Energy Management for MapReduce Clusters

Willis Lang and Jignesh M. Patel
Computer Sciences Department
University of Wisconsin-Madison, USA
{wlang, jignesh}@cs.wisc.edu

ABSTRACT

The area of cluster-level energy management has attracted significant research attention over the past few years. One class of techniques to reduce the energy consumption of clusters is to selectively power down nodes during periods of low utilization to increase energy efficiency. One can think of a number of ways of selectively powering down nodes, each with varying impact on the workload response time and overall energy consumption. Since the MapReduce framework is becoming "ubiquitous", the focus of this paper is on developing a framework for systematically considering various MapReduce node power down strategies, and their impact on the overall energy consumption and workload response time.

We closely examine two extreme techniques that can be accommodated in this framework. The first is based on a recently proposed technique called "Covering Set" (CS) that keeps only a small fraction of the nodes powered up during periods of low utilization. At the other extreme is a technique that we propose in this paper, called the All-In Strategy (AIS). AIS uses all the nodes in the cluster to run a workload and then powers down the entire cluster. Using both actual evaluation and analytical modeling we bring out the differences between these two extreme techniques and show that AIS is often the right energy saving strategy.

1. INTRODUCTION

Direct monthly energy costs for data centers make up around 23% of the total amortized monthly operating costs [17]. This cost does not include functionally related energy costs such as power distribution and cooling infrastructure, which Hamilton states will increase the energy costs to 42% of the total monthly operating costs. If we also consider that server costs are consistently falling, then it is estimated that this year, the three year cost of electricity per server will exceed the initial cost of the server itself [9]. Trends show that processor performance doubles (in number of cores) every 18 months while the performance per Watt only doubles every two years [6]. Thus, it should be no surprise that an early EPA study estimates that servers will make up 3% of the total energy consumption in the U.S. in 2011 [4].

One reason contributing to the high server energy costs is that server nodes in cluster environments are typically only 20-30% utilized, and energy efficiency in this range is under 50% [5]. This suggests that 42% of the total monthly operating cost stemming from power [17] can be reduced if we increase energy efficiency of the cluster nodes during low utilization periods.

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

Recently, the Covering Set (CS) method was proposed for cluster energy management [22]. The CS strategy exploits the replication that is provided by a distributed file systems (DFS), which keeps multiple copies of each data block spread across nodes in the cluster. The CS strategy designates some nodes in the system as special nodes, called the CS nodes, and keeps at least one copy of each unique data block on these nodes. Thus, by altering the data placement policy of the underlying DFS, during periods of low utilization, some or all of the non-CS nodes can now be powered down to save energy. For example, if 33% of the nodes are CS nodes, then at 33% utilization only the CS nodes are online. CS is general and allows the CS nodes to be an arbitrary fraction of the total nodes, and also allows for part of the non-CS nodes to stay online.

With CS, the workloads take longer to run when the cluster is partially powered down, as fewer nodes are available to run the workload. Other downsides to CS include significant overprovisioning of space on the CS nodes as well as requiring code modifications in the underlying DFS software (see Section 4.5).

In this paper, we propose an alternative cluster energy management strategy, called the All-In Strategy (AIS). In AIS, rather than increasing the response time of a workload as in the CS strategy, we run the workload (or a batch of workloads) on all the nodes in the cluster. Then, when we are in a low utilization period and the cluster is idle, the cluster is transitioned to a low power state. Thus, in AIS, rather than selectively powering down the nodes as in CS, the cluster essentially wakes up, runs as fast as it can, and then powers down again. One advantage of AIS is that there is a very predictable degradation in the workload response time. This degradation is based on the time it takes for the hardware and the OS to power up and down nodes from deep power saving modes, and efforts such as [24] are pushing to reduce this cost dramatically.

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

Proceedings of the VLDB Endowment, Vol. 3, No. 1

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

Our experimental results show that in many cases, AIS results in lower energy consumption than CS. For example, running TeraSort on a relatively small 77GB dataset on a 24 node cluster at 33% utilization is always 11% more energy efficient with AIS than with CS, and 60% more efficient than an unmanaged cluster. Perhaps more importantly, CS also incurs a 3.6X increase in response time while AIS suffers only a 12% response time degradation. With larger complex workloads, we show these energy gains of AIS over CS improve rapidly, and factors of 2X improvement in energy or more over CS are easily possible.

The key contributions of this paper are:

- We present a framework for designing and evaluating methods that selectively power down MR nodes to save energy. Within this framework, we focus on two extreme techniques – a recently proposed technique called Covering Set (CS), and a new technique called All-In Strategy (AIS).
- Using our framework, we systematically explore factors that benefit each approach, and show that the (simpler) AIS technique is often more effective than the CS technique in reducing energy consumption. During long and computationally complex MR jobs, AIS overcomes its high cluster transitioning costs and provides better response time and energy savings than CS. AIS also does not require storage overprovisioning as is needed for CS, or require modifying the DFS code, and hence can directly be used with existing systems.
- Our work explores the impact of powering down MR nodes on the workload response time (which increases as nodes are powered down). Our analysis brings out the effect of the interaction between the total cluster energy consumption, the computational complexity of the workload, and a key hardware parameter – namely, the relative time it takes to power up and down a server node with respect to the workload response time. As a consequence, our framework points to the tremendous benefits of research that can improve hardware and software mechanisms to improve power up/down costs (e.g., fast deep hibernation by using PCM [26] and/or building data centers using mobile/nettop computing hardware).

The remainder of this paper is organized as follows: Section 2 presents our framework. Section 3 discusses CS and AIS. Results evaluating CS and AIS are presented in Section 4. Related work and conclusions are discussed in Section 5 and Section 6.

2. ENERGY MANAGEMENT FRAMEWORK

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

We present a mathematical model for the energy consumption of a MapReduce cluster during a specified time window υ, when running a workload ω using a cluster with hardware characteristics η. For simplicity, the workload characteristics (ω) and hardware characteristics (η) are considered as abstract meta-models in our model.

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

Table 1: List of Variables in our Framework

More detailed models for capturing these can be plugged into our model. The workload characteristics model describes the job characteristics, such as an expected resource consumption, performance goals, computational complexity, etc. The hardware characteristics describes aspects such as the average power consumption of the hardware when running the workload, time and energy required to power up/down nodes, etc.

