Energy Proportionality and Performance in Data Parallel Computing Clusters

Jinoh Kim, Jerry Chou, and Doron Rotem
Lawrence Berkeley National Laboratory
University of California, Berkeley, CA 94720
{jinohkim,jchou,d_rotem}@lbl.gov

***Abstract*—Energy consumption in datacenters has recently become a major concern due to the rising operational costs and scalability issues. Recent solutions to this problem propose the principle of *energy proportionality*, i.e., the amount of energy consumed by the server nodes must be proportional to the amount of work performed. For data parallelism and fault tolerance purposes, most common file systems used in MapReduce-type clusters maintain a set of replicas for each data block. A *covering set* is a group of nodes that together contain at least one replica of the data blocks needed for performing computing tasks. In this work, we develop and analyze algorithms to maintain energy proportionality by discovering a covering set that minimizes energy consumption while placing the remaining nodes in low-power standby mode. Our algorithms can also discover covering sets in *heterogeneous* computing environments. In order to allow more data parallelism, we generalize our algorithms so that it can discover k-covering sets, i.e., a set of nodes that contain at least k replicas of the data blocks. Our experimental results show that we can achieve substantial energy saving without significant performance loss in diverse cluster configurations and working environments.**

I. Introduction

Energy consumption in scientific and commercial datacenters has increased dramatically with the introduction of high-performance, power-hungry components, such as multicore processors, high capacity memories, and high rotational speed disks. Therefore, the mounting costs of energy in datacenters has recently become a major concern. It is now estimated by EPA that in 2011 datacenters will consume up to 3% of the total energy in the U.S., while their energy consumption is doubling every 5 years [18]. Despite the technological progress and the amount of capital invested, there are significant inefficiencies in datacenters with server utilization measured at around 6% [16]. In this paper, we focus on optimizing energy consumption of compute clusters in datacenters, such as MapReduce clusters [10] often used in scientific computation [11]. The key idea is to achieve this by placing underutilized components in lower power consumption states (i.e., standby mode).

Optimizing energy consumption in datacenters introduces several challenges. As pointed out in [21], [13], [4], *heterogeneity* of cluster nodes may be inevitable due to gradual replacement or addition of hardware over time. The replaced or added hardware should be "brand-new" rather than the same as the old one. Cluster heterogeneity can also be a result of a design choice. For example, the authors of [7] presented a hybrid datacenter model with two-class nodes that have different performance capabilities and power requirements for energy efficiency. In a recent work [21], heterogeneity in a MapReduce cluster was considered for job scheduling and performance improvement. There are several recent research efforts dealing with energy management for MapReduce clusters [15], [14], but heterogeneity in such clusters has not been considered yet. In this paper, we examine how energy consumption can be further optimized by taking into account the different power requirements of the nodes in the cluster.

Another important requirement for energy management is *energy proportionality*, i.e., the ability to adjust energy consumption in proportion to the given workload. As mentioned in [2], server systems consume a substantial amount of energy even in idle mode (over 50% of the peak), although it could be ideally zero. Thus, a datacenter cluster still needs to consume a great deal of energy even under a very low load (e.g., at midnight), since the cluster nodes require substantial power even when no real work is done. Energy-proportionality can be a great benefit in conserving energy especially in clusters with a high degree of load variation, such as the one described in [6] where variations of over a factor of three between peak loads and light loads have been observed. This paper focuses on those two challenges, cluster heterogeneity and energy proportionality in data parallel computing clusters.

One known approach for cluster energy saving is achieved by powering on/off nodes in response to the current workload. For example, we could use cluster nodes in part to handle light loads, and save energy by deactivating the rest of the nodes not in use. In this work, we study the problem of determining which nodes should be activated or deactivated whenever it is determined that workload characteristics have changed.

More specifically, this work focuses on identifying a set of nodes that minimizes energy costs while satisfying immediate data availability for a data set required in computing. This is important since the cost of demand-based power state transitions of nodes for missing data blocks is significant in terms of both energy and performance due to the long latency needed to transition back from standby to active mode. For example, dehibernating (transitioning from standby to active mode) may require 129W for a duration of 100 seconds [14], for a node consuming 114W in idle mode. In a heterogeneous setting, such power requirements can be different from one node to another. To address this, we establish a power consumption

profile for each node, and use this information in locating an optimal node set. In this paper, we refer to a group of nodes that together contain at least one replica of the data blocks needed for performing computing tasks as a CS (*covering subset*).

For high performance computing, the degree of data availability has a critical role in determining the degree of data parallelism. To consider this, we extend our node discovery algorithms to guarantee a certain degree of data availability. In its simplest form, our node discovery algorithm searches for a node set holding a single replica of the data. However, we may need a node set that has more than a single replica for each data item for certain situations. For example, for satisfying performance dictated by service level requirements we may need to activate a node set containing *two* replicas for supporting intermediate loads, rather than using a node set with a single replica.

Our key contributions are summarized as follows:

- We provide mathematical analysis of minimal CS size under the assumption of a uniform data layout as a function of the number of data blocks. We also show the validity of the theoretical model by simulation.
- We present node set discovery algorithms that find an energy-optimized node set with data availability for all required data items, for homogeneous and heterogeneous settings.
- We extend our discovery algorithms to identify a node set with any required degree of data availability, as a means of energy-proportional cluster reconfiguration.
- We present our evaluation results with respect to energy consumption and performance with a rich set of parameter settings. The results show that our techniques can achieve substantial energy saving without significant performance loss in most light workload environments. Also, we show that our power-aware technique can exploit heterogeneity successfully, yielding greater energy saving.

The paper is organized as follows. We first briefly introduce several closely related studies in the next section. In Sections 3 and 4 we present our algorithms for node set discovery for data availability and energy proportionality. In Sections 5 and 6, evaluation setup and results are presented with a rich set of parameters in diverse working environments. We finally conclude our presentation in Section 7. Additionally, the appendix provides the benefits of the CS approach in terms of energy saving.

II. Related Work

The initial work for MapReduce cluster energy management was performed in [15] based on covering subset (CS). In that work, the CS nodes are manually determined, and one replica for each data item is then placed in one of the CS nodes. Under a light load, it would be possible save energy by running the cluster with only the CS nodes activated. To enable this, the authors modified the existing replication algorithm, such that the CS nodes contain a replica of each data item. Failure of CS nodes was not considered, and as a result, any single node failure can make this scheme ineffective. Also, there was no notion of energy proportionality with gradual adjustment; rather the cluster is in either full performance mode with the entire set of nodes activated or in energy mode with only the CS nodes activated.

AIS (All-in Strategy) [14] is a different approach. AIS runs given jobs employing the entire set of nodes in the cluster to complete them as quickly as possible. Upon completion of the jobs, the entire set of nodes are deactivated to save energy until the next run. This makes sense since data parallel clusters are often used for batch-oriented computations [9]. One potential drawback can be that even with small (batched) jobs, AIS still needs to wake up the entire cluster, possibly wasting energy. Both studies (static CS and AIS) did not consider cluster heterogeneity, as we do in this work.

Rabbit [1] provides an interesting data placement algorithm for energy proportionality in MapReduce clusters. The key idea is to place data items in a skewed way across the nodes in the cluster. More specifically, node k needs to store b/k data items, where b is the total number of data items. Thus, a lower-indexed node has a greater number of data items, and it makes it possible to deactivate a higher-indexed node safely without losing data availability. Energy proportionality is also provided by allowing one-by-one node deactivation. Our approach provides energy management for clusters with the existing data layout, while Rabbit introduces its own method of data placement for energy management. Rabbit also does not consider possibility of cluster heterogeneity.

Cardosa et al. considered energy saving in a VM (Virtual Machine)-based MapReduce cluster [5]. Their approach is to place VMs in a timely balanced way, and find a way to minimize the number of nodes to be utilized, so as to maximize the number of nodes that can be idle. Subsequently, idle nodes can be considered as candidates for deactivation to save energy. One essential assumption in this work, that may not be practical, is the availability of a tool for accurate running time estimation for VMs.

