Making Cluster Applications Energy-Aware

Nedeljko Vasić
EPFL Lausanne, Switzerland
nedeljko.vasic@epfl.ch

Martin Barisits
EPFL Lausanne, Switzerland
martin.barisits@epfl.ch

Vincent Salzgeber
EPFL Lausanne, Switzerland
vincent.salzgeber@epfl.ch

Dejan Kostić
EPFL Lausanne, Switzerland
dejan.kostic@epfl.ch

ABSTRACT

Power consumption has become a critical issue in large scale clusters. Existing solutions for addressing the servers' energy consumption suggest "shrinking" the set of active machines, at least until the more power-proportional hardware devices become available. This paper demonstrates that leveraging the sleeping state, however, may lead to unacceptably poor performance and low data availability if the distributed services are not aware of the power management's actions. Therefore, we present an architecture for cluster services in which the deployed services overcome this problem by actively participating in any action taken by the power management. We propose, implement, and evaluate modifications for the Hadoop Distributed File System and the MapReduce clone that make them capable of operating efficiently under limited power budgets.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; D.4.3 [**File Systems Management**]: File Systems; D.4.7 [**Organization and Design**]: Distributed systems

General Terms

Design, Management, Reliability

Keywords

Cluster services, Cluster applications, Energy-Awareness, Storage

1. INTRODUCTION

Power consumption has traditionally represented a critical issue for devices such as notebooks and personal digital assistants (PDAs) since these devices generally run on batteries. Recent research has however focused on reducing power consumption in cluster systems as well, mostly motivated by their operational costs and reliability. For instance, the data centers can consume as much energy as a city if the number of servers reaches a certain level [13]. Moreover, computing in an environment with high temperatures significantly increases the risk of failure [8]. To address the problem of overheating, data centers are forced to spend significant resources on cooling devices, which again increases the overall power consumption.

ACDC'09, June 19, 2009, Barcelona, Spain.

The EPA [7] estimates that U.S. data centers will consume around 100 billion kilowatt hours annually by 2011, representing a $7.4 billion annual cost. The reason for such a large overall cost is twofold. First, data centers and their services have become important for many sectors of the economy. Second, the same study has shown that the consumed energy has more than doubled in a period between 2000 and 2006, mostly because servers have become more power hungry. Therefore, the report suggested certain opportunities for improving the energy efficiency of data centers, including dynamically matching hardware resources to the load.

Most of the work on energy saving focuses on minimizing the energy consumption while keeping the same overall performance level. However, little research [4, 12] has concentrated so far on minimizing the potential damage caused by a power management system when it is forced to reduce the energy consumption when existing solutions which individually address various aspects of energy savings are not sufficient. It is indeed easy to imagine a case where the only action that a power management system can take is to force some of the machines to enter a sleep state. This could be due to, for example, a "thermal emergency" [4, 12], when it is necessary to immediately reduce power consumption to avoid jeopardizing the energy supply of the entire system. In this case, the energy consumption represents the hard constraint of the optimization problem, rather than the objective function as it was the case in most of the previous work on this topic.

In this paper we demonstrate that two rather popular building blocks for distributed applications are neither capable of operating at multiple power levels, nor of dealing with energy consumption limits. We therefore suggest a simple energy-aware design relying on a **common control plane**, whereby the power management defines the goals that have to be accomplished (energy-wise), and lets the deployed services **actively collaborate** by suggesting actions that, from their standpoint, **minimize the impact** on performance and data availability.

2. CASE STUDY

In this section we analyze the capabilities of currently available distributed applications to adapt to radical reductions in energy consumption or multiple power modes. We focus here on two important blocks for developing such applications: 1) Distributed File Systems, and 2) MapReduce, which presents an important programming model for large scale applications aimed at processing and generating large amounts of data in parallel. Both building blocks are provided by an open source project, Hadoop [1], that provides a software framework for distributing and running applications on clusters of servers, inspired by Google's File System [10] and Google's MapReduce [21].

