issuing large-scale processing requests once they find a good way to extract value from the data [12], [37], [48], [74]. Such human-initiated requests have flexible but not indefinite execution deadlines. This makes MIA workloads unpredictable: new data sets, new types of processing, and new hardware are added rapidly over time, as analysts collect new data and discover new ways to analyze existing data [12], [37], [48], [74]. Thus, increasing utilization here is insufficient: First, the workload is dominated by human-initiated jobs. Hence, such clusters must also be provisioned for peak load to maintain good SLOs, and low-priority batch jobs only partially smooth out the workload variation. Second, the workload has unpredictable high spikes compared with regular diurnal patterns for web queries,

resulting in wasted work from batch jobs being preempted upon sudden spikes in the workload. In short, MapReduce has evolved far beyond its original use case of high-throughput batch processing in support of web search-centric services, and it is critical that we develop energy efficiency mechanisms for MIA workloads.

For web services workloads, we design and implement NapSAC, a power proportional cluster constructed out of non power proportional systems. The basic approach is fairly obvious – put idle servers to sleep and wake them up when they are needed. Thus, capital equipment spend tracks peak demand, but energy consumption tracks delivered service. However, realizing this simple goal presents several challenges. The active work must be coalesced into a subset of nodes, so that a significant amount of coarse-grained, i.e., entire node idleness, is obtained. But enough capacity must be available to absorb bursts in demand. Servers exhibit a dramatic increase in delay or loss rate as the requested load approaches the maximum delivered load, which traditional services avoid by having all the resources running all the time. The key observation is that most clustered systems employ a master-worker pattern where the master is responsible for load management. It is natural to extend the master to manage the power state of the workers and to assign work in a manner that is consistent with its power provisioning. We dynamically provision active nodes to track demand, while providing headroom for bursts. Interestingly, the solution is very similar to approach used by electric utilities to match power generation to time varying demand without any explicit protocol between consumers and providers. A set of baseline resources is supplemented by an array of intermittent resources. Intermittent resources are brought on-line or off-line in accordance with estimated demand in the near future. Each of those resources maintain a certain amount of "spinning reserve" to absorb transient bursts and to cover the ramp-up delay in spinning up additional resources. Each seeks to operate at sufficiently high utilization to be efficient, while leaving enough reserve capacity to be reliable.

For MIA workloads, we present BEEMR (Berkeley Energy Efficient MapReduce), an en-

ergy efficient MapReduce system motivated by an empirical analysis of a real-life MIA workload at Facebook. This workload requires BEEMR to meet stringent design requirements, including minimal impact on interactive job latency, write bandwidth, write capacity, memory set size, and data locality, as well as compatibility with distributed file system fault tolerance using error correction codes rather than replication. BEEMR represents a new design point that combines batching [40], zoning [41], and data placement [38] with new analysis-driven insights to create an efficient MapReduce system that saves energy while meeting these design requirements. The key insight is that although MIA clusters host huge volumes of data, the interactive jobs operate on just a small fraction of the data, and thus can be served by a small pool of dedicated machines; whereas the less time-sensitive jobs can run in a batch fashion on the rest of the cluster. These defining characteristics of MIA workloads both motivate and enable the BEEMR design. BEEMR increases cluster utilization while batches are actively run, and decreases energy waste between batches because only the dedicated interactive machines need to be kept at full power.

Our basic results are, for the web services workload, achieving 90% of the savings that an ideal optimal scheme would achieve, assuming a 2x provisioning rule. With our NapSAC design we are able to reduce energy consumption while maintaining acceptable response times for a web service workload based on Wikipedia. With MIA workloads run on BEEMR, we show energy savings of 40-50%. BEEMR highlights the need to design for an important class of data center workloads, and represents an advance over existing MapReduce energy efficiency proposals [38], [40], [41]. Systems like BEEMR become more important as the need for energy efficiency continues to increase, and more use cases approach the scale and complexity of the Facebook MIA workload.

The remainder of the report is organized as follows. We first describe the workloads for which we build NapSAC and BEEMR (Chapter 2). Next, we discuss the design, implementation, evaluation methodology, and results for our NapSAC power proportional web

services cluster (Chapter 3). We do the same for our energy efficient MIA workload architecture, BEEMR (Chapter 4). We conclude with a some closing thoughts and summary of results and contributions (Chapter 5).

Chapter 2

Background

Different workloads exhibit different characteristics, including diurnal patterns, periods of inactivity, and large event-flurries or spikes in activity. System operators must provision for the worst-case workloads in which all the available system resources are requested, and generally this worst-case provisioning is in use all the time, regardless of how likely the worst-case maximum workload is to occur at that point in time. Over-provisioning in this case is a problem if one cares about energy usage of equipment.

Power proportionality refers to power usage corresponding linearly to the rate of work being done. For example, when a web service is handling 100 requests per second the power of the cluster should be half of the power when handling 200 requests per second. To achieve a power-proportional system, we vary the amount of available resources (compute nodes, in our case) to match the incoming workload. An example of this is shown in Figure 2.1. The capacity of the cluster varies dynamically with the request rate by changing the power state of the underlying compute nodes.

The Advanced Configuration and Power Interface (ACPI) specification defines two types of low-power states: idle states (C states) and sleep states (S states) [22]. Idle states re-

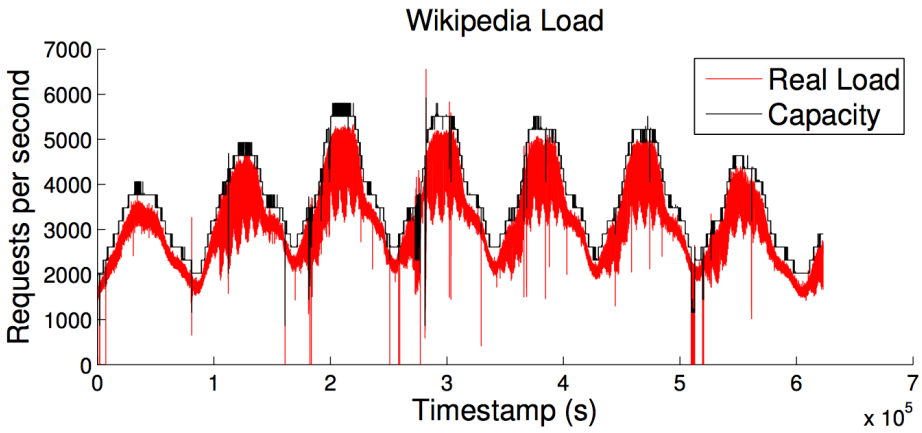


Figure 2.1. A subsampled, two-week Wikipedia trace. The trace exhibits strong diurnal patterns. Typically, clusters supporting workloads such as this would be provisioned for twice the peak observed.

duce processor power consumption by disabling the CPU clock, cache, and other on-chip circuitry. They are characterized by fast transition times and are generally handled automatically by the operating system. Sleep states, on the other hand, include such states as suspend-to-RAM and suspend-to-disk. In these states most of the system is powered down except for the network card which remains active to support Wake-on-LAN (WOL) and thus offer much more aggressive power savings.

2.1 The Wikipedia Workload

We use several days of HTTP traffic data from a trace consisting of a 10% sample of the total Wikipedia traffic [69] as our workload for comparison. We assume a fixed response time and a maximum request rate per system. These parameters are set empirically by using microbenchmarks. We take the maximum request rate with a response time of less than 300ms for the the Wikipedia workload on each system. Figure 2.1 shows a trace of this workload over a two-week period.

Our cluster serves a snapshot of Wikipedia which implements the typical three-tier architecture of web service applications. At the front end we have a load balancer that distributes load among a set of application servers. The application servers generate the requested content using data from the storage (database) layer. This architecture is generic and supports many different applications and storage systems.

2.2 The Facebook Workload

MIA-style workloads have already appeared in several organizations, including both web search and other businesses [12], [37], [48]. Several technology trends help increase the popularity and generality of MIA workloads:

- Industries ranging from e-commerce, finance, and manufacturing are increasingly adopting MapReduce as a data processing and archival system [34].
- It is increasingly easy to collect and store large amounts of data about both virtual and physical systems [12], [26], [38].
- Data analysts are gaining expertise using MapReduce to process big data sets interactively for real-time analytics, event monitoring, and stream processing [12], [37], [48].

In short, MapReduce has evolved far beyond its original use case of high-throughput batch processing in support of web search-centric services, and it is critical that we develop energy efficiency mechanisms for MIA workloads.

We analyze traces from the primary Facebook production MapReduce cluster. The cluster has 3000 machines. Each machine has 12+ TB, 8-16 cores, 32 GB of RAM, and roughly 15 concurrent map/reduce tasks [11]. The traces cover 45 days from Oct. 1 to Nov. 15, 2010, and contain over 1 million jobs touching tens of PB of data. The traces record each

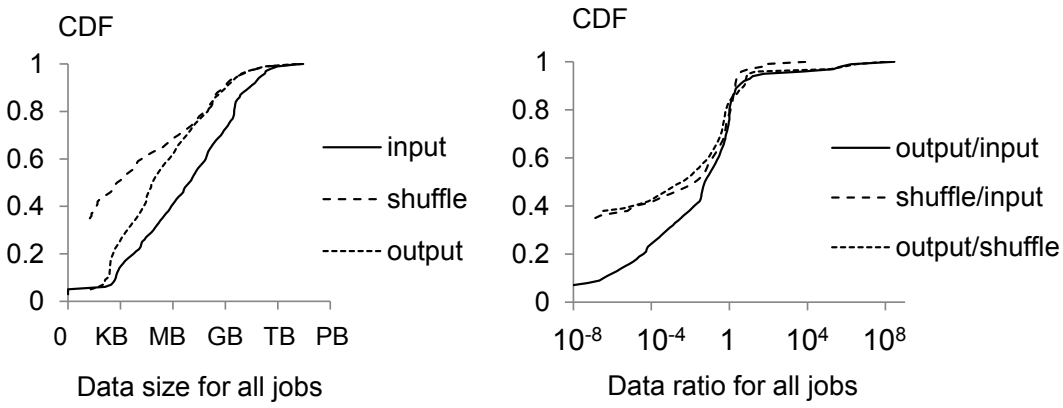


Figure 2.2. CDFs of input/shuffle/output sizes and ratios for the entire 45-day Facebook trace. Both span several orders of magnitudes. Energy efficiency mechanisms must accommodate this range.

job's job ID, input/shuffle/output sizes, arrival time, duration, map/reduce task durations (in task-seconds), number of map/reduce tasks, and input file path.

Figure 2.2 shows the distribution of per-job data sizes and data ratios for the entire workload. The data sizes span several orders of magnitude, and most jobs have data sizes in the KB to GB range. The data ratios also span several orders of magnitude. 30% of the jobs are map-only, and thus have 0 shuffle data. Any effort to improve energy efficiency must account for this range of data sizes and data ratios.

Figure 2.3 shows the workload variation over two weeks. The number of jobs is diurnal, with peaks around midday and troughs around midnight. All three time series have a high peak-to-average ratio, especially map and reduce task times. Since most hardware is not power proportional [6], a cluster provisioned for peak load would see many periods of below peak activity running at near-peak power.

To distinguish among different types of jobs in the workload, we can perform statistical data clustering analysis. This analysis treats each job as a multi-dimensional vector, and finds

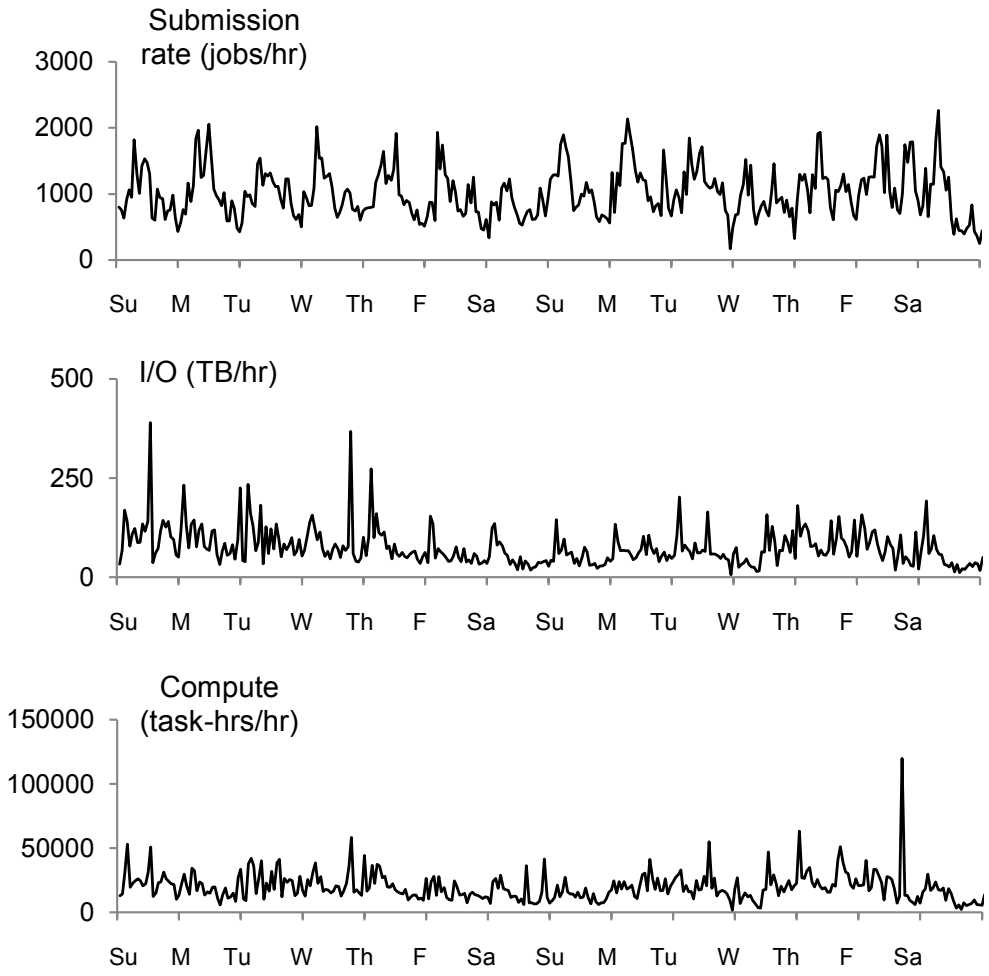


Figure 2.3. Hourly workload variation over two weeks. The workload has high peak-to-average ratios. A cluster provisioned for the peak would be often underutilized and waste a great deal of energy.

# Jobs	Input	Shuffle	Output	Duration	Map time	Reduce time	Label
1145663	6.9 MB	600 B	60 KB	1 min	48	34	Small jobs
7911	50 GB	0	61 GB	8 hrs	60,664	0	Map only transform, 8 hrs
779	3.6 TB	0	4.4 TB	45 min	3,081,710	0	Map only transform, 45 min
670	2.1 TB	0	2.7 GB	1 hr 20 min	9,457,592	0	Map only aggregate
104	35 GB	0	3.5 GB	3 days	198,436	0	Map only transform, 3 days
11491	1.5 TB	30 GB	2.2 GB	30 min	1,112,765	387,191	Aggregate
1876	711 GB	2.6 TB	860 GB	2 hrs	1,618,792	2,056,439	Transform, 2 hrs
454	9.0 TB	1.5 TB	1.2 TB	1 hr	1,795,682	818,344	Aggregate and transform
169	2.7 TB	12 TB	260 GB	2 hrs 7 min	2,862,726	3,091,678	Expand and aggregate
67	630 GB	1.2 TB	140 GB	18 hrs	1,545,220	18,144,174	Transform, 18 hrs

Table 2.1. Job types in the workload as identified by k-means clustering, with cluster sizes, medians, and labels. Map and reduce time are in task-seconds, i.e., a job with 2 map tasks of 10 seconds each has map time of 20 task-seconds. Notable job types include small, interactive jobs (top row) and jobs with inherently low levels of parallelism that take a long time to complete (fifth row). We ran k-means with 100 random instantiations of cluster centers, which averages to over 1 bit of randomness in each of the 6 data dimensions. We determine k, the number of clusters by incrementing k from 1 and stopping upon diminishing decreases in the intra-cluster "residual" variance.

clusters of similar numerical vectors, i.e., similar jobs. Our traces give us six numerical dimensions per job — input size, shuffle size, output size, job duration, map time, and reduce time. Table 2.1 shows the results using the k-means algorithm, in which we labeled each cluster based on the numerical value of the cluster center.

Most of the jobs are small and interactive. These jobs arise out of ad-hoc queries initiated by internal human analysts at Facebook [12], [68]. There are also jobs with long durations but small task times (map only, GB-scale, many-day jobs). These jobs have inherently low levels of parallelism, and take a long time to complete, even if they have the entire cluster at their disposal. Any energy efficient MapReduce system must accommodate many job types, each with their own unique characteristics.

Figures 2.4 and 2.5 show the data access patterns as indicated by the per-job input paths. Unfortunately our traces do not contain comparable information for output paths. Figure 2.4 shows that the input path accesses follow a Zipf distribution, i.e., a few input paths account for a large fraction of all accesses. Figure 2.5 shows that small data sets are accessed frequently; input paths of less than 10s of GBs account for over 80% of jobs, but only a tiny

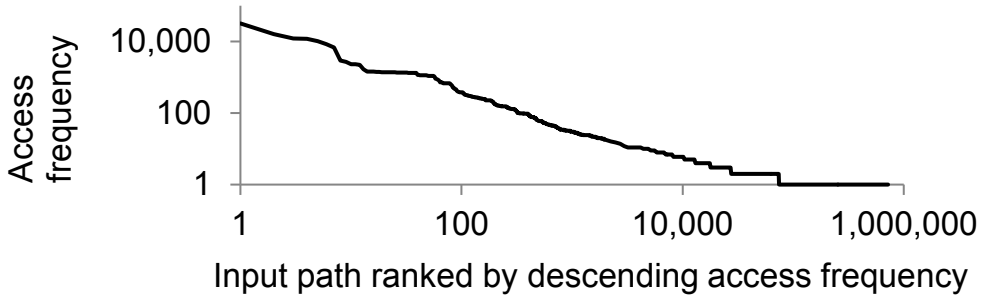


Figure 2.4. Log-log plot of workload input file path access frequency. This displays a Zipf distribution, meaning that a few input paths account for a large fraction of all job inputs.

fraction of the total size of all input paths. Prior work has also observed this behavior in other contexts, such as web caches [13] and databases [30]. The implication is that *a small fraction of the cluster is sufficient to store the input data sets of most jobs.*

Other relevant design considerations are not evident from the traces. First, some applications require high write throughput and considerable application-level cache, such as Memcached. This fact was reported by Facebook in [12] and [68]. Second, the cluster is storage capacity constrained, so Facebook's HDFS achieves fault tolerance through error correcting codes instead of replication, which brings the physical replication factor down from three to less than two [61]. Further, any data hot spots or decreased data locality would increase MapReduce job completion times [2].

Table 2.2 summarizes the design constraints. They represent a superset of the requirements considered by existing energy efficient MapReduce proposals.

2.3 Related Work

Many CPUs have support for Dynamic Voltage/Frequency Scaling (DVFS) which can be used to dynamically reduce CPU performance and save energy when load is low. Many

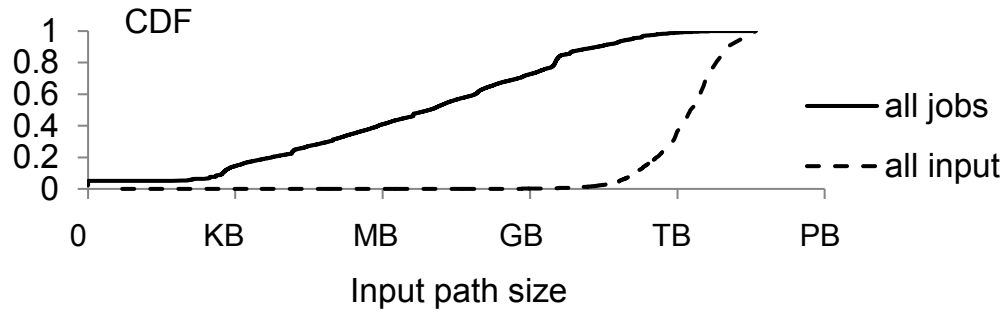


Figure 2.5. CDF of both (1) the input size per job and (2) the size per input path. This graph indicates that small input paths are accessed frequently, i.e., data sets of less than 10s of GBs account for over 80% of jobs, and such data sets are a tiny fraction of the total data stored on the cluster.

papers have explored policies for reducing CPU power with DVFS [33], [72]. In [44], Lorch et al. use predictive models to estimate future load and create a power schedule. Unfortunately, the CPU currently contributes less than 50% to overall system power [8], thus we focus on whole system power management.