Minimum weighted set cover is a classic NP-complete problem. In the problem, we are given a set U of n elements and a collection F of subsets of U each associated with a positive weight. A set cover of U is a collection of subsets, F', of F where the union of the subsets in F' is U. The weight of a cover F' is the sum of the weights of the subsets in it. The problem objective is to find a set cover with the minimum weight. The greedy set cover algorithm is often used since it is NP-complete. The greedy algorithm first selects the most cost-effective subset, i.e., the subset whose cost per element is smallest, and adds that subset to the solution while removing the covered elements and the subset from further consideration. This process is repeated on the remaining subsets and elements until all elements are covered. This simple heuristic was proven to produce a set cover with cost at most a factor of H_n of the minimum cost set cover, where $H_n = O(logn)$ is the nth harmonic number equal to $1 + \frac{1}{2} + \cdots \frac{1}{n}$, and n is the cardinality of U [8].

TABLE I
NOTATIONS

Symbol	Description	Symbol	Description
N	Cluster node set	B	Data block set
CS	CS node set ($CS \subseteq N$)	NCS	non-CS node set ($NCS \subseteq N$)
n	Cluster size	b	Number of data blocks
r	Replication factor	f	Fraction of low-power nodes
$P^{(i)}$	Idle power	$T^{(i)}$	Idle time
$P^{(a)}$	Active power	$T^{(a)}$	Active time
$P^{(p)}$	Peak power	$T^{(p)}$	Peak time
$P^{(s)}$	Standby power	$T^{(s)}$	Standby time
$P^{(u)}$	Activating power	$T^{(u)}$	Activating time
$P^{(d)}$	Deactivating power	$T^{(d)}$	Deactivating time

III. NODE SET DISCOVERY ALGORITHMS

In this section, we present our node discovery algorithms for a set of nodes that minimizes energy consumption subject to data availability constraints. As in [15], we refer to this node set as CS (Covering Subset). We assume that the data set statistics for the next round of computation is readily available and therefore discover the CS based on that information. This leads to a slightly different definition of CS as compared with the definition in [15]. The CS used here is not a static node set, rather it is discovered on demand based on a given list of data blocks required for computation. Thus, our CS must contain a replica for *required* data items instead of the *entire* set of data blocks in the cluster. Since data parallel computing platforms are often used for batch-style processing [9], [14], the data set can be available for the next operational time window.

In this section, we first present a basic algorithm for node discovery that searches a minimal number of nodes for data availability, and then extend it with an energy metric for heterogeneous settings. Table I summarizes notations we used in this paper.

A. A basic method for CS discovery

By definition, CS maintains at least one replica of the required data blocks. Locating such a set is NP-complete as it can be reduced to the well known *set cover* problem [8], as described in the following proposition.

Proposition 3.1: A minimum CS discovery problem $CS(B, S)$ with B required blocks and a set of servers S is NP-complete, the reduction is from a minimum set cover problem $SC(U, F)$, where U is a universe of elements and F is a family of subsets of U.

Proof: We omit the proof since it is trivial. ■

The greedy algorithm for CS discovery is shown in Algorithm 1. In the algorithm, function getReplicaSet() gives the set of data blocks that the node contains.

Figure 1 plots the size of CS for a cluster with size $n = 1024$ under two replicated environments with $r = 3$ and $r = 5$, as a function of the number of required data blocks. As the number of data blocks increases, the CS size also increases. For example, with n data blocks, the CS size ranges 20–30% of the cluster for the two replication settings. This implies that it would be possible to have energy saving of up to 70–80% in this setting. The CS size grows to 60–80% of the cluster for the case where the number of data blocks is $32n$, which is $\sim 2TB$ with the default data block size in MapReduce [10] and Hadoop [12].

Input: Data block set B, Node set S
Output: Covering subset C

```
U ← B;
F ← S;
C ← ∅;
while U ≠ ∅ do
    Select node i ∈ F that maximizes
    |U ⋂ i.getReplicaSet()|;
    U = U − i.getReplicaSet();
    C ← C ⋃ {i};
    F ← F − {i};
end
return C
```

Algorithm 1: ConstructCS(B, S)

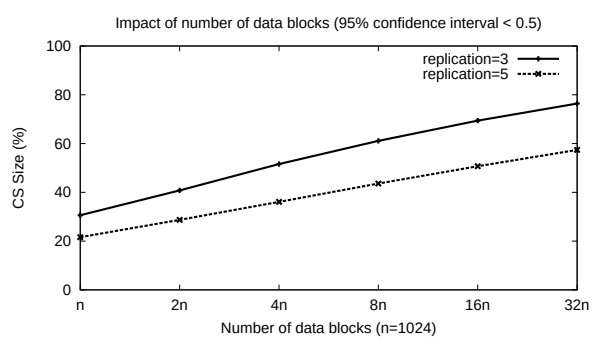


Fig. 1. CS size with respect to the number of data blocks

We briefly discuss the theoretical analysis for the minimal CS size as a function of the number of data blocks in a uniform data distribution. Assume that r copies of each data block are uniformly distributed on n nodes with each node holding at most one of the r copies. As previously defined, CS is a node set that contains at least one replica of each of the given b data items.

Lemma 3.2: Let P be the probability that a randomly selected set of m nodes out of n nodes is CS. Then, P is equal to $\left(1-\prod_{i=0}^{m-1}\left(1-\frac{r}{n-i}\right)\right)^b$.

Proof: The total number of ways for selecting r nodes from the available n nodes to hold the r replicas is $\binom{n}{r}$. From these possible selections, exactly $\binom{n-m}{r}$ do not place a copy in the randomly selected m nodes. We can then calculate the probability that the selected m nodes do not have any replica of a data item d_1 as $P' = \frac{\binom{n-m}{r}}{\binom{n}{r}} = \frac{(n-m)!(n-r)!}{n!(n-m-r)!} = \prod_{i=0}^{m-1}(1-\frac{r}{n-i})$. Due to the fact that the r replicas for each of the b data items are placed independently, we get $P = (1-P')^b$, or $P = (1-\prod_{i=0}^{m-1}(1-\frac{r}{n-i}))^b$. ■

Theorem 3.3: The minimal m such that we can expect at least one CS from any given uniform data layout satisfies: $\binom{n}{m}\left(1-\prod_{i=0}^{m-1}\left(1-\frac{r}{n-i}\right)\right)^b \geq 1$.

Proof: Let $M = \{M_1, M_2, \cdots, M_\ell\}$ be the collection of all sets of size m selected from n nodes. Thus $\ell = \binom{n}{m}$. By Lemma 3.2, we know that the probability of each M_i to be a CS is P. Let X_i be a random variable where,

$$X_i = \begin{cases} 1 & \text{if } M_i \text{ is a CS,} \\ 0 & \text{otherwise.} \end{cases}$$

Then, the expected value of X_i, $E(X_i)$, is equal to P. The expected number of CS is thus,

$$\sum_{i=0}^{l} E(X_i) = \binom{n}{m} P$$

Note that this is true even though the X_i's are not independent. Therefore, the minimal m that ensures existence of at least one CS must satisfy $\binom{n}{m}P \geq 1$. ■

Figure 2 shows the minimal CS size as a function of the number of data blocks in a small system with $n = 20$. The figure compares the analytical results based on our probabilistic model and simulation results, and we can see that they agree with each other. Also, the sub-linear shape of CS size increase over the number of blocks agrees with the mathematical work studied in [20]. Note that we used rack-unaware replication for simulation to assume the equivalent setting.

As described above, the problem of our node set discovery is simply mapped to the set cover problem, and the solution is to locate a set with the minimal size covering the data items in question. However, in a heterogeneous environment where nodes may have different power metrics, locating a minimal-size set would not be sufficient. We present a *power-aware* discovery algorithm as a solution for identifying an optimal node set in a heterogeneous cluster next.

B. Power-aware discovery for heterogeneous clusters

Let us illustrate a heterogeneous cluster with a realistic example. Suppose there are 20 nodes in a cluster with 10 Xeons and 10 Atoms with power profiles as in Table II.

