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 subset* 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 subset that minimizes energy consumption while placing the remaining nodes in low-power standby mode. Our algorithms can also discover covering subset in *heterogeneous* computing environments. In order to allow more data parallelism, we generalize our algorithms so that it can discover k-covering subset, 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 [20]. Despite the technological progress and the amount of capital invested, there are significant inefficiencies in datacenters with server utilization measured at around 6% [18]. In this paper, we focus on optimizing energy consumption of compute clusters in datacenters, such as MapReduce clusters [12] often used in scientific computation [13]. 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 [23], [15], [5], *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 [8] presented a hybrid datacenter model with two-class nodes that have different performance capabilities and power requirements for energy efficiency. In a recent work [23], 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 [17], [16], 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 [3], 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 [7] 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 [16], 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 [2], [12]. 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. In the next section, we briefly discuss the background for this work, including benefits of the CS approach that activates a subset of nodes for energy saving and introduction to several closely related studies 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 paper and present some research topics for future work in Section 7.

II. Background and Related Work

Since our approach in this paper is based on the concept of covering subset (CS), we first discuss the benefits of the CS approach for energy management. We then provide a brief summary of studies related to our work.

A. Energy benefits of CS approach

Here, we discuss energy benefits for the CS approach over a full configuration called NPS (Non Power Saving). As in the

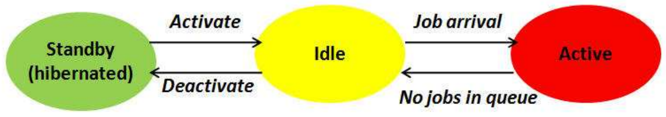


Fig. 1. Power state transition diagram of a node in the cluster (we omit Power-off state.)

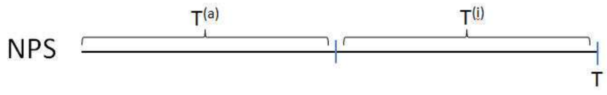
real life, we assume that a node requires a different power level according to their power state, as illustrated in Figure 1. We assume n nodes in the cluster. Thus, n nodes are involved in computation for given jobs for NPS. For the CS approach, in contrast, a part of nodes are involved in computation for those jobs (i.e., m nodes and $m < n$), and the other $n - m$ nodes are deactivated for energy saving. Thus, we can say $m = \alpha n$, where $0 < \alpha \leq 1$. We use $T_x^{(y)}$ and $P_x^{(y)}$ to denote the time duration and power consumption respectively for power state y and node set x. The variable y may take the following values i (idle), a (Active), p (Peak) , s (Standby), u (Activating), and d (Deactivating). The variable x may take the values of individual nodes or sets of nodes based on the context where it is used. For example, it can take the value CS to denote the set of nodes in the covering set, NCS for the set of nodes not in the covering set, and N to denote the set of all nodes. We simply use $T^{(y)}$ and $P^{(y)}$ if the node (set) is obvious, and use T_x and P_x if its power state does not matter. Table I summarizes the notations used in this paper.

Now, we analyze energy consumption for the CS approach (E_{CS}) and NPS (E_{NPS}). We assume that nodes are idle initially. For the CS approach, 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 inversely proportional to the number of active nodes for a job. Under NPS, the complete set of nodes N is active, thus, node active time for NPS would be $T_N^{(a)} = \alpha T_{CS}^{(a)} = \alpha T$, 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. Figure 2 illustrates timing for both approaches in detail.

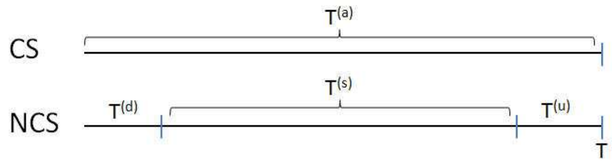
We first compute the lower bound on energy consumption for NPS based on the above assumptions. Since it is true that $P^{(i)} < P^{(a)} \leq P^{(p)}$, NPS energy consumption is simply computed as follows:

$$E_{NPS} \quad \geq \quad nTP^{(i)} \tag{1}$$

We can also compute the upper bound on the CS approach (E_{CS}), as follows:



(a) NPS Timing



(b) CS Timing

Fig. 2. Timing comparison between NPS and CS approach: NPS utilizes the entire set of nodes in the cluster for a given set of jobs. In the CS approach only CS nodes are involved in processing the jobs, while NCS nodes are hibernated during that period.

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

$$\begin{aligned} E_{CS} &= mP^{(a)}T_{CS}^{(a)} \\ &+ (n-m)(P^{(d)}T^{(d)} + P^{(u)}T^{(u)}) \\ &+ (n-m)(P^{(s)}(T_{CS}^{(a)} - (T^{(d)} + T^{(u)}))) \\ &\leq mP^{(p)}T + (n-m)(P^{(p)}(\beta T) + P^{(s)}T(1-\beta)) \\ &= nT(\alpha P^{(p)} + (1-\alpha)(\beta P^{(p)} + (1-\beta)P^{(s)}) \quad (2) \end{aligned}$$

Based on Equation 1 and 2, Figure 3 shows the ratio between two techniques in terms of $lower\_bound(NPS)$: $upper\_bound(CS)$, as a function of β. 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 supports the intuition.