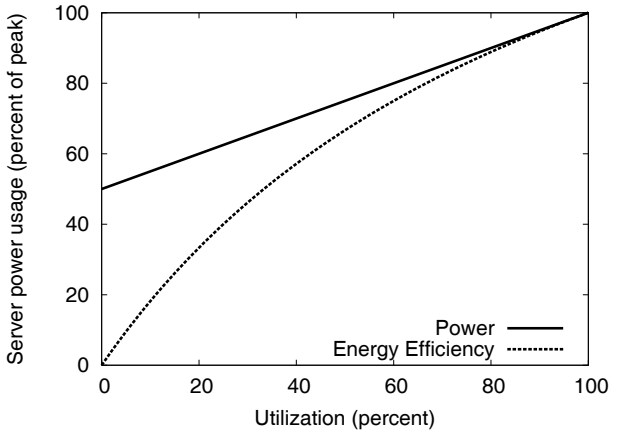

Figure 1: Server power usage and energy efficiency at varying utilization levels, from idle to peak performance. Even an energy-efficient server still consumes about half its full power when doing virtually no work. Source: [2].

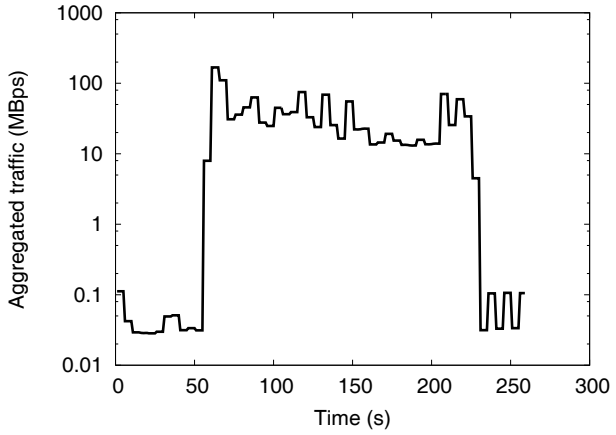

Figure 2: Aggregated network traffic when 30% DataNodes enter the sleep state.

To explore most of the available opportunities for energy savings in cluster computing, we sketch the results of Google's server utilization and energy consumption study [2]. The solid line in Figure 1 represents the power consumption as a function of the utilization, while the dashed line represents the energy efficiency, computed as the ratio between the utilization and the corresponding power value, again as a function of the utilization. The energy efficiency peaks at full utilization and significantly drops as the utilization level decreases. For instance, the power consumption at nil utilization (0%) is still pretty high, around 50%. The typical operating region (lies between 20% and 50% utilization) makes the problem even worse. Thus, Figure 1 also suggests that aggregating the workload by dynamically resizing the active server set is the best decision from the power management point of view, at least until more energy efficient hardware becomes available. By doing so, a certain number of machines can conserve energy through entering some of S-states such as S3 (so called "suspend to RAM"), S4 ("suspend to disk") and S5 ("soft off"). From now on, we will refer to this action as *entering the sleep state*. Exposing the Hadoop's distributed file system (HDFS) and MapReduce to such actions and observing their energy-efficient characteristics are therefore next steps to be tackled.

HDFS contains one master, the so-called *NameNode*, and several *DataNodes*, which are the slaves. As the *NameNode* represents the hot spot of the system, HDFS features also a *secondary NameNode* which ensures correct system operation in case of a master failure. The elementary data storage entities are called blocks, which can be up to 64 MB large. Files larger than this value are partitioned into blocks. If a *DataNode* wants to insert a file, it stores a copy of this file locally and replicates it twice onto remaining *DataNodes* (the default block replication factor is three).

Each *DataNode* periodically sends heartbeat messages to the *NameNode* in order to allow monitoring of its availability. A *DataNode* is declared as dead if no heartbeats are received within a certain time window. The *NameNode* does also periodically verifies if all replicas of the system are available and not corrupted. If a block is found to be under-replicated or erroneous, it is automatically replicated again.