Virtual machines can be used to dynamically add or remove machines in response to change in load. Several recent papers have used machine learning to dynamically provision virtual machines while maintaining quality of service goals [10], [39]. Virtual machines take minutes to boot or migrate and introduce performance overheads. In contrast to this work we operate at a granularity of seconds, giving us more agility in dealing with sharp changes in request rate.

2.3.1 Web Server Clusters

Several papers have proposed putting machines to sleep. PowerNap [46] models how quickly servers would have to transition in and out of sleep states to achieve energy proportionality on web and other workloads. The authors find that a transition time of under 10ms

is required for significant power savings, unfortunately, sleep times on current servers are two orders of magnitude larger. In contrast, we build a power power proportional cluster with current hardware by using predictive provisioning. Chase et al. [15] propose an economic model for allocating resources in a hosting center and putting unused servers into sleep states. A key difference is that our provisioning algorithm explicitly optimizes for energy efficiency utilizing a heterogeneous cluster. We find this heterogeneity essential for obtaining large energy savings with acceptable performance. Gong et al. [16] also trade off user experience with energy usage in the data center. Their work specifically deals with long lived connections as in instant messengers while we concentrate on short lived request-response type of workloads. Pinheiro et al. [55] apply dynamic provisioning strategies to web clusters. However the use of a heterogeneous set of server types and power states that have fast transition times allow us to handle load spikes much better. Finally, we recreate a cluster against a real production class application (Wikipedia) and also evaluate the tradeoffs between user experience and energy usage on traces captured on the Wikipedia servers.

Recent work by Andersen et al. [3] has proposed using embedded processors for reducing energy consumption in I/O based workloads. We also believe that embedded and mobile processors have a place in the data center, however, we consider a broader class of workloads which cannot be run exclusively on embedded systems. Chun et al. [20] make the case for hybrid data centers with the observation that some applications have significantly different performance per watt on different platforms. We agree with this vision and we measure a substantial energy savings when using a heterogeneous cluster over a homogeneous one.

2.3.2 Parallel Computation Frameworks

Prior work includes both energy-efficient MapReduce schemes as well as strategies that apply to other workloads.

Energy Efficient MapReduce

Existing energy efficient MapReduce systems fail to meet all the requirements in Table 2.2. We review them here.

The covering subset scheme [41] keeps one replica of every block within a small subset of machines called the covering subset. This subset remains fully powered to preserve data availability while the rest is powered down. Operating only a fraction of the cluster decreases write bandwidth, write capacity, and the size of available memory. More critically, this scheme becomes unusable when error correction codes are used instead of replication, since the covering subset becomes the whole cluster.

The all-in strategy [40] powers down the entire cluster during periods of inactivity, and runs at full capacity otherwise. Figure 2.3 shows that the cluster is never completely inactive. Thus, to power down at any point, the all-in strategy must run incoming jobs in regular batches, an approach we investigated in [19]. All jobs would experience some delay, an inappropriate behavior for the small, interactive jobs in the MIA workload (Table 2.1).

Green HDFS [38] partitions HDFS into disjoint hot and cold zones. The frequently accessed data is placed in the hot zone, which is always powered. To preserve write capacity, Green HDFS fills the cold zone using one powered-on machine at a time. This scheme is problematic because the output of every job would be located on a small number of machines, creating a severe data hotspot for future accesses. Furthermore, running the cluster at partial capacity decreases the available write bandwidth and memory.

Desirable Property	**Covering subset** [41]	**All-In** [40]	**Hot & Cold Zones** [38]	**BEEMR**
Does not delay interactive jobs	✔		✔	✔
No impact on write bandwidth		✔		✔
No impact on write capacity		✔	✔	✔
No impact on available memory		✔		✔
Does not introduce data hot spots nor impact data locality	✔	✔		✔
Improvement preserved when using ECC instead of replication		✔	✔	✔
Addresses long running jobs with low parallelism				Partially
Energy savings	9-50%[1]	0-50%[2]	24%[3]	40-50%

Table 2.2. Required properties for energy-saving techniques for Facebook's MIA workload. Prior proposals are insufficient. Notes: [1] The reported energy savings used an energy model based on linearly extrapolating CPU utilization while running the GridMix throughput benchmark [31] on a 36-node cluster. [2] Reported only relative energy savings compared with the covering subset technique, and for only two artificial jobs (Terasort and Grep) on a 24-node experimental cluster. We recomputed absolute energy savings using the graphs in the paper. [3] Reported simulation based energy *cost* savings, assumed an electricity cost of $0.063/KWh and 80% capacity utilization.

[]

The prior studies in Table 2.2 also suffer from several methodological weaknesses. Some studies quantified energy efficiency improvements by running stand-alone jobs, similar to [59]. This is the correct initial approach, but it is not clear that improvements from stand-alone jobs translate to workloads with complex interference between concurrent jobs. More critically, for workloads with high peak-to-average load (Figure 2.3), per-job improvements fail to eliminate energy waste during low activity periods.

Other studies quantified energy improvements using trace-driven simulations. Such simulations are essential for evaluating energy efficient MapReduce at large scale. However, the simulators used there were not empirically verified, i.e., there were no experiments comparing simulated versus real behavior, nor simulated versus real energy savings. Section 4.4.8 demonstrates that an empirical validation reveals many subtle assumptions about simulators, and put into doubt the results derived from unverified simulators.

These shortcomings necessitate a new approach in designing and evaluating energy efficient MapReduce systems.

Energy Efficient Web Search-Centric Workloads

MIA workloads require a different approach to energy efficiency than previously considered workloads.

In web search-centric workloads, the interactive services achieves low latency by using data structures in-memory, requiring the entire memory set to be always available [47]. Given hardware limits in power proportionality, it becomes a priority to increase utilization of machines during diurnal troughs [6]. One way to do this is to admit batch processing to consume any available resource. This policy makes the combined workload *closed-loop*, i.e., the system controls the amount of admitted work. Further, the combined workload becomes more *predictable*, since the interactive services display regular diurnal patterns, and with batch processing smoothing out most diurnal variations [6], [28], [47].

These characteristics enable energy efficiency improvements to focus on *maximizing the amount of work done subject to the given power budget*, i.e., maximizing the amount of batch processing done by the system. Idleness is viewed as waste. Opportunities to save energy occur at short time scales, and requires advances in hardware energy efficiency and power proportionality [6], [9], [25], [28], [46], [47], [65].

These techniques remain helpful for MIA workloads. However, the open-loop and unpredictable nature of MIA workloads necessitates additional approaches. Human initiated jobs have both throughput and latency constraints. Thus, the cluster needs to be provisioned for peak, and idleness is inherent to the workload. Machine-initiated batch jobs can only partially smooth out transient activity peaks. Improving hardware power proportionality helps, but remains a partial solution since state-of-the-art hardware is still far from perfectly power

proportional. Thus, absent policies to constrain the human analysts, improving energy efficiency for MIA workloads requires *minimizing the energy needed to service the given amount of work*.

More generally, energy concerns complicate capacity provisioning, a challenging topic with investigations dating back to the time-sharing era [4], [5], [62]. This paper offers a new perspective informed by MIA workloads.

Chapter 3

Front-End Web Services Study

In this chapter, we present the design, implementation, and evaluation of a power-aware cluster manager called NapSAC which performs Server Actuation and Control on a heterogeneous set of machines. It dynamically provisions enough machines to serve the requested load with just enough headroom to accommodate the bursts and allow for re-provisioning to increased demand. NapSAC energy consumption is proportional to the work performed at essentially a Joule per request, despite large variation in the request rate, while static provisioning is only efficient near the peak load. Furthermore, the fine-grained, slight over provisioning required to track the load without incurring a significant congestion penalty is small compared to the worst-case provisioning used in capacity planning.

The system architecture of NapSAC is described in Section 3.1. It employs a master-worker pattern where the master is responsible for load management. The master manages the power state of the workers and to assign work in a manner that is consistent with its power provisioning. Section 3.2 describes our implementation using mechanisms available on current platforms. We explain our evaluation methodology in Section 3.3 Section 3.4 contains our results. We first perform an empirical analysis of typical platforms to establish

the critical parameters and trade-offs that guide the scheduling and provisioning algorithms (Section 3.4.1). As of 2010, leading server products do not actually support going into standby and resuming, other than by actuating the AC power line, so we examine the use of desktop, mobile, and even embedded platforms for use in service applications. The Wake-on-LAN and low-power sleep states available on these platforms are sufficient mechanisms for this design. Sections 3.4.2 through 3.4.4 explore the key axes of the system design space, including prediction and provisioning algorithms, through simulation driven by a production Wikipedia workload. We use the results from this exploration to set parameters in the cluster manager We then evaluate the full system via simulation in Section 3.4.5 We find that under the 2x provision rule we are able to achieve 90% of the savings that a ideal optimal scheme would achieve. Lastly, we validate these results using a prototype cluster built from Atom-based mobile platforms Section 3.4.6. With our design we are able to reduce energy consumption while maintaining acceptable response times for a web service workload based on Wikipedia.

3.1 NapSAC Architecture

NapSAC follows a conventional n-tier Internet service architecture. A large number of clients interact with a web server tier through a load balancing switch. The web server tier is supported by a collection of application servers, which provide dynamic content. Lastly, there is a upon a back-end storage tier, e.g., database servers or a SAN, for persistent data storage [14]. Typically, the front-end servers have large memories for caching, along with local storage. The resources devoted to each tier are sized according to a capacity plan that is based on historical and anticipated workload.

NapSAC augments this general architecture with a cluster manager, as shown in Figure 3.1. The cluster manager coordinates the individual servers to achieve cluster-level power pro-

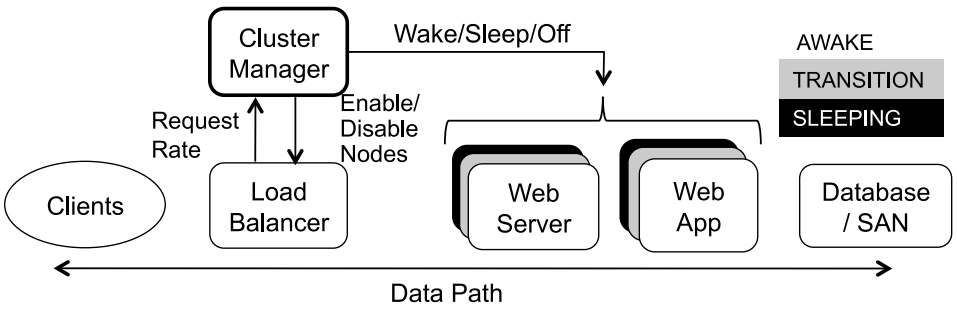


Figure 3.1. NapSAC architecture. We add a cluster manager to the standard multi-tier Internet service architecture. The cluster manager interacts with the load manager to measure and predict incoming load. Based on its prediction, it provisions a number of servers by keeping them active. The rest are put to sleep. The cluster attempts to minimize the servers needed awake by coalescing the workload onto the smallest number of machines possible, and transition the rest to sleep.

portionality. It does this by lowering cluster power consumption during periods of low load, transitioning subsets of front-end servers to low-power sleep states while minimally impacting performance. It seeks to deliver the performance of a fully-powered cluster of the same resources but with less energy consumption. More sophisticated policies might use these capabilities to obtain a specific target power consumption profile with minimum degradation of service.

We note parenthetically that with NapSAC the back-end storage tier machines remain active at all times to provide availability to all of the data. However, in many web service clusters, including Wikipedia, upon which this study is based, the number of front-end servers greatly exceeds the number of storage servers, thereby accounting for the largest proportion of wasted power [73]. Thus, for improving energy efficiency in such clusters, focusing on the front-end servers is most important.

The cluster manager monitors the stream of incoming requests through the load balancer to compute the current load on the cluster and predict future load. It uses this information to determine the minimum set of front-end machines that must be in service. When load is low, exceeded by the processing capacity of the currently active servers, the cluster manager

puts the unneeded servers into a low-power sleep state. When load increases beyond the capacity of the currently active servers, it wakes up servers. As indicated by the shading in Figure 3.1, the servers may be either awake, sleeping or in transition; those in transition may be spinning up and not yet serving requests, or going to sleep after completing current requests. The cluster manager maintains a map of the power state of the nodes and directs the load balancer to send requests to active nodes. To enable the manager to command nodes to transition from one power state to another, nodes are augmented with a daemon that processes sleep and wake-up commands from the cluster manager. Wake-up is triggered by Wake-on-LAN or similar facilities [42].

Our algorithms are equipped to deal with a heterogenous set of nodes, to take advantage of the strengths of different classes of machines. We consider three classes: server, mobile, and embedded Each class has different power demands, processing capacity, ramp up times, and peak efficiencies. Server-class machines consume a lot of power, but tend to be efficient at peak utilization. However, they often lack the kinds of agile, low-power sleep states found in mobile hardware. This means that to save power, they must be turned off, and turning them back on when they are needed can take a long time, on the order of a minute. On the other hand, mobile-class machines have low power consumption and can transition in and out of low-power states very quickly, however, they are not always more efficient at peak utilization. Lastly, embedded hardware is often very efficient, but has much lower processing capacity. These trade-offs suggest that the provisioning algorithm should select different nodes based on the workload. For example, large servers should be used to handle long lasting "base load", while the mobile systems that can be brought up quickly can be used to handle short workload bursts.

The main challenge is to maintain just enough capacity to handle spikes and increases in the average load, given that it takes time to spin up new servers. Doing this involves rate prediction and provisioning, which is performed at an interval of `pwnd` seconds. The clus-

ter manager starts with estimates of (1) the processing capacity of each front-end server, and (2) the amount of time it takes a server to transition out of a low-power state. These estimates are based on past history, measurement, or configuration The processing capacity of each server is described both by a target operating point and a maximum capacity constraint. Typically, the target operating point of a node is at approximately half of the maximum processing rate that a node can deliver effectively. The cluster manager uses its rate prediction algorithm to estimate what the load will be `pwnd` seconds into the future. It is important that `pwnd` be no less than the amount of time it takes to spin up a new node into an active power state, to avoid a situation in which the amount of load exceeds the cluster capacity. The cluster manager attempts to maintain the invariant that the aggregate operating capacity is equal to the predicted load one ramp time ahead, or equivalently that the maximum capacity is twice this amount.

3.1.1 Prediction

Rate prediction is used to hide server ramp-up latency to minimize the impact of power management on performance. By predicting the request rate we can bring up systems ahead of when they become necessary. Some possible prediction algorithms include:

Last Arrival (LA) predicts all future values to be the request rate seen in the last second. This is equivalent to an "on demand" algorithm which brings up nodes only after a violation.

Moving Window Average (MWA) computes the average request rate over the last 'α' seconds. Extrapolation is done by averaging in the request rate over the last second.

Exponentially Weighted Average (EWA) maintains a weighted average over all request history $\texttt{rate} = \alpha\ \texttt{old\_rate} + (1 - \alpha)\ \texttt{current\_rate}$. A rate n seconds in the future is obtained by averaging in the request rate of the last second n times.

NapSAC uses an EWA prediction algorithm with an α value of 0.95. We evaluate this choice in 3.4.2.

3.1.2 Provisioning

The provisioning algorithm must keep enough machines awake such that, if they are utilized at their target operating point, they will be able to handle the load predicted within `pwnd`. The provisioning algorithm uses a knapsack algorithm to select the set of servers to bring up for the predicted workload. For all available systems, the algorithm predicts the load at the time each system could be ramped up. It uses a bin packing algorithm to choose a set of machines, weighted by their efficiency, such that the sum of the system operating points is above the predicted load. This maximizes system utilization while preferring the most efficient systems. Finally, the algorithm subtracts out systems that are already active or have been previously selected to turn on. Any systems still remaining in the list are brought up. The full startup algorithm is shown in Listing 3.1. The shutdown algorithm, shown in Listing 3.2 is largely analogous, but more conservative in turning systems off. It computes the maximum set of machines for all prediction windows. If the current set of nodes is a superset of this, then the excess nodes are turned off. By default, we run these provisioning algorithms every second. Other provisioning intervals are explored in Section 3.4.4

Listing 3.1. Provisioning new servers to handle increased load.

```
for servtyp in servtypes:
  ramptime = servtyp.ramptime
  # predict and start servers
  pred = predict_load(now + ramptime)
  cluster = make_cluster(pred)
          - current_cluster
          - nodes waking up in time
  start servtyp servers in cluster
```

Listing 3.2. Putting unneeded servers into a low-power state when load decreases.

```
for servtyp in reverse(servertypes):
  ramptime = servtyp.ramptime
  maxcluster = emptycluster
  for t in range(1, ramptime):
      pred = predict_load(now + t)
      tmpcluster = make_cluster(pred)
      maxcluster = max(maxcluster,tmpcluster)
  tmpcluster = current_cluster - maxcluster
  turn off servers in tmpcluster
```

3.1.3 Example Behavior

The operation of NapSAC is illustrated in Figure 3.2. The figure also shows a static provisioning line, based on traditional, power-oblivious capacity planning, across the top for comparison. In this example, the load is initially constant. The predictor estimates that this rate will continue, so the manager decides that the active nodes provide adequate head room and distributes requests over the active nodes. Then the request rate begins to increase. A relatively stable estimator is used, so the predicted rate lags the instantaneous rate during this increase. The algorithm is operating on-line, so it cannot tell whether the increase is a transient burst or a change in trend. Either way, the increase is absorbed by headroom maintained by the provisioning algorithm. For a short period, the instantaneous demand is within the active capacity, but then predictor indicates that the demand will exceed the operating capacity within `pwnd`. Thus, the cluster manager initiates the wake up of a sleeping node. By the time this node has come on-line and the available capacity has increased, the instantaneous load has also increased. Sufficient headroom is available for a short period, and eventually additional resources are brought on-line until the full capacity of the cluster is used. Since we assume that the cluster has been provisioned to handle such peaks,

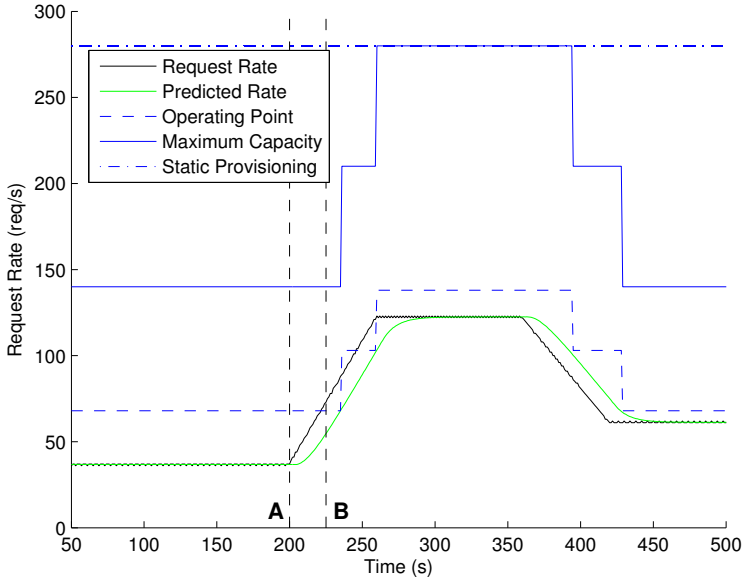


Figure 3.2. Example of the power-proportional NapSAC cluster in action: Initially, load is constant at 40 requests per second. At 200 seconds (A), request rate increases. At 225 seconds (B), demand is predicted to exceed the operating capacity within `pwnd`, so the cluster manager activates a sleeping node. As load increases, it is absorbed the capacity headroom left by the cluster manager, until more nodes are brought online. As load decreases below the aggregate capacity of the cluster, nodes are transitioned back into low-power sleep states.

we do not consider what happens when the full cluster capacity is exceeded. As demand decreases, machine utilization falls, decreasing efficiency. When the load is below the aggregate operating point of the cluster minus one node, a node is spun down. Future requests are distributed over the remaining active nodes. When the last pending request completes, the node informs the cluster manager and puts itself to sleep.