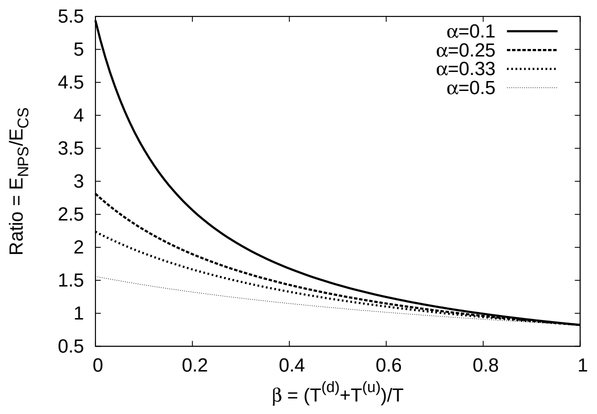


Fig. 3. Impact of power state transition time: As the CS active time (T) increases, energy benefits also increase super-linearly. Here, α is the fraction of active nodes (i.e., $\alpha n =$ the number of active nodes), and β is a ratio of power state transition time for node activation and deactivation to T (i.e., $\beta = \frac{T^{(d)}+T^{(u)}}{T}$).

B. Related Studies

1) MapReduce cluster energy management: The initial work on MapReduce cluster energy management was presented in [17] 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 to 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) [16] is a different approach. AIS runs the 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 [10]. One potential drawback is 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 stores b/k data items, where b is the total number of data items. Thus, a lower-indexed node holds a larger number of data items, this 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 [6]. Their approach places VMs in a timely balanced way, and finds 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.

2) Minimum set cover: 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. Since the exact solution to set cover is NP-complete, a greedy set heuristic algorithm is often used. 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 n^{th} harmonic number equal to $1+\frac{1}{2}+\cdots\frac{1}{n}$, and n is the cardinality of U [9]. A naive implementation has a run time complexity of $O(|U||F|\min(|U|,|F|))$, but a linear-time implementation is also possible [11, Ch.35].

III. Node Set Discovery Algorithms

In this section, we present our CS discovery algorithms for a set of nodes that minimizes energy consumption subject to data availability constraints. 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

```
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 maximizes
       $|U \bigcap i.getReplicaSet()|$;
6      $U = U - i.getReplicaSet()$;
7      $C \leftarrow C \bigcup \{i\}$;
8      $F \leftarrow F - \{i\}$;
9  end
10 return $C$
```

Algorithm 1: FindCS(B, S)

definition in [17]. 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 [10], [16], the data set can be available for the next operational time window. 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.

A. A basic method for CS discovery

By definition, CS maintains at least one replica[1] of the required data blocks. Locating such a set is NP-complete as it can be reduced to the well known *set cover* problem [9], 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 4 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 $\approx 2TB$ with the default data block size in MapReduce [12] and Hadoop [14]. As observed in [2], data popularity in MapReduce clusters is time-varying and

[1]Considering more than a single replica will be discussed in the next section (Section IV) for energy proportionality.

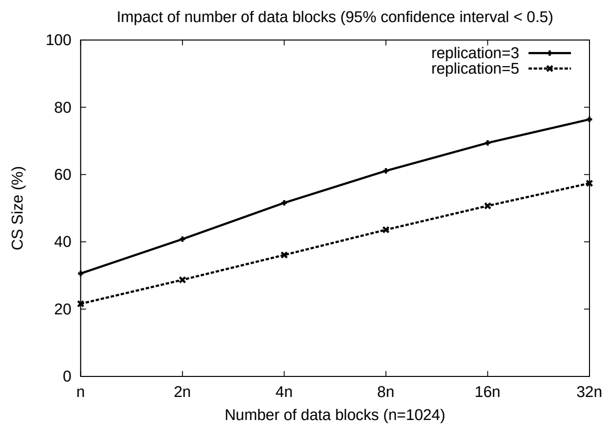


Fig. 4. CS size as a function of the number of data blocks: CS size has a strong correlation with the number of required data blocks. For n data blocks, CS size is 20–30% of the cluster, while it is 60–80% for $32n$ blocks ($\approx 2TB$) in two settings with replication factor 3 and 5.

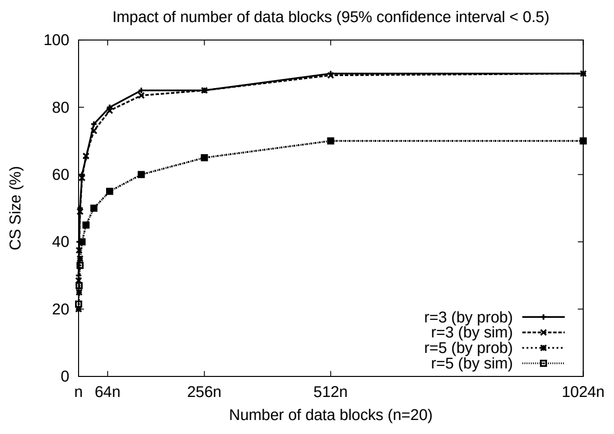


Fig. 5. Minimal CS size: This figure shows the minimal CS size obtained by the mathematical equation in Theorem 3.3 (”by prob”) and simulation (”by sim”). We used $n = 20$ and computed CS size from $b = n$ to $b = 1024n$.

skewed, and thus, there will be substantial opportunities to see a relatively small number of data blocks as a requirement for a certain time window.

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 5 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 [22]. 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 (PA) 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. 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. We will revisit the naive method when discussing Figure 9.

