International Journal of Applied Engineering Research ISSN 0973-4562 Volume 10, Number 17 (2015) pp 38368-38376
© Research India Publications. http://www.ripublication.com

Dynamic Reconstruction of MapReduce Clusters for Achieving Energy Proportionality

M. Kim
Ph.D. Candidate
Department of Computer Engineering
Yeungnam University, Republic of Korea

H. Cho[*]
Professor
Department of Computer Engineering
Yeungnam University, Republic of Korea

Abstract- MapReduce provides a promising tool for processing the massive data sets and becomes a representative paradigm for building large scale data intensive applications. The principle of *energy proportionality* states that we can save energy by activating only a part of cluster nodes in proportion to the amount of input workload. However, achieving the energy proportionality in MapReduce clusters is challenging since arbitrarily deactivating cluster nodes can make a part of data unavailable. In this paper, we propose algorithms for task allocation and node activation in MapReduce clusters. They consolidate the system load to a minimum set of active groups, and thus can save energy significantly. Furthermore, they are rack-aware and thus can reduce energy consumption of power-hungry rack components, such as cooling, power distribution units, and power backup equipment. Experiment results indicate a reduction in energy consumption up to 30% when compared to previous algorithms.

Keywords: Big data, MapReduce, Energy proportionality, Cluster, Node activation.

Introduction

Large data center operating costs have prompted the consideration of energy management policies. Current solutions can be divided into two categories: *cluster-wide* and *local* [1]. Cluster-wide policies achieve *energy proportionality* by reconfiguring the cluster dynamically. When the cluster load is very high, most nodes should turn on and share the load. However, if the cluster load is significantly lower than the peak load, cluster-wide policies consolidate the load on a subset of nodes and turning off the remaining nodes. As a result, the entire cluster can consume energy in proportion to its load level. Local policies put unused resources in a low-power state. Typical examples include dynamic voltage scaling and power-aware storage management.

This paper focuses on cluster-wide energy management for MapReduce clusters. MapReduce has emerged as an important paradigm for building large scale data intensive applications [2]. It provides a simple and powerful programming model that enables easy development of distributed applications to process vast amounts of data on large clusters of commodity machines. Recent empirical studies show that MapReduce clusters have to support highly diverse workloads consisting of short interactive jobs and long-running batch jobs [3][4]. Furthermore, some workload has high peak-to-average ratios [5]. A cluster provisioned for the peak load would be often underutilized and waste a great deal of energy. This means that MapReduce clusters have high potential to save energy by cluster-wide energy management.

In this paper, we assume that the cluster is partitioned into several groups. Each group includes one replica of every data item. To save energy, we exploit both *inter-group strategy* and *intra-group strategy*. The inter-group strategy consolidates system load to a minimum set of active groups so that nodes in other groups can stay at low-power mode for a long time. The intra-group strategy leverages rack-aware node activation to reduce the number of active racks for each group. A recent study on data center power management shows that rack-based power management can lead to several times more energy savings than other solutions that focus only on the power consumed by node [6]. The contribution of this paper can be summarized as follows.

- We first propose *task allocation algorithms* that select target nodes to execute input tasks. The goal is to consolidate the system load to a minimum set of active groups. We present two alternatives on node selection, group-based and locality-based, which shows an interesting trade-off between energy consumption and performance.
- Then we propose *node activation algorithms* that determines a node to be activated when the system load increases. Considering the activation relationship of group, rack, and node, we propose three node activation algorithms, entire activation, rack-based activation, and locality-based activation. They also exploit a trade-off between energy consumption and performance.
- We develop an experiment model of MapReduce cluster and evaluate the proposed algorithms with respect to energy consumption and performance.