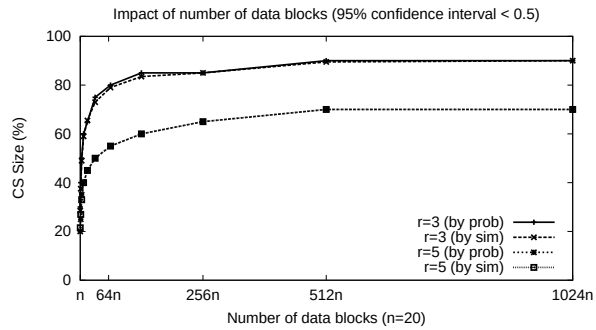


Fig. 2. Minimal CS size

We can see that Xeons consume ten times more energy than Atoms. In such an environment, a CS with two Xeon nodes as a minimal subset may require a greater power level than a CS with ten Atom nodes. The former power requirement is $2 \cdot 315W + 8 \cdot 18W + 10 \cdot 2W = 794W$ at peak, while the latter only requires $10 \cdot 33.8W + 10 \cdot 18W = 518W$. At the idle state, the former requires 683W and the latter does 436W.

However, any technique that naively selects low-power nodes for CS discovery may not work that well. For example, in the above example, if Xeons consume only half watts than that in the table, i.e., $P^{(p)} = 315/2W = 157.5W$ and $P^{(s)} = 18/2W = 9W$, where $P^{(p)}$ stands for peak power and $P^{(s)}$ does standby power, then the power requirement for a CS with two Xeons becomes $2 \cdot 157.5W + 8 \cdot 9W + 10 \cdot 2W = 407W$, which is smaller than the energy requirement for a CS with ten Atom nodes. Hence, we need a more sophisticated approach to locate an optimal CS in heterogeneous settings, as discussed next.

Formally, CS power requirement is $P_{CS}^{(a)} + P_{NCS}^{(s)}$, where $P_{CS}^{(a)}$ is power for CS in active state and $P_{NCS}^{(s)}$ is power for non-CS nodes in standby. The energy consumption (E) for a given period of time (T) is then simply $E = (P_{CS}^{(a)} + P_{NCS}^{(s)}) \times T$. If we assume that T is fixed, our objective in identifying CS is to minimize $P_{CS}^{(a)} + P_{NCS}^{(s)}$. In other words, what we want to do here is to discover nodes for CS whose aggregated energy consumption can be minimized during time period T. This can be rewritten as follows for power P:

$$\begin{aligned} P &= P_{CS}^{(a)} + P_{NCS}^{(s)} \\ &= \sum_{x \in CS} P_x^{(a)} + \sum_{y \in NCS} P_y^{(s)} \\ &= \sum_{x \in CS} \left(P_x^{(a)} + P_x^{(s)} - P_x^{(s)}\right) + \sum_{y \in NCS} P_y^{(s)} \\ &= \sum_{x \in CS} \left(P_x^{(a)} - P_x^{(s)}\right) + \sum_{x \in CS} P_x^{(s)} + \sum_{y \in NCS} P_y^{(s)} \\ &= \sum_{x \in CS} \left(P_x^{(a)} - P_x^{(s)}\right) + \sum_{y \in N} P_y^{(s)} \qquad (1) \end{aligned}$$

Since the second part in Equation 1 is a constant, we can then map the node set discovery problem in a heterogeneous

setting to a *weighted set cover* problem with an energy metric $(P_i^{(a)} - P_i^{(s)})$ as the weight associated with each node i. More precisely, the goal of the node set discovery problem can be cast as follows. Let G be the set of all possible covering subsets for a required set of data blocks. For covering subset $g \in G$, we define its weight $w(g)$ as the sum of weights of its nodes, i.e.,:

$$w(g) = \sum_{x \in g} \left(P_x^{(a)} - P_x^{(s)} \right) \tag{2}$$

Then, our goal is to find a covering subset q, such that $w(q) \leq w(g)$ for all $g \in G$.

Proposition 3.4: A minimum CS discovery problem $CS(B, S)$ in a heterogeneous setting is NP-complete, and it can be reduced to a minimum weighted set cover problem $WSC(U, F)$, where U is a universe and F is a family of subsets of U.

Proof: As in Proposition 3.1, given a CS problem $CS(B, S)$, we can construct a corresponding set cover problem $SC(U, F)$, where for each set $f_k \in F$, we set its weight to $P_k^{(a)} - P_k^{(s)}$. Let $C \subset F$ be the minimum weighted set cover of $SC(U, F)$. Define $C' = \{s_i | u_i \in C\}$, then it is easy to see C' is also the minimum weighted set of nodes covering all blocks in B. Reversely, weighted set cover can be reduced to the heterogeneous CS discovery problem, and the reduction is in polynomial time. ■

For an active node, its power consumption can scale from idle to peak based on workloads. That is, $P^{(a)}$ can vary over time depending on jobs running on the node. Thus, it is difficult to estimate $P_i^{(a)}$ for a given time period. In this work, we simply chose the mean between these two extreme values, $P_i^{(\bar{a})} = (P_i^{(i)} + P_i^{(p)})/2$, and use this for weight w_i for node i. However, this can be replaced with any other relevant measure.

Our power-aware algorithm for node set discovery in a heterogeneous setting is illustrated in Algorithm 2. This greedy algorithm selects a node that minimizes $\frac{P_n^{(\bar{a})} - P_n^{(s)}}{|U \bigcap n.getReplicaSet()|}$. In other words, the algorithm prefers a node with smaller power requirement but with the greater number of data blocks for CS.

One interesting part in the algorithm is the addition of line 10: after obtaining a CS set by the weighted set cover algorithm, we run the (non-weighted) set cover algorithm once more. It is a "reduction" phase. We observed that this reduction can reduce the size of the CS set by removing redundant nodes. Indeed, a greedy technique yields good approximation, but may not be an optimal. To explain this more in detail, here is an example. Suppose $r = 3$ and two-class nodes, high-power and low-power nodes (as shown in Table II), reside in the cluster with a ratio 5:5. Then, the probability that a specific data block has no replica in any low-power node is $p = (\frac{1}{2})^3 = 0.125$. In other words, 12.5% of data blocks should only be found in high-power nodes probabilistically. If we consider those essential high-power nodes first as elements of the CS set, we could eliminate some redundant low-power

Input: Data block set B, Node set S
Output: Covering subset C

1 $U \leftarrow B$;
2 $F \leftarrow S$;
3 $C \leftarrow \emptyset$;
4 **while** $U \neq \emptyset$ **do**
5 Select node $i \in F$ that minimizes $\frac{P_i^{\bar{a}} - P_i^{(s)}}{|U \bigcap i.getReplicaSet()|}$;
6 $U = U - i.getReplicaSet()$;
7 $C \leftarrow C \bigcup \{i\}$;
8 $F \leftarrow F - \{i\}$;
9 **end**
10 $C \leftarrow ConstructCS(B, C)$;
11 return C

Algorithm 2: ConstructPACS(B, S)