Formally, the overall power requirement in the CS approach 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:

$$\begin{aligned} P &= P_{CS}^{(a)} + P_{NCS}^{(s)} \\ &= \sum_{x \in CS} P_x^{(a)} + \sum_{y \in NCS} P_y^{(s)} \\ &= \sum_{x \in CS} \left(P_x^{(a)} + P_x^{(s)} - P_x^{(s)} \right) + \sum_{y \in NCS} P_y^{(s)} \\ &= \sum_{x \in CS} \left(P_x^{(a)} - P_x^{(s)} \right) + \sum_{x \in CS} P_x^{(s)} + \sum_{y \in NCS} P_y^{(s)} \\ &= \sum_{x \in CS} \left(P_x^{(a)} - P_x^{(s)} \right) + \sum_{y \in N} P_y^{(s)} \end{aligned} \tag{3}$$

Since the second part in Equation 3 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.,:

$$w(g) = \sum_{x \in g} \left(P_x^{(a)} - P_x^{(s)} \right) \tag{4}$$

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

```
    **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  $C \leftarrow FindCS(B, C)$;
11  return $C$
```

Algorithm 2: FindPACS(B, S)

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 a smaller power requirement but with a 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. We call it a “reduction” phase. We observed that this reduction can decrease the size of CS by removing inessential nodes. Indeed, a greedy technique yields good approximation, but the resulted set may not be an optimal. To explain this more in detail, here is an example. Suppose $r = 3$ and the cluster consists of the equal number of high-power and low-power nodes. 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 the elements of CS, it is possible to eliminate some redundant low-power nodes because the high-power nodes also keep some other data blocks. This simple optimization helps to reduce CS size, as shown in Figure 6.

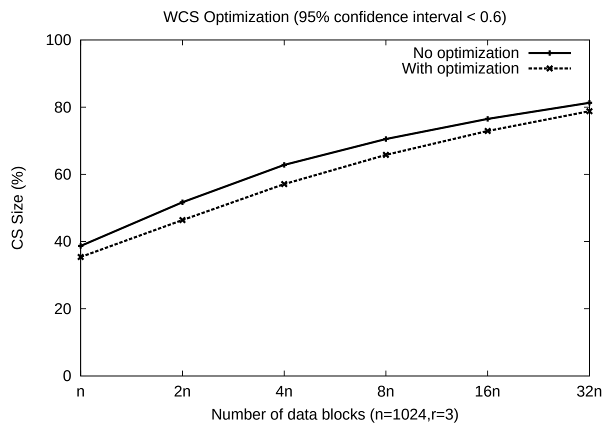


Fig. 6. CS Size with optimization for the power-aware algorithm: The simple optimization reduces CS size by 2–5% by removing inessential, redundant nodes from CS.

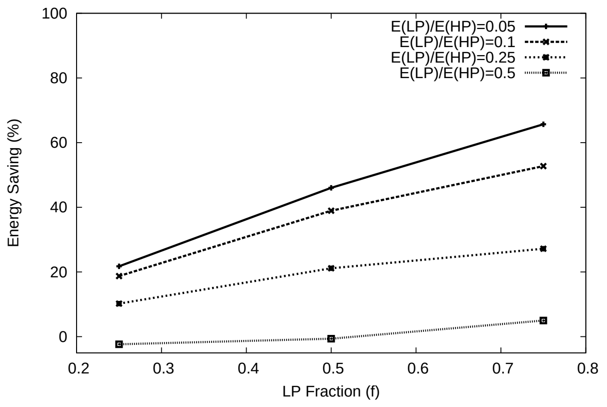


Fig. 8. Energy saving of the power-aware discovery (PA) over the basic algorithm (Basic): PA saves more energy where energy ratio between LP and HP (i.e., $E(LP)/E(HP)$) is smaller, and vice versa.

Figure 7 compares the power-aware CS discovery with the basic CS method. 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 7(a)) and percentage of LP nodes (Figure 7(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 7(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.

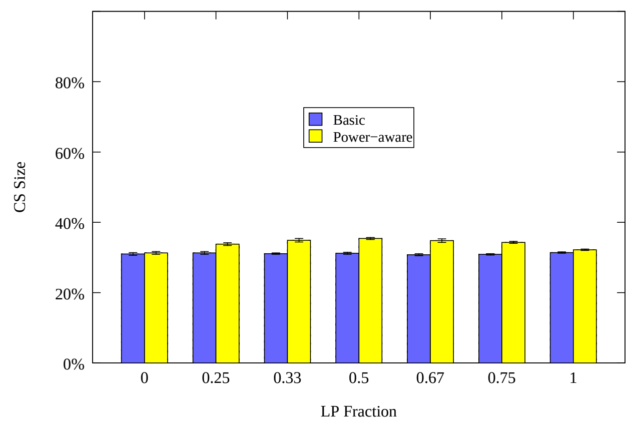
A short discussion about energy benefits of the power-aware discovery technique over the basic algorithm. Figure 8 shows this as a function of fraction of LP nodes with respect to energy ratio between LP and HP. For example, $E(LP)/E(HP) = 0.5$ means that LP node energy consumption is 1/2 of energy consumption of a HP node. We can see that the power-aware discovery achieves more energy saving, as the energy ratio between LP and HP nodes goes high. Past studies, such as [8], [21], observed that $E(LP)/E(HP)$ is 1/10 or even smaller. In such cases, it is expected that more than 42% energy saving can be achieved over the basic CS method where LP fraction is 0.5 ($f = 0.5$). All these suggest that *the power-aware technique is beneficial to discover CS closer to the optimal in terms of energy efficiency*.