[*] Corresponding author: hrcho@yu.ac.kr

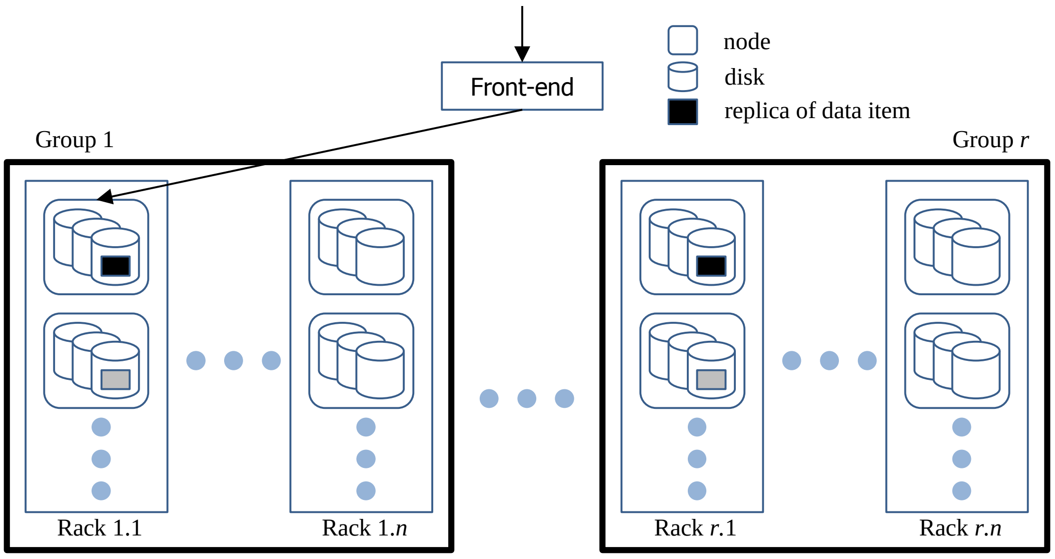


Fig.1. Cluster architecture

The rest of this paper is organized as follows. Section 2 presents related work and explains its limitations. Section 3 describes the cluster architecture that we assume in this paper. Section 4 proposes algorithms for task allocation and node activation. Section 5 describes the experiment model and Section 6 analyzes the experiment results. Finally, the concluding remarks appear at Section 7

Related Work

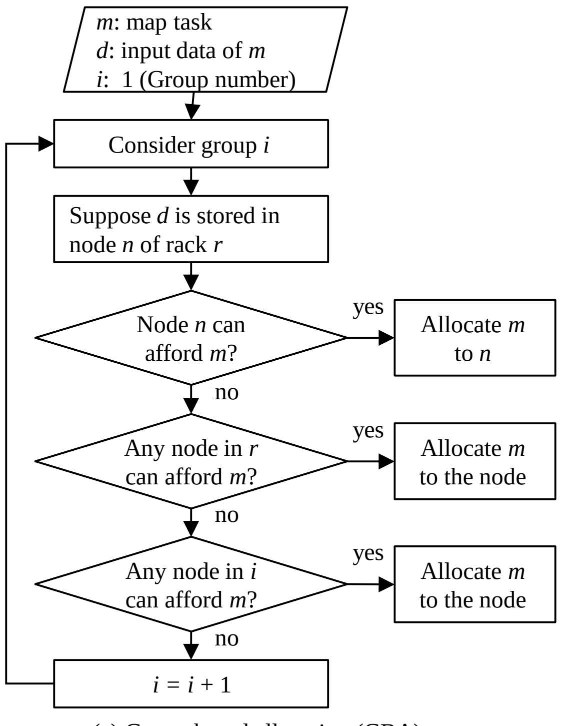
A number of studies have reported cluster-wide energy management methods for MapReduce clusters. The *covering set* (CS) method [7] is the initial work. It keeps one replica of every data item within a small subset of nodes called CS nodes. CS nodes remain fully powered to preserve data availability while the rest is powered down. However, this method supports limited energy proportionality. The cluster is in either full performance mode with every node activated or energy mode with only the CS nodes activated.

Sierra [8] motivates our work the most. It extends the CS method by partitioning the cluster into γ groups, where γ is a replication factor. Each node belongs to one of the groups so that one replica of each data is placed to each group. Similar to our work, Sierra reconfigures the cluster by activating or deactivating each group according to the system load. However, it may suffer from two limitations. First, it lacks an elaborate task allocation scheme that determines which nodes are going to execute incoming tasks. As we will describe at Section 4, the number of idle nodes can be different according to the task allocation algorithm, especially when the variance of cluster load is significant. By maximizing the number of idle nodes, we can increase the probability of deactivating racks and groups. The next limitation of Sierra is that it does not consider the order of node activation for each group. We will show that activating nodes at random order causes unnecessary energy consumption and longer execution time for each task.

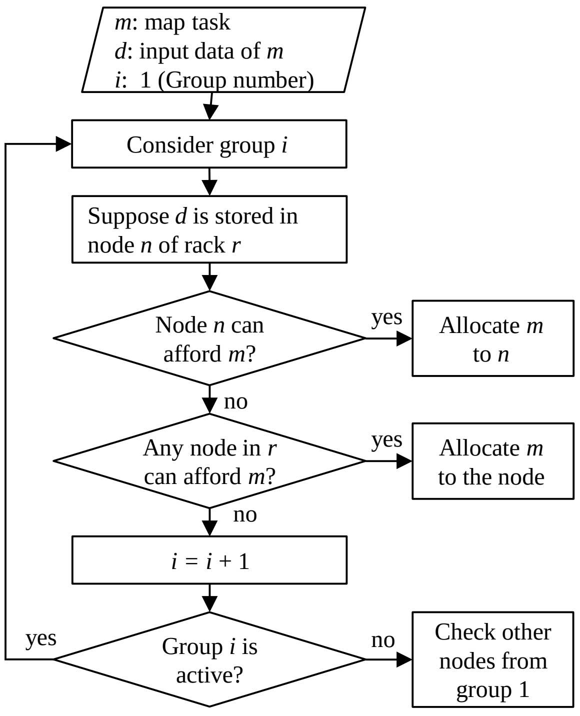
GreenHDFS [9] divides the cluster into disjoint hot and cold groups. The frequently accessed data is placed in the hot group, which is always powered on. Similarly, BEEMR [5] partitions the cluster into disjoint interactive and batch group. The interactive group is always powered on. It runs all of the interactive jobs and stores every data for the jobs. Both algorithms can save energy by deactivating another group (cold or batch) for a long time. The performance loss of active group can also be minimal since the number of replicas of frequently accessed data is not reduced. The drawback of both algorithms is that any jobs should experience considerable delay when they access data stored in the deactivated group.