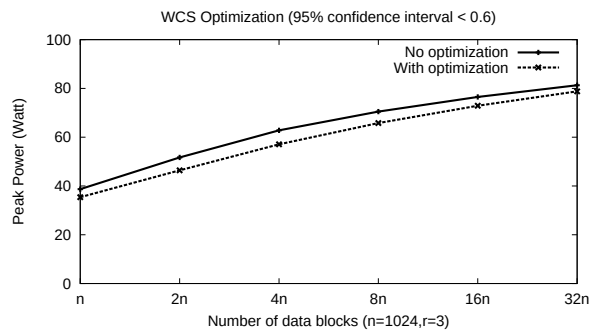
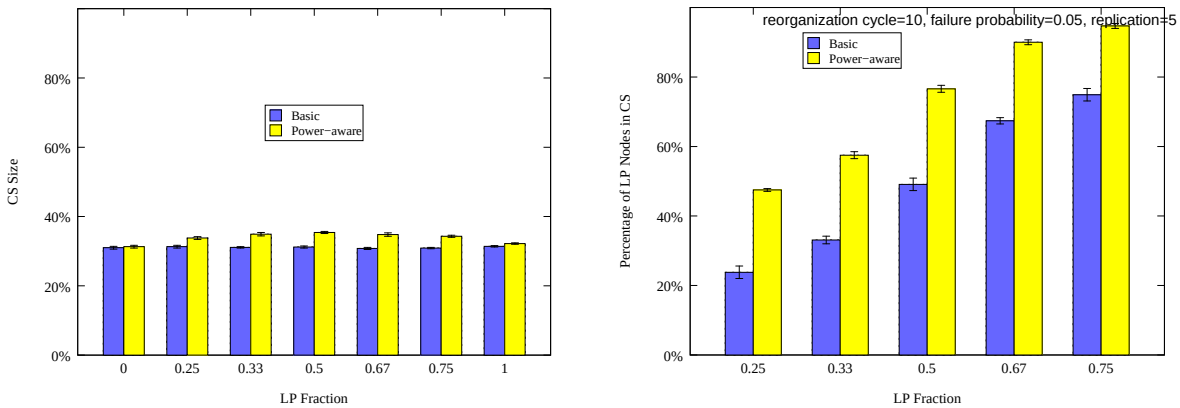


Fig. 3. CS Size with optimization for the power-aware algorithm

nodes because the high-power nodes also keep some other data blocks. This simple optimization helps to reduce the CS size, as shown in Figure 3.

Figure 4 compares the power-aware CS discovery algorithm with the basic CS algorithm. As above, we considered two classes of nodes, low-power (LP) and high-power (HP), based on Table II. The two figures show CS size (Figure 4(a)) and percentage of LP nodes (Figure 4(b)) in the resulted CS, as a function of fraction of LP nodes in the cluster. In this experiment, we set the number of data blocks $b = n$ and replication factor $r = 3$. We can see that the power-aware algorithm yields a slightly bigger set for CS, but not that significant (the max gap is smaller than 4%). Figure 4(b) shows the power-aware algorithm takes a greater number of LP nodes for CS. Even with 0.25 for LP fraction, around 50% of nodes in the CS are LP nodes, while it is 25% with the basic algorithm. This power-optimized CS technique can significantly reduce energy consumption over the basic CS technique in heterogeneous settings, as we will show in Section VI-A.

A short discussion about energy benefits of the power-aware discovery technique over the basic one. Figure 5 shows this as a function of fraction of LP nodes with respect to energy ratio between LP and HP. For example, $E(LP)/E(HP) = 0.5$ means that LP node energy consumption is 1/2 of energy



(a) CS Size

(b) Percentage of low-power nodes

Fig. 4. Comparison of basic and power-aware CS discovery

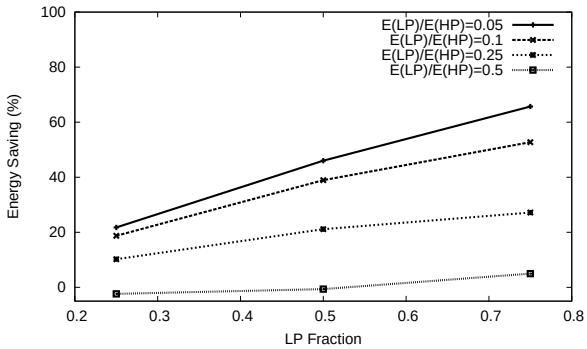


Fig. 5. Energy saving of power-aware search over basic algorithm (in idle)

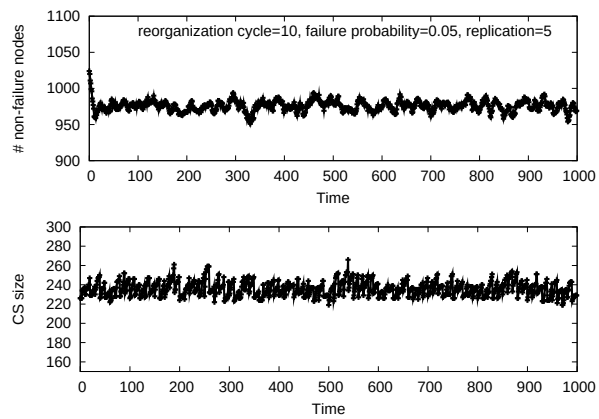


Fig. 6. CS reorganization under node failures

consumption for HP node. We can see that the power-aware search achieves more energy saving, as the energy ratio between LP and HP nodes goes high. Past studies, such as [7], [19], observed that 1:10 or even smaller ratio for LP:HP energy consumption. In such cases, 40% more energy saving can be achieved over basic CS when LP fraction is 0.5. All these suggest that *the power-aware technique would be beneficial to discover CS closer to the optimal in terms of energy efficiency*.

C. Incremental CS reorganization for node failure

Here, we briefly discuss the issue of CS reorganization in case of cluster configuration changes due to node failure. We assume that a new CS set is constructed periodically or on demand. Thus, any configuration change can be accounted at every construction time. However, there may be node failures, and as a result, some data blocks can be unavailable from the CS set. To deal with such failure cases, it is possible to reorganize CS incrementally by adding some nodes to keep the CS effective. Upon detection of any failure that affects the CS set, we can perform the CS discovery algorithm with inputs of the missing data blocks from the CS set and a set of non-CS nodes (i.e., NCS). The resulting set can then be added to the CS set. The incremented set may not be optimal, but still effective with required data availability. At the end of the time window for which the current CS is effective, a full reorganization is initiated to find an optimal node set for the new set of data blocks.

Figure 6 shows an example of CS reorganization over time under a node failure environment. We assumed that node failure probability is 0.005 for each node at every time unit. Probabilistically, at each time unit around 5 nodes suffer a failure in a cluster with $n = 1024$. Thus, at each time step, there would be an incremental reorganization if any CS node suffers a failure. We assume that a failed node is recovered after a deterministic amount of time (10 time units), and that a full reorganization takes place at every 10 time units. In the figure, the upper plot shows the number of nodes that do not experience failure, while the bottom plot shows CS size changes over time. In the upper plot, we can see that nodes fail and recover back, and the CS size varies accordingly in the bottom plot. As shown in the figure, CS size varies up and down over time with incremental reorganizations (increasing CS size) and full reorganizations (minimizing CS size) from the bottom one.

In this section, we have discussed node set discovery for CS that provides a single replica availability for data blocks in requirement. This can be extended to guarantee a higher

degree of data availability, e.g., two replicas for each required data block. In the next section, we discuss how we can achieve this, and show how this idea can be used to provide energy proportionality in a cluster.

IV. Multi-level Node Set Discovery

Here we discuss how it is possible to provide energy proportionality in this framework. In [14], the authors considered several strategies for node deactivation for non-CS nodes to support the CS approach. By deactivating (and activating) nodes one by one according to the current load, it is possible to get energy proportionality, but as the authors indicated, there may be load inequality between nodes because the number of replicas for each data block may be different for a certain time. For example, if we deactivate one node (and all the other nodes are active), there will remain $r-1$ blocks for the data blocks kept in that node, while the other blocks are maintained based on replication factor (r). This implies a possibility of load imbalance. For these types of complications, we do not rely on a node selection strategy for achieving energy proportionality. Instead, we propose a multi-level CS discovery that gives different degrees of data availability based on performance requirements for the given workload.

In our multi-level CS approach, different CS levels provide different degrees of data availability. For example, a CS set in level 2 in our framework gives 2-replica availability for the required data blocks (we call it *CS-2*). Therefore, there can be a series of CS sets from *CS-1* to *CS-r* (usually equivalent to n). In this section, we describe how we can discover such CS sets for a certain degree of data availability.

The problem of identifying *CS-k* can be mapped to the *set multicover* problem with coverage factor k, where k denotes the minimal number of times each object in question appears in the resulting set.

Proposition 4.1: The CS-k(B,S) problem is NP-complete, the reduction is from the set multicover problem $SMC(U, F, k)$, where U is a universe, F is a family of subsets of U, and a required coverage factor k.