Next we observe how the HDFS behaves under realistic actions taken by the cluster power management system. To emulate the scenario in which the power management module intentionally lets randomly chosen 30% of the *DataNodes* enter the sleep state, we kill the HDFS processes on these machines. Figure 2 shows the aggregate network traffic during this experiment. The leftmost side of the graph corresponds to the system's standby traffic, resulting from heartbeat signals, report messages and other background tasks (10 to 30 kB/s on average). After the HDFS processes are killed, the *NameNode's* heartbeat messages fail and the system instantly starts to replicate all the blocks belonging to the sleeping nodes. The reason for this so-called **panic phase** is that the system checks the availability of each stored block, and if the number of replicas is below a certain threshold, it automatically replicates the block onto other machines. The resulting system peak traffic is around 110 MB/s.

These results show that implementing a power-saving strategy without an energy-aware HDFS design generates heavy network traffic, which in addition is completely unnecessary - the data stored on sleeping nodes is temporarily unavailable and overall durable. Furthermore, replicating blocks already stored on sleeping DataNodes wastes space and interferes with the system operation. An even worse scenario can occur if the power management system observes the additional load caused by panic phase, interprets this as a user requirement, and starts waking up the machines. This could cause undesirable cycles of powering machines on and off.

Another relevant metric is represented by the number of temporarily unavailable blocks and files caused by certain nodes entering the sleep state. Figure 5 depicts the fraction of unavailable blocks, of the blocks that have only one replica, and of unavailable files. A discouraging fact is that, even with a reasonable reduction in energy consumption of 30%, there is a high fraction of unavailable blocks (5%) leading to a full 25% of all files being temporarily unavailable. Moreover, 20% of the blocks are only replicated once, implying that these blocks could eventually become unavailable too in case of failure.

As a next step, we run a similar set of experiments with MapReduce. Hadoop's MapReduce framework is a programming model which enables programmers who do not have much experience with parallel programming to rapidly write jobs that are automatically split into multiple independent simple tasks (mappers and reducers), in a completely parallel way. This allows easy access to the resources of a large cluster and is appropriate for executing jobs such as data mining, web indexing, scientific computation, etc. Similarly

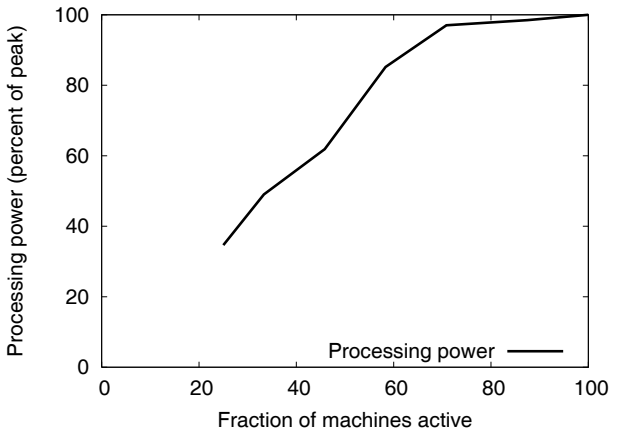

Figure 3: MapReduce processing power as a function of the number of active machines.

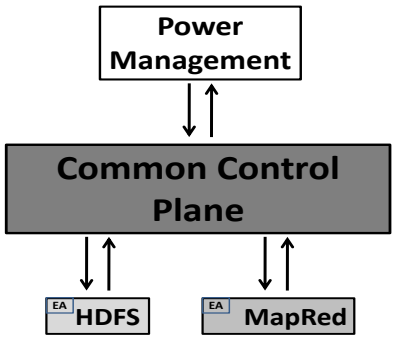

Figure 4: Proposed approach: Common control plane connecting power management on one side and HDFS/MapReduce on the other.

to Hadoop's distributed file system, MapReduce consists of one master node running *JobTracker* and multiple slaves. Each slave runs one *TaskTracker* instance. The master node is responsible for assigning component tasks to slaves (scheduling), monitoring all jobs, re-executing tasks that have failed or are too slow, etc.