AIS (All-in Strategy) [10] is a totally different approach. It runs the given jobs with the entire set of nodes in the cluster to complete them as quickly as possible. Upon completion of the jobs, every node is deactivated to save energy until the next run. One potential drawback is that even with small jobs, AIS still needs to wake up the entire cluster, possibly wasting energy. Furthermore, it cannot support real-time jobs that require fast response time and deadline constraint [11].

Recently, iPACS [12] presents a CS set discovery algorithm that finds an energy-optimized node set to satisfy the given data requirement. Unlike Sierra where each group includes a replica of entire data set, the CS set contains a replica of required data set under the assumption that the data set can be available for the next time window. To achieve the energy proportionality, iPACS also proposes a multi-level CS discovery algorithm. The level in this case means the number of replicas. When the system load increases, iPACS finds a higher level CS set that contains more replicas of the required data set. The problem of iPACS is the difficulty of predicting the data set of the next time window. If the CS set does not include the data set actually accessed by some jobs, the jobs should experience considerable delay.



(a) Group-based allocation (GBA)



(b) Locality-based allocation (LBA)

Fig.2. Task allocation algorithms for map task

Cluster Architecture

Figure 1 shows the baseline cluster architecture. It follows the conventional master-slave architecture of Hadoop [13]. Basically, the cluster is a *shared-nothing* architecture, where each node has its own set of private disks and only the owner node can directly access its disks. Each file is split into smaller data blocks and these blocks are distributed across the cluster. A block is replicated to different nodes to ensure data availability. The cluster stores γ replicas for each block, and γ is called as a *replication factor*. The default setting of γ is three in Hadoop.

The cluster is partitioned into γ groups, and each group stores one replica of every data item. This means that there are γ replica for each data item. Group 1 is a *primary group*. It has a role to CS nodes [7] and thus always powered on. As a result, the primary group can ensure the immediate availability of data items required for real-time jobs. Other groups may be activated or deactivated according to the system load. This way we can support the principle of *energy proportionality* such that the amount of energy consumed by the cluster is proportional to the amount of work performed.

We assume that each group consists of *n* distinct racks and all nodes in a rack are in the same group. This permits the entire rack to be turned off when the owner group is deactivated, allowing additional power savings by turning off rack-wide equipment such as switches [6][8]. We also assume that there is more bandwidth available between nodes on the same rack compared to nodes on different rack. Note that this assumption is effective for most configurations of current data centers. For example, Hadoop prefers within-rack transfers to off-rack transfers when placing MapReduce tasks on nodes [13].

The front-end is a single point of contact for a client wishing to execute a MapReduce job. Typically, a MapReduce job is divided into a number of map tasks and reduce tasks. The front-end allocates each task to a node in some active group. The task allocation has to be performed in an energy-efficient manner not to activate excessive groups compared to the current load. Furthermore, it tries to take advantage of data locality to reduce the amount of inter-node or inter-rack communications. We will describe the proposed task allocation algorithms at Section 4.A. The front-end also monitors the system load and compares it to the processing capacity of active groups. If the system load increases, the current set of active nodes would not afford additional tasks. Then the front-end has to turn on new nodes. There are some alternatives to select new nodes to be activated. We will discuss this issue at Section 4.B.

Proposed Algorithms

In this section, we first describe task allocation algorithms that determine a node to execute a task. Then we propose node activation algorithms that determine a node to be activated to process excessive load.

A. Task allocation algorithms

We propose two task allocation algorithms, named *group-based allocation* (GBA) and *locality-based allocation* (LBA). Figure 2 presents the details of both algorithm for allocating a map task. They take advantage of data locality to reduce inter-node and inter-rack communications.

Specifically, they first check if the task is *data-local*, that is, running on the same node that the required data resides on. If the node has available slots for the task, the task can be allocated to the node. Otherwise, both algorithms check if the task is *rack-local*: on the same rack, but not the same node, as the input data. Some task is neither data-local nor rack local. For the task, GBA checks if any other node in the current group is available to execute it. On the other hand, LBA checks if the task is data-local or rack-local for the next group. If it is not data-local and rack-local for every active group, then LBA checks the availability of other nodes from group 1 again.