Proof: The reduction algorithm is the same as proof 3.4. Since there is a one-to-one mapping between the block $b_i \in B$ and the element $u_i \in U$, any element that is covered k times in $SMC(U, F, k)$ also appears k times in the result set of *CS-k(B,S)*, and vice versa. Also, the reduction remains in polynomial time. ■

In [3], the authors presented an $O(k|U||F|)$ time greedy heuristic for the $SMC(U, F, k)$ problem with an approximation factor of $(1 + \ln a)$ from optimal where $a = \max_i |F_i|$.

The greedy heuristic makes a selection of a new set in each iteration. The selected set must include the max number of elements that have not been covered k times yet. We employ this greedy heuristic for our multi-level CS discovery.

Figure 7 shows the CS size compared to the cluster size, as a function of the number of data blocks in two replicated environments ($r = 3$ and $r = 5$). As shown in the figure, *CS-1* and *CS-2* have different sizes. For example with $b = 4n$ and $r = 3$, the CS size is around 50% and 80% of the cluster for *CS-1* and *CS-2*, respectively. From those sets, we can select the one with a desired data availability while considering the (expected) workload. By doing so, our multi-level CS technique can be used for achieving energy-proportionality in the cluster.

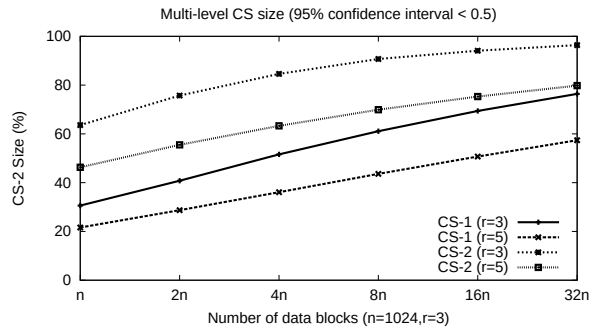


Fig. 7. Multi-level CS size

TABLE III
Parameters

Symbol	Description	Default value
n	Cluster size	1,024
b	Number of data blocks	$16n$
r	Replication factor	3
f	Fraction of low-power nodes	0.5
ξ	Number of jobs	1000
λ	Job arrival rate	0.5
χ	Number of tasks	n
τ	Task processing time	Normal(300,0.1)

V. Evaluation Methodologies

For evaluation, we developed a simulator based on OMNeT++ [17] providing a discrete event simulation framework. Our simulator performs with power measures from [7], [14] shown in Table II. In the table, *MaxThread* is the max number of threads that can be concurrently run in the node, and *Capacity* refers to processing capacity. Thus in the table, we can see that an Atom node can accommodate 4 concurrent tasks at max, and its processing capacity is 0.36 of that of a Xeon. For example, if a Xeon node can run 100 instructions in a unit time, an Atom node can perform 36 instructions for that moment.

We conducted experiments extensively with a diverse set of parameters summarized in Table III. We assume data placement follows the basic MapReduce replication properties (hence, almost close to a uniform data layout). We then inject a series of jobs to the simulator based on job arrival rate (λ). We assume λ follows an exponential distribution. Since we are more interested in light loads for energy saving, we use $\lambda = 0.5$ by default in our experiments.

Each job requires χ parallel tasks, and the processing time is defined by τ and node capacity. The task processing time (τ) consists of computation time (c) and additional time for networking for data transfer (d), and is described as $\tau = distribution(c, d)$. We assume that the computation time

TABLE II
POWER MODEL: SPECPOWER RESULTS FROM [7] AND NODE HIBERNATION COSTS FROM [14].

Platform	$P^{(i)}$	$P^{(p)}$	$P^{(s)}$	$T^{(d)}$	$T^{(u)}$	MaxThread	Capacity
HP (Xeon)	259.5W	315.0W	18.0W	11s	100s	8	1
LP (Atom)	25.6W	33.8W	2.0W	11s	100s	4	0.36

is deterministically calculated based on c and node capacity by the equation of $computation_time = c/Capacity$, whereas the data transfer time is determined by a probabilistic distribution. Since no previous work identified distributions for data transfer time in data parallel computing clusters, we employed two distribution models, normal and exponential, in this study. For example, $\tau = Normal(300, 0.1)$ implies $300s$ for computation time, and a positive random value (v) from a normal distribution $Normal(0, 0.1)$ determines the additional time for data transfer, with an equation of $download_time = v \times c/Capacity$. Thus, the overall task completion time is $(1 + v) \times c/Capacity$. For exponential distributions, we randomly choose a value from the given exponential distribution with $mean = d \times c/Capacity$, and we use the chosen value as the data transfer time.

We compare the following techniques in terms of energy consumption and average turnaround time: (1) NPS without reconfiguration (hence no energy management); (2) AIS (All-in Strategy); (3) Basic CS; and (4) Power-aware CS. NPS fully utilizes nodes in the cluster, and nodes are in idle after jobs are completed. AIS also utilizes entire nodes for jobs, but keeps the cluster deactivated as soon as completing jobs until the next job comes. The Basic CS technique constructs CS dynamically without considerations of node heterogeneity, while the power-aware CS technique takes heterogeneity into account in construction.

Initially, the entire cluster is "on" for NPS and AIS, while only CS nodes selected by each algorithm are active for our *CS*-based techniques. After completing all injected jobs (i.e., ξ), we measured aggregated energy consumption and average turnaround time for each technique, and compared the measured results. We repeated experiments and provide 95% confidence intervals for statistical consideration.

VI. EXPERIMENTAL RESULTS

A. Impact of fraction of low-power nodes

In this experiment, we explore the impact of the LP fraction in the cluster. We moved LP fraction from 0 to 1 for this. By definition, the two extremes (i.e. $f = 0$ and $f = 1$) refer to homogeneous settings (i.e., $f = 0$ for all high-power node setting and $f = 1$ for all low-power node setting), while the others mixes both classes of nodes based on the fraction.

As shown in Figure 8, the power-aware technique yields the same results as the basic CS technique for the both extremes, showing around 30% energy saving. However, we can see that the power-aware technique further improves energy saving in any heterogeneous setting. With $f = 0.75$, the power-aware technique improves energy saving over 50%, as shown in Figure 8(a). For turnaround time, no significant deviations for the techniques were observed, as in Figure 8(b). This indicates that our power-aware technique enables to improve energy saving with little performance loss by exploiting cluster heterogeneity.

B. Impact of the number of data blocks

Next, we investigate the impact of the number of data blocks since the CS size has strong correlation with this parameter. To see this, we used a diverse set of the number of data blocks, from $b = n$ (i.e., 64GB) to $b = 32n$ (i.e., 2TB). Figure 9 shows the results with respect to both energy and performance. We can see linear increases of energy consumption as the number of blocks increases for CS techniques, since a greater number of data blocks results in a larger CS. environment. The basic technique shows 30–60% energy saving, while the power-aware yields 40–70% saving with no noticeable performance degradation.

C. Impact of job arrival rate

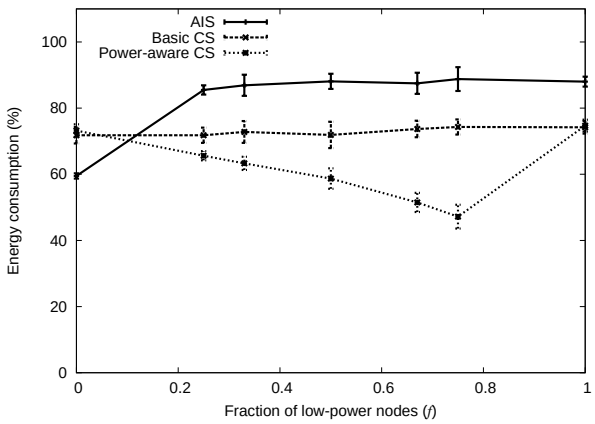
By default, we used job arrival rate $\lambda = 0.5$, since we are interested more in light load environments. In this experiment, we discuss the experimental results under varied job arrival rates. We employed a multiple set of job arrival rates from $\lambda = 0.25$ (for light load) to $\lambda = 2$ (for heavy load) in this experiment.