In order to analyze the energy efficiency characteristics of MapReduce we vary the overall energy consumption by repeating the same experiment, but decreasing the number of active machines in the cluster by one in each round. Each experiment includes the same job (word counting problem) and the same input set (10GB) that we found to be reasonably large for our cluster (details on the cluster are reported in the Evaluation). The results are reported in Figure 3, which shows the processing power as a function of the number of machines used. We assume that all machines consume the same amount of energy. One can observe that the processing power scales almost linearly with the number of machines used, up to a point where the system encounters a network bottleneck. This is consistent with the previously reported findings [22], which indicate that the MapReduce scheduler is not best suited for non-homogeneous environments. In our example, once we start adding machines across the bottleneck links, the energy efficiency drops significantly. It is obvious from this scalability curve that MapReduce should keep its computation locally in order to achieve the best energy-efficiency.

Overall, one might conclude that aggregating load on a fewer number of machines is a wise decision to address the problem of the discouraging energy-efficiency characteristics for modern servers. However, we also show that emerging distributed applications, beside being energy-efficient, have to be the active participants of the power management if we want to avoid poor performance as seen in this section. Therefore, we propose a model that has a communication channel between the cluster power management and services running on the cluster for collaborative power-management. The next section gives more detail of the proposed design.

3. APPROACH

The model we propose is depicted in Figure 4. In this architecture, each distributed service running on the cluster cooperates with the common control plane, which acts as a glue between the cluster power management interface on one side, and all deployed services (only HDFS and MapReduce in this specific case) on the other. The common control plane assures that the needs of the distributed services as well as those of the power management are enforced in a way that maximizes system objectives and observes power budgets.

The cluster's power management actions may be initiated by a human administrator or, under certain circumstances, automatically. An example for the first case might be a thermal emergency that requires 30% of the machines to enter the sleep state. An example for the second case might be provided by a cluster being intentionally forced to operate, in an automatic way, at a lower power level during nighttime, when the load is assumed to be lower.

One of the common control plane's responsibilities is to report the cluster's power management decisions to the active distributed services. It is worth mentioning that each communication channel is active in both directions, meaning that the distributed services collaborate in the process and provide their feedback on which machines are most appropriate for sleeping. By doing so they minimize the impact on the performance, while fulfilling the power management's demands. Once the common control plane has collected the necessary feedback from the running services, it makes a decision and forwards the list of machines that are going to be "sacrificed" to the power management, which then reacts on this decision.

Although the general idea sketched above looks straightforward, there are a few points worth discussing, mostly concerning the design of a particular service running on the cluster. For instance, it is unacceptable to have a fraction of unavailable blocks as high as the one seen in the previous section when only 30% of the nodes are sleeping. Therefore, we need to introduce changes in the original version of HDFS and MapReduce in order to make them behave as active participants in the power management actions. The changes are labeled as EA, which stands for Energy Aware design for cluster applications, in Figure 4.

4. DESIGN AND IMPLEMENTATION

Section 2 shows that the design of both Hadoop's DFS and MapReduce is not energy-aware, despite the fact that they might rely on machines featuring efficient local power management schemes. We present in the following what be believe are mandatory attributes for distributed applications, such as HDFS or MapReduce, to really behave in an energy-proportional way, i.e., capable of efficiently operating at multiple power levels.

4.1 Energy-Aware HDFS design

First, we need to make Hadoop capable of handling nodes which we intentionally let enter the sleep state, instead of getting into a

panic phase. Therefore, we have to deal with a new kind of node status, namely **sleeping**. The main purpose of this status is to differentiate dead nodes (i.e., unrecoverable ones) from those which have intentionally been put to sleep. We assume that a node going to sleep is healthy, at least at the time we shut it down, and therefore that all the data it holds is only temporarily unavailable. On the contrary, if a node is dead, we do not know exactly what caused the problem - the physical machine where the DataNode is hosted could have crashed due to hardware failure, the network could be temporarily down, or the DataNode process itself could have hung. In this case, it would be better for the distributed file system to replicate the missing blocks.