Note that GBA allocates a task to any node in group 2 only if every node in group 1 cannot afford new tasks. This means that GBA can consolidate the system load to the limited number of groups. As a result, more groups can be deactivated and the amount of energy reduction would be significant. LBA adopts a different approach. It allocates the task to group 2 in case of data-local or rack-local even though some nodes in group 1 are available. Since LBA takes advantage of data locality more significantly, it could outperform GBA with respect to the execution time. However, more groups would be activated at the same time and thus the amount of energy reduction is limited.
Allocating reduce tasks are rather simple. To exploit data locality, they can be allocated to the same node or same rack of preceding map tasks. If the node or the rack is not available, the front-end will simply take any available node from group 1.

B. Node activation algorithm
If the set of active groups cannot afford current system load, the front-end has to activate a new group. Considering to the activation relationship of group, rack, and node, we propose the following node activation algorithms.

- **Entire activation (EA)**: When a group is activated, every rack and node in the group is activated at the same time.

- **Fractional activation (FA)**: When a group is activated, only part of entire racks and nodes of the group are activated according to the system load. To select racks and nodes to be activated, the front-end considers the required data of yet-to-be-run tasks. There are two algorithms of the fractional activation.

  - **Rack based activation (FA-RBA)**: It first turns on a rack that includes the most required data of yet-to-be-run tasks. Within the rack, nodes are activated in order according to the system load. Only if every node in the active racks is activated, FA-RBA turns on a new rack.

  - **Locality based activation (FA-LBA)**: It selects an inactive node that includes the most required data of yet-to-be-run tasks. If the node is included in inactive rack, then FA-LBA turns on the rack first and then the node.

Note that those algorithms show an interesting trade-off between energy consumption and performance. First, EA should spend energy the most since all members of the group are activated regardless of the system load. However, it can cope with the following increase of system load without any turn-on delay. FA activates only part of the group according to the system load. If the system load increases thereafter, the rest should be activated with considerable turn-on delay. This means that FA has some limitation to adapt dramatic load variation. However, it can save energy significantly compared to EA.

The trade-off between energy consumption and performance is also existing between FA-RBA and FA-LBA. FA-RBA tries to reduce the number of active racks. Since 70% of data center power goes toward rack-related components, such as cooling, power distribution units, the switch gear, and power backup [6], reducing the number of active racks must contribute to save energy significantly. On the other hand, FA-LBA just activates nodes that store the required data the most regardless of rack activation. As a result, it is possible that many racks of a group are activated while there are very small number of active nodes for each rack. FA-LBA increases the probability of data-local at the cost of high energy consumption.

Experiment Model

To evaluate the performance of proposed algorithms, we developed a simulation model of the MapReduce cluster using CSIM discrete-event simulation package [14]. Table 1 shows the simulation parameters. The parameter values will be used for most experiments unless otherwise noted. Many of their values are adopted from [6].

TABLE.1. Simulation Parameters

Parameters	Value
Number of groups	3
Number of racks	150
Number of nodes	3000
Number of Map slots per node	10
Number of Reduce slots per node	5
Rack transition time	300 sec
Node transition time	30 sec
Maximum duration of staying at idle state	15 min
Energy consumption of idle node	100 w/h
Energy consumption of active node	500 w/h
Rack power overhead for each node	50%
Fixed energy consumption of active rack	5 kw/h
Intra-rack network bandwidth	64 Gbps
Inter-rack network bandwidth	128 Gbps
Data size of each group	80 TB
Load skew factor (θ)	0.0 : 1.0
Portion of hot data	4%
Probability of accessing hot data	80%

The cluster consists of three groups and each group has 50 racks. The first group is a primary group and is always powered on. The other groups can be deactivated according to the system load. We assume a homogeneous