Figure 10 shows energy and performance as a function of λ. We see no significant changes for our CS techniques, except that power-aware CS somewhat degraded in a heavy workload environment $\lambda = 2$. Interestingly, AIS could save energy with very small job arrival rates. This is because very light loads can help to reduce the number of cluster power transition and can lengthen deactivation period at the same time. In this experiment, AIS yielded around 30% energy saving when $\lambda = 0.25$. However, little energy saving has been observed with greater job arrival rates than that with AIS.

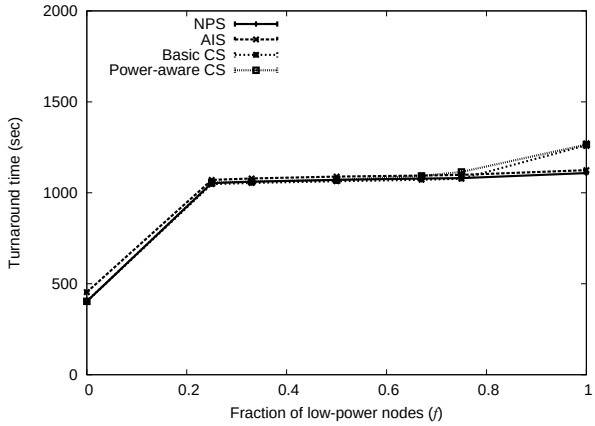
D. Impact of data transfer distributions

In this experiment, we employ several distribution models to consider data transfer times in the cluster. As mentioned, we consider normal and exponential distributions. For the normal distribution, we used three standard deviation values, 0.05, 0.1, and 0.25. For the exponential distribution, we used the same values as above but for the mean for the distribution. Again, these values are used to determine data transfer time according to the given distribution model, as described in Section V.

Figure 11 shows the results. In the figure, $Norm(\sigma)$ represents a normal distribution with standard distribution σ, while $Exp(\mu)$ is for an exponential distribution with mean

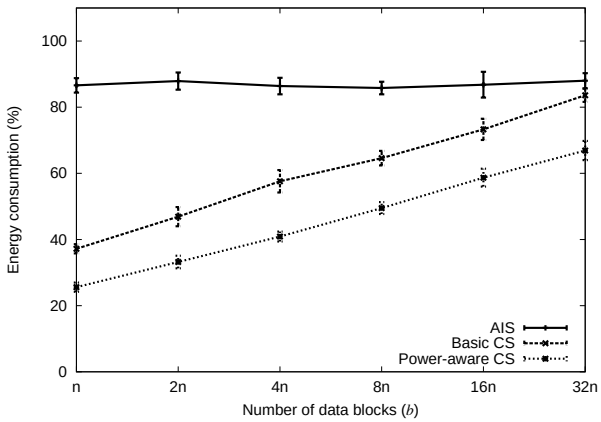


(a) Energy consumption

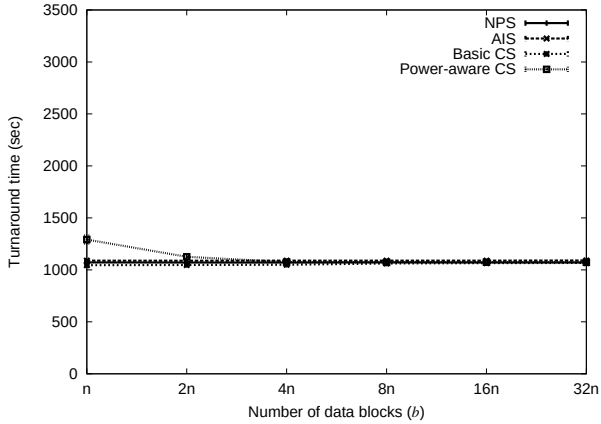


(b) Average turnaround time

Fig. 8. Impact of fraction of low-power nodes in the cluster

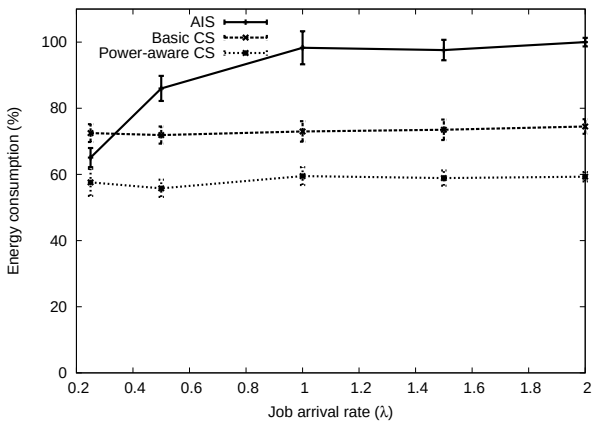


(a) Energy consumption

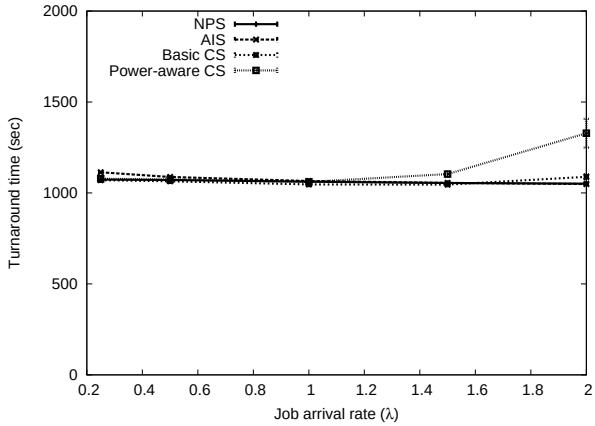


(b) Average turnaround time

Fig. 9. Impact of the number of data blocks

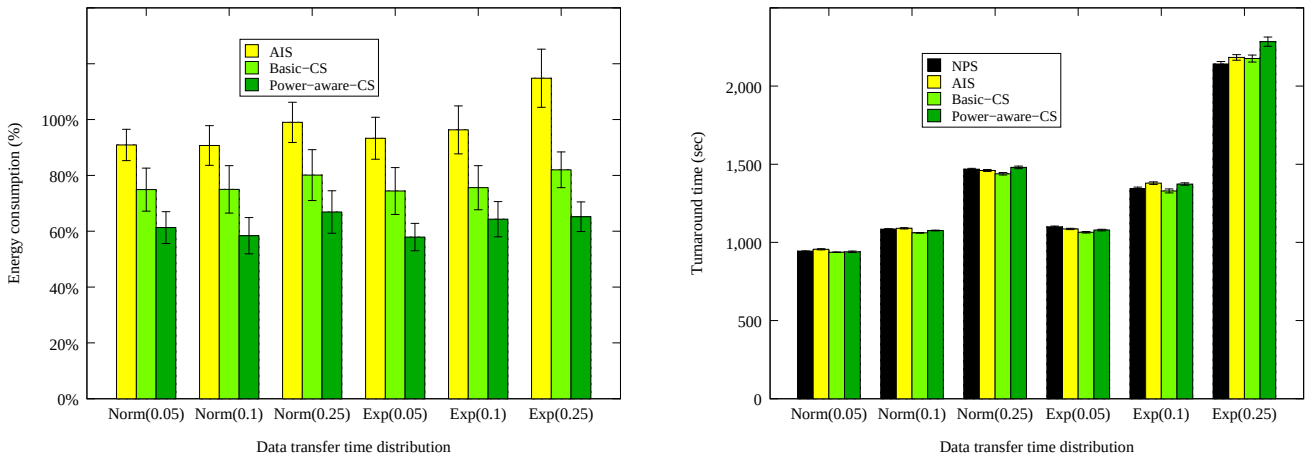


(a) Energy consumption



(b) Average turnaround time

Fig. 10. Impact of job arrival rate



(a) Energy consumption

(b) Average turnaround time

Fig. 11. Impact of data transfer distribution

μ. For a diverse set of distribution models, we can see that our *CS*-based techniques consistently save energy around 30% for Basic and 40% for power-aware CS, but without any significant performance loss.