We mainly assumed two classes of nodes, LP and HP, for heterogeneity. In reality, however, a cluster may have more than two generations of nodes. To consider more heterogeneous environments, we compose clusters with some other classes of nodes, in addition to LP and HP, with synthetic power values chosen based on the LP measures. We suppose ML (“Medium-Low”) has twice of LP power measures, and MH (“Medium-High”) has four times of LP measures (i.e., $P_{ML} = 2 \times P_{LP}$ and $P_{MH} = 4 \times P_{LP}$). Then peak power for each class of node is $P^{(p)}_{LP}$ =25.6W, $P^{(p)}_{ML}$ =51.2W, $P^{(p)}_{MH}$ =102.4W, and $P^{(p)}_{HP}$ =259.5W.

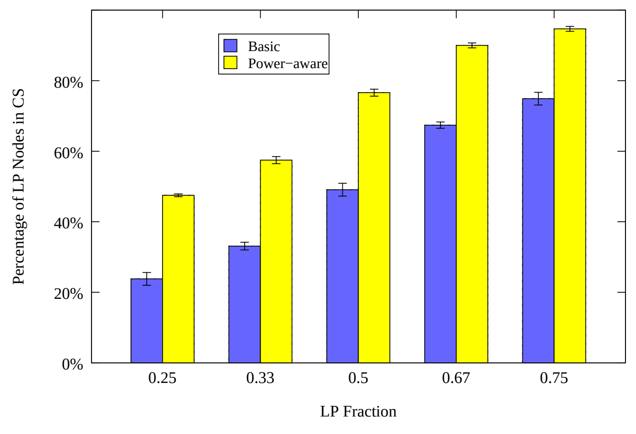
Figure 9 compares three CS discovery algorithms with respect to aggregated peak power of CS. Here, “Naive” is a CS discovery method that simply chooses the lowest power nodes first in a greedy manner. For each configuration in the figure, we assume the number of each class of node is equivalent in the cluster. We consider three configurations: LP/HP, LP/MH/HP, and LP/ML/MH/HP (i.e., four classes of nodes in the cluster). Naive finds CS with smaller peak power than Basic. However, the figure shows that the difference between Naive and Basic is reduced in more heterogeneous environments: it achieves 67% of Basic in the LP/HP configuration, but becomes 83% and 89% in the other two configurations. In contrast, our PA algorithm shows fairly consistent results between 62–73% of the peak power as compared with Basic. Comparing idle power between Naive and PA to Basic shows that Naive requires 67–89% of Basic, whereas PA requires 66–72% of Basic, although not shown in the figure. We repeated five times for this experiment, and report average in the figure. The 95% confidence intervals are smaller than 3.8%.

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,



(a) CS Size



(b) Percentage of low-power nodes

Fig. 7. Comparison of basic and power-aware CS discovery: The left-side figure compares CS size (relative to the cluster size) between Basic and PA algorithms, and the right-side figure compares the percentage of LP nodes in CS.

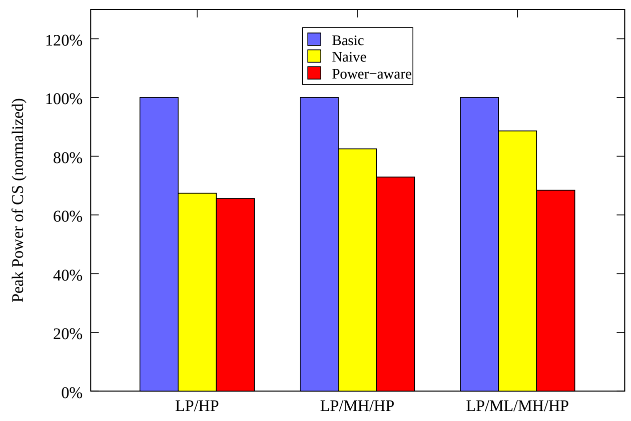


Fig. 9. Comparison of CS algorithms in peak power ($P_{CS}^{(p)}$): The figure plots aggregated peak power of CS in a normalized form, under three different heterogeneous environments. LP/HP for a cluster with two classes of nodes, LP/MH/HP for three classes of nodes, LP/ML/MH/HP for four classes of nodes. ML and MH power values are synthetically defined: $P_{ML} = 2 \times P_{LP}$ and $P_{MH} = 4 \times P_{LP}$.

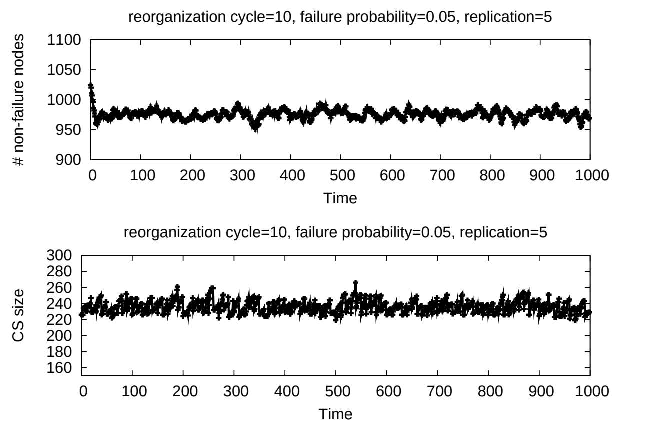


Fig. 10. CS reorganization under node failures: The top figure plots the number of non-failure nodes, while the bottom figure shows the changes of CS size by incremental reorganization due to CS node failure.

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 10 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 [16], 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 [4], 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 11 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.