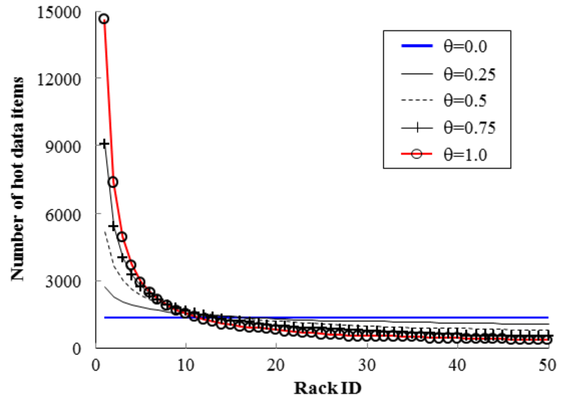


Fig.3. Distribution of hot data items on various θ

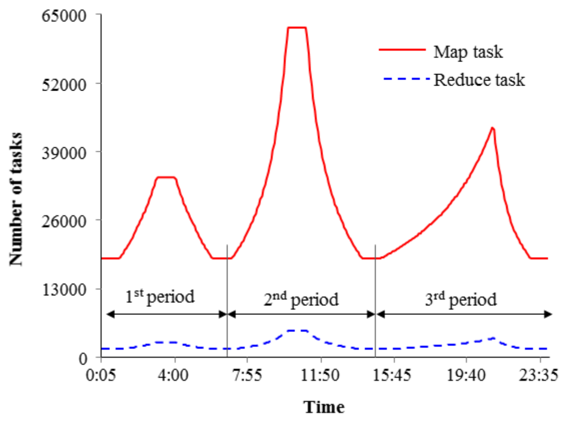


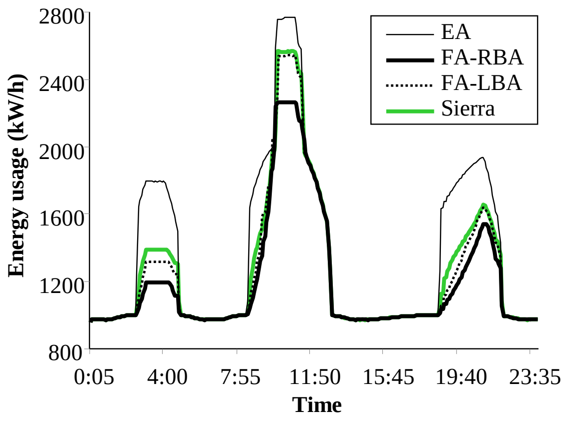
Fig.4. Load variation mode

computing environment where the hardware specification of each device type is same. Specifically, a rack includes 20 nodes and every node has 10 map slots and 5 reduce slots. That means a node can execute 10 map tasks and 5 reduce tasks at the same time. Each node can stay at one of three execution states: *active*, *idle*, and *off*. An off node does not consume any energy. A node is in active state when it executes some tasks. When the node completes every assigned task, it moves to the idle state. We assume that the energy consumption of idle node is much lower than that of active node. Furthermore, if any node in non-primary group stays at the idle state for longer than the threshold (15 min), the front-end turns off the node. The time taken by the node to go between idle and off states is set to 30 seconds.

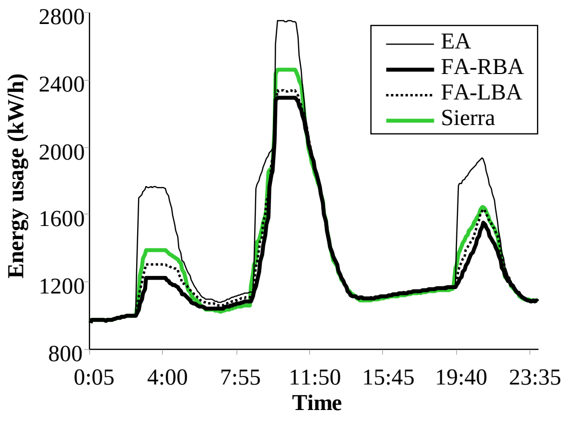
The state of rack is modeled similarly. If any nodes in a rack is active, the rack is also active. If every node in the rack is in off state, it moves to the off state. The transition delay taken by the rack is set to 300 seconds same to [6]. An active rack consumes energy in proportion to the number of active nodes. Similar to [6], we configure rack overhead to be 50%; i.e, the support-infrastructure like cooling system on each rack consumes 50% as much energy as the nodes on the rack. Since the rack may also have equipments like interconnect bay and power backup that spend energy independent of node states, we also model that there is significant fixed energy consumption for each rack [15].

A group contains one replica for each data item. We configure that the data size of each group is 80 TB, and it is evenly distributed to the nodes. To model the access skew, we categorize the entire data items into *hot set* and *cold set*. A data item in hot set has a high probability of being accessed by tasks. The load skew factor, θ, determines how much portion of hot set is assigned to each rack. Suppose that D_{hot} represents the size of entire hot set. We set D_{hot} to 3.2 TB which corresponds to 4% of the entire data set (= 80 TB). If each group inclusdes N racks, the size of hot set in K-th rack is determined by the following Zipf-like distribution expression, $D_{hot} * \frac{1/k^{\theta}}{\sum_{i=1}^{N}(1/i^{\theta})}$. If θ is set to 0, every rack has the same number of hot data items. On the other hand, if θ is set to 1, the first few racks store most of hot data items. Figure 3 shows the number of hot data items assigned to every rack of a group on different settings of θ. The size of a data item is set to 64 MB.

Figure 4 depicts a load variation model used throughout the experiments. It consists of three periods, each of which has a load increasing stage and a load decreasing stage. The first period lasts for the first seven hours. It is intended to model the initial situation of the cluster where only the primary group is active and every node of the primary group does not cache anything. The second period starts after the first period. In this period, the load increases up to the maximum cluster capacity and then decreases dramatically. The last period starts near fifteen hours. Unlike the previous periods, the load increases slowly and thus we can compare the performance of proposed algorithms under different load increasing speed. We believe that our load variation model can capture most of real workload characteristics related to MapReduce clusters [5, 9].