E. Impact of the number of tasks

We next present our evaluation results showing how the number of tasks impacts to energy and performance. We arranged the number of tasks (χ) from $n/4$ to $4n$ (i.e., $\chi = [n/4, n]$) for each job. As shown in Table II, the maximum number of threads that the cluster can concurrently accommodate is approximately $8 \times 512 + 4 \times 512 = 3n$ in the basic experimental setting with the fraction of low-power nodes $f = 0.5$. Thus, a workload with $\chi > 3n$ can be considered as a heavy load in the basic setting.

Figure 12 shows both energy consumption and average turnaround time for each technique. We can see that Basic CS achieves 30% energy saving compared to NPS, even with the heavy workload $\chi = 4n$. Power-aware CS further improves energy saving by considering power metrics of nodes for CS discovery, yielding around 40% saving. However, our techniques showed some degrees of performance degradation with the heavy workload, as shown in the figure. This suggests that any energy-optimized CS can degrade performance in non-light workload environments, and we address this problem by providing the concept of multi-level CS sets that will be evaluated in Section VI-G. In the figure, AIS yields no significant energy saving, and hence no performance penalty.

F. Impact of computation time

We examine the impact of computation time. In addition to $\tau = 300s$ used in the basic setting, we examined the techniques with three more processing times, $100s$, $600s$, and $1200s$, to consider diverse classes of applications. The results showed no noticeable impact of this parameter for both energy and performance, as plotted in Figure 13.

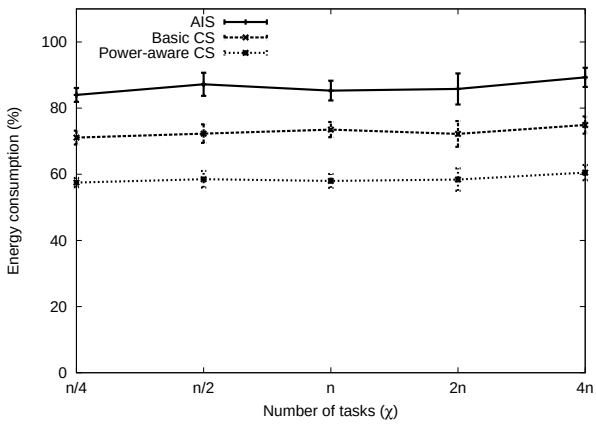
G. Evaluation of multi-level CS

Finally, we present the impact of multi-level CS sets. To see the impact more clearly, we used a greater replication factor $r = 5$ and a smaller data blocks $b = n$, in this experiment. There can thus be four CS sets from *CS-1* to *CS-4* in addition to the entire cluster. For those CS sets, we varied λ to see how the CS sets respond to different loads.

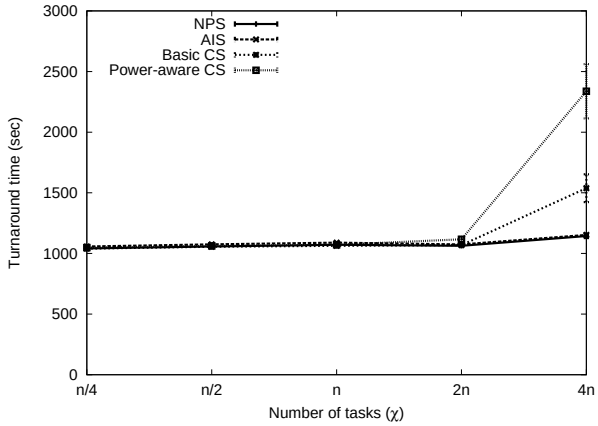
Figure 14 shows the results. From the figure, we can see that each CS level gives a different degree of energy saving. For $\lambda = 1$, even with *CS-4*, it saves 20% of energy compared to NPS on average. The figure also shows that *CS-3* achieves 50% energy saving in the same setting, while *CS-2* and *CS-1* further increase saving to 70%. With respect to performance, we can see that a lower level CS shows a greater turnaround time. Thus, any appropriate CS can be chosen based on load intensity to maximize energy saving with performance guarantees.

VII. Conclusions

Energy consumption in commercial and scientific datacenters has recently become a major concern due to the rising operational costs and scalability issues. For data parallelism and fault tolerance purposes, most common file systems used in MapReduce-type clusters maintain a set of replicas for each data block. Our basic idea in this work is to identify a subset of nodes, called a covering subset, that can provide a required degree of data availability for a given set of data blocks. In this work, we developed algorithms to maintain energy proportionality by discovering a covering set that minimizes energy consumption while placing the remaining nodes in low-power standby mode. In particular, we consider heterogeneity in determining a power-optimized covering set. For evaluation, we conducted experiments with a variety of parameters, such as job arrival rate and data transfer distribution. The experimental results show that power management based on our covering set algorithms can significantly reduce energy consumption, up to 70% compared to a non-power saving

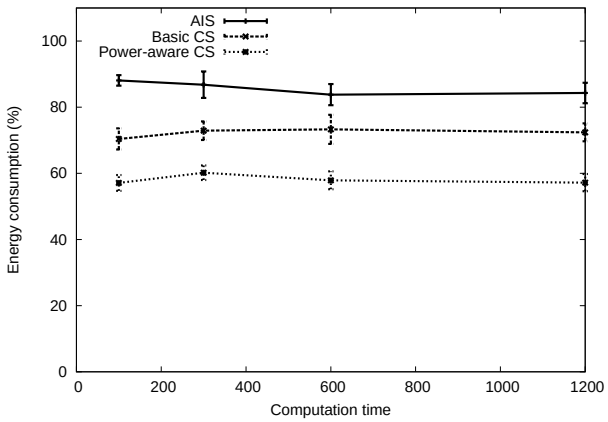


(a) Energy consumption

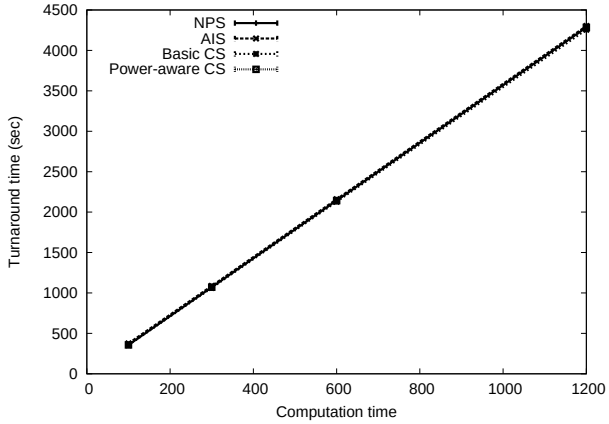


(b) Average turnaround time

Fig. 12. Impact of the number of tasks

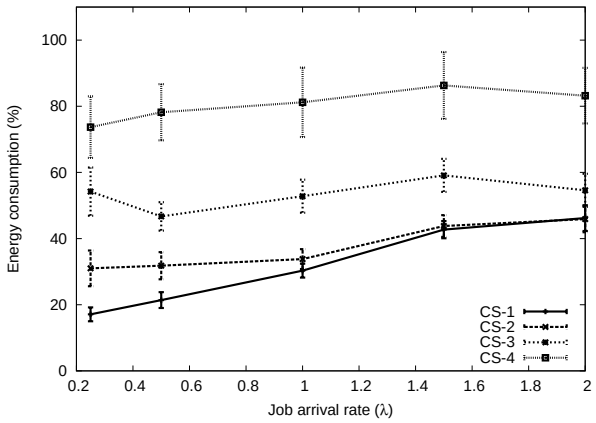


(a) Energy consumption

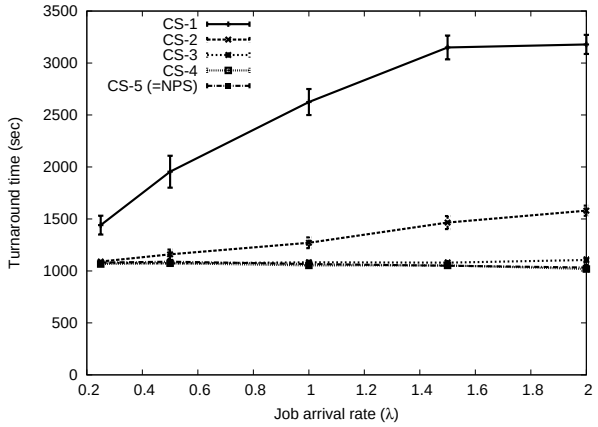


(b) Average turnaround time

Fig. 13. Impact of computation time



(a) Energy consumption



(b) Average turnaround time

Fig. 14. Impact of CS level

configuration, with little performance loss. In particular, the experimental results show that our algorithms can enhance energy saving in a heterogeneous environment by considering power metrics of individual nodes in the construction of a covering set. The results also show that our extended algorithm can be used to provide a coarse-grained level of energy proportionality based on covering sets with different degrees of data availability (thus providing different degrees of data parallelism). In the future we plan to also work on efficient scheduling algorithms for activating/deactivating nodes based on anticipatory analysis of future workloads.