If, on the other hand, a node is sleeping, we know that the machine and the hosted data are safe; we therefore do not need to replicate again the temporarily unavailable blocks. In order to take this aspect into account, we modified the block inspection function. This has been implemented by checking if an unavailable block is found: when this occurs, we prevent the replication from being executed if the corresponding node is sleeping. Thus, by introducing the **sleeping layer**, we prevent the occurrence of a panic phase while some machines are put to sleep.

Second, we need to introduce the ability of operating at multiple power levels. However, we recall that the previous section demonstrated an unacceptably high fraction of unavailable blocks when HDFS is exposed to a reasonably aggressive power management action (30% machines were put to sleep). The reason lies in the fact that HDFS randomly chooses machines on which a block would be replicated. Therefore, a large number of choices for picking machines for sleeping leads to data unavailability. The similar study about load balancing and data availability is conveyed by Saito *et al.* [19]. They defined the seggroup that represents the set of machines storing a block. In short, as the number of seggroups increases, the extra load is spread more evenly. However, having too many seggroups reduces the system's reliability, since it increases the number of combinations of inactive machines (caused by sleeping) that lead to data loss. Since our goal is to minimize the number of unavailable blocks while machines are sleeping, we set the number of seggroups per machine to one. For instance, in a set of 6 machines ({a,b,...,f}) we would have 2 seggroups ({a,b,c} and {d,e,f}). Thus, we should be able to let 2/3 of all machines enter the sleep state while still having a block availability of 100%, assuming that the default number of replicas is 3.

Finally, the proposed design assumes that services have to actively collaborate by suggesting actions - machines appropriate for sleeping in our case - that, from their standpoint, minimize the harm at system level. This translates into minimizing the number of unavailable blocks, or making sure that all of them are still available, if possible. Therefore, we implemented a function that cycles through all combinations, selecting the most suitable one.

4.2 Energy-Aware MapReduce design

As seen in Figure 3, the processing power scales linearly with the power consumption up to a certain level; however, from that point onwards adding more machines increases the processing power only marginally. We analyze reasons for such behavior, in the hope that it will provide us with the insight useful for the energy-aware design.

Each MapReduce task is monitored and its current progress is used when *JobTracker* looks for *stragglers* - nodes that are available but perform poorly. MapReduce speculatively launches a backup task for each task whose progress is slower than the average progress score by a certain fraction (20% by default). For instance, a map task's progress score represents the fraction of the input data read. Nodes that need to fetch the input data (for mappers) or map outputs (for reducers) from other racks are therefore usually marked as stragglers because they are significantly slower than the average tasks' speed. In this case, adding more machines across the bottleneck links (details on the used topology are reported in the Evaluation) does not in general necessarily increase the processing power, and sometimes even decreases it, given that the stragglers are competing with the other nodes for the same resources, such as the network and disk I/O.

Based on the previous observation, we implemented a function that acts on the power management's decision by choosing sleeping candidate machines that will maximize the processing power of MapReduce. In other terms, we try to maximize the number of tasks that will fetch input data locally (from their own rack) instead of facing the network bottleneck. By doing so, we succeeded in maximizing the performance from the users' standpoint, while still fulfilling the power management's demands.

4.3 Discussion

So far, we have considered the scenario in which only the services such as HDFS and MapReduce are involved in power management actions. Additional applications that are built on top of these services cannot fully be insulated from power-management. For example, since only the application might be aware of some of its data placement policies, it too has to be connected to the control plane. Further, it is likely that some application-specific knowledge can be leveraged to enhance the overall benefit. Take a data storage application as an example. This application would probably prefer maintaining availability of the current, most commonly used data relative to historical records which can temporarily be made unavailable. In this case, the information about historical records would be propagated to the common control plane where the final energy-saving decision would be taken.

5. EVALUATION

We report in this section our preliminary results. We use 24 machines for all our experiments. In order to closely reproduce a realistic environment with bottlenecks in the data center network, we separate our machines into two logical racks with 17 and 7 machines, respectively. The connection between the racks is capped at 100 Mbps. This is especially important for MapReduce as its performance is significantly affected by data locality. We install Sun Java 1.5 and Hadoop version 0.18.3 which serves as the baseline. We generate eight one GByte files. Each file is inserted on a different node and replicated afterwards. Overall the system handles 120 unique blocks, 360 blocks including replicas and 24 GB of data in total.

