Achieving Power-Efficiency in Clusters without Distributed File System Complexity

Hrishikesh Amur, Karsten Schwan

Achieving Power-Efficiency in Clusters without Distributed File System Complexity

Hrishikesh Amur
amur@gatech.edu
College of Computing, Georgia Tech

Karsten Schwan
schwan@cc.gatech.edu
College of Computing, Georgia Tech

Abstract

Power-efficient operation is a desirable property, particularly for large clusters housed in datacenters. Recent work has advocated turning off entire nodes to achieve power-proportionality, but this leads to problems with availability and fault tolerance because of the resulting limits imposed on the replication strategies used by the distributed file systems (DFS) employed in these environments, with counter-measures adding substantial complexity to DFS designs. To achieve power- efficiency for a cluster without impacting data availability and recovery from failures and maintain simplicity in DFS design, our solution exploits cluster nodes that have the ability to operate in at least two extreme system- level power states, characterized by minimum vs. maximum power consumption and performance. The paper describes a cluster built with power-efficient node prototypes and presents experimental evaluations to demonstrate power-efficiency.

1 Introduction

Power-proportionality, according to Barroso and Hölzle [8], is a desired characteristic for any computing system. For datacenters, however, while significant prior work has enabled the CPU to operate in an increasingly power-efficient manner, this is not the case for other system components like memory and storage. Previous research [9, 12, 15] has attempted to attain system-level power-efficiency by suggesting that significant savings in power can be obtained by turning off/hibernating nodes. This would allow, for example, the reduction of power usage via the exploitation of temporal variations in load such as those seen in user-facing services such as web email [15, 11]. We term these approaches *coarse-grained*, because they turn off entire nodes, and this not only results in loss of node state, but it also makes the data stored on their local disks unavailable to other machines. This issue is of particular importance to datacenters that use distributed file systems (DFS), because a DFS uses replication to guard against loss of data in case of infrastructure failure. In fact, a data layout policy based on a randomized approach, such as those used in the Hadoop Distributed File System (HDFS) [1] and GFS [10], severely restricts the number of nodes that can be turned off. In response, alternate data layout policies, such as those proposed in [5, 15], attempt to bias data storage in a manner that allows performance to be traded off with the number of nodes turned off in a cluster. These policies maintain a primary copy of the data on a small subset of cluster nodes, which represents the lowest power setting. Availability is guaranteed as long as at least that subset of nodes is kept on. Next, additional copies of the data are stored on progressively larger sized node subsets, i.e., larger power settings. This allows performance to be scaled up, by effectively increasing the number of nodes across which system load can be balanced. But any approach that turns off entire servers to save power loses the flexibility to work well in a variety of scenarios. There are two specific problems with the coarse-grained approach towards power-efficiency for data-intensive applications using a DFS:

- *Placing new data:* When new data enters the system, we are faced with the dilemma of whether to wake up sleeping servers to store data or to write the data (maybe temporarily) to servers that are already on. As we show in Section 2.1, both options fail to yield universally satisfactory results.
- *Dealing with failures:* Server and disk failures are common occurrences in datacenters. Policies that rely on storing copies of the entire data on subsets of the cluster typically use a randomized scheme to distribute data within the subset for maximizing throughput. This is problematic because, to maintain the replication level of the data on the unavail-

able disks, an entire subset of nodes needs to be turned on in the event of a single failure.

Neither of these problems is insurmountable, of course, but their solutions will place further constraints on the design of the underlying DFS in addition to those that make it power-efficient. This leads to increased complexity in DFS design and implementation, as discussed in more detail in Section 2.1. This paper explores an alternative, *fine-grained*, approach to attaining power-efficiency in data-intensive systems supported by a DFS. The approach is termed as *fine-grained* since it leverages future hardware capabilities with which, at system level and under software control, one can decide for each platform exactly which and how many components, including memory, are turned on vs. off. The resulting cluster node, detailed in Section 3, has the following properties:

- *Simplicity:* DFS design is kept simple by keeping *all* of the disks on all of the time. As a result, data availability and recovery from failure use the same strategies as those used in traditional DFS designs.
- *Extreme power-performance states:* Each cluster node operates in one of two node-level power-performance states. The first state, termed *max-perf*, yields maximum performance and consumes maximum power. The second state, called *io-server*, offers sufficient minimum CPU power and amounts of 'on' memory to service its storage and the network. In *io-server* state, the node allows data to be read and written to it over the network at the same rate as the *max-perf* mode, but it cannot carry out any of the other computations desired by data-intensive applications.
- *Low relative minimum power consumption:* The power consumption of each cluster node is significantly lower in *io-server* state compared to *max-perf* state. This relies on the ability to turn platform components off in the interest of improved power management allowing the entire system to operate at multiple power and performance levels. We further discuss this assumption in Section 3.