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

Thus, using the variables in Table 1, the total energy consumption, denoted as $E(\omega, \upsilon, \eta)$, is:

$$E(\omega, \upsilon, \eta) = (P_{tr}T_{tr}) + (P_w^n + P_w^{\bar{n}})T_w + (P_{idle}^m + P_{idle}^{\bar{m}})T_{idle} \quad (1)$$

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

$$\upsilon = T_{tr} + T_w + T_{idle} \quad (2)$$

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

$$min(E(\omega, \upsilon, \eta)) \quad | \quad T_w \leq \tau \quad (3)$$

Based on this model, we can see that there are several approaches to reduce the cluster energy consumption. From Equation 1, we see that one can reduce the energy consumption by reducing the idle energy consumption, $(P_{idle}^m + P_{idle}^{\bar{m}})T_{idle}$, by powering down part of the cluster. But powering down part of the cluster implies that the job has fewer nodes/resources to run, which potentially impacts the execution time of the workload (T_w). From Equation 1 this means that the energy cost to run the workload could rise as T_w increases. The rate of increase in the workload energy consumption with fewer nodes will depend on the workload characteristics, and primarily the computational complexity of the workload.

From Equation 1, we also observe that the time to transition nodes between powered up and down states (T_{tr}) can have a significant impact on the energy consumption, especially when the workload energy component in Equation 1 is small; i.e., when the workload execution time is small, schemes that require powering up and down often will consume significant energy in transitions.

Finally, from Equation 1, we can see that reducing the power drawn by online idle nodes P_{idle}^m can have a big impact on energy management schemes.

3. ENERGY MANAGEMENT STRATEGIES

In this section, we use the framework developed in Section 2 to consider two cluster energy management strategies. These two

strategies are: a) Covering Set (CS) – a recently proposed data placement and power down strategy to reduce the energy consumption of MapReduce clusters, and b) All-In Strategy (AIS) – a technique that we propose in this paper.

First, we present an overview of CS and AIS (Section 3.1), followed by various extensions to CS that are needed to make it a practical solution (Section 3.2), followed by a discussion of the AIS strategy (Section 3.3). We compare both techniques in Section 4.

3.1 Overview of CS and AIS

The CS strategy powers down nodes to reduce the idle energy consumption in Equation 1. In an ideal case, CS knows the workload perfectly ahead of time, and can power down just the right number of nodes at the start of the workload execution to reduce idle energy consumption to zero. However, as discussed in Section 2, such powering down of nodes can increase the response time of the workload, which in turn can increase the energy consumed during the workload execution. Thus, CS can only power down nodes such that it still adheres to performance constraints (Equation 3). CS also changes the data placement policy of the DFS so that one copy of the data is always online. The original CS work does not describe a strategy for powering down nodes. In Section 3.2 we discuss various node power down strategies for CS.

The AIS strategy is to trade idle energy consumption for transitioning energy consumption in Equation 1. It takes an extreme view and toggles the entire cluster between "all-nodes-on" and "all-nodes-off" modes. It uses all the nodes in the system to run the workload as fast as it can (i.e., minimizes T_w Equation 1), and then at the end of the workload execution, powers down all the nodes to reduce the idle energy cost. The price AIS pays is a high transitioning energy cost. The scale of this increase in transitioning cost is determined by the power (P_{tr}) and length of time (T_{tr}) of the cluster transition. While this transitioning power (P_{tr}) may be similar to the power when the cluster is fully on and running a workload, T_{tr} is solely defined by the capabilities of the hardware and the operating system.

3.2 Covering Set (CS)

The Covering Set (CS) strategy, proposed recently by Leverich and Kozyrakis [22] aims to reduce the energy consumption of clusters by changing the data placement policy in a DFS. The main idea is that in a DFS, such as GFS [14, 15] and HDFS [8], every data block is replicated three times. The cluster energy consumption can be reduced if the server powers down some nodes. But, powering down some nodes can make some data unavailable. To avoid this case, CS changes the data placement policy so that one copy of every data block is kept on a set of nodes. These nodes constitute the Covering Set nodes and are never powered down. To reduce energy consumption, non-CS nodes can be powered down. The CS nodes can be any arbitrary fraction of the total nodes in the system. For example, the CS nodes could be 25% of the total nodes in the system, which implies that up to 75% of the nodes could be powered down when running a workload.

The CS method proposed in [22] does not include any strategy to determine which nodes to power down when the overall system utilization drops. To use CS practically, one needs such a method which we briefly describe below (see Appendix A for more details). Also, to use CS in practice one also needs a power up method, which is the reverse of the power down method, and omitted in the interest of space.

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

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

3.3 All-In Strategy (AIS)

The CS technique described in the previous section (Section 3.2), has a few drawbacks. First, the CS strategy requires modifying the DFS code to alter the data placement strategy. As a result, it is not a broad generic solution and it is tied to the specific replication strategies used by the DFS. (It also makes it harder to use in cases where a single system may have different data sets with varying replication factors.) Second, CS does not explicitly consider the impact on response time. As we will see for workloads like distributed Grep that have linear computation complexity, this is manageable, but for workloads that have worse than linear complexity, this is problematic as running on fewer nodes can result in rapid response time degradation, which may not be acceptable. This means that to use CS, one would need a detailed workload run time estimation technique for all the cluster configurations that CS might transition to. Finally, CS requires good workload prediction as the system has to determine how many nodes to power down, and for how long. Compared to CS, AIS does not require modifying the data placement strategy or a detailed node power down strategy, and can trivially calculate the workload response time degradation.

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

We exploit this idea of eventual data availability and develop a new scheme for cluster energy management that addresses the shortcomings of CS outlined above. This new strategy is called the All-In Strategy (AIS). The AIS mechanism is simply to run the MapReduce job on all the nodes in the system and power down the entire system when there is no work. (Here we could have a mechanism to transition the entire system which could include everything – the compute nodes, rack power supply, other power supplies, routers, etc., to and from a low power state. Or, we could have a mechanism to transition selected parts of the entire system, e.g., only the compute nodes. The benefits are larger if more and more power hungry components provide mechanisms for quick transitions to and from power savings mode. AIS provides one more argument for building data centers out of traditionally fast transitioning mobile parts.)