3.2 Implementation

Our implementation of NapSAC is shown in Figure 3.3. The cluster consists of 16 Atom nodes, 8 Beagleboard nodes, and 4 Nehalem nodes. We evaluate these platforms in Sec-

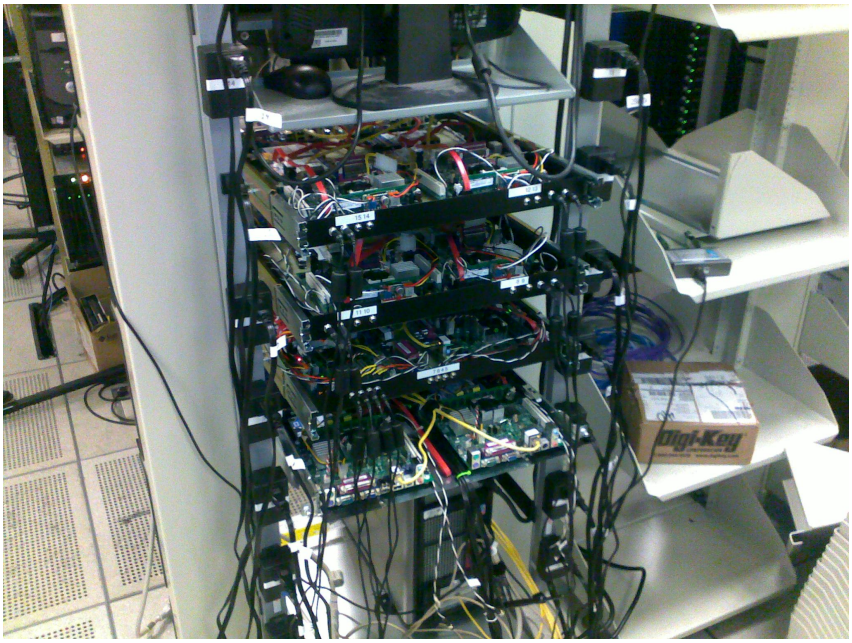


Figure 3.3. A photograph of the actual cluster including the 16 Atom back-end nodes and the Core 2 Duo that hosts both the cluster manager and the load balancer. (The 8 Beagleboards and 4 Nehalem nodes are not pictured here.)

tion 3.4.1. These nodes sit on a 10 Gbps switch behind our cluster manager and load balancer, which runs on Intel Core 2 Duo-based desktop hardware. All of the cluster nodes run Linux; the servers with a patch to disable ARP, as explained below.

We explore the use of three classes of machines in our implementation: server, mobile, and embedded. In the server class, we evaluate a high-end machine with a Nehalem architecture quad-core Intel Xeon X5550. The mobile class platform is a dual-core Intel Atom 330. Finally, the embedded system is an ARM-based Beagle Board. Although our cluster contains machines from each class, our whole-cluster prototype evaluations center primarily upon the mobile-class platform. This is because it is easier to see their power and performance contribution when added or removed from a cluster than the low-power, low-performance embedded system, and the server-class nodes do not support sleep states.

Each node in our cluster, except the load balancer and cluster manager, runs the web server

Lighttpd. In our implementation, all of the servers and the load balancer share an IP address. However, the servers are modified so that only the load balancer responds to ARP requests for the IP address, ensuring that all traffic destined for the system will pass through it. Thus when a new request arrives from a client, the load balancer chooses an application server from the pool of machines specified by the cluster manager and forwards the request in an Ethernet frame addressed to that application server. Then, the response is sent directly from this chosen application server back to the client.

For the load balancer software we use the low-level software-based Linux Virtual Sever (LVS) [75]. LVS has a direct routing mode where it forwards packets at the Ethernet layer; this way it can support a large number of connections and is oblivious to packet contents. LVS worked well for our purposes with one exception: the process of frequently adding and removing machines from the set of machines to load balance among caused the load balancer to lock up after several iterations. We suspect this is due to a bug in LVS which is probably not frequently encountered. We explain our temporary workaround in Section 3.2. The cluster manager runs on the same machine as the load balancer, although this is not necessary. The manager uses the MWA predictor with $\alpha = .95$. Provisioning is done every 1 second, with an over provisioning factor of 30%.

Our implementation currently supports two web service applications for the purposes of evaluation. The first is a simple CPU-intensive PHP application, and the second is MediaWiki. We configure MediaWiki to serve a snapshot of Wikipedia, which implements the three-tier architecture typical of web service applications. Wikipedia makes ample use of caching, using front-end Squid reverse web cache servers, distributed memory cache memcached, PHP bytecode caches, and database caches. Wikipedia also uses several LVS servers for load-balancing. The application servers run the MediaWiki application using a PHP interpreter. A PHP accelerator is also used to cache the PHP byte code. A snapshot of Wikipedia data is stored in a central MySQL database accessible to all of the application

servers. To host the back-end database, we use one Nehalem running a MySQL server containing a full copy of the English Wikipedia article data, totaling approximately 22 GB. The application servers also maintain a local disk cache of the recently accessed content. In particular, they each use a PHP bytecode cache called APC, along with the built-in MediaWiki FileCache, instead of memcached or dedicated Squid servers.

3.3 Evaluation Methodology

We evaluate our design through experiments on our prototype implementation as well as a simulator. We first evaluate the hardware platforms from which our prototype is constructed. We use data gathered from these evaluations to feed into our simulator. With the simulator, we explore several different rate prediction and provisioning schemes in detail to evaluate trade-offs. Then, we test the performance of the entire system using our prototype driven by a real-world trace from Wikipedia.

3.3.1 Simulator

The simulator models a cluster with an arbitrary number of compute nodes. For each compute node it has a performance model and a power model. We construct the simulator performance model empirically by running a sample workload on the real hardware described in Section 3.1 and Section 3.4.1. The performance model describes the maximum sustainable request rate for a given type of request. We consider any requests sent at a higher rate as "violations" because on a real system they may have response times that violate the desired maximum latency. We also maintain a power model for each node which consists of the system power states and system efficiency. This includes idle power, peak power and sleep states as shown in Table 3.1. There are also power states for transitions

such as “going to sleep state” and “resuming from sleep.” These states are active for a fixed amount of time corresponding to the wake up and sleep times of the machines.

The simulator inputs a request trace with timestamp and request type. Each request is assigned to an active node in the virtual cluster according to the scheduling algorithm. For web requests this is done using a weighted round-robin algorithm where machines receive requests in proportion to their maximum request rate. If a node is assigned requests faster than its maximum request rate, the excess requests are marked as “violations.” A provisioning algorithm is run at a fixed interval to reconfigure the cluster. This updates the power states of nodes in the cluster, potentially putting systems into transition states. Internally, the simulator maintains an event queue which is executed at fixed time steps for efficiency reasons. This is necessary for quickly executing week-long traces consisting of over a billion requests. The request trace we use for evaluation is a 7 day HTTP log consisting of a 10% sample of the total Wikipedia.org traffic [69].

3.3.2 Prototype

We evaluate both the hardware platforms we use for the individual nodes from which our cluster is constructed, as well as the operation of the cluster as a whole.

Individual Nodes

In our evaluation of hardware platform, we measured energy efficiency, agility, overall performance, and total power consumption.

Efficiency. Informally, one key feature of a good platform is one that can process a large number of requests per unit energy expended. Thus, our metric of energy efficiency is Joules per successful web request, where a successful web request is one whose response time is 300 ms or less.

Agility. The ideal platform would also be agile, meaning it would have a fast ramp rate. The ramp rate is how quickly a node can transition between an active state and an idle low-power sleep states; we measure this as well.

Power. To measure power consumption, we used a power meter to collect and sum the instantaneous power readings once a second throughout the duration of each test.

Performance. To measure performance, we use a PHP application that performs some CPU-intensive task on each request. In a full production system, this script would be replaced by the access of a back-end database server. However, we use this simple replacement for node characterization so that there is no possibility of the database server being a bottleneck.

Entire Cluster

To evaluate the cluster as a whole, we use two different workloads. For the first, we design a request trace that emulates the trace used in the simulator, so we can verify that our simulator matches the behavior of the cluster and vice-versa. This trace runs against the simple CPU-intensive PHP application. The second workload we use is subsampled from a real two-day Wikipedia trace and runs against MediaWiki. We compress the timescale down to approximately 20 minutes to obtain the workload shown in Figure 3.12. This subsampling is necessary because we are constrained by the number of actual nodes in our cluster; to run the full trace we would require approximately 200 Atoms.

In order to generate enough traffic to saturate the cluster, we found it necessary to implement our own load generation software. Many existing load generators either depend upon the `select()` call which does not fare well with a large number of open sockets or use a distributed load generation framework. The distribution of load generation resulted in time synchronization issues which we found to be an issue when attempting to replay traces.

Our software instead uses the efficient `epoll/libevent` mechanism for generation and handling of network requests.

To our knowledge, at the time of our experiments, no commodity servers supported S-states, which we use in our design for putting nodes to sleep. In particular, the Nehalem servers do not support S-states. Thus, our first working implementation uses mostly Atoms.

Lastly, to address the LVS bug mentioned in Section 3.2 we use a temporary workaround that requires reloading the LVS kernel modules. This results in a few anomalous data points but these are quite easy to detect and explain, as will be shown in the next section.

3.4 Results

In this section, we evaluate our design using a combination of simulation and experiments on our prototype cluster. We begin by examining the nodes from which our prototype is constructed. We then evaluate the prediction and provisioning algorithms used by the cluster manager. Lastly, we test the operation of the design as a whole, first on the simulator, then validating these results on our real cluster.

3.4.1 Hardware

In the server class, we evaluate a high-end machine with a Nehalem architecture quad-core Intel Xeon X5550. This server has high power consumption, with a maximum power use of 248W and idle power use of 149W. Like nearly all commercially available servers, it has no support for low-power S-states, so its only available low-power state is when it is powered off. The time it takes the machine to power back on and resume all necessary services is approximately 60 seconds.

The mobile class platform is a dual-core Intel Atom 330, which supports both suspend-to-

	Peak Power	Idle Power	Power in S3	Wake Time from S3	Wake Time from Off	Max Request Rate	Min Energy per Response
Server	248 W	149 W	-	-	60s	310 req/s	0.6 J/resp
Mobile	28 W	22 W	1W	2.4s	40s	28 req/s	1.0 J/resp
Embedded	3.4 W	3.4 W	-	-	33s	3 req/s	1.1 J/resp

Table 3.1. We evaluate each of three different classes of hardware platforms: server, embedded, and mobile. We measured peak, idle, and low-power sleep state (S3) power consumption, the time it takes for each to transition into and out of this low power state, the maximum rate of requests each can sustain before latency begins to grow unbounded, and the minimum number of Joules required to service a request.

RAM (S3) and suspend-to-disk sleep states, can be brought out of S3 in 2.4 seconds, and can be powered on in 40s. This system uses 28W when fully utilized, 22W when idle, and 1W in S3. These low-power states and fast transition times make it agile and an asset to a cluster turning machines on and off.

Finally, the embedded system is an ARM-based Beagle Board, which only consumes 3.4 Watts, whether fully utilized or idling. While it does not support low-power S-states, the system's power usage could be further reduced by enabling power management, though we did not. The embedded system requires 33s to fully power on and resume services.

The results for each node are summarized in Table 3.1.

To test the efficiency and performance of each node, we send 1000 web requests to the server under test at a constant rate. The requests go to our CPU-intensive PHP script. We send requests at higher and higher rates until we find the saturation point, or knee, after which individual response times increase exponentially.

Figures 3.4, 3.5, and 3.6, show efficiency along with response time as load increases. These figures allow us to identify the ideal operating range for each node, which we provide as parameters to our simulator and cluster manager. In the ideal operating range, also highlighted in the figures, nodes achieve both low Joules per response and low latency. At levels of load below this ideal range, efficiency is poor. Requests arrive sparsely and thus

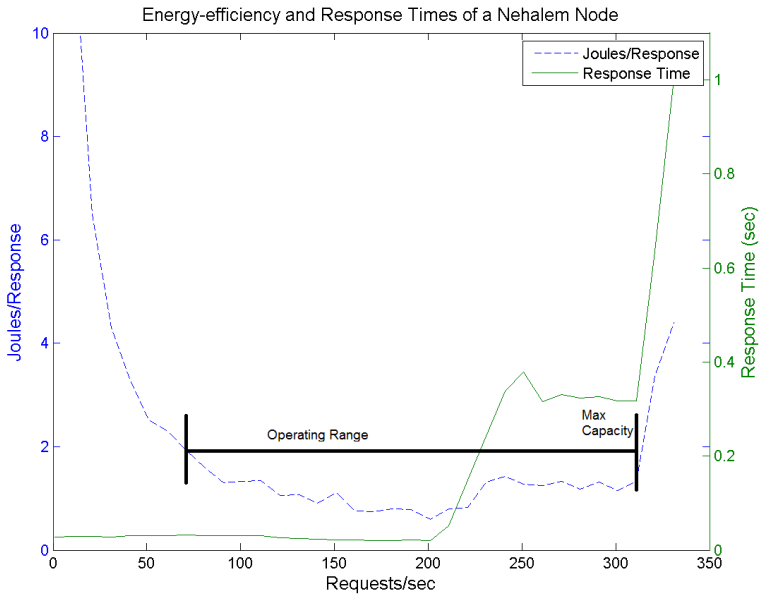


Figure 3.4. Server class hardware platform. We measured the energy efficiency and response times at increasing request rates for a Nehalem server. We highlight the ideal operating range (in terms of request load sustained) where the platform achieves both request latencies and low Joules per request, corresponding to better efficiency. The operating range here is approximately 75 to 310 requests per second.

are all easily handled by the under-utilized CPU. As load increases, requests can still be serviced in a reasonable time, leading to higher efficiency. But at high levels of load, the processor becomes saturated and response times spike dramatically. This saturation point thus marks the uppermost bound on the ideal operating range.

The high-end server has an operating range of 75 to 310 requests per second. The mobile node has an operating range of 7 to 28 requests per second. The embedded node has a range of 1 to 3 requests per second. Interestingly, despite very different processing capacities, the three different types of nodes exhibit comparable energy efficiency within this operating range, consuming around 1 Joule per response.

While the traditional operating set point is 50% of a node's maximum capacity, these results

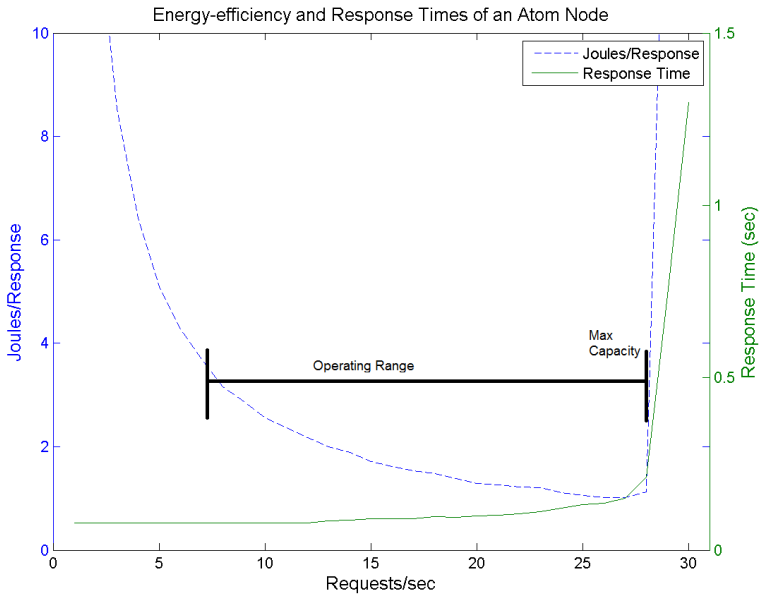


Figure 3.5. Mobile class hardware platform. We measured the energy efficiency and response times at increasing request rates for an Atom-based board. We highlight the ideal operating range (in terms of request load sustained) where the platform achieves both request latencies and low Joules per request, corresponding to better efficiency. The operating range here is approximately 7 to 28 requests per second.

suggest that in practice, this set point is tunable over a wide range of rates. The operating range creates a guard band which allows deviations from the set point while still maintaining good response times and energy efficiency. Operating at the lower end of the ideal range can lead to over provisioning since machines are treated as if they can handle only a small fraction of their actual maximum capacity. This route is the most conservative, with a moderate power penalty and greater guarantees against high latency. Operating at higher ends of the ideal range lead to less over provisioning and greater efficiency, but incurs a higher risk of service degradation. If the operating set point is too aggressively, the computing resources may actually be under provisioned, causing violations when unexpected spikes in the workload force machines beyond their maximum request capacity. Large increases in response times can also causes a drop in energy efficiency, so it is critical to avoid over

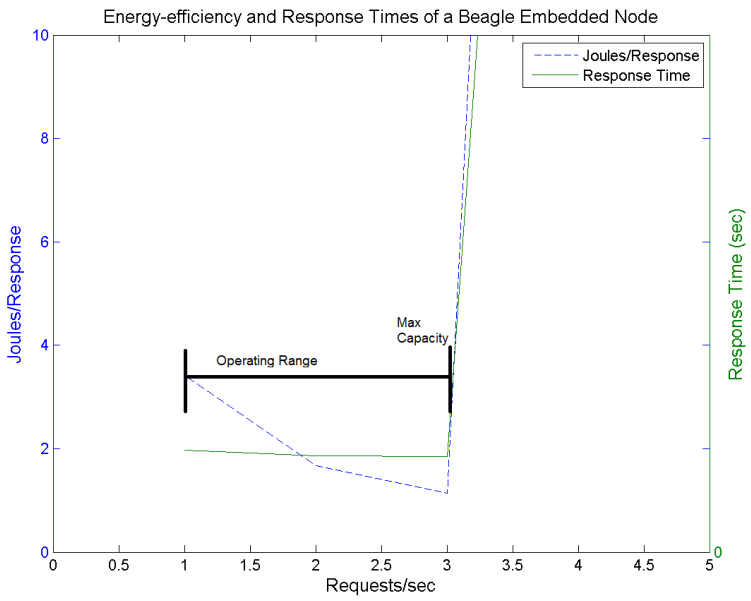


Figure 3.6. Embedded class hardware platform. We measured the energy efficiency and response times at increasing request rates for a BeagleBoard. We highlight the ideal operating range (in terms of request load sustained) where the platform achieves both request latencies and low Joules per request, corresponding to better efficiency. The operating range here is approximately 1 to 3 requests per second.

saturating the system. These results demonstrate that the choosing a proper operating set point turns out to be a crucial step in having an energy efficient system.