Given these properties, the cluster can be made to operate in a power-efficient manner by using a control algorithm that scales the number of nodes in the cluster maintained in *max-perf* state according to the current system load. To experimentally demonstrate this fact in this paper, we build a platform prototype that possesses the aforementioned properties. We do so by coupling a commodity full-featured server, termed 'Obelix', with a low-power system, called 'Asterix', the latter consisting of an embedded processor with minimal memory. Disk storage is shared by the two systems, with only one of them having exclusive access to the disks at any time. This architecture enables the node to operate in two power states. During times of high compute load, the node operates in *max-perf* state with Obelix on and Asterix off. When the node is required simply for availability of the data stored on its disks, it operates in *io-server* state, by turning Obelix off after turning Asterix on, which results in system power being dominated by the storage. The architecture of the node is described in detail in Section 4. Our research makes the following technical contributions:

- We show that coarse-grained approaches that involve turning entire nodes off in a cluster to make it operate in a power-efficient manner introduce additional constraints on DFS design as detailed in Section 2.1.
- We present an alternative, fine-grained solution that relies on hardware with flexibility in turning system components on and off and show that this allows a cluster to operate in a power-efficient manner, obviating complex solutions in the DFS.
- We present a prototype of our design explained in Section 3 that is built from commodity hardware consisting of a powerful system coupled with a low-power system as explained in Section 4.
- We show that the I/O performance of the Hadoop DFS [1] when a DFS node is in *io-server* state is comparable to that in *max-perf* state in Section 5.1.

2 Motivation

2.1 Problems

In this section, we give a more detailed description of the problems that prevent nodes from being easily turned off.

2.1.1 Dealing with New Data

Consider a data-intensive distributed application running on a cluster using a DFS. The objective is to allow some of its nodes to be turned off without loss in data availability. This can be done without loss in performance by using an appropriate data layout if the load is sufficiently low. The following options exist when new data arrives and has to be stored:

1. Turn on some/all of the sleeping nodes.
2. Store the data on the nodes that are already on.
3. Use a lazy write scheme where the data is temporarily written to available nodes and later transferred to the sleeping nodes once they are turned on.

There are issues with all three choices. The first option will not be beneficial if the new data is not going to be used immediately, since this will simply cause the nodes that were woken up to be turned off again. This is especially wasteful if the application receives new data frequently, which is not unusual, with examples including applications like Picasa or Flickr. Further, for interactive applications, current wakeup latencies from system hibernate states, in seconds, are unacceptable. The second option has the disadvantage that new data will only be present on the nodes that are currently on. This means that applications operating on such data cannot benefit from the aggregate system I/O bandwidth of all the nodes. Exploiting such bandwidth requires data movement, which is also undesirable. The third option is the most promising, but care is required to ensure that the additional usage of I/O bandwidth for lazy writing does not impact the performance of running applications. Further, turned off nodes might need to be turned on periodically in order to avoid excessive amounts of data movement at some single period of time. More generally, it should be apparent from this discussion that attaining power-efficiency via implementation of any of these options adds complexity to what originally constituted a straightforward data layout policy, without guaranteeing high performance in all scenarios.

2.1.2 Response to Failures

A second issue with turning off entire nodes concerns failures. A cluster built with commodity hardware typically experiences a significant number of individual machine and disk failures. Therefore, any design of a power-efficient cluster must consider recovery from failures as norm rather than exception. Technically, this requires the cluster to possess rebuild parallelism and it means that the cluster must not require a large amount of time to re-replicate data from a failed disk. For this, a candidate data-layout could have the nodes divided into mutually exclusive subsets P_i with $i \in \{1, ..., r\}$, where r is the replication factor. The nodes in each P_i contain an entire copy of the data. A randomized data layout policy to select nodes within each set P_i is used. Power-efficiency is obtained with this arrangement, and the minimum power-performance setting corresponds to keeping the nodes in P_1 on, and the maximum corresponds to keeping all of the nodes on. A number of intermediate settings are also possible. Unfortunately, with a single failure, all nodes sharing data with the failed node must be turned on to minimize rebuild time. This will lead to a large number of nodes being turned on for a single failure. As a result, there needs to be a carefully calibrated data layout policy that ensures that each node shares its data with only a limited number of other nodes. The outcome is a three-way compromise between minimizing rebuild time, minimizing the number of servers turned on to rebuild, and adding substantial complexity to the data layout policy. Solutions able to avoid such compromises would be preferable.