In cases where there is a consistent low utilization period, AIS would batch the MR jobs in a queue, and periodically power up the entire system and run the entire batch of jobs on the cluster

"Down" means idle state ($114W$) transitioning to offline state, "Up" is the reverse.

State	Down Time(s)	Down Cost(J)	Up Time(s)	Up Cost(J)	State Cost(W)
Stopgrant	1	114	1	114	112
Hibernate	11	1300	100	12900	10
Off	27	3200	156	20000	10

Table 2: Costs for different types of offline states available on our MR nodes. Hibernate and Shutdown draw 10W because the motherboard/NIC is still powered on (for IPMI).

(and then power down). For instance, for the default FIFO queue scheduler in Hadoop or the add-on Fair scheduler, AIS could batch intermittantly arriving jobs to then submit all the jobs in the batch simultaneously. This idea mirrors the QED idea of energy efficient batching of database queries at a single node [19], and requires techniques for making the decision of how long to batch the jobs. These decisions could be guided by the delays that the job can tolerate (see Equation 3), and other workload characteristics. Workload prediction models, such as [7], would be used to guide the energy management framework. We leave such complex workload management as part of future work. (Note that CS would need such techniques too, so there is a broader set of research agendas on developing the decision making algorithms for system transitions.)

A crucial aspect for AIS is the cost to transition between low power states (T_{tr}, in Equation 1, Section 2) and the energy consumed in the idle state. There are a number of choices for these parameters that are offered by modern hardware. Consider Table 2 where we present *all* the available power up and down characteristics of one of our cluster nodes (details are presented in Section 4.1). In this table, it is clear that the hibernate state is the ideal energy efficient state to use; it is faster than full shutdown and consumes much less in its "off" state cost than the stopgrant state. Technology on the horizon, such as phase change memory [26], and systems research, such as automatically transitioning hardware [24], will help reduce the transitioning costs further.

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

4. EVALUATION

In this section, we compare CS and AIS using actual end-to-end response time and high resolution energy measurements taken on a Hadoop cluster. We used sort and scan jobs as was used in [22, 28]. This section largely focusing on *single-user latency-sensitive* environments. This type of environment can be found using Hadoop-On-Demand by Yahoo! that partitions user specific virtual cluster partitions of the physical cluster [1]. An evaluation of *multi-user throughput-sensitive* environments is found in Section 4.3.2 and Appendix C.

4.1 Experimental Background

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

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

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

We used the distributed Grep and the Terasort workload. We chose these two workloads because of their striking differences and also because other studies [22, 28] have relied on these workloads. The distributed Grep workload is a map-only file scan job with variable selectivity and requires little additional space on disk. For the Grep workload, we ran a three character query against the Teragen dataset as was done in [28]. The Terasort workload stresses both map and reduce components of the MR framework. It needs to read and write significant amounts of intermediate data and the size of the output is equal to the size of input.

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

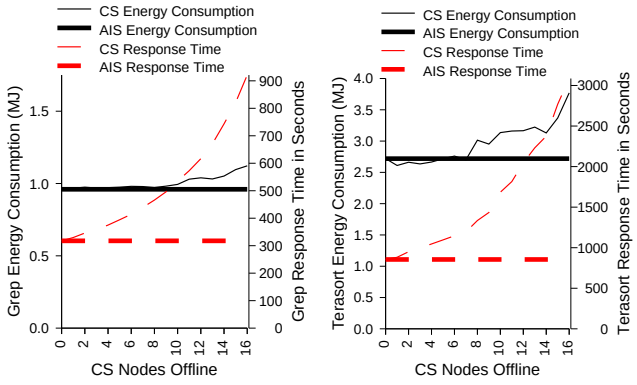
We also ran the workloads on a 96 node cluster with a 4X larger data set and measured the response time (though not the power as we did not have enough instruments to accurately measure power for 96 nodes). The response time behavior on 96 nodes is similar to the 24 node case, and both matched the analytical model (found in Appendix B). In the interest of space we do not present any results from the 96 node runs in this paper.

4.2 Workload-Only Evaluation

Now, consider a best-case scenario for both CS and AIS in which the MR cluster already exists in the state that the strategy needs (see Section 2). In other words, for CS, the system has already powered down the desired number of nodes. For AIS, the cluster is fully powered up. Since our framework expects the cluster to be returned to the state in which it originated, the best-case scenario means that no transitioning costs are needed by either method and there is no idle time (Equation 1 in Section 2). In other words, we only measured the actual time and energy consumed when running the actual workload (we relax this assumption in the next experiment).

Figures 1 (a) and (b) show our actual measured CS and AIS results for the Grep and Terasort workloads respectively. In each figure, the workload energy consumption is plotted on the left y-axis and the response time on the right y-axis.

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



(a) Grep (b) Terasort

Figure 1: 77GB Grep and Terasort workload (no transitioning/idle) response time and energy consumption on a 24 node cluster using CS and AIS.

a non-linear increase in response time for non-linear jobs. We can see this result in the response time curves of Figures 1 (a) and (b). Grep in Figure 1 (a) follows a response time degradation exactly proportional to $M = N/(N-i)$ for an N node cluster with i nodes powered down. Similarly, Terasort in Figure 1 (b) shows a response time degradation proportional to $MlnM$ consistent with sort complexity (see analysis in Appendix A.4).

Our measured energy results show that CS steadily consumes *more* energy to run the same workload with fewer online nodes. Figure 1 (b) shows that this increase for Terasort is about 39% between *performance mode* (all nodes being powered up) and when all the non-CS nodes are powered down. Since in this scenario, AIS consumes the same amount of energy as performance mode, AIS is up to 39% more energy efficient than CS for Terasort. For linear Grep, AIS is 17% more energy efficient than CS when all the non-CS nodes are powered down (Figure 1 (a)). This is because CS' offline nodes still draw 10W (Table 2).