3.4.2 Rate Prediction

Next, we evaluate the three rate predictors mentioned in Section 3.1. We also briefly considered a predictor that extrapolates the current rate of change of the request rate, but found that it performs very poorly because it is extremely sensitive to transient spikes. For each predictor, we evaluate its accuracy using a week-long Wikipedia trace. At each time step t in the trace we use a predictor to predict the value n seconds in the future; we compare this to the actual rate at time $t + n$. We show the root mean square error of these predictions

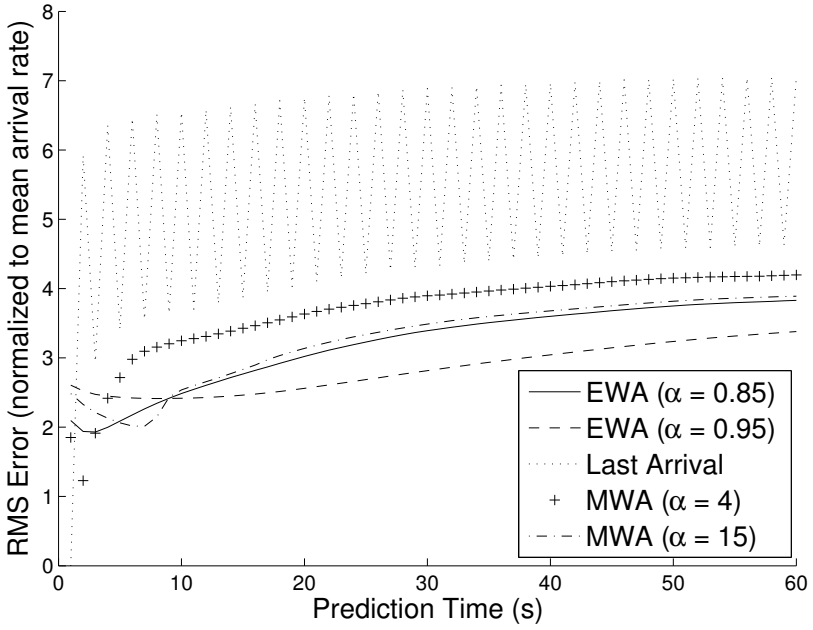


Figure 3.7. We compare rate predictors based on the root mean squared prediction error over a one-week request trace. For our predictors, we try three different kinds of moving window averages, with varying α values, which control how much history the predictor takes into account. These prediction algorithms are given in detail in Section 3.1. An exponentially weighted average with high values of α performs best.

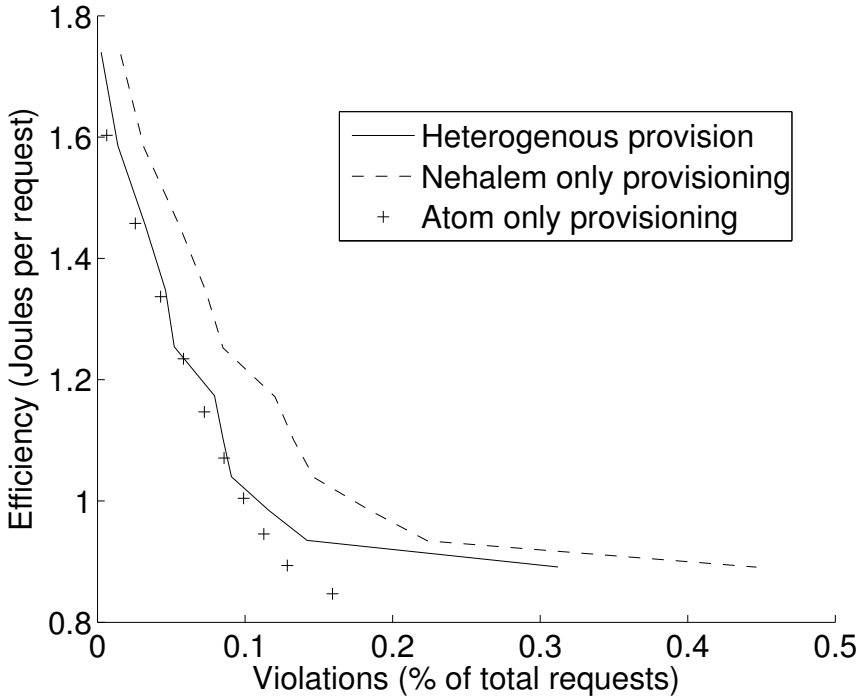


Figure 3.8. We run a week-long Wikipedia trace-driven workload against NapSAC , operating on a heterogenous cluster consisting of server- and mobile-class nodes, as well as over homogenous clusters consisting of either type individually. We measure both efficiency in terms of Joules per request as well as performance in terms of the number of violations, or responses returned in excess of 300 ms. The agile, mobile-class cluster performs the best in both dimensions, showing that using smaller, agile nodes helps, despite their lower processing capacity.

normalized by the mean request rate in Figure 3.7. These values can be interpreted as a measure of the typical error of the predictor as a percent of the average request rate. We also consider different α values for the predictors. We see that all averaging algorithms perform similarly well, with errors of less than 4% of the mean. Using more history, a larger α value, improves the prediction. Overall, the exponential weighted average with an α value of 0.95 is found to perform best for long prediction windows.

3.4.3 Provisioning Algorithm

We evaluate our provisioning algorithm using two metrics: energy efficiency and number of violations. The efficiency is a measure of how well NapSAC is saving energy. It is measured in Joules per request; lower numbers indicate more energy savings. Violations are used as a proxy measure of performance impact. They indicate that server capacity has been temporarily exceeded and some of the requests may incur excess latency due to queuing. Violations occur when a sudden burst of requests exceed the maximum capacity of the cluster nodes. Hence there is a trade-off between the aggressiveness of the power management and the number of violations that occur. We tune the aggressiveness of our provisioning algorithm by setting the operating point of each node to a fraction of the maximum capacity. This over provisioning ensures that some bursts can be absorbed without impacting performance.

We experiment with both a mix of heterogeneous platforms rather as well as a single type of node. We simulate the operation of NapSAC over each type of cluster over a week-long Wikipedia trace. Figure 3.8 compares a mixed cluster of Atom and Nehalem machines to a homogeneous cluster of either one individually, using the provisioning algorithms described above with a 1 second provisioning interval. For each configuration we use a range of over provisioning from 100% to 0%, shown along the length of the curve. Each over provisioning setting gives us an efficiency and a number of violations over the request trace. 100% over provisioning is shown in the upper left, indicating poor efficiency but no violations and 0% over provisioning occurs in the lower right. For 0.1% violations the Nehalem only cluster requires substantially more over provisioning and has worse efficiency than the heterogeneous and atom only clusters. Moreover, as we decrease the over provisioning, the efficiency of the Nehalem cluster does not significantly improve, but the number of violations increases. This occurs because the less agile Nehalem nodes cannot keep up with

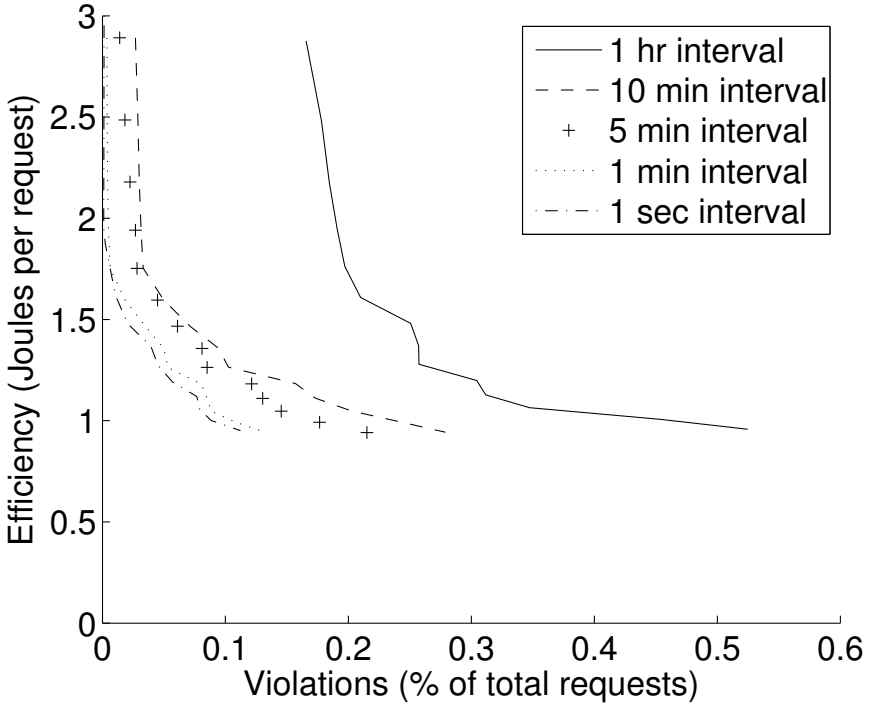


Figure 3.9. We run NapSAC on a cluster of mobile-class nodes while experimenting with different values for the provisioning window (`pwnd`). We measure both efficiency in terms of Joules per request as well as performance in terms of the number of violations, or responses returned in excess of 300 ms. Smaller provisioning windows are better.

higher frequency load spikes. Overall, we see that using the more agile Atom nodes, alone or in combination with the Nehalems leads to better efficiency and fewer violations. For all subsequent comparisons we simulate a heterogeneous cluster.

3.4.4 Provisioning Interval

The provisioning algorithm is executed at regular intervals of size `pwnd` to reconfigure the cluster. Figure 3.9 shows the effects of running at different provisioning intervals. We consider a range of over provisioning factors from 10% to 230%, shown along the length of the curve. Provisioning intervals under 10 minutes all perform comparably and much better

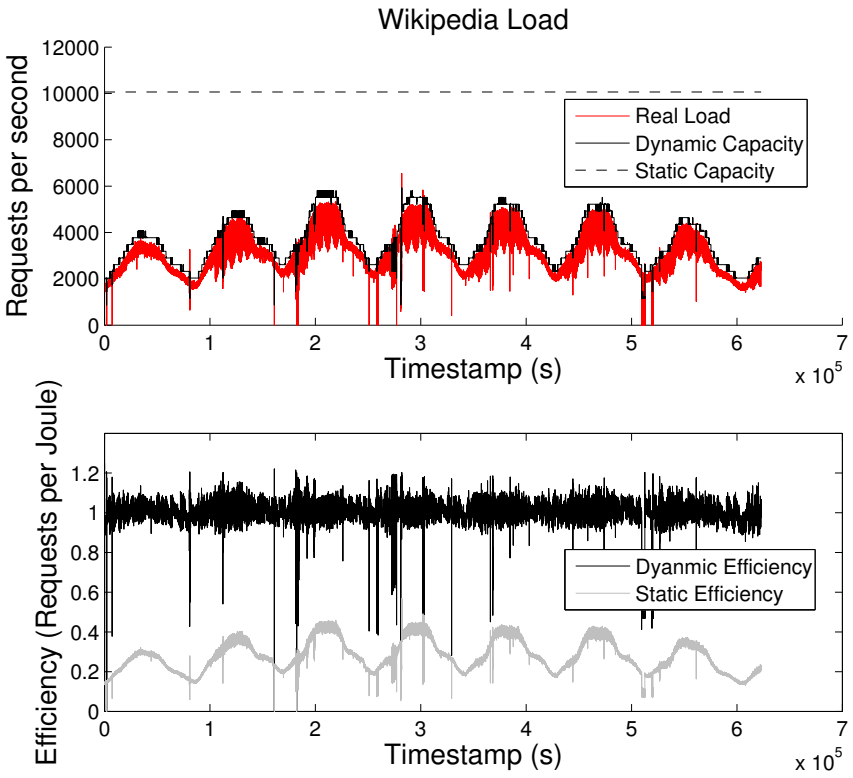


Figure 3.10. We simulate running a week-long Wikipedia trace-driven workload against our NapSAC cluster using the best settings from previous experiments. We compare this against a static provisioning scheme. We measure the request rate, provisioned capacity, power consumption, efficiency, and violations over the duration of the experiment. NapSAC efficiency is relatively constant and does not vary with request rate, meaning the cluster is indeed power proportional. Further, we witness relatively few violations.

than a `pwnd` of one hour. Higher provisioning rates lead to lower violations and less over provisioning.

3.4.5 Efficiency and Performance

We now use our simulator to compare our cluster manager to (1) a baseline static provisioning scheme and (2) an oracle scheme. For the static scheme, the cluster is sized using the rule-of-thumb of provisioning for twice the peak load observed in the trace. For the

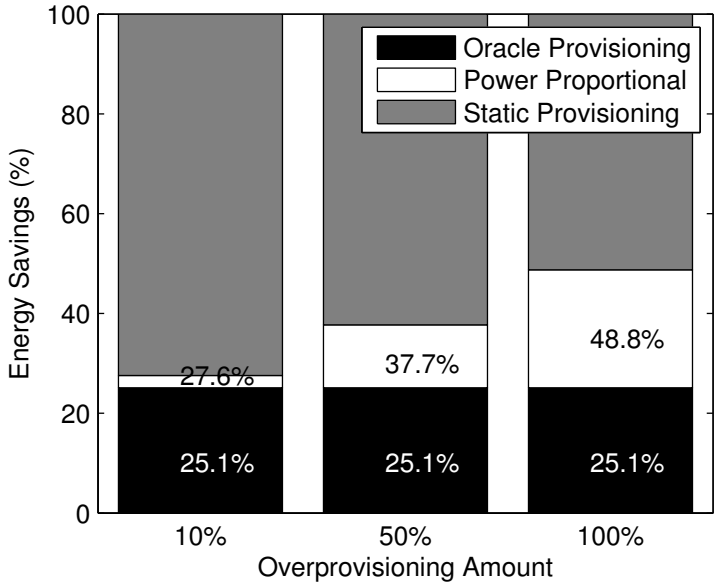


Figure 3.11. We simulate running a week-long Wikipedia trace-driven workload against our NapSAC cluster using the best settings from previous experiments. We compare this against a static provisioning scheme and an oracle scheme which has perfect knowledge of the future. We measure the amount of energy each scheme saves for varying amounts of over provisioning. NapSAC comes to within 90 percent of the savings attained by the oracle scheme with only 10 percent over provisioning, vastly out-performing traditional, static provisioning.

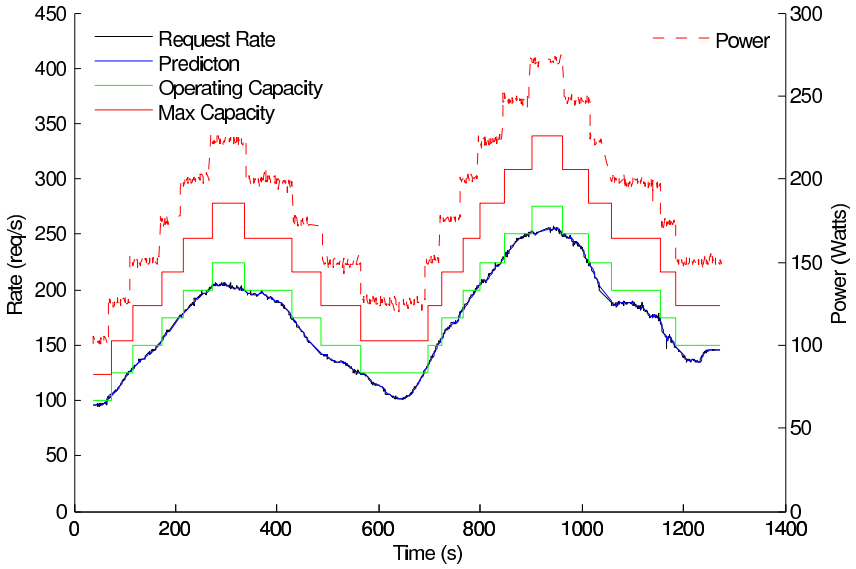
oracle scheme, we assume an omnipresent cluster manager that can see the future and keep the minimum number of nodes needed to handle the load active at any time. We configure NapSAC using the best performing settings from the previous experiments. For rate prediction, use EWA with $\alpha = .95$. We also use a heterogeneous cluster and a provisioning interval of 1 second.

We simulate this system running on a two-week production trace from Wikipedia. Figure 3.10 shows how the cluster is provisioned over time, along with the energy efficiency achieved. The efficiency of our dynamic algorithm is fairly constant and generally independent of of the workload. Figure 3.11 summarizes our energy savings at different levels of provisioning. NapSAC consistently saves over half of the energy consumed by the baseline static provisioning scheme. At ten percent provisioning, we have only 0.15 percent violations and we achieve more than 70 percent power savings. Moreover, we are within 90 percent of the oracle.

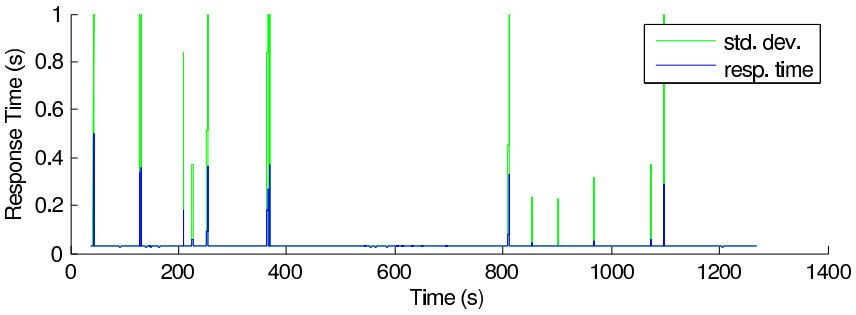
3.4.6 Validation

Lastly, we evaluate NapSAC using our prototype. We run our subsampled, two-day Wikipedia trace against our cluster and measure the resulting efficiency as well as violations.

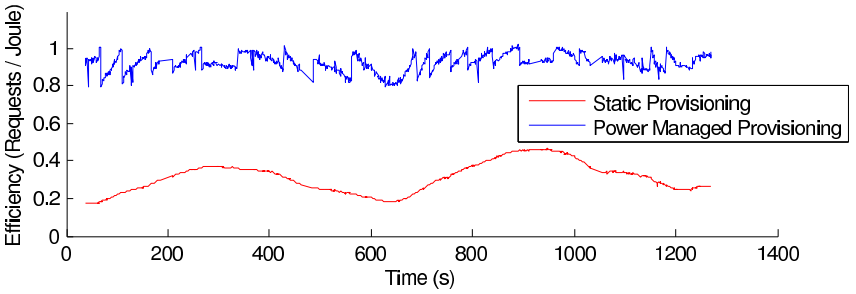
The results of this experiment are shown in Figure 3.12. NapSAC is again able to provision to closely follow the load to maintain efficiency without any violations. Response times remain well under 200 ms, discounting the handful of anomalous cases due to the LVS reload bug. Moreover, the efficiency attained by the NapSAC cluster comes close to the maximum achievable efficiency of the underlying nodes. This demonstrates NapSAC can achieve power proportionality in a real-world setting.



(a) Workload tracking



(b) Response times



(c) Efficiency

Figure 3.12. We ran a two-day Wikipedia trace-driven workload against our real, NapSAC power proportional cluster, measuring request rate, provisioned capacity, power consumption, violations, and efficiency. We see that these results closely match those of the simulator. Efficiency remains relatively constant throughout the experiment, showing that our cluster is indeed power proportional. Moreover, it achieves this without incurring any performance violations.

Chapter 4

Parallel Computation Framework Study

In this chapter, we present BEEMR, an energy efficient MapReduce system motivated by the real-life Facebook workload described in Chapter 2. In that chapter, we provided an analysis of a Facebook cluster trace to quantify the empirical behavior of a MIA workload. In this chapter, we will see how that analysis factors into our design. BEEMR combines novel ideas based this empirical data with existing MapReduce energy efficiency mechanisms. We detail the BEEMR architecture in Section 4.1. We also developed an improved evaluation methodology to quantify energy savings and account the complexity of MIA workloads. This methodology is described in Section 4.3 and the implementation Section 4.2. We evaluate the various parameters of our system and present energy efficiency results in Section 4.4. In particular, we examine the effect of cluster size, batch interval length, per-job task slots, map to reduce ratio, and interruptible threshold value in Sections 4.4.1 through 4.4.5. We then evaluate the overhead and sensitivity of BEEMR in Sections 4.4.6 and 4.4.7. Lastly, we validate that the results of from our simulation can be

obtained in real-life using EC2. These results are presented in 4.4.8. With MIA workloads run on BEEMR, we show energy savings of 40-50%. This represents a step towards both highlighting and improving the energy efficiency of a demanding, real-world use case for MapReduce.