2.2 Exploiting System-level Power States

With multi-core processors ubiquitous, power management trends have moved towards turning chip components off by power-gating rather than using voltage scaling. We envision that this trend will apply at the system level too and anticipate that controls will be provided to turn off system components entirely or to a minimum performance level. In Section 3, we show how the ability to run a node at two extreme system power-performance states enables power-efficiency.

3 Design

This section describes the architecture of a system that allows individual system components to be turned off and the design principles used to build a node which allows a cluster built with such nodes to operate in a power-efficient manner.

3.1 Power-Efficient Nodes

Modern systems operate with high idle power consumed by non-CPU components. Future systems will likely have capabilities to determine under software control which and how many system components, including memory and storage, are turned on vs. off. Such systems also promise to allow an entire node to operate in a power-efficient manner, by providing additional system-level power states. Currently, ACPI defines a number of system S-states that include working(S0), standby(S3), hibernate(S4), and powered-off(S5). While previous work has leveraged these states by switching nodes from S0 to S3 and back, these states do not allow the system to function in a power-efficient manner , because all of these states except S0 define the CPU to be off. Clearly, it is beneficial for the system to have additional states with certain proportions of the memory, storage and cores turned off but still allowing the system to function at a lower performance level. This will enable cluster-level power-efficiency in a simple manner, without complexity of system software like the DFS. We next explain how we use node-level power-efficiency to attain that property for entire DFS-based clusters.

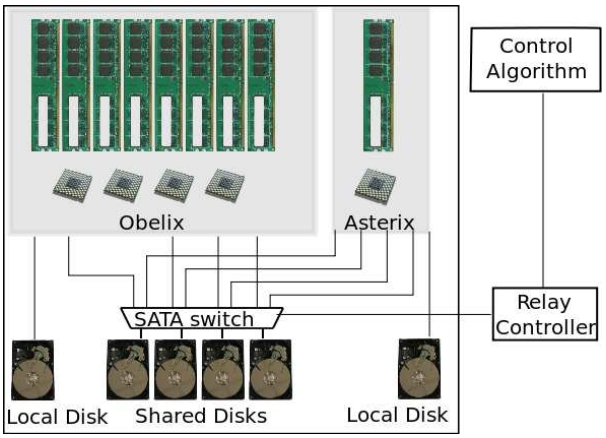

Figure 1: Node Architecture

3.1.1 Keep Disks On

By keeping the disks of nodes on at all times, we maintain DFS simplicity. Specifically, in a cluster environment, the DFS is responsible for guaranteeing availability as well as a requested level of fault tolerance for all data. But turning disks off to achieve power-efficiency requires potentially complex counter-measures to restore the original availability and fault tolerance properties of the DFS. Such counter-measures are unnecessary if disks are always kept on.

This is also beneficial because Pinheiro et al. showed that increased power-cycling of disks increased the probability of disk failure [14]. Keeping disks on may increase power consumption in the low-power states, but it also guarantees near maximum cluster I/O throughput at all times. Fortunately, the total power consumption of storage and network in a datacenter are relatively low, shown to be roughly equal to 15% of total IT equipment power usage [8]. This limits the deleterious effects of disks being kept on, on overall system power-efficiency. We believe this to be an acceptable compromise for the benefits gained.

3.1.2 *max-perf* and *io-server* States

A power-efficient node allows the system to operate in different states varying in power and performance. We utilize two of these states: the maximum performance state, termed *max-perf* and a state that has the minimal amount of CPU and memory left 'on' to service the storage and the network, termed *io-server* state. The purpose of the *io-server* state is to offer sufficient resources to provide access to the data stored on the node to other nodes. It need not support any computations performed on behalf of other system software or for applications running on the cluster. This paper assumes that both system software like the DFS and applications like MapReduce can be modified easily to schedule computation only on cluster nodes that are not in the *io-server* state. A control algorithm can determine the number of nodes that need to be in *max-perf* state to handle the current compute load on the cluster. The remaining nodes are then transitioned into *io-server* state.