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

4.3 Workloads with Idle Periods

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

4.3.1 Latency-sensitive Workloads

Now let us consider a scenario in which CS and AIS need to transition nodes to minimize idle energy costs (114W/node), which would happen when the cluster is underutilized. Consider a 1032 second window, which is the time it takes for CS to run the Grep workload without idle cost, including powering down all non-CS nodes (11s), execute the workload (921s with 8 online nodes), and then powering up 16 nodes to return to performance mode (100s). If we can not power down 16 nodes (due to performance limitations as discussed in Section 2), the workload will finish and the cluster will consume idle energy. In performance mode, CS runs Grep in 318s and then it will have to idle 24 nodes for the rest of the period which consumes tremendous amounts of energy as shown in Figure 2 – see the bar corresponding to zero nodes powered down.

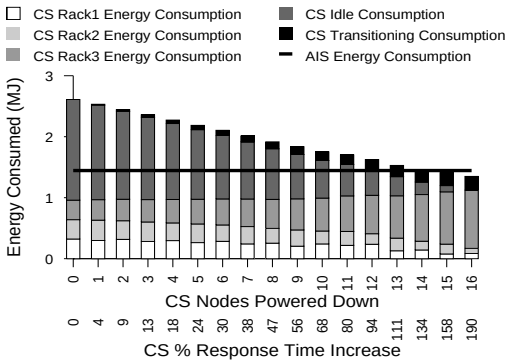


Figure 2: Cluster energy consumption (transitioning and idle) of Grep over a 1032s time window.

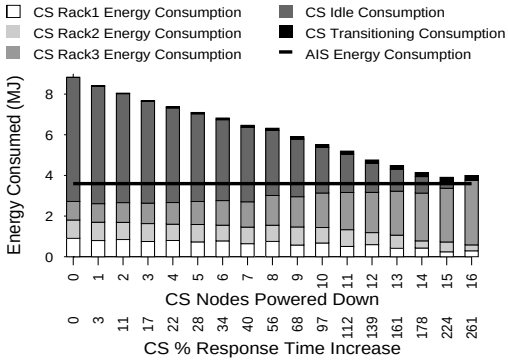


Figure 3: Cluster energy consumption (transitioning and idle) of Terasort over a 3197s time window.

If CS powers down nodes prior to running Grep, Grep will take longer to return and the idle costs diminish but transition costs increase (shown in Figure 2 where the black bars increase as more nodes are powered down). When 16 nodes are offline, there is no idle energy cost.

Similarly, in Figure 3 we present the same analysis for Terasort but over a 3197 second window where CS can run Terasort on 8 nodes (3086s) and perform all round-trip transitions (111s) to performance mode. Similarly, the energy consumption over the window decreases as we lengthen the workload running time to fill 3197 seconds and erase all idle time cost.

In Figures 2 and 3, the energy consumption of AIS during the respective time windows includes the cost to power up the 24 node cluster, run the workload, power it back down, and draw 10W per node (see Table 2) while they are powered down for the rest of the time period. For Grep, AIS consumes less energy during the 1032s window than CS most of the time until CS powers down 13 or more nodes. Since AIS has the overhead of transitioning, this cost makes AIS less desirable for this short workload. However, for the Terasort workload, we notice that AIS *always* consumes less energy during the 3197s window. Due to the complexity of the workload, AIS' overhead costs are less than the energy consumption increases of CS. Consequently, AIS saves 10% in energy over CS even at CS' more efficient operating state.

Consider an example of a scenario where the response time performance requirements (Equation 3) cause CS to consume more energy than AIS because it cannot power down sufficient nodes to eliminate idle cost. Let the tolerable level of Grep response time degradation be 50% ($\tau = 450s$). Figure 2 shows that during this underutilized period, CS will consume 33% *more* than AIS because CS can only power down 8 nodes and draws idle power.

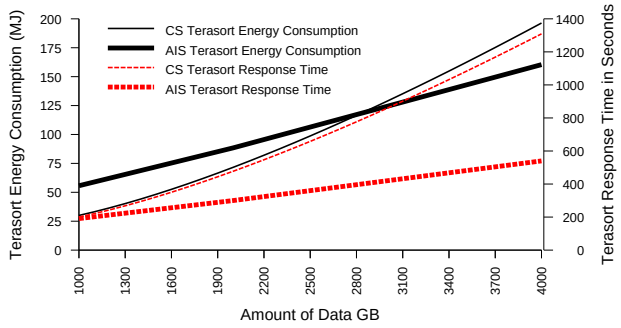


Figure 4: Analytic comparison between CS and AIS response time and energy cost for Terasort. CS uses 50% of the 2000 node cluster. Hibernate parameters are found in Table 2. Average operating power is 150W for each online node.

This problem is even worse with a super-linear complexity job such as Terasort. If acceptable response time $\tau = 1300s$ (in Equation 3) is 1.5X the performance mode response time, then CS can only power down 7 nodes (less than for Grep). Then Figure 3 shows that CS consumes 80% *more* than AIS!

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

4.3.2 *Throughput-sensitive Workloads*

We have also evaluated AIS and CS on throughput-sensitive workloads, and observed that AIS can save significantly more energy than CS given a fixed level of throughput degradation. We ran a heterogeneous workload of Grep and Terasort jobs (similar to [22]) and used CS and AIS to manage the cluster energy consumption. In our results we found that when AIS batches jobs, it consumes 26% less energy than CS given a acceptable throughput degradation (3%). These results are not surprising, and follow the same intuition from the evaluations in Section 4.3; CS results in rapid response time degradation which impacts both the energy consumption and throughput. In the interest of space, additional details can be found in Appendix C.

4.4 Effects of Workload and Hardware

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

The response time of any AIS job is simply the performance mode (all nodes online) response time plus transitioning time. Similarly the energy cost for AIS is simply the performance mode cost plus the transitioning costs. For CS, we have shown that the workload complexity determines the response time when nodes are powered down (Section 4.2). Energy modeling for CS similarly requires incorporation of the workload complexity and also the transitioning costs. For space, the details and accuracy of our models can be found in Appendix B.

Given the results of Section 4.3, we have shown that the main factor that affects AIS is the transitioning costs that it has to incur. But the question is how does this transitioning cost affect AIS' potential advantage over CS?