4.1 BEEMR Architecture

BEEMR is an energy efficient MapReduce workload manager. The key insight is that the interactive jobs can be served by a small pool of dedicated machines with their associated storage, while the less time-sensitive jobs can run in a batch fashion on the rest of the cluster using full computation bandwidth and storage capacity. This setup leads to energy savings and meet all the requirements listed in Table 2.2.

The BEEMR cluster architecture is shown in Figure 4.1. It is similar to a typical Hadoop MapReduce cluster, with important differences in how resources are allocated to jobs.

The cluster is split into disjoint interactive and batch zones. The interactive zone makes up a small, fixed percentage of cluster resources — task slots, memory, disk capacity, network bandwidth, similar to the design in [5]. The interactive zone is always fully powered. The batch zone makes up the rest of the cluster, and is put into a very low power state between batches [35].

As jobs arrive, BEEMR classifies them as one of three job types. Classification is based on empirical parameters derived from the analysis in Section 2.2. If the job input data size is less than some threshold `interactive`, it is classified as an interactive job. BEEMR seeks to service these jobs with low latency. If a job has tasks with task duration longer than some threshold `interruptible`, it is classified as an interruptible job. Latency is not a concern for these jobs, because their long-running tasks can be check-pointed and re-

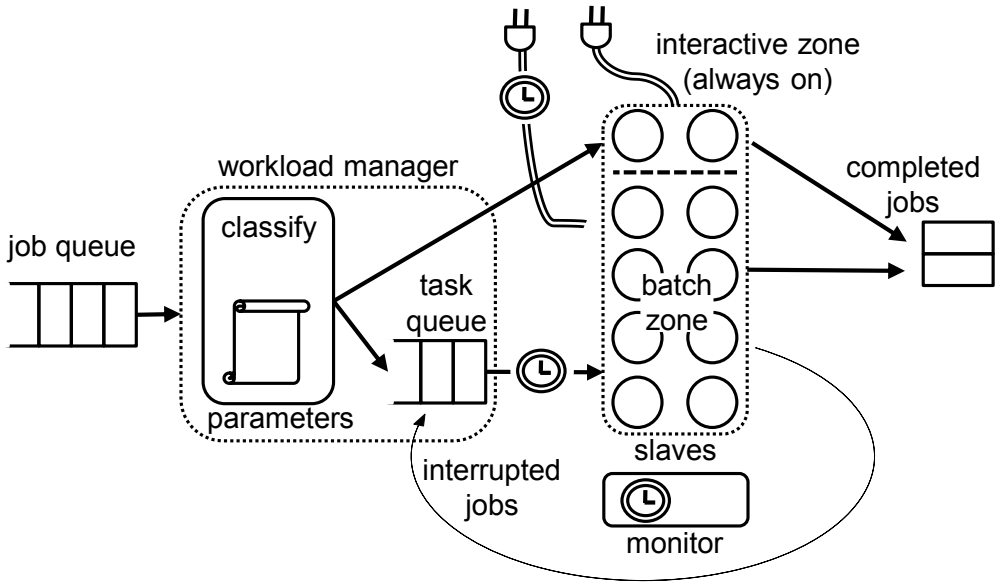


Figure 4.1. The BEEMR workload manager (i.e., job tracker) classifies each job into one of three classes which determines which cluster zone will service the job. Interactive jobs are serviced in the interactive zone, while batchable and interruptible jobs are serviced in the batch zone. Energy savings come from aggregating jobs in the batch zone to achieve high utilization, executing them in regular batches, and then transitioning machines in the batch zone to a low-power state when the batch completes.

sumed over multiple batches. All other jobs are classified as batch jobs. Latency is also not a concern for these jobs, but BEEMR makes best effort to run them by regular deadlines. Such a setup is equivalent to deadline-based policies where the deadlines are the same length as the batch intervals.

The interactive zone is always in a full-power ready state. It runs all of the interactive jobs and holds all of their associated input, shuffle, and output data (both local and HDFS storage). Figures 2.4 and 2.5 indicate that choosing an appropriate value for `interactive` can allow most jobs to be classified as interactive and executed without any delay introduced by BEEMR. This `interactive` threshold should be periodically adjusted as workloads evolve.

The interactive zone acts like a data cache. When an interactive job accesses data that is not in the interactive zone (i.e., a cache miss), BEEMR migrates the relevant data from the batch zone to the interactive zone, either immediately or upon the next batch. Since most jobs use small data sets that are reaccessed frequently, cache misses occur infrequently. Also, BEEMR requires storing the ECC parity or replicated blocks within the respective zones, e.g., for data in the interactive zone, their parity or replication blocks would be stored in the interactive zone also.

Upon submission of batched and interruptible jobs, all tasks associated with the job are put in a wait queue. At regular intervals, the workload manager initiates a batch, powers on all machines in the batch zone, and run all tasks on the wait queue using the whole cluster. The machines in the interactive zone are also available for batch and interruptible jobs, but interactive jobs retain priority there. After a batch begins, any batch and interruptible jobs that arrive would wait for the next batch. Once all batch jobs complete, the job tracker assigns no further tasks. Active tasks from interruptible jobs are suspended, and enqueued to be resumed in the next batch. Machines in the batch zone return to a low-power state. If a batch does not complete by start of the next batch interval, the cluster would remain fully powered for consecutive batch periods. The high peak-to-average load in Figure 2.3 indicates that on average, the batch zone would spend considerable periods in a low-power state.

BEEMR improves over prior batching and zoning schemes by combining both, and uses empirical observations to set the values of policy parameters, which we describe next.

4.1.1 Parameter Space

BEEMR involves several design parameters whose values need to be optimized. These parameters are:

Parameter	Units or Type	Values
`totalsize`	thousand slots	32, 48, 60, 72
`mapreduceratio`	map:reduce slots	$1:1$, $27:14$, (≈ 2.0), $13:5$ (≈ 2.6)
`izonesize`	% total slots	10
`interactive`	GB	10
`interruptible`	hours	6, 12, 24
`batchlen`	hours	1, 2, 6, 12, 24
`taskcalc`	algorithm	default, actual, latency-bound

Table 4.1. Design space explored. The values for `izonesize` and `interactive` are derived from the analysis in Section 2.2. We scan at least three values for each of the other parameters.

- `totalsize`: the size of the cluster in total (map and reduce) task slots.
- `mapreduceratio`: the ratio of map slots to reduce slots in the cluster.
- `izonesize`: the percentage of the cluster assigned to the interactive zone.
- `interactive`: the input size threshold for classifying jobs as interactive.
- `interruptible`: task duration threshold for classifying jobs as interruptible.
- `batchlen`: the batch interval length.
- `taskcalc`: the algorithm for determining the number of map and reduce tasks to assign to a job.

Table 4.1 shows the parameter values we will optimize for the Facebook workload. For other workloads, the same tuning process can extract a different set of values. Note that `totalsize` indicates the size of the cluster in units of task slots, which differs from the number machines. One machine can run many task slots, and the appropriate assignment of task slots per machine depends on hardware capabilities.

Another parameter worth further explanation is `taskcalc`, the algorithm for determining the number of map and reduce tasks to assign to a job. An algorithm that provides appropriate task granularity ensures that completion of a given batch is not held up by long-running tasks from some jobs.

BEEMR considers three algorithms: *Default* assigns 1 map per 128 MB of input and 1 reduce per 1 GB of input; this is the default setting in Hadoop. *Actual* assigns the same number of map and reduce tasks as given in the trace and corresponds to settings at Facebook. *Latency-bound* assigns a number of tasks such that no task will run for more than 1 hour. This policy is possible provided that task execution times can be predicted with high accuracy [29], [50].

4.1.2 Requirements Check

We verify that BEEMR meets the requirements in Table 2.2.

1. Write bandwidth is not diminished because the entire cluster is fully powered when batches execute. Table 2.1 indicates that only batch and interruptible jobs require large write bandwidth. When these jobs are running, they have access to all of the disks in the cluster.

2. Similarly, write capacity is not diminished because the entire cluster is fully powered on during batches. Between batches, the small output size of interactive jobs (Table 2.1) means that an appropriate value of `izonesize` allows those job outputs to fit in the interactive zone.

3. The size of available memory also remains intact. The memory of the entire cluster is accessible to batch and interruptible jobs, which potentially have large working sets. For interactive jobs, the default or actual (Facebook) `taskcalc` algorithms will assign few tasks per job, resulting in small in-memory working sets.

4. Interactive jobs are not delayed. The interactive zone is always fully powered, and designated specifically to service interactive jobs without delay.

5. BEEMR spreads data evenly within both zones, and makes no changes that impact

data locality. Nonetheless, Figure 2.4 suggests that there will be some hotspots inherent to the Facebook workload, independent of BEEMR.

6. BEEMR improves energy efficiency via batching. There is no dependence on ECC or replication, thus preserving energy savings regardless of fault tolerance mechanism.

7. Long jobs with low levels of parallelism remain a challenge, even under BEEMR. These jobs are classified as interruptible jobs if their task durations are large, and batch jobs otherwise. If such jobs are classified as batch jobs, they could potentially prevent batches from completing. Their inherent low levels of parallelism cause the batch zone to be poorly utilized when running only these long jobs, resulting in wasted energy. One solution is for experts to label such jobs a priori so that BEEMR can ensure that these jobs are classified as interruptible.

4.2 Implementation

BEEMR involves several extensions to Apache Hadoop.

The job tracker is extended with a wait queue management module. This module holds all incoming batch jobs, moves jobs from the wait queue to the standard scheduler upon each batch start, and places any remaining tasks of interruptible jobs back on the wait queue when batches end. Also, the scheduler's task placement mechanism is modified such that interactive jobs are placed in the interactive zone, and always have first priority to any available slots.

The namenode is modified such that the output of interactive jobs is assigned to the interactive zone, and the output of batch and interruptible jobs is assigned to the batch zone. If either zone approaches storage capacity, it must adjust the fraction of machines in each zone, or expand the cluster.

The Hadoop master is augmented with a mechanism to transfer all slaves in the batch zone in and out of a low-power state, e.g., sending a "hibernate" command via `ssh` and using Wake-on-LAN or related technologies [42]. If batch intervals are on the order of hours, it is acceptable for this transition to complete over seconds or even minutes.

Accommodating interruptible jobs requires a mechanism that can suspend and resume active tasks. The current Hadoop architecture makes it difficult to implement such a mechanism. However, suspend and resume is a key component of fault recovery under Next Generation Hadoop [52]. We can re-purpose for BEEMR those mechanisms.

These extensions will create additional computation and IO at the Hadoop master node. The current Hadoop master has been identified as a scalability bottleneck [64]. Thus, it is important to monitor BEEMR overhead at the Hadoop master to ensure that we do not affect cluster scalability. This overhead would become more acceptable under Next Generation Hadoop, where the Hadoop master functionality would be spread across several machines [52].

4.3 Evaluation Methodology

The evaluation of our proposed algorithm involves running the Facebook MIA workload both in simulation and on clusters of hundreds of machines on Amazon EC2 [1].

The Facebook workload provides a level of validation not obtainable through stand-alone programs or artificial benchmarks. It is logistically impossible to replay this workload on large clusters at full duration and scale. The high peak to average nature of the workload means that at time scales of less than weeks, there is no way to know whether the results capture transient or average behavior. Enumerating a multi-dimensional design space would also take prohibitively long. Any gradient ascent algorithms are not possible, simply

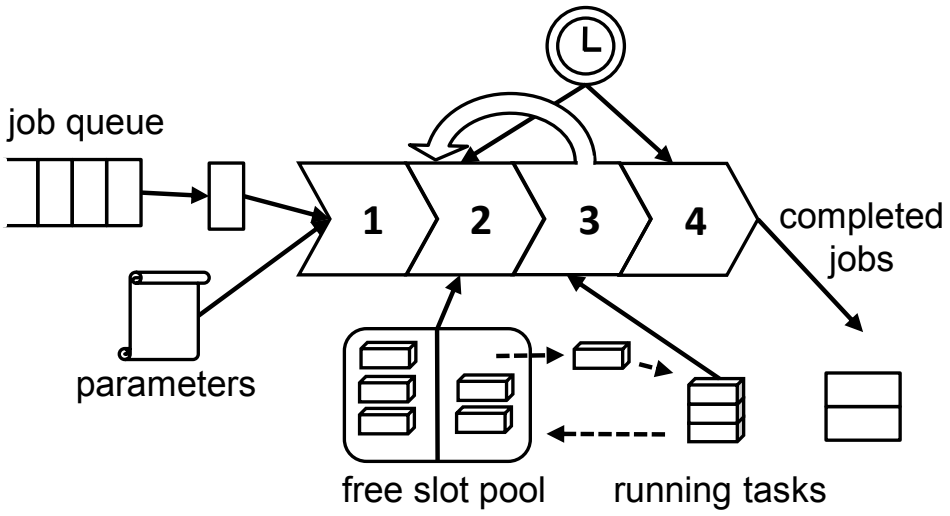


Figure 4.2. A high-level view of the simulation algorithm. For each simulated second, the following executes: 1. The simulator dequeues newly arrived jobs (arrival pattern given in the trace), classifies the job as interactive, batch, or interruptible, and applies the task granularity policy. 2. The simulator checks for available map or reduce slots, checks the batch policy to see which jobs can be run at the present time, and assigns slots to jobs in round robin, fair scheduler fashion. 3. The simulator removes completed tasks and returns the corresponding slot back to the free slot pool. For each active job, it checks to see if the job has more tasks to run (go back to step 2) or is complete (go to step 4). 4. The job is marked complete and the job duration recorded.

because there is no guarantee that the performance behavior is convex. Combined, these concerns compel us to use experimentally validated simulations.

The simulator is optimized for simulation scale and speed by omitting certain details: job startup and completion overhead, overlapping map and reduce phases, speculative execution and stragglers, data locality, and interference between jobs. This differs from existing MapReduce simulators [51], [71], whose focus on details make it logistically infeasible to simulate large scale, long duration workloads. The simulator assumes a simple, fluid-flow model of job execution, first developed for network simulations as an alternative to packet-level models [43], [58]. There, the motivation was also to gain simulation scale and speed. Section 4.4.8 demonstrates that the impact on accuracy is acceptable.

Simulated job execution is a function of job submit time (given in the trace), task assignment time (depends on a combination of parameters, including batch length, and number of map and reduce slots), map and reduce execution times (given in the trace), and the number of mappers and reducers chosen by BEEMR (a parameter). Figure 4.2 shows how the simulator works at a high level.

We empirically validate the simulator by replaying several day-long workloads on a real-life cluster (Section 4.4.8). This builds confidence that simulation results translate to real clusters. The validation employs previously developed methods to "replay" MapReduce workloads independent of hardware [18]. The techniques there replays the workload using synthetic data sets, and reproduces job submission sequences and intensities, as well as the data ratios between each job's input, shuffle, and output stages.

We model the machines as having "full" power when active, and negligible power when in a low power state. Despite recent advances in power proportionality [6], such models remain valid for Hadoop. In [17], we used wall plug power meters to show that machines with power ranges of 150W-250W draw 205W-225W when running Hadoop. The chattiness of the Hadoop/HDFS stack means that machines are active at the hardware level even when they are idle at the Hadoop workload level. The simple power model allow us to scale the experiments in size and in time.

Several performance metrics are relevant to energy efficient MapReduce: (1) Energy savings: Under our power model, this would be the duration for which the cluster is fully idle; (2) Job latency (analogous to "turn around time" in multiprogramming literature [27]): We measure separately the job latency for each job class, and quantify any tradeoff against energy savings; (3) System throughput: Under the MIA open-loop workload model, the historical system throughput would be the smaller of `totalsize` and the historical workload

arrival rate. We examine several values of `totalsize` and quantify the interplay between latency, energy savings, and other policy parameters.

Table 4.1 shows the parameter values used to explore the BEEMR design space.

4.4 Results

The evaluation spans the multi-dimensional design space in Table 4.1. Each dimension illustrates subtle interactions between BEEMR and the Facebook workload.

4.4.1 Cluster Size

Cluster size is controlled by `totalsize`. Under provisioning a cluster results in long queues and high latency during workload peaks; over provisioning leads to arbitrarily high baseline energy consumption and waste. Over the 45-days trace, the Facebook workload has an average load of 21029 map tasks and 7745 reduce tasks. Since the workload has a high peak-to-average ratio, we must provision significantly above the average. Figure 4.3 shows the detailed cluster behavior for several cluster sizes without any of the BEEMR improvements. We pick a one-to-one map-to-reduce-slot ratio because that is the default in Apache Hadoop, and thus forms a good baseline. A cluster with only 32000 total slots cannot service the historical rate, being pegged at maximum slot occupancy; larger sizes still see transient periods of maximum slot occupancy. A cluster with at least 36000 map slots (72000 total slots) is needed to avoid persistent long queues, so we use this as a baseline.

4.4.2 Batch Interval Length

Energy savings are enabled by batching jobs and transitioning the batch zone to a low-power state between batches. The ability to batch depends on the predominance of interactive

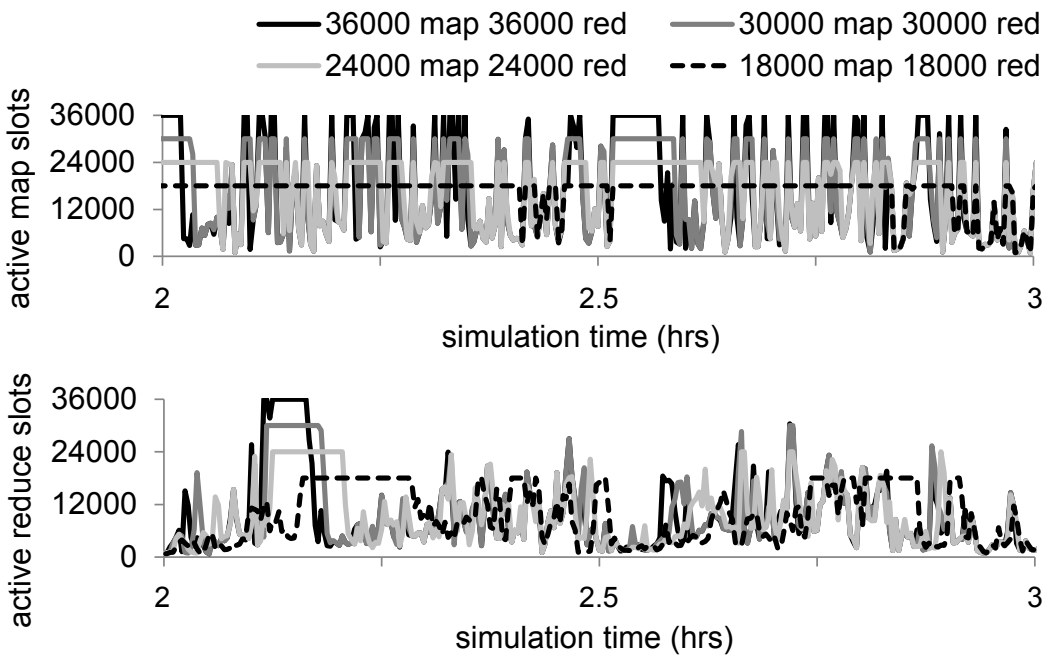


Figure 4.3. The number of concurrently active tasks for clusters of different sizes (in terms of total task slots, `totalsize`).