V. Evaluation Methodologies

For evaluation, we developed a simulator based on OMNeT++ [19] providing a discrete event simulation framework. We ran our simulation using the power consumption data given in [8], [16] shown in Table II. The measurement information does not include power requirements for node activation and deactivation, and we simply assume they are equal to the peak power (i.e., $P^{(u)} = P^{(d)} = P^{(p)}$), based on the observations in [16]. 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 for a finite time interval, an Atom node can perform 36 instructions for that amount of time.

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 use average computation time (c) defined below as the unit time for λ in this paper. For example, if $\lambda = 1$ and $c = 100s$, it means that a single job is enqueued in the system at every 100 seconds on average. We assume job arrival 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, but we explore the impact of λ as well.

Each job requires χ parallel tasks, and task processing time (τ) is determined by computation overhead (c), data transfer

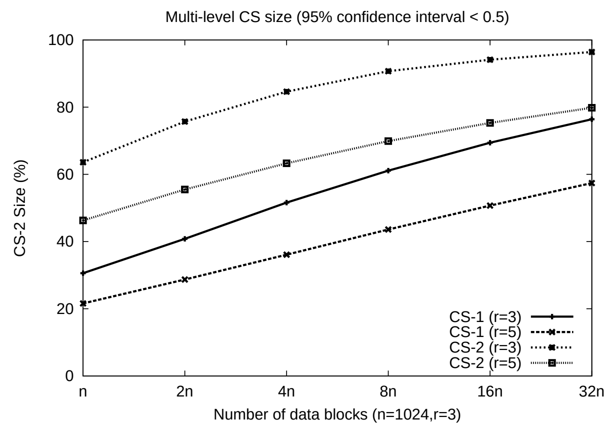


Fig. 11. Multi-level CS size: This figure plots CS size for CS-1 and CS-2 under two replicated environments, $r = 3$ and $r = 5$, as a function of the number of data blocks. Since it satisfies $size(\text{CS-}x) < size(\text{CS-}y)$ for $x < y$, it is possible to choose CS-i $(1 \leq i \leq r)$ based on load intensity for energy proportionality.

TABLE III
PARAMETERS

Symbol	Description	Default value
n	Cluster size	1,024
b	Number of data blocks	$16n(\approx 1TB)$
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
c	Average computation time	300 sec
d	Average data transfer overhead	0.1
τ	Task processing time	$Normal/c/d$

TABLE II
POWER MODEL: SPECPOWER RESULTS FROM [8] AND NODE HIBERNATION COSTS FROM [16].

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

overhead (d), and node capacity ($Capacity$). We calculate computation time in a deterministic way based on c and node capacity by the equation: $compute\_time = c/Capacity$. In contrast, data transfer time is determined probabilistically. Since no previous work identified distributions for data transfer time in data parallel computing clusters, we employ two distribution models, *normal* and *exponential*, in this study. For *normal*, we pick a random number (v_n) from $N(0, d)$, and the data transfer time is computed by $data\_time = |v_n| \times c/Capacity$. For *exponential*, we select a random number (v_e) from an exponential distribution with $mean = d$, and then compute $data\_time = v_e \times c/Capacity$. That is, d is used to define standard deviation for *normal*, while it determines mean for *exponential*. Finally, a task processing time is computed by adding those two elements, i.e., $\tau = compute\_time + data\_time$. Due to randomness, tasks may have different τ even in a single job.

We compare the following techniques in terms of energy consumption and average turnaround time:

- NPS, without energy management;
- AIS, All-in Strategy;
- Basic, basic CS discovery based on Algorithm 1;
- PA, power-aware CS discovery based on Algorithm 2.

NPS fully utilizes nodes in the cluster, and nodes are in idle mode after jobs are completed. AIS also utilizes the entire set of nodes for processing jobs, but keeps the cluster deactivated as soon as all jobs are completed until the next job arrives. Basic constructs CS dynamically without considerations of node heterogeneity, while PA takes heterogeneity into account.

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 each experiment ten times and provide 95% confidence intervals.

VI. EXPERIMENTAL RESULTS

A. Impact of fraction of low-power nodes

In this experiment, we explore the impact of the fraction of LP nodes in the cluster. We varied the fraction of LP nodes from 0 to 1. 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 intermediate values of f represent mixtures of both classes of nodes.

As shown in Figure 12, PA yields the same results as Basic for both extremes, showing around 30% energy saving compared to NPS regardless of fraction of LP nodes in the cluster. In contrast, we can see that PA further improves energy saving in heterogeneous settings. With $f = 0.75$, PA improves energy saving over 50%, as shown in Figure 12(a). For turnaround time, no significant differences among the various techniques were observed, as in Figure 12(b). This indicates that our PA technique can improve energy saving with little performance loss by exploiting cluster heterogeneity.

AIS also achieves energy saving, but it is less than 20%. One interesting observation is that AIS shows the best result in energy saving in a cluster that only has HP nodes, yielding 40% energy saving. This is because AIS can quickly complete a set of jobs in the queue, and then move to low-power mode in that configuration. However, in the environment where LP nodes exist, slower nodes (i.e., LP nodes in our experiment due to the low capacity) can delay job completion, and it results in the reduced number of opportunities to get into power-saving mode. In the HP-only configuration ($f = 0$), the number of cluster deactivation to low-power mode is 2.5–3 times greater than the other configurations.