To explore this question, let us model the differences between CS and AIS when we have powered down 50% of a 2000 node cluster. Furthermore, we increase the amount of data that needs to be processed given constant transitioning parameters such as the node power up time. We assume that each node draws 150W when

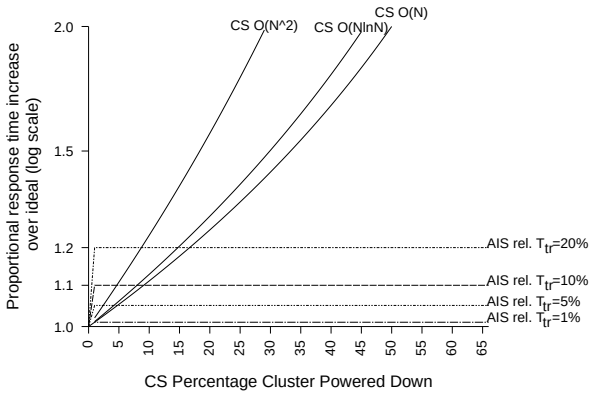


Figure 5: An analysis on the effect of workload complexity and relative T_{tr} on CS and AIS workload response time.

running a job (an average from our empirical results) and the cluster nodes have the hibernate transitioning characteristics as in Table 2.

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

Figure 4 shows the energy consumption and response time of CS and AIS for Terasort on this 2000 node cluster as we increase the amount of data to be sorted. As the results in Figure 4 show, the relative rather than the absolute transition time is the important factor since as we increase the workload length, AIS' transitioning penalties will be overcome. Thus, the relative transitioning time is a significant factor in determining the feasibility of AIS. When running a 1TB sort job with half the nodes in the cluster, CS provides better energy efficiency and response time characteristics than AIS. However, at this point, $T_{tr} = 111s$ is about half of the performance mode response time ($200s$). Now as the data size increases, the workload response time increases while the T_{tr} factor remains constant. As this happens, the advantage of AIS becomes apparent – beyond a 2.8TB data set (1.4GB/node), AIS is both faster and more energy efficient than CS.

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

Figures 5 and 6 compares the response time and energy consumption characteristics respectively, of both methods for workloads with varying computational complexity, as we increase the proportion of the cluster that is powered down by CS. In these figures, we show four different cases for AIS, with *relative* T_{tr} values of 1%, 5%, 10%, and 20%. Since our observed transitioning power (P_{tr} in Equation 1) is approximately equal to workload power (P_w^n), the rel. T_{tr} also translates to the relative increase in AIS energy consumption.

In these figures, we have presented the proportional increase in response time and energy consumption for both CS and AIS over an "ideal" case in which the hardware is perfectly energy-proportional, for three different classes of jobs with linear, sort, and quadratic computation costs.

From Figure 5, we observe that across all workloads, even with the largest 20% relative T_{tr}, AIS generally has a better workload response time than CS. This is not surprising as AIS runs the workload in *performance mode*, and its response time degradation is based only on the transitioning overhead. AIS has worse response time than CS only when the relative T_{tr} is very large.

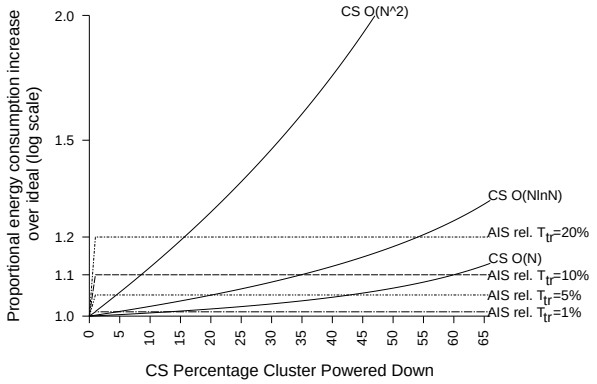


Figure 6: An analysis on the effect of workload complexity and relative T_{tr} on CS and AIS workload energy consumption.

Looking at the energy consumption in Figure 6, we notice that if the relative T_{tr} is large (e.g., 20%), then AIS will consume too much energy in transitioning, and will only be more effective with large complex workloads, where the computational complexity of the workload is high (e.g., polynomial or worse). AIS' energy consumption drops rapidly as the relative T_{tr} decreases. For example, with a relative T_{tr} of 1%, AIS is the preferred strategy for all three workloads. In this environment, for a sort workload, if the cluster is powered down by 66% by CS, then AIS saves more than 30% in energy consumption over CS.

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

Table 3 presents a two dimensional summary of the factors that affect the energy efficiency of AIS and CS. These factors are the computational complexity of the workload and efficiency of node transitioning (T_{tr}). Table 3 shows that AIS is favoured when the workload computational complexity is high. Furthermore, when AIS and CS provide about the same benefits (linear-rel. $T_{tr} = 5\%$ and sort-rel. $T_{tr} = 10\%$), AIS is preferred when the fraction of idle time is high and CS needs to power down a large proportion of the nodes. Finally, when the T_{tr} factor is small, AIS is preferred even when the computational complexity is linear. Of course, this summary is caveated with the assumption that the CS response time degradation is acceptable. As shown in Section 4.3, if the CS response time is unacceptable, AIS is the preferred method.

4.5 Discussion

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

4.5.1 Drawbacks of CS

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

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

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

Table 3: Summary of the two main factors that discriminate CS from AIS: Workload Complexity and Relative Transitioning Cost. This summary is based on the workload energy consumption since the response time performance of AIS is better than that for CS in the vast majority of cases.

Response Time Degradation – As discussed in Section 4.3, Figure 2 and 3 shows that CS can only save energy when it commits exactly the right amount of resources such that all the idle time in a given time window is erased (Equation 1). However, this requires that the workload is willing to tolerate a potentially large response time penalty (constraint τ in Equation 3). If this response time penalty is not acceptable and CS must commit more resources and incur more idle energy, Figure 2 and 3 shows that AIS will consume less energy than CS for the majority of the cases. Section 4.3 shows that with an acceptable 50% increase in response time, AIS can save up to 80% of CS' consumption.

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

4.5.2 Hybrid Approaches

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

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

5. RELATED WORK

The problem of increasing energy consumption in large-scale data processing environment has received considerable attention in the context of data center construction and operation [11, 16, 17, 23, 27]. All these efforts have resulted in dramatic improvements in the energy efficiency of data centers, and can largely be used orthogonally to software methods to reduce energy consumption.