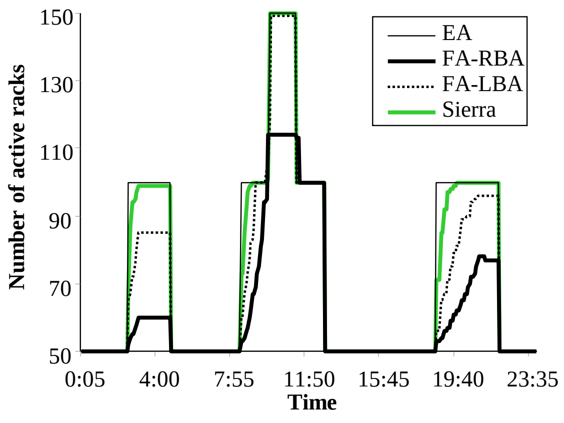


(a) Task allocation algorithm = GBA

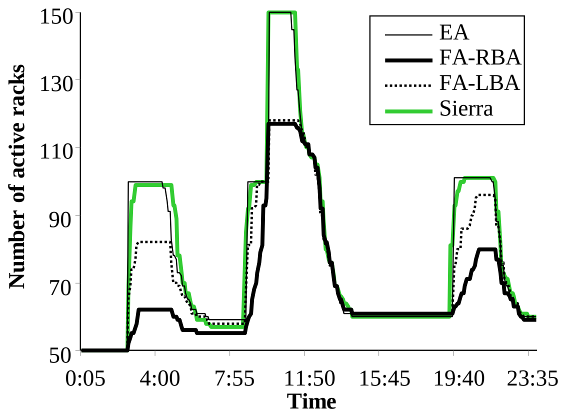


(b) Task allocation algorithm = LBA

Fig.6. Energy usage ($\theta = 0.5$)



(a) Task allocation algorithm = GBA



(b) Task allocation algorithm = LBA

Fig.5. Number of active racks ($\theta = 0.5$)

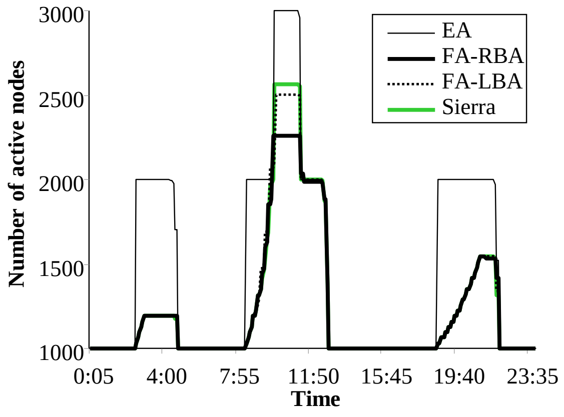
Experiment Results

We implement two versions of task allocation algorithms, GBA and LBA, and four versions of node activation algorithms, EA, FA-RBA, FA-LBA, and Sierra [8] that chooses the activated node at random. Performance metrics are *energy usage* and *response time* (turnaround time). The energy usage is the aggregate power consumed by every rack and node. We use the number of active racks and active nodes as a secondary metric to explain the variation of the energy usage. The response time in seconds is measured as the difference between when a task is submitted and when the task successfully commits.

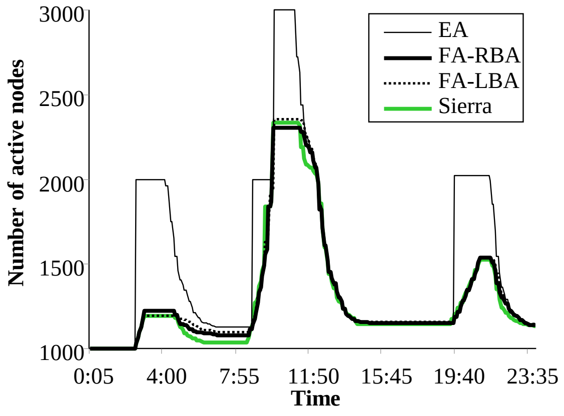
A. Energy usage

We first compare the energy usage. Figure 5 shows the experiment results. As expected, EA consumes energy the most because it turns on every rack and node in the activated group without regard to the system load. On the other hand, the fractional activation algorithms can save energy by turning on part of racks and nodes of the group according to the system load.

Among the fractional activation algorithms, FA-RBA performs best. It can save energy up to 30% compared to EA as Figure 5 shows. This is because FA-RBA turns on a new rack only if all nodes in active racks are active. Figure