First we run the experiments where 30% of machines are intentionally put to sleep and observe the behavior of both, the original HDFS and HDFS with energy-aware design that actively participate in power management's actions. In contrast towith the baseline HDFS, Figure 5 shows that the proposed design dramatically reduces block unavailability. Additionally, the presence of at least 2 replicas for each block proves the system's resilience to failure.

Finally, Figure 6 demonstrates the same experiment with MapReduce involved. Energy-aware design of MapReduce takes locality into consideration when letting nodes enter the sleeping state. By doing so, it minimizes the impact on performance when power management system reduces power consumption by 30%. Indeed, corresponding performance is significantly better, close to the one when all machines are up and running.

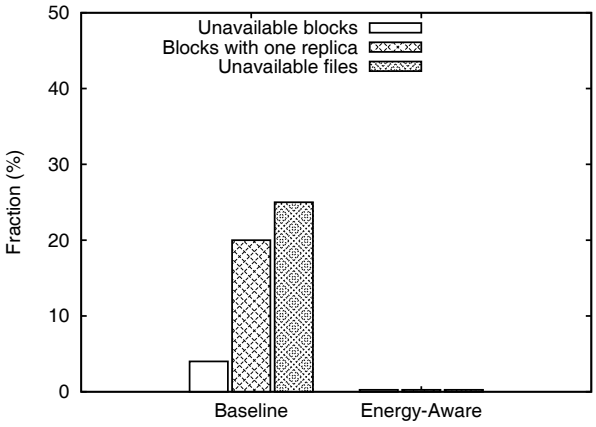

Figure 5: Block availability after leveraging the cluster power management's machine sleeping design. 30% of DataNodes were put to sleep. Baseline against energy-aware HDFS.

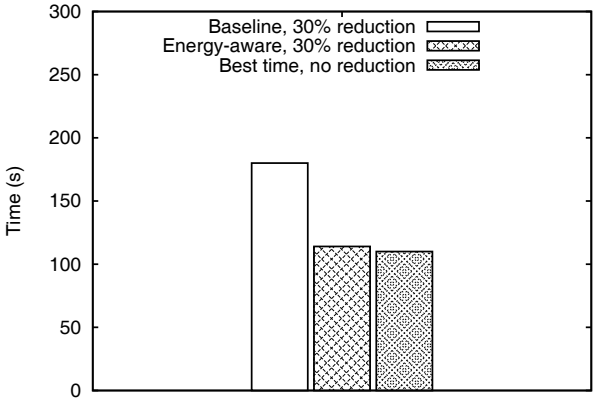

Figure 6: Time to finish the word counting problem after leveraging the cluster power management's machine sleeping design. 30% of DataNodes were put to sleep. Baseline against energy-aware MapReduce.

6. RELATED WORK

As already mentioned, a large body of research has focused so far on energy savings in laptop computers, embedded devices, etc. Specifically, most of the attention has been concentrated on optimizing the energy consumption for processors [20, 17], memories [14], or disks [5]. Interesting research work on offloading computation from these devices onto non-battery operated computers [18] has also been reported.

Recently, energy savings for data centers hosting clusters of thousands servers are becoming more and more important since the energy cost represents a significant share of the total ownership cost. Energy savings techniques used for data centers are usually different than those used for battery-operated devices because of their different purpose and workload. The work in this area can be classified into two groups, local and cluster-wide techniques, with most of them summarized in [3]. The local techniques include efforts based on DVS policies and batching requests [6, 9] to address high CPU energy consumption, or multi-speed disks [11] for tackling the energy consumption in the storage subsystem.