A desired property for systems is energy proportionality, which is currently lacking in modern servers [5]. Major components, such as CPU and disk, are now under high scrutiny for improving their energy characteristics under varying utilizations [5, 10, 36]. Software efforts such as the Tickless Kernel project, aim to change the way the OS kernels operate when the server is idle [37]. Efforts by the systems community to develop energy efficiency metrics can be found in [2, 18, 34, 35]. Recent work has examined how direct CPU power control mechanisms can effect energy savings and workload response time [19] while single node database energy efficiency was discussed in [40]. Chen et al. recently presented a study on MR operating variables [12].

Studies into shutting down online web servers were discussed in [30, 32]. Shutting down a replicated parallel database environ-

ment was analyzed in [20]. Other related methods [29, 31] either rely on learning request skew, specialized hardware, and data migration. Increasing utilization can be done by consolidation using a virtual machine (VM) solution [3, 13, 33, 39]. However, using VMs when running data intensive services, like the ones we consider in this paper, is challenging for a number of reasons such as performance penalties from VM overhead, homogeneous performance from heterogeneous hardware [25], and costs of VM migration and overprovisioning. Weddle et al. [42] described a RAID-based system to turn off disks to save energy.

As we have mentioned in our discussions on reducing cluster node transitioning costs, recent discussions on fundamentally new cluster server design are highly relevant to ideas such as CS and AIS [17, 21, 24, 41]. Hardware advances, such as low-power non-volatile Phase-Change Memory [26], solid state/flash memory, and large arrays of cheap low power processors such as Intel's Atom, may be the key towards achieving cost-effective, energy-efficient servers that transition between online and offline states efficiently.

6. CONCLUSIONS

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

7. ACKNOWLEDGEMENTS

We would like to thank the reviewers of this paper for their constructive comments. This research was supported in part by a grant from the Microsoft Jim Gray Systems Lab, Madison, WI and also in part by the National Science Foundation under grant IIS-0963993.

8. REFERENCES

[1] http://www.infoq.com/articles/hadoop-interview.

[2] SPEC Power. http://www.spec.org/power_ssj2008.

[3] VMware Infrastructure Architecture Overview White Paper. http://www.vmware.com/pdf/vi_architecture_wp.pdf.

[4] Report To Congress on Server and Data Center Energy Efficiency. In *U.S. EPA Technical Report*, 2007.

[5] L. A. Barroso and U. Hölzle. The Case for Energy-Proportional Computing. *IEEE Computer*, 40(12), 2007.

[6] C. Belady. In the Data Center, Power and Cooling Costs More than the IT Equipment it Supports. *Electronics Cooling*, 23(1), 2007.

[7] R. Bianchini and R. Rajamony. Power and Energy Management for Server Systems. In *IEEE Computer*, 2004.

[8] D. Borthakur. The Hadoop Distributed File System: Architecture and Design. In hadoop.apache.org, 2007.

[9] K. G. Brill. Data Center Energy Efficiency and Productivity. In *The Uptime Institute - White Paper*, 2007.

[10] E. V. Carrera, E. Pinheiro, and R. Bianchini. Conserving Disk Energy in Network Servers. In *ICS*, 2003.

[11] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing Energy and Server Resources in Hosting Centers. In *SOSP*, 2001.

[12] Y. Chen, L. Keys, and R. Katz. Towards Energy Efficient Hadoop. In *UC Berkeley Technical Report*, number UCB/EECS-2009-109, 2009.

[13] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *NSDI*, 2005.

[14] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, 2004.

[15] S. Ghemawat, H. Bobioff, and S.-T. Leung. The Google File System. In *SOSP*, 2003.

[16] J. Hamilton. Where Does Power Go In DCs and How To Get It Back? http://mvdirona.com/jrh/TalksAndPapers/JamesRH_DCPowerSavingsFooCamp08.ppt, 2008.

[17] J. Hamilton. Cooperative Expendable Micro-slice Servers (CEMS): Low Cost, Low Power Servers for Internet-Scale Services. In *CIDR*, 2009.

[18] S. Harizopoulos, M. A. Shah, J. Meza, and P. Ranganathan. Energy Efficiency: The New Holy Grail of Database Management Systems Research. In *CIDR*, 2009.

[19] W. Lang and J. M. Patel. Towards Eco-friendly Database Management Systems. In *CIDR*, 2009.

[20] W. Lang, J. M. Patel, and J. F. Naughton. On Energy Management, Load Balancing and Replication. In *SIGMOD Record*, 2009.

[21] W. Lang, J. M. Patel, and S. Shankar. Wimpy Node Clusters: What About Non-Wimpy Workloads? In *DaMoN*, 2010.

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

[23] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments. *ISCA*, 2008.

[24] D. Meisner, B. T. Gold, and T. F. Wenisch. PowerNap: Eliminating Server Idle Power. In *ASPLOS*, 2009.

[25] A. Noll, A. Gal, and M. Franz. CellVM: A Homogeneous Virtual Machine Runtime System for a Heterogeneous Single-Chip Multiprocessor. In *Workshop on Cell Systems and Applications*, 2008.

[26] Numonyx Memory Solutions. www.numonyx.com/Documents/WhitePapers/PCM_Basics_WP.pdf.

[27] C. D. Patel and A. J. Shah. Cost Model for Planning, Development and Operation of a Datacenter. http://www.hpl.hp.com/techreports/2005/HPL-2005-107R1.pdf.

[28] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A Comparison of Approaches to Large-Scale Data Analysis. In *SIGMOD*, 2009.

[29] E. Pinheiro and R. Bianchini. Energy Conservation Techniques for Disk Array-Based Servers. In *ICS*, 2004.

[30] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems. In *Workshop on Compilers and Operating Systems for Low Power*, 2001.

[31] E. Pinheiro, R. Bianchini, and C. Dubnicki. Exploiting Redundancy to Conserve Energy in Storage Systems. In *SIGMETRICS*, 2006.

[32] K. Rajamani and C. Lefurgy. On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters. In *IEEE ISPASS*, 2003.

[33] P. Ranganathan, P. Leech, D. Irwin, and J. Chase. Ensemble-level Power Management for Dense Blade Servers. In *ISCA*, 2006.

[34] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis. JouleSort: a balanced energy-efficiency benchmark. In *SIGMOD*, 2007.