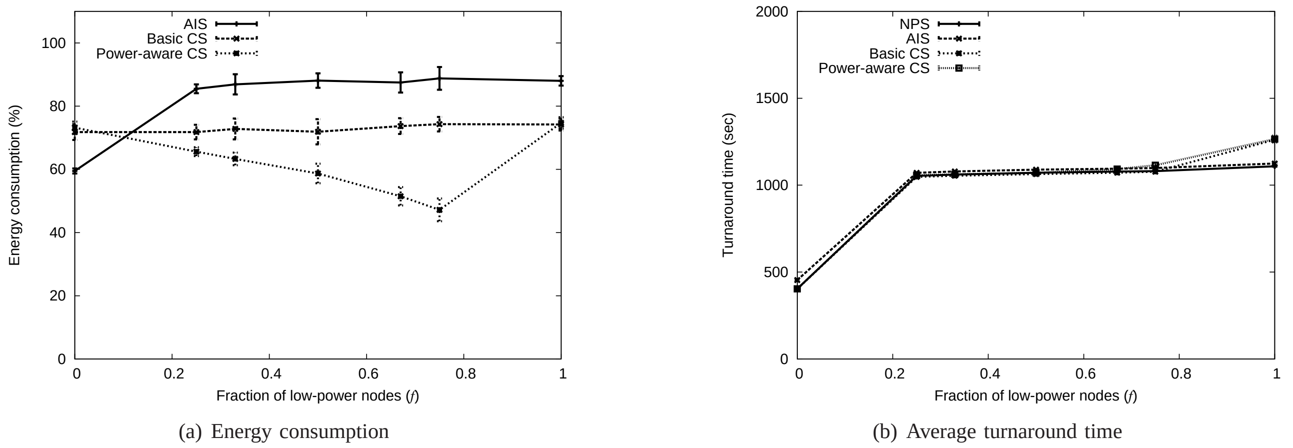
B. Impact of the number of data blocks

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 values for the number of data blocks, from $b = n$ (i.e., 64GB) to $b = 32n$ (i.e., 2TB). Figure 13 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. Basic shows 30–60% energy saving, while PA yields 40–70% saving with no noticeable performance degradation.

C. Impact of job arrival rate

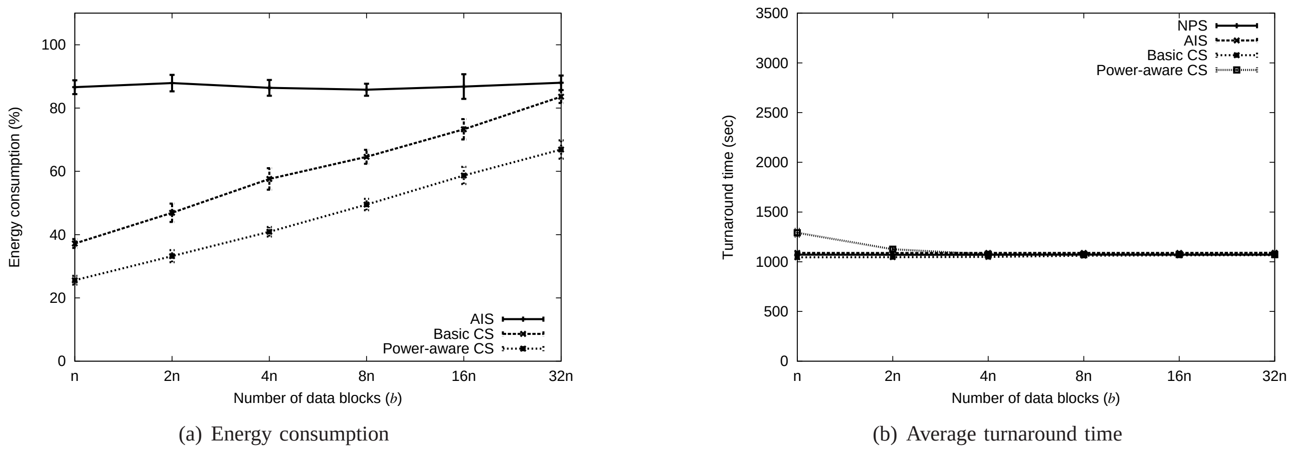
By default, we used job arrival rate $\lambda = 0.5$, since we are more interested in light load environments. In this experiment, we discuss the experimental results under varied job arrival rates. We employed a multiple set of job arrival rates from $\lambda = 0.25$ (for a light load) to $\lambda =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

Figure 14 shows energy and performance as a function of λ. We see no significant changes between our CS techniques, except that PA somewhat degraded in a heavy workload environment $\lambda = 2$. Interestingly, AIS saves energy more well with smaller λ. This is because AIS could lengthen deactivation periods in a light load. In this experiment, AIS yielded around 30% energy saving when $\lambda = 0.25$. However,



(a) Energy consumption (b) Average turnaround time

Fig. 12. Impact of fraction of low-power nodes in the cluster



(a) Energy consumption (b) Average turnaround time

Fig. 13. Impact of the number of data blocks

little energy saving has been observed with greater job arrival rates ($\lambda > 0.5$) for AIS.

D. Impact of computation time

We examine the impact of computation time. In addition to $c = 300s$ used by default, we examined the techniques with three more computation times, $c = 100s$, $c = 600s$, and $c = 1200s$ to consider a diverse range of applications that have different computation requirements. The results showed no noticeable impact of this parameter for both energy and performance, as plotted in Figure 15.