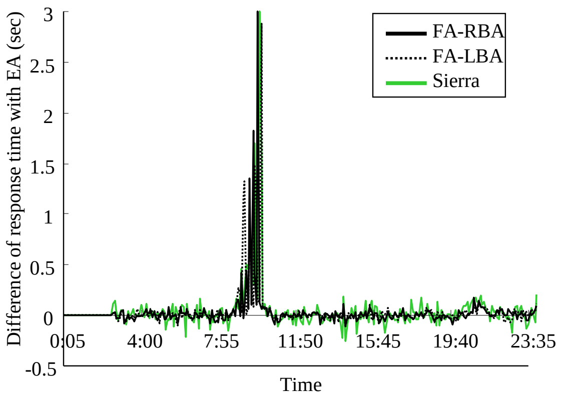


(a) Task allocation algorithm = GBA

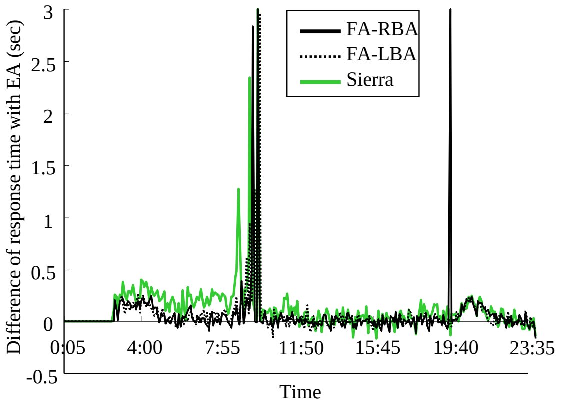


(b) Task allocation algorithm = LBA

Fig.7. Number of active nodes (θ = 0.5)



(a) Task allocation algorithm = GBA



(b) Task allocation algorithm = LBA

Fig.8. Difference of response time from EA (θ = 0.5)

6.a and 6.b show that the number of active racks is the smallest at FA-RBA. Both FA-LBA and Sierra consume more energy due to large number of active racks. The difference of the number of active nodes is not significant among the fractional activation algorithms as Figure 7.a and 7.b show. Since Sierra selects a new node at random manner, the selected node may be any rack in the group. This means that Sierra would turn on most racks for each activated group. FA-LBA also turns on more racks compared to FA-RBA. This is because FA-LBA selects a new node that stores the requested data items the most, even though the node is in inactive rack. It prefers to support fast response time rather than reduce energy consumption.

Task allocation algorithms also affect the energy usage. Specifically, LBA consumes more energy than GBA. The difference is significant between the second period and the third period. Since GBA tries to distribute the system load to a minimal number of groups, it has more chances to turn off any group compared to LBA. From Figure 6.a and 7.a, we can guess that GBA activates the primary group only between the second period and the third period since the number of active racks and nodes correspond to the size of a group. On the other hand, as Figure 6.b and 7.b illustrate, LBA turns on more racks and nodes at that duration, which means that LBA activates additional groups.

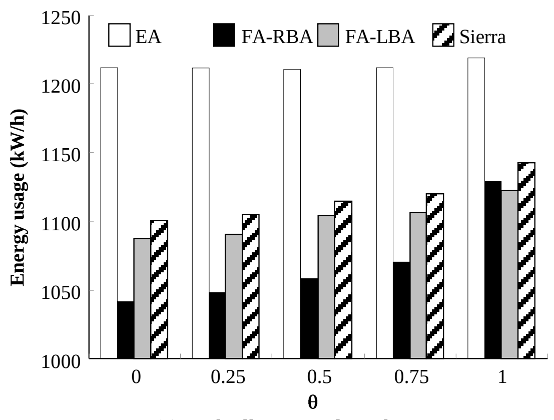
B. Response time

We also compare the response time of proposed algorithms. Figure 8 shows the difference of response times of proposed algorithms from EA. We select EA as a baseline since EA turns on every rack and node immediately for an active group. As a result, it may not experience any transition delay for any tasks allocated to the group.

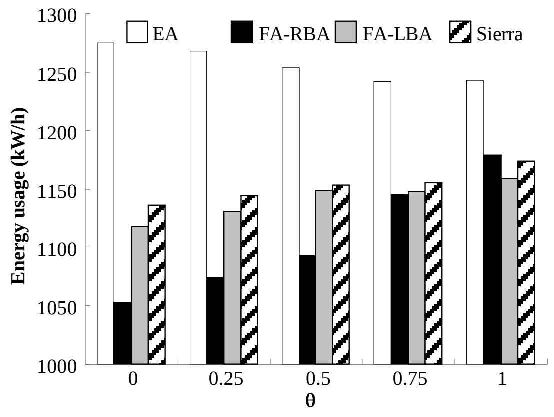
For the most part, the differences of response time between node activation algorithms are not significant. This is especially true for GBA as Figure 8.a shows. In LBA, the difference increases at the first period, and Sierra performs worst among them. At the second period, every fractional algorithm performs worse dramatically for some point where the load increases up to the maximum cluster capacity. Since every rack and node in off state should be activated, many tasks experience transition delay waiting for the rack and node will be active. Similar phenomenon also occurs once for FA-RBA at the third period. FA-RBA tries to reduce the number of active racks, and thus it may suffer from large transition delay when the system load increases suddenly and requires many racks being activated.

C. Access skew

The last experiment compares the energy usage on various settings of θ. Figure 9 shows the average energy usage at the first period. When θ is 0, FA-RBA performs best and it can save energy about 20% of EA. Both FA-LBA and Sierra



(a) Task allocation algorithm = GBA



(b) Task allocation algorithm = LBA

Fig.9. Average energy usage on various θ

consume much energy compared to FA-RBA. Since every rack of a group has same number of hot data items when θ is 0, the access set of tasks distribute the entire racks. This means that FA-LBA turns on most racks while very few nodes would be activated at each rack. As a result, the fixed energy consumption of active rack takes significant part of aggregate energy consumption. FA-RBA can reduce the fixed energy consumption since it tries to minimize the number of active racks.