3.1.3 Low Power Consumption in *io-server* State

A critical requirement for power-efficiency at the node-level is that power consumption is low in *io-server* state compared to *max-perf* state. We posit that this requirement is easily met by heterogeneous, multi-core systems, where select low-power cores with limited memory operating 'near' I/O devices [3] can be used to realize the *io-server* state, and all cores and memory 'on' represents the *max-perf* state. We emulate such an architecture by building a prototype using two different systems as explained in Section 4 and achieve the requirement of low power consumption in *io-server* state.

4 Implementation

In this section, we describe a prototype built according to the design principles stated in Section 3. For the prototype, the two states of operation are implemented by coupling a commodity full-featured server with a low-power embedded system. The architecture of such a node is shown in Figure 1. The server, called Obelix, and the low-power system, called Asterix, are connected to the same set of shared SATA disks through a SATA switch, which acts as a multiplexer and allows one of the two to have exclusive access to the shared disks. The switch is controlled through a relay device that actuates the disk hand-off during the transition. We run hypervisors on both Obelix and Asterix, and the application running on the cluster is run inside a separate virtual machine(VM). When there is compute load to be run on the node, the node operates in *max-perf* state, where Obelix has control over the shared disks and Asterix hibernates. When the compute load is low enough for Obelix to be turned off, we use the capabilities of the hypervisor to reduce the VM memory and the number of virtual CPUs, perform the disk-handoff via the relay, and migrate the reduced-size VM to Asterix. Obelix may then hibernate. In the *io-server* mode, therefore, the power consumed is only due to the embedded system and shared disks, both of which are quite small compared to Obelix' power usage. We next demonstrate this fact experimentally.

5 Experimental Evaluation

Our evaluation testbed consists of 4 nodes, each made up of a server, Obelix, and a low-power embedded sys-

Mode	HDFS Datanode Throughput (MB/s)								Power (W)		tput/Watt (MB/s/W)	
	domU		dom0		Linux		domU*		domU*		domU*	
	R	W	R	W	R	W	R	W	R	W	R	W
Oblx.	71.4	48.6	75.3	65	78.1	69.9	73	63.2	112	110	0.65	0.57
Astx-I	10.17	4.8	63.6	31.3	78.2	33.5	57.5	27.5	77	76	0.74	0.36
Astx-II	45.7	28.3	69.2	56.6	77	65.6	67.4	54	41	39	1.64	1.38

Table 1: Throughput was measured for Hadoop DFS configured on a single datanode. domU vs. dom0 shows the overhead of the split-driver implementation in Xen, dom0 vs. Linux shows the virtualization overhead. domU* shows performance with direct I/O device access. Asterix-II gives higher performance/Watt compared to Obelix.

tem. We evaluate two generations of low-power nodes: Asterix-I and Asterix-II. The server configuration consists of an Intel Core2 Xeon dual-core processor with 3GB of memory. Asterix-I consists of an Intel Tolapai, an embedded system-on-a-chip (SoC) processor clocked at a maximum of 1.2GHz and Asterix-II consists of a dual-core Intel Atom based system clocked at 1.6GHz. Each of the systems has 1 GB of DDR2 RAM and a Gigabit Ethernet card. The shared disk, which is used for all I/O in the experiments, is a 7200 RPM Seagate Barracuda SATA disk. Disk ownership is switched between Obelix and Asterix nodes using a 3Gbps SATA switch, which is controlled from software using a USB relay device [2]. In *max-perf* state, the VM runs on Obelix with 4 virtual CPUs(VCPUs) and 2GB of memory. In *io-server* mode, the VM runs with 1 VCPU and 512MB of memory on Asterix-I and 4 VCPUs and 512MB on Asterix-II. A Dell PowerConnect 5324 Gigabit Ethernet switch is used as the network inter-connect.