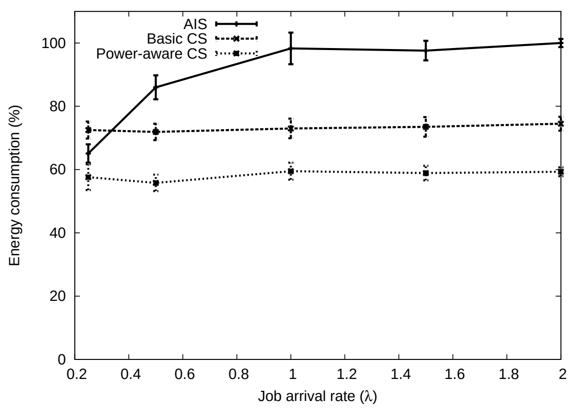
E. Impact of data transfer overhead

In this experiment, we employ several distribution models to consider indeterministic data transfer overhead. As explained in the previous section, we consider two distributions for this — *normal* and *exponential* distributions. Again, we model data transfer overhead by specifying standard deviation for normal distribution and mean for exponential distribution. We used $d = 0.05, 0.1, 0.25$ for this experiment.

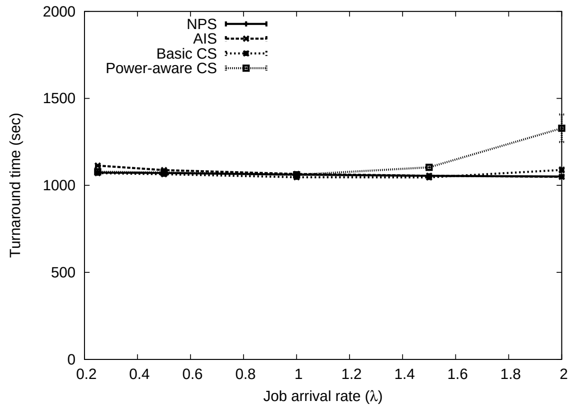
Figure 16 shows the results. In the figure, $Norm(d)$ and $Exp(d)$ represent normal and exponential distributions with data transfer overhead d, respectively. For a diverse set of distributions, we can see that our *CS*-based techniques consistently save energy around 30% for Basic and 40% for PA without significant performance losses.

F. Impact of the number of tasks

We next present our evaluation results showing the impact of the number of tasks on energy and performance. We varied the number of tasks (χ) from $n/4$ to $4n$ (i.e., $\chi = [\frac{n}{4}, 4n]$) for each job. Figure 17 shows the experimental results as a function of the number of tasks. Basic achieves 30% energy saving as compared to NPS, and PA further improves energy saving up to 40%. However, the CS-based techniques showed some extent of performance degradation with the heavy workload where $\chi = 4n$. This result is not surprising as CS-based techniques are designed for light workloads. Nonetheless, we address this problem by providing the concept of multi-level CS sets, as discussed next. As shown in the figure, AIS yields no significant energy saving, and hence no performance penalty.

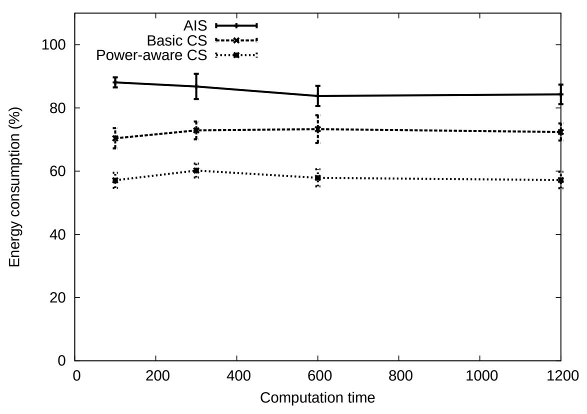


(a) Energy consumption

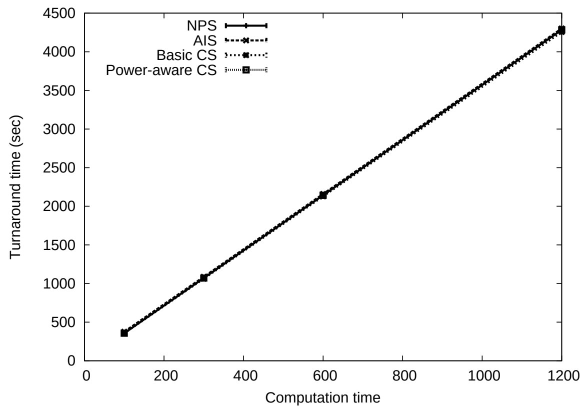


(b) Average turnaround time

Fig. 14. Impact of job arrival rate



(a) Energy consumption



(b) Average turnaround time

Fig. 15. Impact of computation time

G. Evaluation of multi-level CS

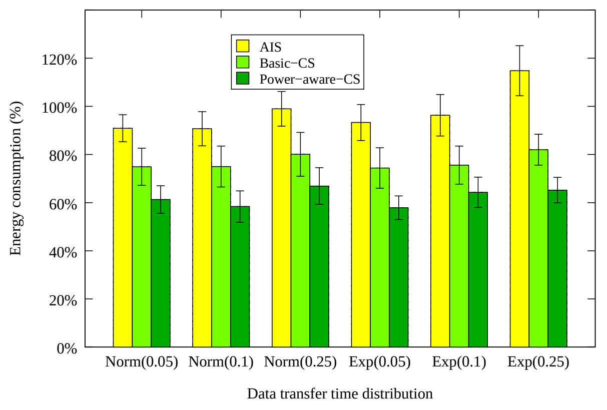
Finally, we present performance and energy consumption of multi-level CS sets. To see the impact more clearly, we used a greater replication factor $r = 5$ and a smaller value for the number of data blocks $b = n$, in this experiment. Thus there can exist four CS sets from *CS-1* to *CS-4*, in addition to the entire cluster. Then we varied λ to see how the CS sets respond to different loads.