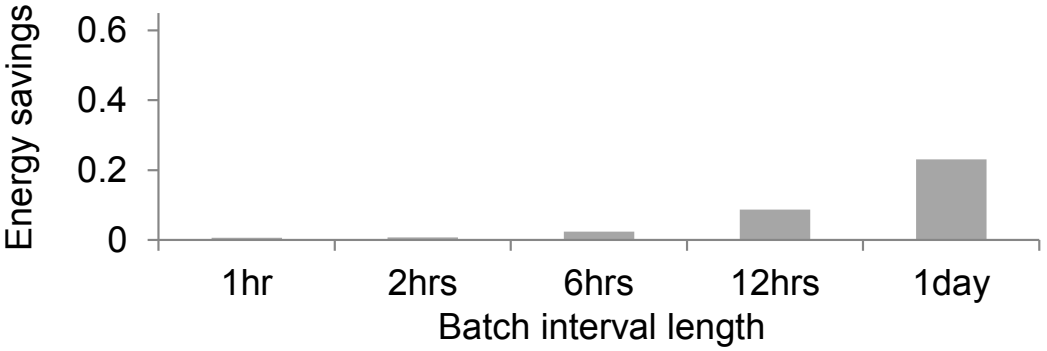


Figure 4.4. Energy savings for different batch interval lengths as given by `batchlen`. Energy savings are non-negligible for large batch intervals only. Note that `taskcalc` is set to default, `mapreduceratio` is set to 1:1, `totalsize` is set to 72000 slots, and `interruptible` is set to 24 hours.

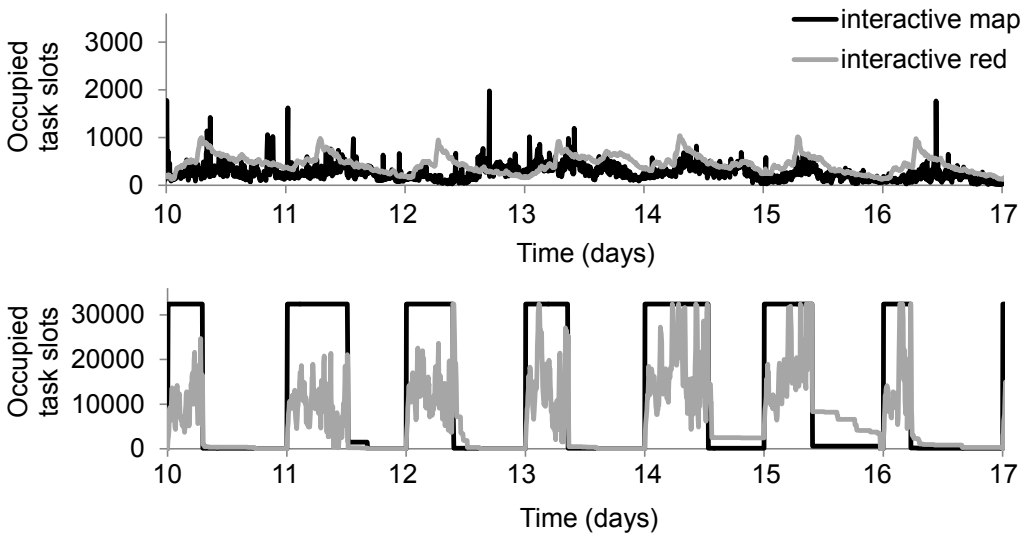


Figure 4.5. Active slots for a `batchlen` of 24 hours. Showing slot occupancy in the interactive zone (top) and in the batch zone (bottom). Showing one week's behavior. Note that `taskcalc` is set to default, `mapreduceratio` is set to 1:1, `totalsize` is set to 72000 slots, and `interruptible` is set to 24 hours.

analysis in MIA workloads (Section 2.2). We consider here several static batch interval lengths. A natural extension would be to have dynamically adjusted batch intervals to enable various deadline driven policies.

We vary `batchlen`, the batching interval, while holding the other parameters fixed. Figure 4.4 shows that energy savings, expressed as a fraction of the baseline energy consumption, become non-negligible only for batch lengths of 12 hours or more. Figure 4.5 shows that map tasks execute in near-ideal batch fashion, with maximum task slot occupancy for a fraction of the batch interval and no further tasks in the remainder of the interval. However, reduce slot occupancy rarely reaches full capacity, while "dangling" reduce tasks often run for a long time at very low cluster utilization. There are more reduce tasks slots available, but the algorithm for choosing the number of task slots limits the amount of parallelism. During the fifth and sixth days, such dangling tasks cause the batch zone to remain at full

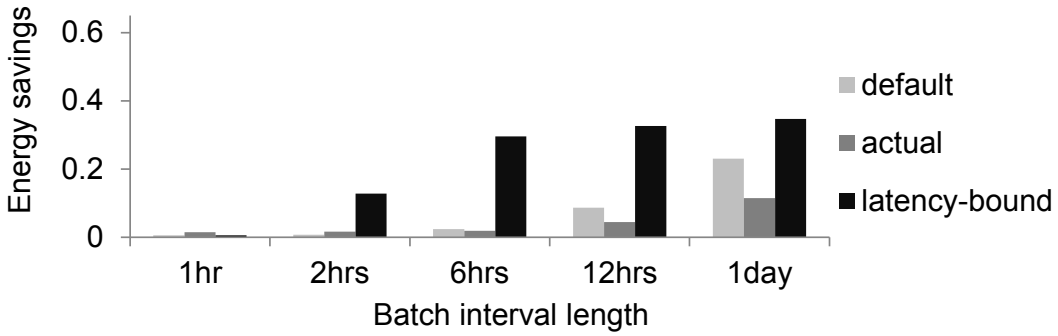


Figure 4.6. Energy savings for different `taskcalc` algorithms. Note that `mapreduceratio` is set to 1:1, and `interruptible` is set to 24 hours. The *actual* (Facebook) algorithm does worst and the *latency-bound* algorithm does best.

power for the entire batch interval. Fixing this requires improving both the algorithm for calculating the number of tasks for each job and the ratio of map-to-reduce slots.

4.4.3 Task Slots Per Job

The evaluation thus far considered only the default algorithm for computing the number of tasks per job, as specified by `taskcalc`. Recall that we consider two other algorithms: *Actual* assigns the same number of map and reduce tasks as given in the trace and corresponds to settings at Facebook. *Latency-bound* assigns a number of tasks such that no task will run for more than 1 hour. Figure 4.6 compares the default versus actual and latency-bound algorithms. The actual policy does the worst, unsurprising because the task assignment algorithm at Facebook is not yet optimized for energy efficiency. The latency-bound policy does the best; this indicates that good task execution time prediction can improve task assignment and achieve greater energy savings.

Observing task slot occupancy over time provides insight into the effects of `taskcalc`. Using the actual algorithm (Figure 4.7(a)), slots in the interactive zone reach capacity more frequently, suggesting that the Facebook algorithm seeks to increase parallelism to decrease

the amount of computation per task and lower the completion latency of interactive jobs. In contrast, tasks in the batch zone behave similarly under the default and Facebook algorithm for the week shown in Figure 4.7(a). Aggregated over the entire trace, the actual policy turns out to have more dangling tasks overall, diminishing energy savings.

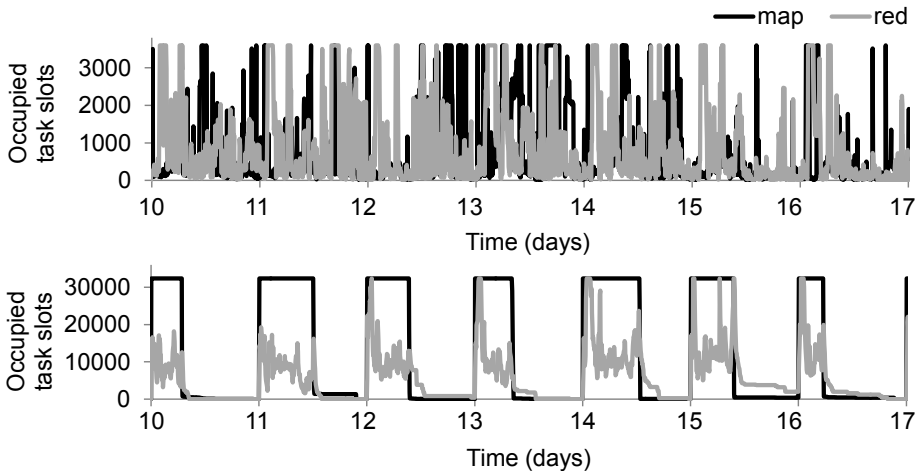
In contrast, task slot occupancy over time for the latency-bound policy eliminates all dangling tasks of long durations (Figure 4.7(b)). This results in high cluster utilization during batches, as well as clean batch completion, allowing the cluster to be transitioned into a low-power state at the end of a batch. There is still room for improvement in Figure 4.7(b): the active reduce slots are still far from reaching maximum task slot capacity. This suggests that even if we keep the total number of task slots constant, we can harness more energy savings by changing some reduce slots to map slots.

4.4.4 Map to Reduce Slot Ratio

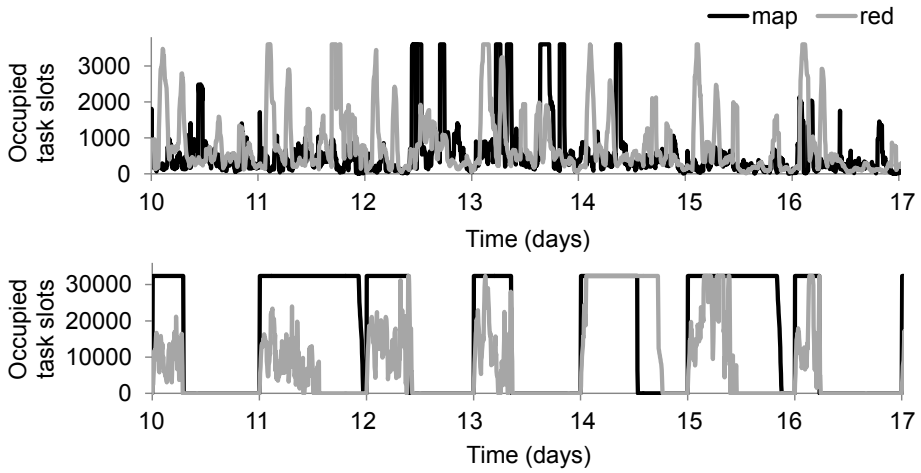
The evaluation thus far illustrates that reduce slots are utilized less than map slots. Changing `mapreduceratio` (i.e., increasing the number of map slots and decreasing the number of reduce slots while keeping cluster size constant) should allow map tasks in each batch to complete faster without affecting reduce tasks completion rates. Figure 4.8 shows that doing so leads to energy efficiency improvements, especially for the latency-bound algorithm.

Viewing the task slot occupancy over time reveals that this intuition about the map-to-reduce-slot ratio is correct. Figure 4.9(a) compares batch zone slot occupancy for two different ratios using the default algorithm. With a larger number of map slots, the periods of maximum map slot occupancy are shorter, but there are still dangling reduce tasks. The same ratio using the latency-bound algorithm avoids these dangling reduce tasks, as shown in Figure 4.9(b), achieving higher energy savings.

Nevertheless, the latency-bound algorithm still has room for improvement. During the fifth



(a) *Actual* (Facebook)



(b) *Latency-bound*

Figure 4.7. Slot occupancy over time in the interactive zone (top graph) and batch zone (bottom graph). Showing one week's behavior. Note that `batchlen` is set to 24 hours, `mapreduceratio` is set to 1:1, and `interruptible` is set to 24 hours.

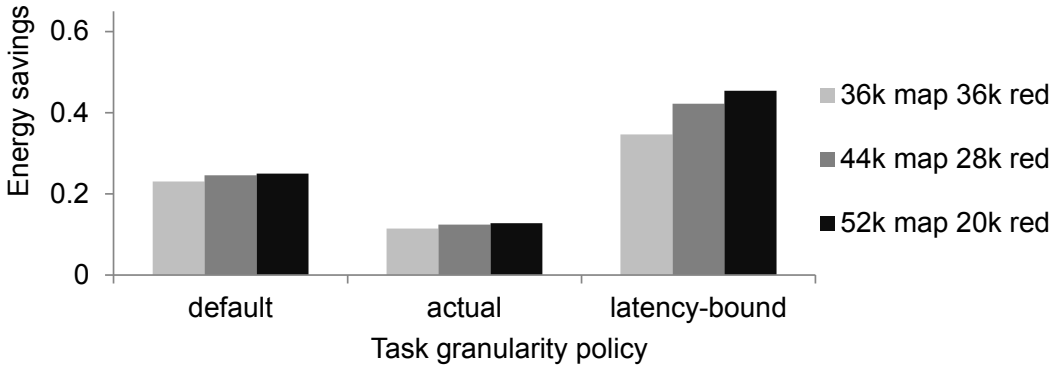


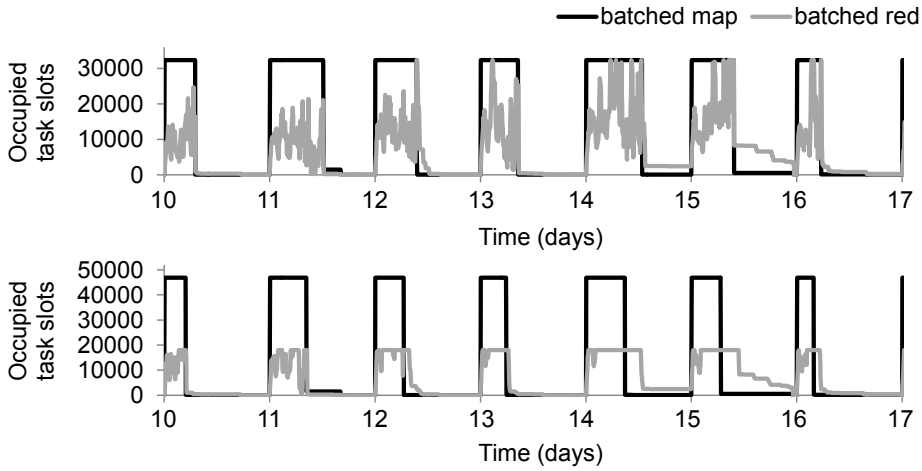
Figure 4.8. Energy savings for different values of `mapreduceratio`. Increasing the number of map slots increases energy savings for all `taskcalc` algorithms, with the improvement for *latency-bound* being the greatest. Note that `totalsize` is set to 72000 slots, `batchlen` is set to 24 hours, and `interruptible` is set to 24 hours.

and sixth days in Figure 4.9(b), the batches are in fact limited by available reduce slots. Figure 4.10 shows that neither static policy for map versus task ratios achieve the best savings for all days. A dynamically adjustable ratio of map and reduce slots is best. A dynamic ratio can ensure that every batch is optimally executed, bottlenecked on neither map slots nor reduce slots.

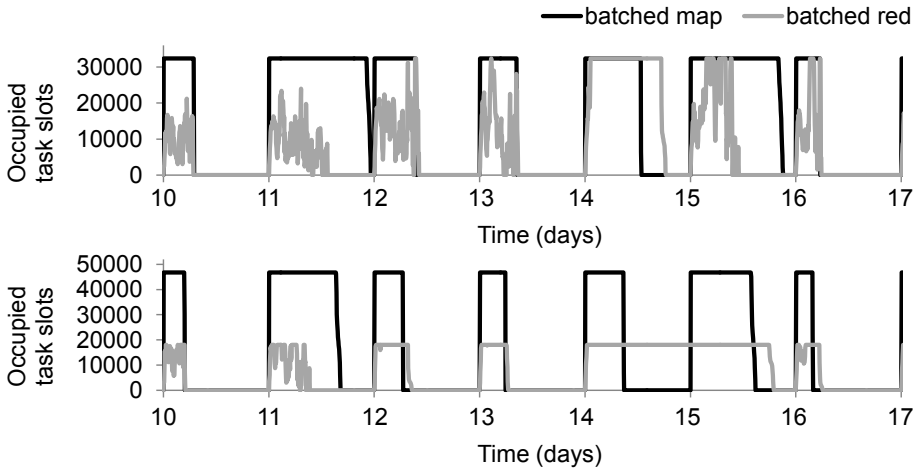
4.4.5 Interruptible Threshold

The last dimension to evaluate is `interruptible`, the task duration threshold that determines when a job is classified as interruptible. In the evaluation so far, `interruptible` has been set to 24 hours. Decreasing this threshold should cause more jobs to be classified as interruptible, and fewer jobs as batch. A lower interruptible threshold allows faster batch completions and potentially more capacity for the interactive zone, at the cost of higher average job latency, as more jobs are spread over multiple batches.

Figure 4.11 shows the energy saving improvements from lowering `interruptible`.



(a) *Default*



(b) *Latency-bound*

Figure 4.9. Batch zone slot occupancy over time using a `mapreduceratio` of 1:1 for the top graph, and a `mapreduceratio` of 13:5 for the bottom graph. Showing one week's behavior. Note that `batchlen` is set to 24 hours and `interruptible` is set to 24 hours.

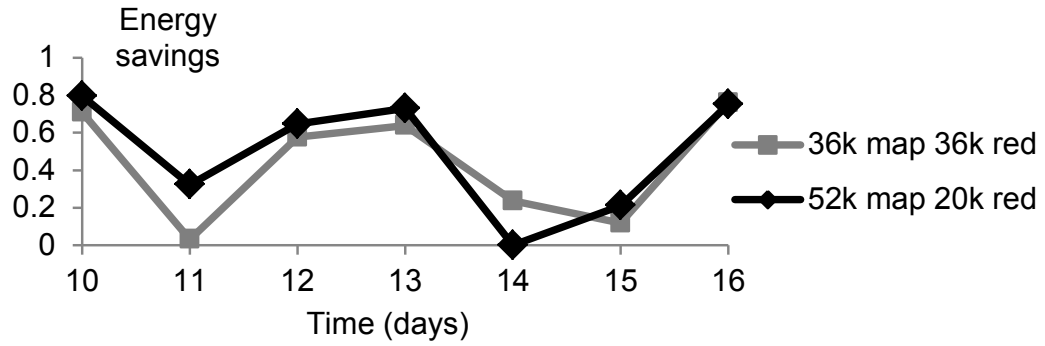


Figure 4.10. Energy savings per day for the latency-bound policy comparison in Figure 4.9(b). Daily energy savings range from 0 to 80%. Neither static policy achieves best energy savings for all days.

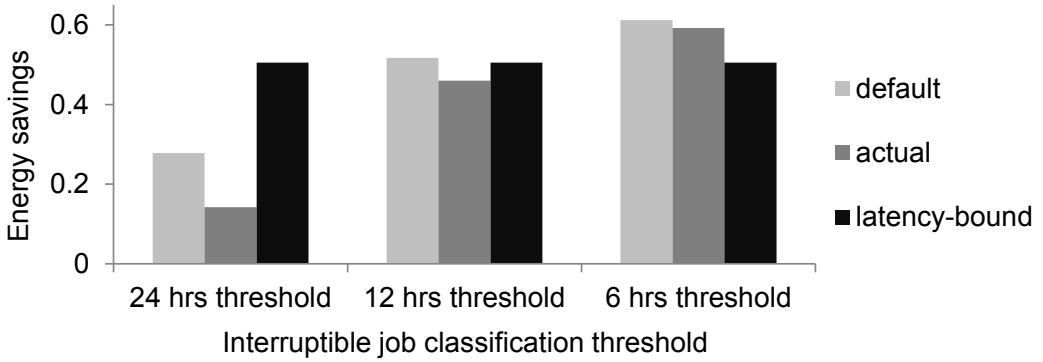


Figure 4.11. Energy savings for different values of `interruptible`. Lowering the threshold leads to increased energy savings for actual and default algorithms. Note that `mapreduceratio` is set to 13:5 and `batchlen` is set to 24 hours. Note that for actual and default algorithms, having a low `interruptible` causes the queue for waiting interrupted jobs to grow without limit; the latency-bound policy is preferred despite seemingly lower energy savings (Section 4.4.5).

(The latency-bound algorithm, by design, does not result in any interruptible jobs, unless the `interruptible` is set to less than an hour, so the energy savings for the latency-bound algorithm are unaffected.) Actual and default algorithms show considerable energy savings improvements, at the cost of longer latency for some jobs. It would be interesting to see how many cluster users and administrators are willing to make such trades.