[35] S. Rivoire, M. A. Shah, P. Ranganathan, C. Kozyrakis, and J. Meza. Models and Metrics to Enable Energy-Efficiency Optimizations. *Computer*, 2007.

[36] S. Sankar, S. Gurumurthi, and M. R. Stan. Intra-disk Parallelism: An Idea Whose Time Has Come. In *ISCA*, 2008.

[37] S. Siddha, V. Pallipadi, and A. V. D. Ven. Getting Maximum Mileage Out of Tickless. In *Linux Symposium*, 2007.

[38] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 2005.

[39] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu. Delivering Energy Proportionality with Non Energy-Proportional Systems - Optimizing the Ensemble. In *HotPower*, 2008.

[40] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the Energy Efficiency of a Database Server. In *SIGMOD*, 2010.

[41] V. Vasudevan, J. Franklin, D. Andersen, A. Phanishayee, L. Tan, M. Kaminsky, and I. Mararu. FAWNdamentally Power-efficient Clusters. In *USENIX*, 2009.

[42] C. Weddle, M. Oldham, J. Qian, A. Wang, P. Reiher, and G. Kuenning. PARAID: A gear-shifting power-aware RAID. *Trans. Storage*, 2007.

APPENDIX

A. CS POWER DOWN STRATEGIES

This section details some of the different power down strategies for the CS approach. A CS power down strategy's main goal is to be simple and help maintain predictable response time degradation. As [22] did not detail a power down strategy, we first detail our observations of a purely random power down order.

A.1 Random Power Down

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

However, randomly selecting non-CS nodes for powering down can result in suboptimal performance, as explained below. To begin this discussion, consider a distributed Grep workload on a 24 node Hadoop cluster. (More details about the system setup and workload can be found in Section 4.1.) In this case, the system has three racks and each rack had 8 nodes. The CS set was set to the nodes in the third rack. (Similar issues as those described below happen, if the CS nodes are spread across the racks.)

Now, consider selecting nodes at random for powering down from the two non-CS racks. Figure 7 shows the effect on response time for the Grep workload as nodes are powered down. Also plotted in this figure is the theoretical ideal response time curve ($N/(N-i)$) for Grep. As can be seen from this figure, there is a significant degradation in response time when the 9th node is powered down. The reason for this degradation is as follows: first, recall that for each data block, HDFS keeps one replica on a node on the same rack, and another replica on a node on another rack. Second, because of the HDFS replication policy, a natural way to produce a CS node set is to allocate an entire rack to the CS nodes (in our 3 rack case). Third, Hadoop tries to schedule Map and Reduce jobs so that they work on the data that is local (called "data local" tasks), but the Hadoop scheduler will assign tasks to work on remote blocks if some nodes have no additional unprocessed local blocks. These remote tasks incur additional overhead as they interfere with the disk activity at the remote node (which is presumably running a data local task), and incurs additional delays because of the network activity. Fourth, as non-CS nodes are powered down, the probability that nodes in the CS rack have the only copy of the data increases. Finally, if by chance there is a disproportionate number of nodes in one non-CS rack that are turned off, then the chance that some node in the CS rack will end up with a disproportionately larger number of single replica blocks increases. This node will then be the bottleneck as some blocks on that node will probably have to be fetched remotely by other nodes for processing. In fact, this is precisely what happens in Figure 7 when the 9th node is powered down and the fraction of non-data local nodes increases rapidly over the previous case when the 8th node was turned off. Consequently, as can be seen in Figure 7, the response time degrades rapidly when the 9th node is taken down.

Thus, simple random powering down has the drawback of resulting in surprising jumps in response time.

A.2 Load Balanced (LB) Power Down

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

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

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

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

A.3 Round – Robin Random (RRR) Power Down

This scheme simply goes through the non-CS racks in a round robin fashion and selects a random node (that is not powered down) in each rack as the next "victim". Thus, in the case above, where we have two non-CS racks (see Appendix A.1) this strategy will first select, at random, a node from the first non-CS rack for powering down. In the next iteration, it will select a victim from the second non-CS rack, and in a subsequent iteration it will return back to the original first non-CS rack for victim selection, and so on. The difference between this strategy and a purely random strategy is that we do not allow any two physical racks to have their number of powered down nodes to differ by more than one. In this way, we minimize the number of single replica blocks that are created by each node power down.

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

A.4 Comparing the Power Down Schemes

Figure 8 shows the corresponding behavior of the load balanced (Appendix A.2) and the round-robin random (Appendix A.3) schemes compared to the ideal behavior (as was shown in Figure 7).

Figure 8 shows two important points. First, the response time of both the Round-Robin Random (RRR) and the Load Balanced (LB) schemes match the theoretical ideal case. Second, in this case both the RRR and LB methods have nearly identical response time. The simplicity of the RRR method (see Appendix A.3), implies that it is a better method for use with the Covering Set technique.

We have analyzed these three schemes for a 24 node Terasort workload. Unlike Grep, the Terasort workload has an $O(NlnN)$ computational complexity. Figure 9 shows similar Terasort results as in Figure 8. That is, RRR and LB provide similar response time degradation and very closely follow the ideal $O(NlnN)$.

As we have seen here, the computational complexity models are good for modeling CS response time degradation. Further, we will need these response time models for modeling CS energy consump-

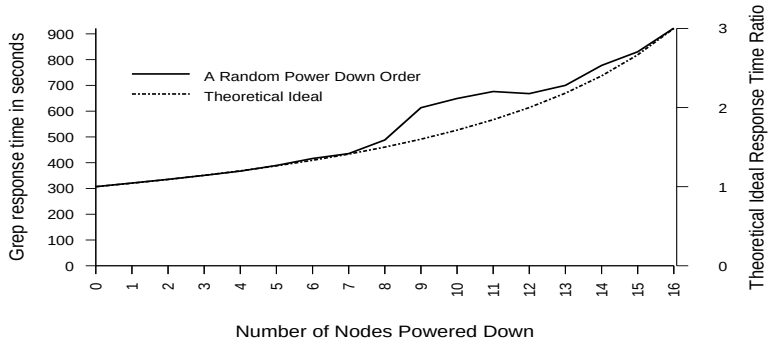


Figure 7: An example of load imbalance with a bad power down order on a MapReduce cluster using a CS data placement. A Grep workload was run on a 24 node cluster with a CS of 8 nodes. Ideally as i nodes are powered down in an N node cluster, the amount of work at each node increases by a factor of $N/(N-i)$.