ACKNOWLEDGMENTS

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

REFERENCES

[1] H. Amur, J. Cipar, V. Gupta, G. R. Ganger, M. A. Kozuch, and K. Schwan. Robust and flexible power-proportional storage. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 217–228, 2010.

[2] L. A. Barroso and U. Holzle. The case for energy-proportional computing. *Computer*, 40:33–37, 2007.

[3] P. Berman, B. DasGupta, and E. Sontag. Randomized approximation algorithms for set multicover problems with applications to reverse engineering of protein and gene networks. *Discrete Appl. Math.*, 155(6-7):733–749, 2007.

[4] R. Bianchini and R. Rajamony. Power and energy management for server systems. *Computer*, 37(11):68–74, 2004.

[5] M. Cardosa, A. Singh, H. Pucha, and A. Chandra. Exploiting spatio-temporal tradeoffs for energy efficient MapReduce in the cloud, 2010.

[6] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing and server resources in hosting centers. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 103–116, 2001.

[7] B.-G. Chun, G. Iannaccone, G. Iannaccone, R. Katz, G. Lee, and L. Niccolini. An case for hybrid datacenters. *SIGOPS Oper. Syst. Rev.*, 44(1):76–80, 2010.

[8] V. Chvàtal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.

[9] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. MapReduce online. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 21–21, 2010.

[10] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 10–10, 2004.

[11] T. Gunarathne, T.-L. Wu, J. Qiu, and G. Fox. MapReduce in the clouds for science. In *CloudCom*, pages 565–572, 2010.

[12] Hadoop:, http://hadoop.apache.org/.

[13] T. Heath, B. Diniz, E. V. Carrera, W. Meira, Jr., and R. Bianchini. Energy conservation in heterogeneous server clusters. In *PPoPP '05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 186–195, 2005.

[14] W. Lang and J. M. Patel. Energy management for MapReduce clusters. In *VLDB '10*, 2010.

[15] J. Leverich and C. Kozyrakis. On the (in)efficiency of Hadoop clusters. *SIGOPS Oper. Syst. Rev.*, 44(1):61–65, 2010.

[16] http://www.mckinsey.com/clientservice/bto/pointofview/pdf/revolutionizing_data_center_efficiency.pdf.

[17] OMNeT++ Network Simulation Framework, http://www.omnetpp.org/.

[18] http://www.federalnewsradio.com/pdfs/epadatacenterreporttocongress-august2007.pdf.

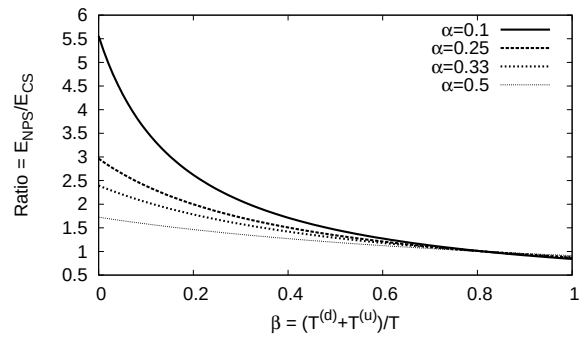


Fig. 15. Impact of power state transition time

[19] A. S. Szalay, G. C. Bell, H. H. Huang, A. Terzis, and A. White. Low-power amdahl-balanced blades for data intensive computing. *SIGOPS Oper. Syst. Rev.*, 44(1):71–75, 2010.

[20] C. Vercellis. A probabilistic analysis of the set covering problem. *Annals of Operations Research*, pages 255–271, 1984.

[21] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica. Improving MapReduce performance in heterogeneous environments. In *OSDI*, pages 29–42, 2008.

APPENDIX

Here, we discuss energy benefits for CS over a full configuration (or NPS). We assume n nodes in the cluster. Thus, n nodes are involved in computation for given jobs for NPS. For CS, in contrast, a part of nodes are involved in computation for those jobs (i.e., m $(< n)$ nodes), and the other $n-m$ nodes are deactivated for energy saving. Thus, we can say $m = \alpha n$, where $0 < \alpha \leq 1$. Similar with the notations for power states in Table III, we use $T^{(x)}$ for the associated time for power state x, i.e., the amount of time spending in state x.

Now, we analyze energy consumption for CS (E_{CS}) and NPS (E_{NPS}). We assume that nodes are idle initially. For CS, CS nodes are active for a time period $T_{CS}^{(a)}$ for computation, while non-CS nodes are deactivated and activated again during the time period. That is, $T_{NCS}^{(d)} + T_{NCS}^{(s)} + T_{NCS}^{(u)} = T_{CS}^{(a)}$. We denote $T = T_{CS}^{(a)}$ for simplicity. We also assume that job completion time is proportional to the number of active nodes for a job. Thus, node active time for NPS would be $T_{NPS}^{(a)} = \alpha T_{CS}^{(a)} = \alpha T$, where $\alpha = m/n$ as defined above. To represent the power state transitioning time, we define β as $T^{(u)} + T^{(d)} = \beta T$, where $0 \leq \beta \leq 1$. Thus, the extreme values of β imply that: if $\beta = 0$, there is zero transitioning time; and if $\beta = 1$, computation time is negligible compared to transitioning time.

We first compute the maximum bound of energy consumption for NPS based on the above assumptions.

$$\begin{aligned} E_{NPS} &= n(P^{(a)}T_{NPS}^{(a)} + P^{(i)}(T - T_{NPS}^{(a)})) \\ &\leq n(\beta P^{(p)}T + P^{(i)}(T - \beta T)) \\ &= nT(\alpha P^{(p)} + (1-\alpha)P^{(i)}) \end{aligned} \quad (3)$$

We next compute the CS case (E_{CS}):

$$
\begin{aligned}
E_{CS} &= mP^{(a)}T_{CS}^{(a)} \\
&+ (n-m)(P^{(d)}T^{(d)} + P^{(u)}T^{(u)}) \\
&+ (n-m)(P^{(s)}(T_{CS}^{(a)} - (T^{(d)} + T^{(u)}))) \\
&\leq mP^{(p)}T + (n-m)(P^{(p)}(\beta T) + P^{(s)}T(1-\beta)) \\
&= nT(\alpha P^{(p)} + (1-\alpha)(\beta P^{(p)} + (1-\beta)P^{(s)}) \qquad (4)
\end{aligned}
$$

Based on Equation 3 and 4, Figure 15 shows the ratio between $E_{NPS} : E_{CS}$ as a function of β from the equations. For the graphs, we plugged in the Xeon measures in Table II. It is intuitive that E_{CS} has a greater energy benefit with a smaller power state transitioning time. As seen from the figure, E_{CS} has no benefit if $\beta > 0.9$. However, if T is relatively large compared to the transitioning time, energy benefits increase super-linearly. The figure also plots energy ratio with diverse α values (i.e., as a function of m). Intuitively, energy saving can grow up with a smaller α (i.e., a smaller set of CS nodes), and the figure agrees with the intuition.

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.