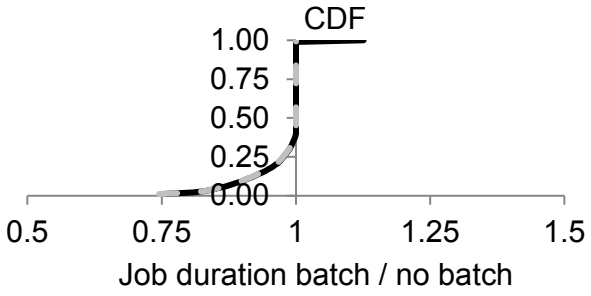
Lowering `interruptible` too much would cause the queue of waiting interruptible jobs to build without bound. Consider the ideal-case upper bound on possible energy savings. The Facebook workload has a historical average of 21029 active map tasks and 7745 active reduce tasks. A cluster of 72000 task slots can service 72000 concurrent tasks at maximum. Thus, the best case energy savings is $1 - (21029 + 7745)/72000 = 0.60$. As we lower `interruptible`, any energy "savings" above this ideal actually represents the wait queue building up.

The best policy combination we examined achieves energy savings of 0.55 fraction of the baseline, as shown Figure 4.11, with `taskcalc` set to default and `interruptible` set to 6 hours. This corresponds to 92% of this ideal case.

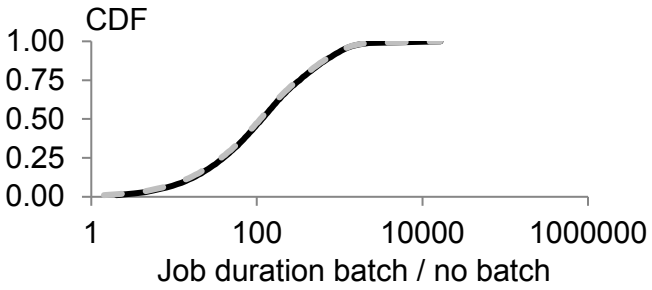
4.4.6 Overhead

The energy savings come at the cost of increased job latency. Figure 4.12 quantifies the latency increase by looking at normalized job durations for each job type. BEEMR achieves minimal latency overhead for interactive jobs, and some overhead for other job types. This delayed execution overhead buys us energy savings for non-interactive jobs.

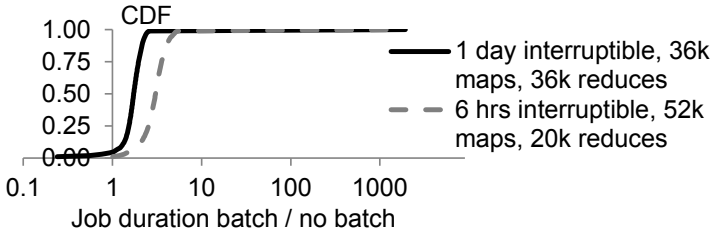
For interactive jobs, more than 60% of jobs have ratio of 1.0, approximately 40% of jobs have ratio less than 1.0, and a few outliers have ratio slightly above 1.0. This indicates that a dedicated interactive zone can lead to either unaffected job latency, or even improved job latency from having dedicated resources. The small number of jobs with ratio above 1.0



(a) Interactive jobs



(b) Batch jobs



(c) Interruptible jobs

Figure 4.12. Latency ratio by job type between BEEMR with `totalsize` set to 72000, `taskcalc` set to default, and (1) `batchlen` set to 24 hours, `mapreduceratio` set to 1:1, or (2) `batchlen` set to 6 hours, `mapreduceratio` set to 13:5; versus the baseline with no batching. A ratio of 1.0 indicates no overhead. Some interactive jobs see improved performance (ratio < 1) due to dedicated resources. Some batch jobs have very long delays, the same behavior as delayed execution under deadline-based policies. Interruptible jobs have less overhead than batch jobs, indicating that those are truly long running jobs. The delayed execution in non-interactive jobs buys us energy savings.

is caused by peaks in interactive job arrivals. This suggests that it would be desirable to increase the capacity of the interactive zone during workload peaks.

For batched jobs, the overhead spans a large range. This is caused by the long batch interval, and is acceptable as a matter of policy. A job that arrives just after the beginning of one batch would have delay of at least one batch interval, leading to large latency. Conversely, a job that arrives just before a batch starts will have almost no delay. This is the same delayed execution behavior as policies in which users specify, say, a daily deadline.

For interruptible jobs, the overhead is also small for most jobs. This is surprising because interruptible jobs can potentially execute over multiple batches. The result indicates that interruptible jobs are truly long running jobs. Executing them over multiple batches imposes a modest overhead.

4.4.7 Sensitivity

The evaluation thus far has set a `totalsize` of 72000 task slots and discovered the best parameter values based on this setting. A cluster size of 72000 forms a conservative baseline for energy consumption. Using BEEMR on larger clusters yields more energy savings, as shown in Figure 4.13.

BEEMR extracts most, but not all, of the ideal energy savings. The discrepancy arises from long tasks that hold up batch completion (Section 4.4.2) and transient imbalance between map and reduce slots (Section 4.4.4). If the fraction of time that each batch runs at maximum slot occupancy is already small, then the effects of long tasks and map/reduce slot imbalance are amplified. Thus, as cluster size increases, the gap between BEEMR energy savings and the ideal also increases. One way to narrow the gap would be to extend the batch interval length, thus amortizing the overhead of long tasks holding up batch completion and transient map/reduce slot imbalance. In the extreme case, BEEMR can achieve

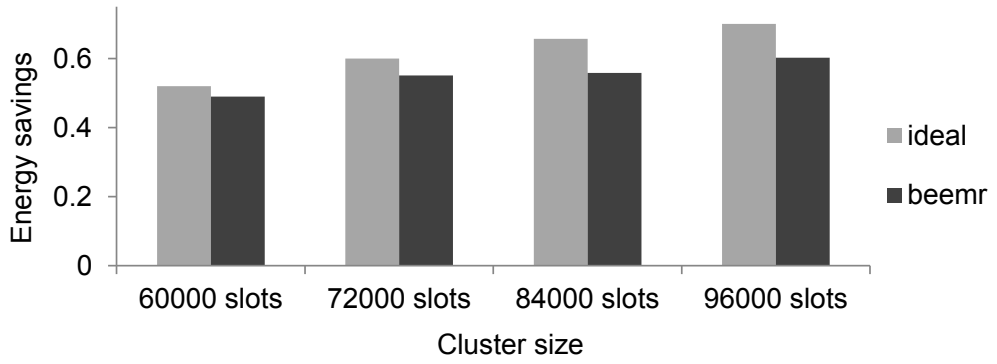


Figure 4.13. Ideal and observed energy savings for different cluster sizes. Both increase as cluster size increases. Note that `batchlen` is set to 24 hours, `taskcalc` is set to default, `mapreduceratio` is set to 13:5, and `interruptible` is set to 6 hours.

arbitrarily close to ideal energy savings by running the historical workload in one single batch.

4.4.8 Validation

Empirical validation of the simulator provides guidance on how simulation results translate to real clusters. The BEEMR simulator explicitly trades simulation scale and speed for accuracy, making it even more important to quantify the simulation error.

We validate the BEEMR simulator using an Amazon EC2 cluster of 200 "m1.large" instances [1]. We ran three experiments: (1) a series of stand-alone sort jobs, (2) replay several day-long Facebook workloads using the methodology in [18], which reproduces arrival patterns and data sizes using synthetic MapReduce jobs running on synthetic data, (3) replay the same workloads in day-long batches. For Experiments 1 and 2, we compare the job durations from these experiments to those obtained by a simulator configured with the same number of task slots and the same policies regarding task granularity. For Experiment 3, we compare the energy savings predicted by the simulator to that from the EC2 cluster.

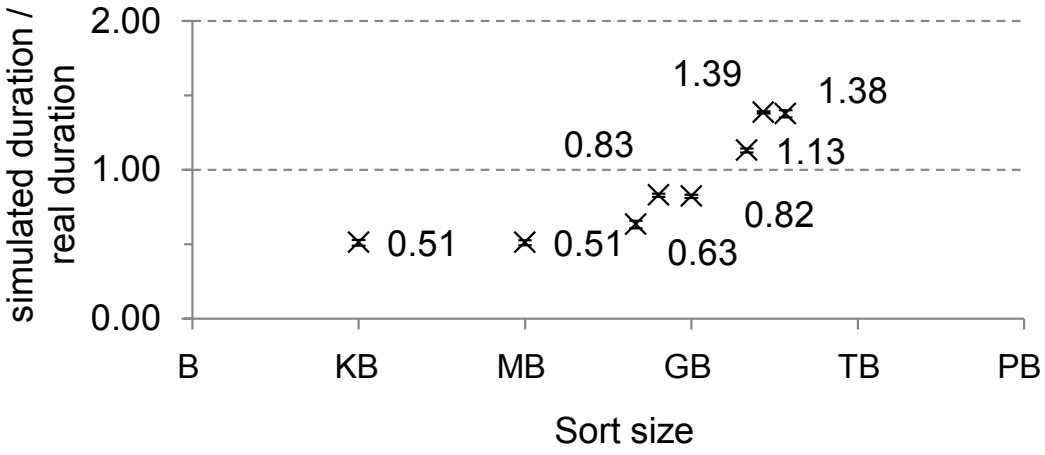


Figure 4.14. Simulator validation for stand-alone jobs. Showing the ratio between simulated job duration and average job duration from 20 repeated measurements on a real cluster. The ratio is bounded for both large and small jobs and is very close to 1.0 for sort jobs of size 100s of MB to 10s of GB.

These experiments represent an essential validation step before deployment on the actual front-line Facebook cluster running live data and production code.

Figure 4.14 shows the results from stand-alone sort jobs. This ratio is bounded on both ends and is very close to 1.0 for sort jobs of size 100s of MB to 10s of GB. The simulator underestimates the run time (the ratio is less than 1.0) for small sort sizes. There, the overhead of starting and terminating a job dominates; this overhead is ignored by the simulator. The simulator overestimates the run time (the ratio is greater than 1.0) for large sort sizes. For those jobs, there is non-negligible overlap between map and reduce tasks; this overlap is not simulated. The simulation error is bounded for both very large and very small jobs.

Also, there is low variance between different runs of the same job, with 95% confidence intervals from 20 repeated measurements being barely visible in Figure 4.14. Thus, pathologically long caused by task failures or speculative/abandoned executions are infrequent; not simulating these events causes little error.

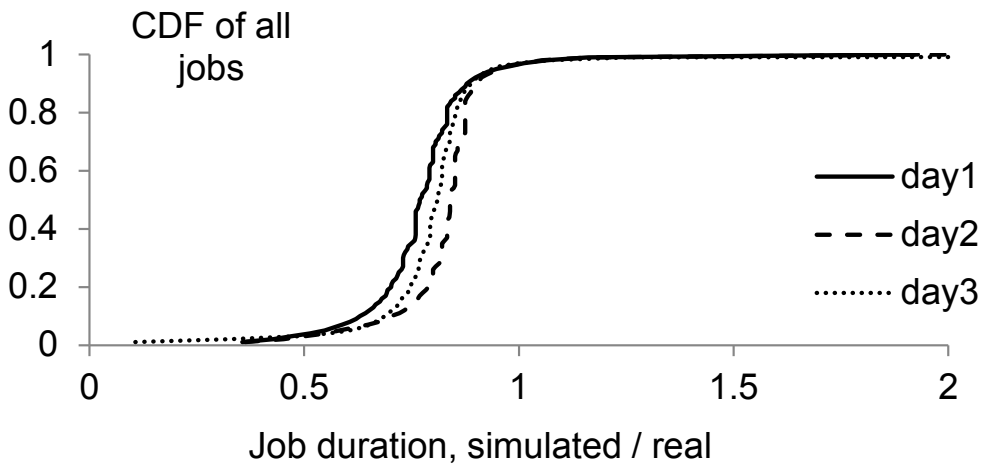


Figure 4.15. Simulator validation for three day-long workloads, without batching. Showing the ratio between simulated and real job duration. This ratio is bounded on both ends and is very close to 0.75 for the vast majority of jobs.

Figure 4.15 shows the results of replaying one day's worth of jobs, using three different day-long workloads. The ratio is again bounded, and close to 0.75 for the majority of jobs. This is because most jobs in the workload have data sizes in the MB to GB range (Figure 2.2). As explained previously, job startup and termination overhead lead to the simulator to underestimate the duration of these jobs.

Figure 4.16 shows the validation results from batching the three day-long workloads. The simulation error varies greatly between three different days. The average error is 22% of the simulated energy savings (top graph in Figure 4.16). We identify two additional sources of simulator error: (1) The BEEMR simulator assumes that all available task slots are occupied during the batches. However, on the EC2 cluster, the task slot occupancy averages from 50% to 75% of capacity, a discrepancy again due to task start and termination overhead — the scheduler simply cannot keep all task slots occupied. Adjusting the simulator by using a lower cluster size than the real cluster yields the bottom graph in Figure 4.16, with the error decreased to 13% of the simulated energy savings. (2) The BEEMR simulator

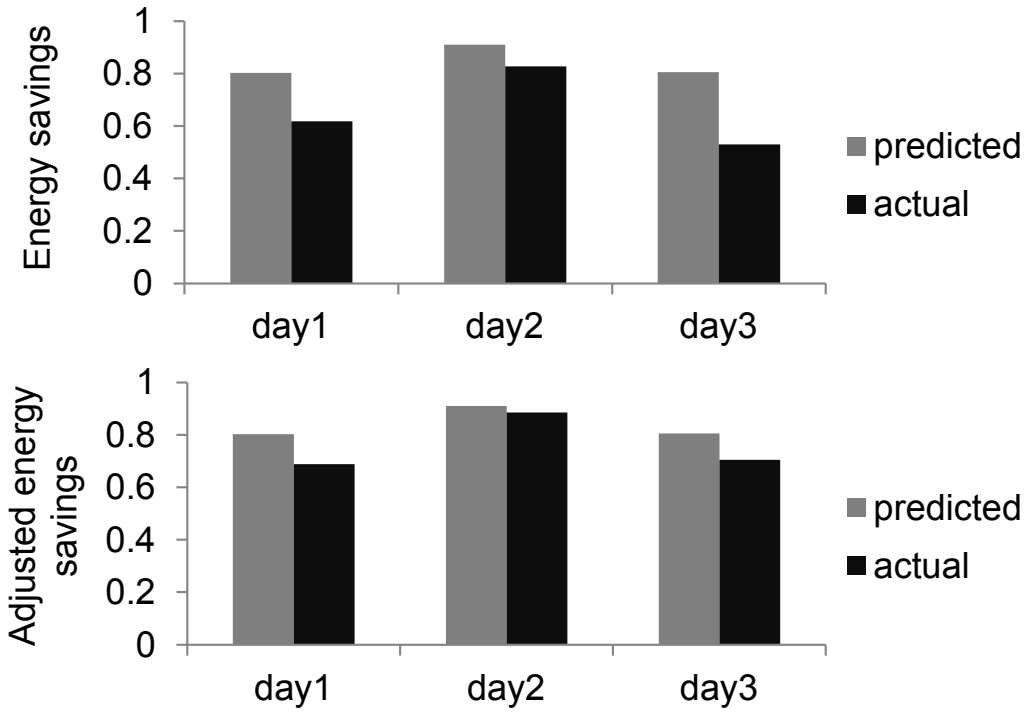


Figure 4.16. Simulator validation for three different day-long workloads, with `batchlen` set to 24 hours. Showing the predicted versus actual energy savings (top graph, average 22% simulation error), and the predicted versus actual energy savings after adjusting for the slot occupancy capacity on the real-life cluster (bottom graph, average 13% simulation error).

assumes that task times remain the same regardless of whether the workload is executed as jobs arrive, or executed in batch. Observations from the EC2 cluster reveals that during batches, the higher real-life cluster utilization leads to complex interference between jobs, with contention for disk, network, and other resources. This leads to longer task times when a workload executes in batch, and forms another kind of simulation error that is very hard to model.

Overall, these validation results mean that the simulated energy savings of 50-60% (Section 4.4.5) would likely translate to 40-50% on a real cluster.

Chapter 5

Conclusion

In this chapter, we first present some general discussion. We then consider avenues of future work related to each of the studies we considered. We close with a summary of contributions.

5.1 Discussion

In this section, we examine some often asked questions, including:

- Is transitioning machines into a low-power state the right approach?
- How can we improve system methodologies to provide the more transparent, understandable, and ideally, repeatable results?
- Are the workloads in the case studies we examine general enough for us to draw broader conclusions?

We do not claim to have an answer for all of these questions, but we hope that the work here provides some guidance.

5.1.1 Increasing Load versus Decreasing Power

There is a lively debate about what is the best way to handle workloads with inherent high peak to average ratios—save energy by transitioning machines to low-power states during below peak periods, or increase utilization of machines by sending more work to smooth out the peaks? Running machines at high utilization is desirable, since until energy prices get considerably higher, equipment depreciation of already purchased equipment will dominate the cost of power. Idle machines represent arguably wasted investments [6]. However, it remains an open problem to increase cluster utilization to consistently high levels without major performance degradation remains an open problem [36].

In some respects, powering down machines does not conflict with this viewpoint. Most workloads have inherent high peak-to-average ratios that must be serviced within some deadline or SLO constraints, implying that it is inevitable to have some degree of over provisioning and low utilization. In our web services cluster, we strive to drive up active machines to respectable levels of utilization. BEEMR shifts a subset of the workload in time and executes those jobs in batches with high utilization. The 40-50% energy savings can be viewed as 40-50% head room in the cluster for additional latency insensitive jobs.

5.1.2 Power Cycles versus Reliability

Transitioning machines to low-power states is one way to achieve power proportionality while more power proportional hardware is being developed. Large-scale adoption of this technique has been limited by worries that power cycling increases failure rates.

There have been few published, large-scale studies that attribute increased failure rates to power cycling. Some authors have observed a correlation between the two, but point out that correlation could come simply from failed systems needing more reboots to restore [56]. To identify a causal relationship would require a more rigorous methodology, comparing

mirror systems servicing the same workload, with the only difference being the frequency of power cycles.

One such comparison experiment ran for 18 months on 100s of machines, and found that power cycling has no effect on failure rates [53]. Larger scale comparisons have been stymied by the small amount of predicted energy savings, and uncertainty about how those energy savings translate to real systems. BEEMR gives empirically validated energy savings of 40-50%. This represents more rigorous data to justify further exploring the thus far unverified relationship between power cycles and failure rates.

5.1.3 Methodology

Evaluating the energy efficiency of large scale distributed systems presents significant methodological challenges. We attempt to strikes a balance between scale and accuracy.

Simulation versus replay. The inherent difference among workloads means that the best energy efficiency mechanisms would be highly workload dependent. In our two case studies, we use two very different approaches, though they both rely on the same basic mechanism of consolidating load and transitioning unneeded machines to a low-power state. Even within workloads, behavior varies between use cases and over time (Figures 4.10 and 5.1). Thus, only evaluation over long durations can reveal the true historical savings (Figure 4.10). Day- or week-long experiments are unrealistic, especially to explore multiple design options at large scale. Hence, we are compelled to use simulations.

Choice of simulator. For our MapReduce case study, we considered using Mumak [51]and MRPerf [71]. Mumak requires logs generated by the Rumen tracing tool [60], which is not yet in universal use and not used at Facebook. MRPerf generates a simulation event per control message and per packet, which limits simulation scale and speed. Neither simulator has been verified at the multi-job workload level. Additionally, to our knowledge, there are

no widely used web service simulators. Thus, we developed both the NapSAC and BEEMR simulators. Both intentionally trade simulation detail and accuracy to gain scale and speed. We also verify the simulator at the workload level (Sections 3.4.6 and 4.4.8).