The energy consumption of FA-RBA increases in proportion to θ, and it consumes more energy than FA-LBA when θ is 1. Note that the first few racks store most of hot data items when θ is 1. Let us suppose that 'hot rack' is one that stores hot data items the most. FA-LBA assigns a task to the hot rack only if it accesses hot data items. The execution time of the task should be relatively short since it does not incur inter-rack communication. On the other hand, in FA-RBA, the hot rack may be assigned to tasks accessing other data items. This causes to make the execution time of the tasks longer. Since many tasks have to access data items in hot rack, the execution time of the tasks may be also prolonged. As a result, more racks should be required to provide map/reduce slots for incoming tasks and thus FA-RBA may consume much energy.

Concluding Remarks

In this paper, we consider cluster-wide energy management for MapReduce clusters. We first propose task allocation algorithms that select target nodes to execute incoming tasks. Then we propose node activation algorithms that determine a new node to be activated when the system load increases. The proposed algorithms consolidate the system load to a minimum set of active groups, and thus can save energy significantly. Furthermore, they are rack-aware and thus can reduce energy consumption of power-hungry rack components, such as cooling, power distribution units, and power backup equipment.

To evaluate the performance of proposed algorithms, we develop a simulation model of MapReduce clusters. The important experiment results are summarized as follows. First, the group-based allocation can reduce the number of active racks and thus can save energy considerably compared to the locality-based allocation. Next, the rack-aware fractional activation outperforms other node activation algorithms with regard to the energy usage. The performance improvement is up to 30% when compared to entire activation algorithm. Furthermore, its response time is comparable to other algorithms in most cases.

References

[1] Cho, H., 2012, "Energy management for a real-time shared disk cluster," J. Supercomputing, 62(3), pp. 1338-1361.
[2] Sakr, S., Liu, A., and Fayoumi, A., 2013, "The family of MapReduce and large-scale data processing systems," ACM Comp Surveys, 46(1), pp. 1-44.
[3] Chen, Y., Alspaugh, S., and Katz, R., 2012, "Interactive analytical processing in big data systems: a cross-industry study of MapReduce workloads," J. Proc. of VLDB Endowment, 5(12), pp. 1802-1813.
[4] Ren, K., Kwon, Y., Balazinska, M., and Howe, B., 2013, "Hadoop's adolescence - an analysis of Hadoop usage in scientific workloads," J. Proc. of VLDB Endowment, 6(10), pp. 129-139.
[5] Chen, Y., Alspaugh, S., Borthakur, D., and Katz, R., 2012, "Energy efficiency for large-scale MapReduce workloads with significant interactive analysis," Proc. 7th ACM European Conf. Computer Syst. (EuroSys'12), pp. 43-56.
[6] Ganesh, L., Weatherspoon, H., Matian, T., and Birman, K., 2013, "Integrated approach to data center power management," IEEE Trans. Computers, 62(6), pp. 1086-1096.
[7] Leverich, J., and Kozyrakis, C., 2010, "On the energy (in)efficiency of hadoop clusters," Operating Syst. Review, 44(1), pp. 61-65.
[8] Thereska, E., Donnelly, A., and Narayanan, D., 2011, "Sierra: practical power-proportionality for data center storage," Proc. 6th ACM European Conf. Computer Syst. (EuroSys'11), pp. 169-182.
[9] Kaushik, R., Bhandarkar, M., and Najrstedt, K., 2010, "Evaluation and analysis of GreenHDFS: a self-adaptive, energy-conserving variant of the Hadoop

distributed file system," Proc. 2nd IEEE Int. Conf. Cloud Comp. Tech. and Sci. (CloudCom'10), pp. 274-287.
[10] Lang, W., and Patel, J. M., 2010, "Energy management for MapReduce clusters," J. Proc. of VLDB Endowment, 3(1-2), pp. 129-139.
[11] Borthakur, D., et al, 2011, "Apache hadoop goes realtime at Facebook," Proc. 2011 ACM SIGMOD, pp. 1071-1080.
[12] Kim, J., Chou, J., and Rotem, D., 2014, "iPACS: Power-aware covering sets for energy proportionality and performance in data parallel computing clusters," J. Parallel Distrib. Comput., 74(1), pp. 1762-1774.
[13] White, T., 2012, Hadoop - The definitive guide (3rd Edition), O'Reilly.
[14] Mesquite Software, Inc., 2009, User's guide of CSIM20 simulation engine.
[15] Patil, V., and Chaudhary, V., 2013, "Rack aware scheduling in HPC data centers: an energy conservation strategy," Cluster Comput., 16(3), pp. 559-573.