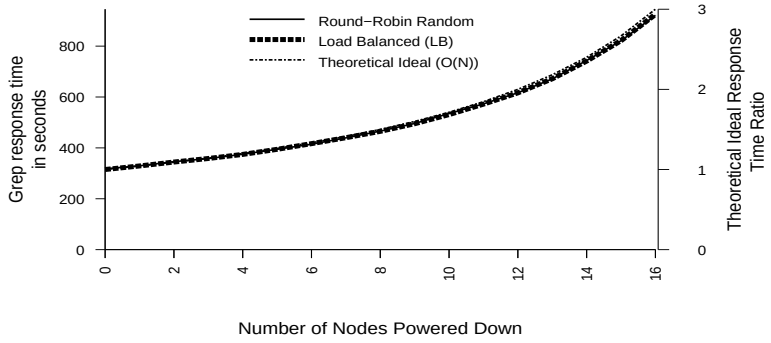


Figure 8: $77GB$ Grep workload response time on a 24 node cluster as i nodes are powered down. A comparison of the effects of the RRR and Greedy power down strategy and the ideal $O(N)$ degradation.

tion. This is discussed in Section 4.4 and Appendix B.

B. MODELING VALIDATION

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

CS on the other hand, is more complex in its modeling of response time and energy consumption. We have already presented results showing that the computational complexity of the workloads can be used to accurately model the response time degradation of CS as more nodes are powered down (Appendix A.4).

If we recall Equation 1 from Section 2, the workload energy consumption is $E_w = (P_w^n + P_w^{\bar{n}})T_w$, which considers the power drawn by both the online and offline nodes during the actual workload execution. The effect of CS powering down nodes is that the idle energy shrinks, and eventually reduces to zero. As CS powers down more nodes, (P_w^n) decreases while ($P_w^{\bar{n}}$) increases. Using Equation 1, we can model the energy consumption of the workload under CS when we substitute T_w with the workload complexity models just described. Any transitioning and idle costs are straightforward to include as we just add them to the workload cost. For simplicity, we do not include them in our modeled analysis of CS.

Figure 10 shows a comparison of the observed and modeled energy consumption of the Grep and Terasort workloads using the workload energy consumption $(P_w^n + P_w^{\bar{n}})T_w$ from Equation 1. For this figure, we used the power data shown in Table 2, along with using O(N) and O(NlnN) complexities to model the response times of Grep and Terasort respectively. The results shown in Figure 10 demonstrate that the models are quite accurate in predicting the energy consumption of CS: an average error of 1% and 4% for Grep and Terasort jobs respectively.

C. THROUGHPUT – SENSITIVE WORKLOADS

In this section we discuss empirical results showing the abilities of CS and AIS to handle a throughput-sensitive MapReduce cluster. Here we are trading throughput for energy efficiency instead of the response time/energy efficiency trade-offs, as in Section 4.

For AIS, the method that we employ is the batching method described in Section 3.3. AIS keeps the MR cluster powered down while jobs are batching [19, 38]. When enough MR jobs are collected, AIS powers up the cluster and submits all the jobs. The job collection, or batching delay, effectively degrades throughput.

In contrast, CS runs the jobs as they arrive but can process them

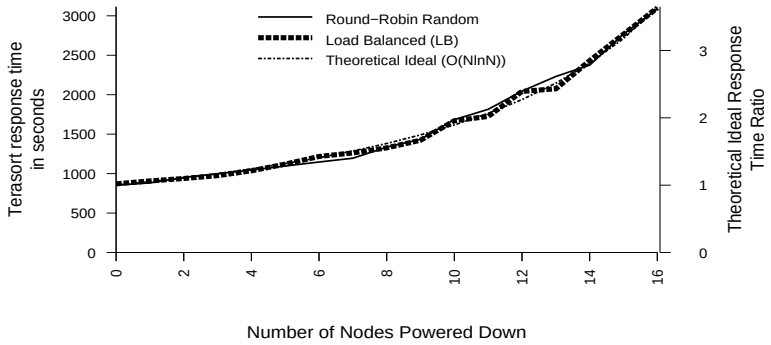


Figure 9: $77GB$ **Terasort workload response time on a** 24 **node cluster as** i **nodes are powered down. A comparison of the effects of the RRR and Greedy power down strategy and the ideal** $O(NlnN)$ **degradation.**

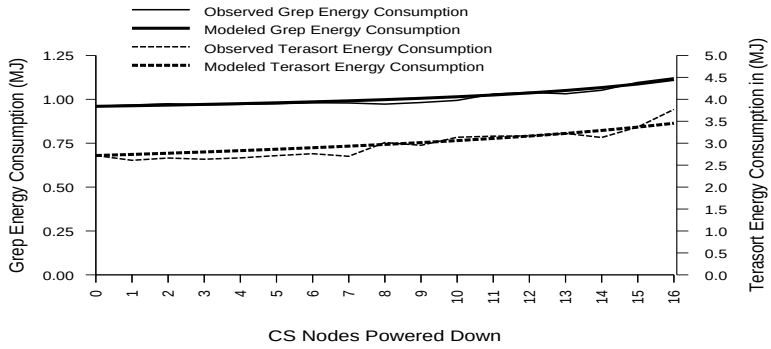


Figure 10: Comparing the observed and modeled energy consumption of the Grep and Terasort workloads for CS. The average error is 1% and 4% for the Grep and Terasort models respectively.

with some MR cluster nodes powered down. However, as fewer nodes are available for the job, this also lowers throughput.

Our throughput workload mimics that of [22] whereby sort and scan jobs are injected into the MR cluster. We use the same Terasort and Grep jobs of Section 4 whereby each job runs on a 77GB dataset.

We evaluate a heterogeneous job workload consisting of four sort and four scan jobs, randomly ordered and individually submitted to the 24 node cluster in 850 second intervals. Recall from Section 4.2, that the Grep job can run in about 300 seconds and the Terasort job can run in 850 seconds.

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

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

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