Figure 18 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 the average. The figure also shows that *CS-3* achieves 50% energy saving in the same setting, while *CS-2* and *CS-1* further increase energy saving to 70%. With regards 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.

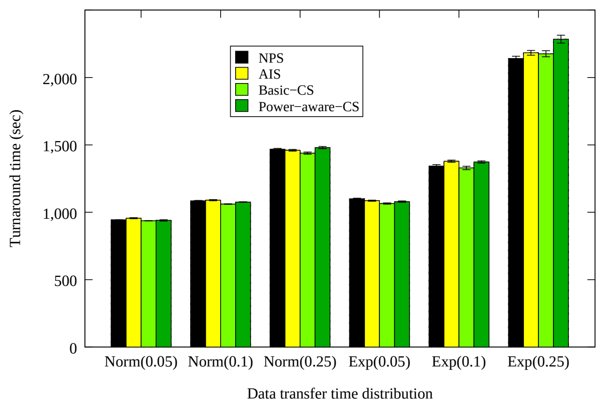
VII. Conclusions

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 subset 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 subset. 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 subset algorithms can significantly reduce energy consumption, up to 70% compared to a non-power saving 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 subset. The results also show that our extended

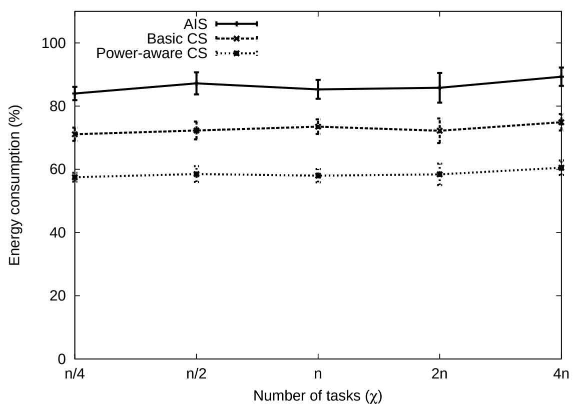


(a) Energy consumption

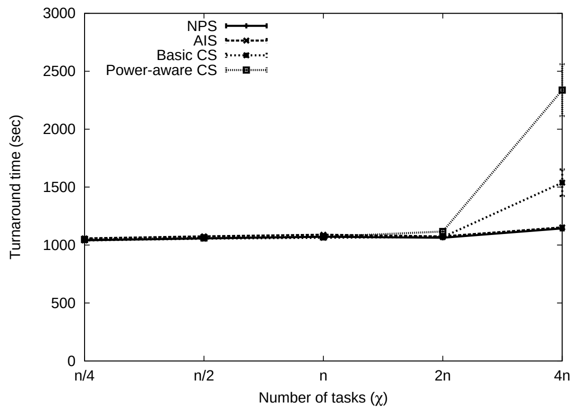


(b) Average turnaround time

Fig. 16. Impact of data transfer distribution



(a) Energy consumption



(b) Average turnaround time

Fig. 17. Impact of the number of tasks

algorithm can be used to provide a coarse-grained level of energy proportionality based on covering subset 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] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris. Scarlett: coping with skewed content popularity in mapreduce clusters. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 287–300, 2011.

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

[4] 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.

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

[6] M. Cardosa, A. Singh, H. Pucha, and A. Chandra. Exploiting spatio-temporal tradeoffs for energy efficient MapReduce in the cloud. Technical Report TR 10-008, University of Minnesota, April 2010.

[7] 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.

[8] 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.

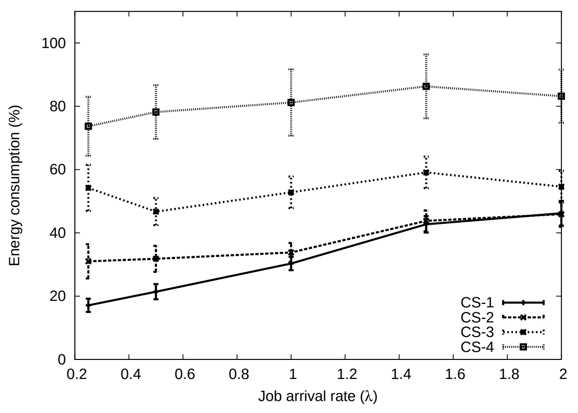
[9] V. Chvàtal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.

[10] 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.

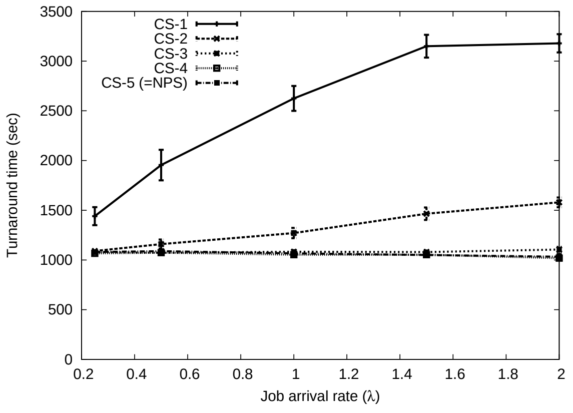
[11] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[12] 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.

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



(a) Energy consumption



(b) Average turnaround time

Fig. 18. Performance and energy consumption for multi-level CS sets

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

[15] 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.

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

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

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

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

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

[21] 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.

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

[23] 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.