Achieving the magnitude of energy savings that is sufficient to address thermal emergencies [4, 12] while keeping performance penalties marginal might require cluster-wide techniques. For instance, if the server temperature is too high, these approaches can reduce its load (or turn off the server altogether, if necessary) by shifting the load to other machines that are unaffected by the emergency. The existing work [4, 15, 12] assumes that any given request might be served by a number of the currently active servers. This sort of policy is possible in scenarios in which each server is largely stateless, as is the case with the first two tiers in 3-tier Web services. However, cluster applications might be heterogeneous and the previous assumption does not hold in general case. For instance, there is no server which can replace the server holding the last replica of a file. Similarly, only the applications know which servers are essential to their functioning. In contrast to previous solutions, our approach suggests that all services deployed in a cluster actively participate in the power management actions. Furthermore, we explore what needs to be done in order to make any service deployed to the cluster power-aware and capable of operating in multiple power modes.

Given that various energy-saving solutions individually address different aspects of energy savings, deploying them together might potentially cause some dangerous or suboptimal interactions. Therefore, Raghavendra *et al.* [16] propose a solution to this problem that coordinates different individual approaches in an efficient and stable way.

7. CONCLUSION

We demonstrate that important classes of distributed applications do not gracefully operate with limited power budgets. We believe that energy-aware design for cluster applications and services and their active participation in power management actions will be required for reliable, high performance, and low cost data centers. We therefore propose a new approach for making cluster applications energy aware, and demonstrate the efficiency of our approach using a prototype implementation of HDFS and MapReduce.

8. REFERENCES

[1] Hadoop distributed file system.

[2] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 2007.

[3] Ricardo Bianchini and Ram Rajamony. Power and energy management for server systems. *Computer*, 2004.

[4] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. *SIGOPS*, 2001.

[5] Fred Douglis, P. Krishnan, and Brian N. Bershad. Adaptive disk spin-down policies for mobile computers. In *MLICS*, 1995.

[6] Mootaz Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy conservation policies for web servers. In *USITS*, 2003.

[7] U.S. EPA. Report to congress on server and data center energy efficiency. Technical report, 2007.

[8] Wu-chun Feng. Making a case for efficient supercomputing. *Queue*, 2003.

[9] Krisztián Flautner, Steve Reinhardt, and Trevor Mudge. Automatic performance setting for dynamic voltage scaling. *Wirel. Netw.*, 2002.

[10] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *SIGOPS*, 2003.

[11] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mahmut Kandemir, and Hubertus Franke. Drpm: dynamic speed control for power management in server class disks. *SIGARCH*, 2003.

[12] Taliver Heath, Ana Paula Centeno, Pradeep George, Luiz Ramos, Yogesh Jaluria, and Ricardo Bianchini. Mercury and freon: temperature emulation and management for server systems. In *ASPLOS*, 2006.

[13] Kyong Hoon Kim, Rajkumar Buyya, and Jong Kim. Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters. In *CCGRID*, 2007.

[14] Alvin R. Lebeck, Xiaobo Fan, Heng Zeng, and Carla Ellis. Power aware page allocation. *SIGOPS*, 2000.

[15] Eduardo Pinheiro, Ricardo Bianchini, Enrique V. Carrera, and Taliver Heath. Dynamic cluster reconfiguration for power and performance. *Compilers and operating systems for low power*, pages 75–93, 2003.

[16] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. No "power" struggles: coordinated multi-level power management for the data center. *SIGARCH*, 2008.

[17] Arun Rangasamy, Rahul Nagpal, and Y.N. Srikant. Compiler-directed frequency and voltage scaling for a multiple clock domain microarchitecture. In *CF*, 2008.

[18] Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. Saving portable computer battery power through remote process execution. *SIGMOBILE*, 1998.

[19] Yasushi Saito, Svend Frolund, Alistair Veitch, Arif Merchant, and Susan Spence. Fab: building distributed enterprise disk arrays from commodity components. *SIGOPS*, 2004.

[20] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced cpu energy. In *OSDI*, 1994.

[21] Hung-chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and D. Stott Parker. Map-reduce-merge: simplified relational data processing on large clusters. In *SIGMOD*, 2007.

[22] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy H. Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *OSDI*, 2008.