Choice of power model. One accurate way to measure system power is by a power meter attached at the machine wall socket. For NapSAC, we use this method to profile machines individually. However, this method does not scale to clusters of 1000s of machines. An alternative is to use empirically verified power models, which are yet to be satisfactorily developed for MapReduce and other workloads. The translation between SPECpower [66] measurements and MapReduce remains unknown, as it is between MapReduce workload semantics and detailed CPU, memory, disk, and network activity. For BEEMR, we chose an on-off power model, i.e., machines have "max" power when on and "zero" power when off. This simple model allow us to scale the experiments in size and in time.

Towards improved methodology. The deliberate tradeoffs we had to make reflect the nascent performance understanding and modeling of large scale systems. We encourage the research community to seek to overcome the methodology limitations of this study.

5.1.4 MIA Generality Beyond Facebook

MIA workloads beyond Facebook lend themselves to a BEEMR-like approach. We analyzed four additional Hadoop workloads from e-commerce, telecommunications, media, and retail companies. These traces come from production clusters of up to 700 machines, and cover 4 cluster-months of behavior. The following gives a glimpse of the data. We are seeking approval to release these additional workloads.

One observation that motivated BEEMR is that most jobs access small files that make up a small fraction of stored bytes (Figure 2.5). This access pattern allows a small interactive zone to service its many jobs. Figure 5.1shows that such access patterns exist for all

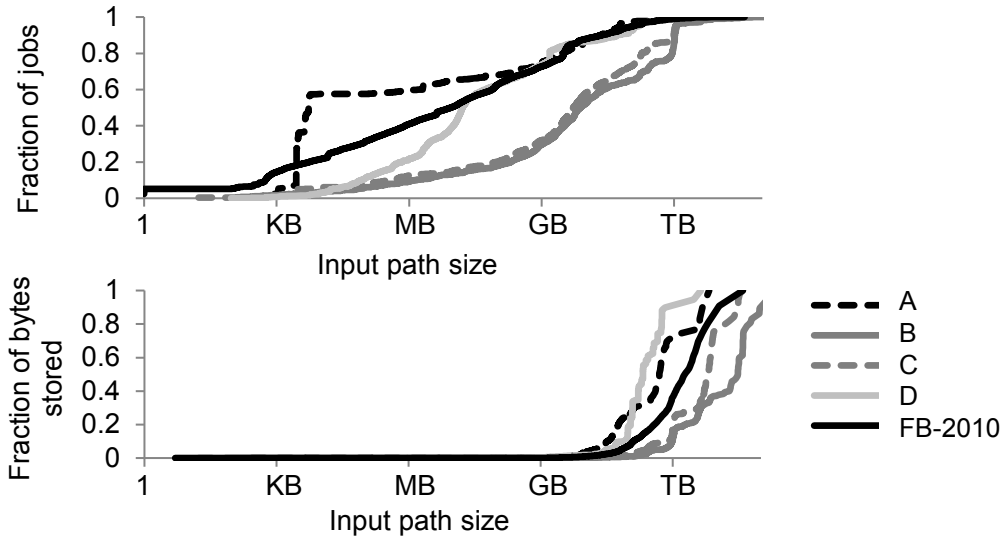


Figure 5.1. Access patterns versus input path size. Showing cumulative fraction of jobs with input paths of a certain size (top) and cumulative fraction of all stored bytes from input paths of a certain size (bottom). Contains data from Figure 2.5 for the FB-2010 workload, and four additional workloads from e-commerce, telecommunications, media, and retail companies.

workloads. For FB-2010, input paths of $<$ 10GB account for 88% of jobs and 1% of stored bytes. For workloads A and D, the same threshold respectively accounts for 87% and 87% of jobs, and 4% and 2% of stored bytes. For workloads B and C, input paths of $<$ 1TB accounts for 86% and 91% of jobs, as well as 12% and 17% of stored bytes.

Another source of energy savings comes from the high peak-to-average ratio in workload arrival patterns (Figure 2.3). The cluster has to be provisioned for the peak, which makes it important to achieve energy proportionality either in hardware or by workload managers such as BEEMR. For the five workloads (Facebook and workloads A through D), the peak-to-average ratios are: 8.9, 30.5, 23.9, 14.5, and 5.9. BEEMR potentially extracts higher energy savings from workloads with higher peak-to-average arrival ratios, though the exact energy savings and the tradeoff between policy parameters is workload specific. These ad-

ditional workloads give us confidence that the BEEMR architecture can generalize beyond the Facebook workload.

5.2 Future Work for Power Proportional Web Services

Our NapSAC cluster manager is optimized for those workloads exhibiting statistically similar traits to the Wikipedia workload. While we realize this lack of workload diversity limits the generalization of our results, we felt it was more compelling to show an actual use-case of our cluster manager paired with actual production workload traces rather than leave it unoptimized with only synthetic scenarios demonstrating different request interarrival times.

The current cluster manager makes simplifying assumptions that can be extended in the future to support more diverse workloads. One such assumption is that the distribution of response times holds true for all requests. This assumption breaks down with requests for both static and dynamic content as well as in systems with caches where requests have only some probability of hitting in the cache. Handling different types of requests requires the cluster manager to determine the type of content and to know the distribution of response times for each type. The distribution of response times for dynamic requests poses a more difficult challenge than simple static pages. Dealing with cache hits and misses requires that the cluster manager have knowledge about the hit ratios.

A second simplifying assumption our cluster is based on is that every backend node acts as a web server that has an exact replica of all the data. However, in a more multipurpose cluster, different backends might serve special purposes, from web server to database server or application server, and even machines dedicated to the same purpose might contain different sets of data, as in a distributed file system. The current round-robin server provisioning scheme would not work with this mix of machines since this provisioning treats every ma-

chine equally with regards to being turned on or off, so choosing the particular machine to run an incoming request on requires the scheduler to keep more detailed state information about each machine, though it is possible round-robin provisioning could be used within pools of special purpose machines. Dealing with machines containing different data for the same applications is more complicated and requires further study.

In conjunction with differently purposed nodes comes mixed workloads, including batch jobs. Nodes in our cluster are currently provisioned based on their current and maximum request rate capacities, so the difficulty with batch jobs lies in deciding what the equivalent of maximum request rate should be and how to determine whether a given node can accept new jobs. CPU capacity and utilization might be a reasonable first-order estimation for these types of jobs, though it would require frequent probing from the cluster manager to determine these levels.

5.3 Future Work for MapReduce in General

Designing and evaluating BEEMR has revealed several opportunities for future improvements to MapReduce.

1. The BEEMR policy space is large. It would be desirable to automatically detect good values for the policy parameters in Table 4.1.

2. The ability to interrupt and resume jobs is desirable. This feature is proposed under Next Generation Hadoop for fast resume from failures [52]. Energy efficiency would be another motivation for this feature.

3. A well-tuned `taskcalc` algorithm can significantly affect various performance metrics (Section 4.4.3). However, choosing the correct number of tasks to assign to a job

remains an unexplored area. Given recent advances in predicting MapReduce execution time [29], [50], we expect a dedicated effort would discover many improvements.

4. The chatty HDFS/Hadoop messaging protocols limits the dynamic power of machines to a narrow range. There is an opportunity to re-think such protocols for distributed systems to improve power proportionality.

5. The disjoint interactive and batch zones can be further segregated into disjoint interactive and batch clusters. Segregated versus combined cluster operations need to balance a variety of policy, logistical, economic, and engineering concerns. More systematic understanding of energy costs helps inform the discussion.

6. The gap between ideal and BEEMR energy savings increases with cluster size (Section 4.4.7). It is worth exploring whether more fine-grained power management schemes would close the gap and allow operators to provision for peak while conserving energy costs.

5.4 Summary of Contributions

In an effort to tackle the problem of ever-rising energy usage from data centers, we presented the design of (1) a power proportional cluster that uses a power-aware cluster manager and a set of heterogeneous machines to harness idleness to achieve significant energy savings, and (2) an architecture for an energy efficient MapReduce cluster that relies on keeping a small always-on cache for interactive jobs and batching the remainder to achieve savings.

NapSAC

In our first study, we characterize three different types of compute nodes representing server, mobile, and embedded class systems with the metric of Joules per successful web request. Despite having different levels of performance, they all exhibit comparable efficiencies within their own energy efficient operating ranges, defined as the range of request rates a node can service with low energy per response and low response times. Finding the proper operating set point within this range is crucial to maintaining both top performance and energy efficiency. Machines should have their operating set point fall within these energy efficient ranges. The level of over provisioning within a cluster affects this operating point and should be carefully considered by system operators.

To study how over provisioning and different provisioning schemes impact the energy and performance of a front end web service cluster, numbers obtained from individual nodes were used in simulations for estimating future incoming web server work rates in our Wikipedia-based workload. Taking the best algorithms from the simulation, we compare the efficiency of our power proportional cluster to static provisioning and an oracle algorithm. The standard approach to capacity planning is to provision for twice the peak load, we consider this the baseline, static provisioning approach. We achieve over 70% power savings compared to static provisioning and we are within 90% of the optimal. In addition, these saving can be achieved while minimally impacting performance. As a large proportion of servers in the real world are idle for large portions of the day, we expect our system design to enable major energy savings when deployed at large scale. We implemented NapSAC cluster manager on a set of mobile class compute nodes along with the optimal provisioning algorithm from our simulations and demonstrated the use of our design on a set of actual Wikipedia workload traces. Using relatively simple, stable estimations for fu-

ture work rates and simple round-robin scheduling, we are still able to attain improvements over conventional 2x capacity planning.

BEEMR

For our second study, we performed an analysis of a Facebook cluster trace to quantify the empirical behavior of a MIA workload. From this, we developed the BEEMR framework which combines novel ideas with existing MapReduce energy efficiency mechanisms. We improved upon current evaluation methodology to quantify energy savings achieved by BEEMR and account for the complexity of MIA workloads. BEEMR is able to cut the energy consumption of a cluster almost *in half* (after adjusting for empirically quantified simulation error) without harming the response time of latency-sensitive jobs or relying on storage replication, while allowing jobs to retain the full storage capacity and compute bandwidth of the cluster. BEEMR achieves such results because its design was guided by a thorough analysis of a real-world, large-scale instance of the targeted workload. We dubbed this widespread yet under-studied workload MIA. The key insight from our analysis of MIA workloads is that although MIA clusters host huge volumes of data, the interactive jobs operate on just a small fraction of the data, and thus can be served by a small pool of dedicated machines; the less time-sensitive jobs can run in a batch fashion on the rest of the cluster.

Acknowledgements

Many thanks to my collaborators Yanpei Chen, Andrew Krioukov, Prashanth Mohan, and Laura Keys, as well as to the many individual student reviewers in the EECS department who provided helpful feedback on various versions of text, including Peter Alvaro, Ali Ghodsi, Matei Zaharia, and Kay Ousterhout, among others. Also thanks to Druhba Borathkor and Stephen Dawson-Haggerty for their useful comments. Lastly, thanks to my advisor Professor Randy Katz and Professor David Culler for help initiating this line of research and providing much guidance along the way.

References

[1] Amazon Web Services. Amazon Elastic Computing Cloud. http://aws.amazon.com/ec2/.

[2] G. Ananthanarayanan et al. Scarlett: coping with skewed content popularity in mapreduce clusters. In *Eurosys 2011*.

[3] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. Fawn: A fast array of wimpy nodes. *SOSP '09*, October 2009.

[4] R. H. Arpaci et al. The interaction of parallel and sequential workloads on a network of workstations. In *SIGMETRICS 1995*.

[5] I. Ashok and J. Zahorjan. Scheduling a mixed interactive and batch workload on a parallel, shared memory supercomputer. In *Supercomputing 1992*.

[6] L. A. Barroso. Warehouse-scale computing: Entering the teenage decade. In *ISCA 2011*.

[7] L. A. Barroso. The price of performance. *ACM Queue*, 3(7):48–53, 2005.

[8] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

[9] R. Bianchini and R. Rajamony. Power and energy management for server systems. *Computer*, Nov 2004.

[10] P. Bodik, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson. Statistical machine learning makes automatic control practical for internet datacenters. *HotCloud'09*, 2009.

[11] D. Borthakur. Facebook has the world's largest Hadoop cluster! http://hadoopblog.blogspot.com/2010/05/facebook-has-worlds-largest-hadoop.html.

[12] D. Borthakur et al. Apache Hadoop goes realtime at Facebook. In *SIGMOD 2011*.

[13] L. Breslau et al. Web Caching and Zipf-like Distributions: Evidence and Implications. In *INFOCOM 1999*.

[14] E. A. Brewer. Lessons from giant-scale services. *IEEE Internet Computing*, 5(4):46–55, 2001.

[15] J. S. Chase, D. C. Anderson, P. N. Thakar, A. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centres. In *SOSP*, pages 103–116, 2001.

[16] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 337–350, Berkeley, CA, USA, 2008. USENIX Association.

[17] Y. Chen, L. Keys, and R. H. Katz. Towards Energy Efficient MapReduce. Technical Report UCB/EECS-2009-109, EECS Department, University of California, Berkeley, Aug 2009.

[18] Y. Chen et al. The Case for Evaluating MapReduce Performance Using Workload Suites. In *MASCOTS 2011*.

[19] Y. Chen et al. Statistical Workloads for Energy Efficient MapReduce. Technical Report UCB/EECS-2010-6, EECS Department, University of California, Berkeley, Jan 2010.

[20] B.-G. Chun, G. Iannaccone, G. Iannaccone, R. Katz, G. Lee, and L. Niccolini. An energy case for hybrid datacenters. *HotPower'09*, 2009.

[21] J. Corbet. LWN.net 2009 Kernel Summit coverage: How Google uses Linux. 2009.

[22] H.-P. Corporation, I. Corporation, M. Corporation, P. T. Ltd., and T. Corporation. Advanced configuration and power interface specification, June 2009.

[23] S. Dawson-Haggerty, A. Krioukov, and D. E. Culler. Power optimization - a reality check. Technical Report UCB/EECS-2009-140, EECS Department, University of California, Berkeley, Oct 2009.

[24] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Comm. of the ACM*, 51(1):107–113, January 2008.

[25] Q. Deng et al. Memscale: active low-power modes for main memory. In *ASPLOS 2011*.

[26] EMC and IDC iView. Digital Universe. http://www.emc.com/leadership/programs/digital-universe.htm.

[27] S. Eyerman and L. Eeckhout. System-level performance metrics for multiprogram workloads. *Micro, IEEE*, 28(3):42 –53, May-June 2008.

[28] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA 2007*.

[29] A. Ganapathi et al. Statistics-driven workload modeling for the cloud. In *ICDEW 2010*.

[30] J. Gray et al. Quickly generating billion-record synthetic databases. In *SIGMOD 1994*.

[31] Gridmix. HADOOP-HOME/mapred/src/benchmarks/gridmix2 in Hadoop 0.20.2 onwards.

[32] S. V. L. Group. Data center energy forecast. `http://svlg.org/campaigns/datacenter/docs/DCEFR_report.pdf`, 2009.

[33] D. Grunwald, P. Levis, K. I. Farkas, C. B. M. III, and M. Neufeld. Policies for dynamic clock scheduling. In *OSDI*, pages 73–86, 2000.

[34] Hadoop World 2011. Hadoop World 2011 Speakers. `http://www.hadoopworld.com/speakers/`.

[35] Hewlett-Packard Corp., Intel Corp., Microsoft Corp., Phoenix Technologies Ltd., Toshiba Corp. Advanced Configuration and Power Interface 5.0. `http://www.acpi.info/`.

[36] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI 2011*.

[37] M. Isard et al. Quincy: fair scheduling for distributed computing clusters. In *SOSP 2009*.

[38] R. T. Kaushik et al. Evaluation and Analysis of GreenHDFS: A Self-Adaptive, Energy-Conserving Variant of the Hadoop Distributed File System. In *IEEE CloudCom 2010*.

[39] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. In *ICAC '08:*, 2008.

[40] W. Lang and J. Patel. Energy management for mapreduce clusters. In *VLDB 2010*.

[41] J. Leverich and C. Kozyrakis. On the Energy (In)efficiency of Hadoop Clusters. In *HotPower 2009*.

[42] P. Lieberman. White paper: Wake on lan technology, June 2006.

[43] B. Liu et al. A study of networks simulation efficiency: Fluid simulation vs. packet-level simulation. In *Infocom 2001*.

[44] J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with pace. In *SIGMETRICS/Performance*, pages 50–61, 2001.

[45] D. McCullagh. Turbotax e-filing woes draw customer ire. 2007.

[46] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: eliminating server idle power. In *ASPLOS '09*, 2009.

[47] D. Meisner et al. Power management of online data-intensive services. In *ISCA 2011*.

[48] S. Melnik et al. Dremel: interactive analysis of web-scale datasets. In *VLDB 2010*.

[49] A. K. Mishra et al. Towards characterizing cloud backend workloads: insights from Google compute clusters. *SIGMETRICS Perform. Eval. Rev.*, 37:34–41, March 2010.

[50] K. Morton et al. ParaTimer: a progress indicator for MapReduce DAGs. In *SIGMOD 2010*.

[51] Mumak. Mumak: Map-Reduce Simulator. `https://issues.apache.org/jira/browse/MAPREDUCE-728`.

[52] A. Murthy. Next Generation Hadoop Map-Reduce. Apache Hadoop Summit 2011.

[53] D. Patterson. Energy-Efficient Computing: the State of the Art. Microsoft Research Faculty Summit 2009.

[54] Personal email. Communication regarding release of Google production cluster data.

[55] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Dynamic cluster reconfiguration for power and performance. pages 75–93, 2003.

[56] E. Pinheiro et al. Failure trends in a large disk drive population. In *FAST 2007*.

[57] N. Rasmussen. Electrical efficiency modeling of data centers. Technical Report White Paper #113, APC, 2006.

[58] G. F. Riley, T. M. Jaafar, and R. M. Fujimoto. Integrated fluid and packet network simulations. In *MASCOTS 2002*.

[59] S. Rivoire et al. Joulesort: a balanced energy-efficiency benchmark. In *SIGMOD 2007*.

[60] Rumen: a tool to extract job characterization data from job tracker logs. `https://issues.apache.org/jira/browse/MAPREDUCE-751`.

[61] A. Ryan. Next-Generation Hadoop Operations. Bay Area Hadoop User Group, February 2010.

[62] J. H. Saltzer. A simple linear model of demand paging performance. *Commun. ACM*, 17:181–186, April 1974.

[63] R. K. Sharma, C. E. Bash, C. D. Patel, R. J. Friedrich, and J. S. Chase. Balance of power: Dynamic thermal management for internet data centers. *IEEE Internet Computing*, 9(1):42–49, 2005.

[64] K. Shvachko. HDFS Scalability: the limits to growth. *Login*, 35(2):6–16, April 2010.

[65] D. C. Snowdon et al. Accurate on-line prediction of processor and memory energy usage under voltage scaling. In *EMSOFT 2007*.

[66] SPEC. SPECpower 2008. http://www.spec.org/power_ssj2008/.

[67] the green grid. The green grid data center power efficiency metrics: Pue and dcie, 2007.

[68] A. Thusoo et al. Data warehousing and analytics infrastructure at Facebook. In *SIGMOD 2010*.

[69] G. Urdaneta, G. Pierre, and M. van Steen. Wikipedia workload analysis for decentralized hosting. *Elsevier Computer Networks*, 53(11):1830–1845, July 2009.

[70] U.S. Environmental Protection Agency. Epa report on server and data center energy efficiency. *ENERGY STAR Program*, 2007.

[71] G. Wang et al. A simulation approach to evaluating design decisions in MapReduce setups. In *MASCOTS 2009*.

[72] M. Weiser, B. B. Welch, A. J. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *OSDI*, pages 13–23, 1994.

[73] Wikipedia.org.

[74] M. Zaharia et al. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *EuroSys 2010*.

[75] W. Zhang. Linux virtual server for scalable network services. *Ottawa Linux Symposium*, 2000.