5.1 I/O Performance

In this section we seek to show that the I/O bandwidth available from a low-power node (Asterix-I and II) is comparable to that available from a server, but at a significantly lower cost in power. This would allow a number of nodes in the cluster to operate in *io-server* state and provide comparable I/O performance at lower power during periods of low compute load. To show this, we set up Hadoop Distributed File System (HDFS) on a single node with 10GB of data and measure the read (write) performance from (to) the node for different types of nodes as shown in Table 1. The buffer caches are flushed on all nodes before each run. Since the mechanism for a node to move from *max-perf* to *io-server* state involves a VM migration from the Obelix to Asterix node, we also evaluate the overhead of virtualization on all the nodes by running on plain Linux. In particular, the split-driver implementation in Xen causes high overhead on the low-power nodes (domU vs. dom0 in Table 1) in addition to the base virtualization overhead (dom0 vs. Linux). To avoid the former, we provide direct access to the disk and network device to the VM (domU*). With direct access to the I/O devices, we can see that the read and write throughputs in the case of Obelix and Asterix-II nodes are comparable, but with lower power consumption in the case of the latter. The performance of Asterix-I suffers because it has a single core at 1.2GHz and is affected by virtualization and frequent context switch overheads. We also show a performance/power metric, where Asterix-II is the clear winner.

5.2 Cluster Power-Efficiency

In this section, we demonstrate experimentally that a cluster with low-power nodes operates in a more power-efficient manner than a default cluster during periods of low system load. We consider a distributed grep program implemented with Hadoop on a 10GB dataset stored on a cluster of 4 nodes. The number of nodes in *max-perf* state is varied, while making sure that Hadoop schedules the map and reduce threads only on these nodes. Power consumed by an Obelix node in *max-perf* mode is 252W. The remaining nodes are in *io-server* state, storing data as part of the DFS but not running any computation. In the default case, the Obelix node consumes 111W in *io-server* state. However, if the Obelix nodes are coupled with Asterix nodes, then the VM can be moved to the low-power node in *io-server* state. The power consumption is 77W in the case of Asterix-I and 39W for Asterix-II. In our prototype cluster, we only show results for Asterix-I nodes, and project the expected power savings for a cluster with Asterix-II nodes in Table 2.

6 Related Work

Although significant previous work has dealt with CPU power management, system- level power-proportionality has proved elusive due to the increasing power consumed by non-CPU components, which has led to a more holistic view of system power management [13]. Chase et al. discuss how to turn servers on and off based on demand in datacenters [9] and more recent work [12, 15, 5] specifically tackles the problem of designing a distributed file system to be power-proportional. We

Servers in *max-perf* state	Throughput (MB/s)		Power (W)			Throughput / Watt (KB/s/W)		
	Default	Ast-I	Default	Ast-I	Ast-II(proj.)	Default	Ast-I	Ast-II(proj.)
1	22	19.1	585	483	369	38.51	40.49	53
2	38	36.2	726	658	582	53.6	56.34	63.69
3	59	58.5	867	833	795	69.68	71.91	75.35
4	78	79	1008	1008	1008	79.24	80.6	80.6

Table 2: Comparison of power efficiency for clusters with different configurations for distributed grep. Hadoop was modified to schedule the map/reduce tasks only to the nodes in *max-perf* state, while the I/O bandwidth of all the nodes in the cluster can be used. The throughput/Watt values when Obelix, Asterix-I and Asterix-II nodes are used to implement the *io-server* state are shown. Since we did not have enough Asterix-II nodes to build a cluster, the total power for this case is projected. We also project the throughput/Watt for Asterix-II using the fact that the I/O bandwidth available is comparable between Asterix-I and Asterix-II as shown in Table 1.

present an alternative solution that relies on node-level power-proportionality available in future systems through system-level control over which and how many components to turn off. The advantages of providing this control has also been explored previously in work on barely-alive servers [6] where free memory on servers is used for cooperative data caching by switching to a "barely-alive" power state. Somniloquy [4] allows certain networking tasks to be performed in the background without waking the processor from deep S-states. We believe that these are techniques that can be viewed as policies that target specific applications (in the same way that our technique targets DFS) that use the underlying mechanism that provides control over which system components to turn off. In this way, they are complementary to our technique. FAWN also uses low-power embedded systems for data-intensive computing [7].

7 Conclusions and Future Work

Achieving power-efficiency in clusters is hindered by the fact that non-CPU system components do not operate power-proportionally. Earlier work suggests solutions that hibernate/turn off machines, including modifications to the distributed file system (DFS) required to maintain data availability and fault tolerance. Unfortunately, such solutions lead to a significant increase in DFS complexity. This paper shows that a power-efficient node with controls over which system components to turn off is sufficient for clusters to operate in a power-efficient manner without making significant changes to the DFS. We construct a prototype cluster with nodes that can operate in two extreme modes: one with maximum performance and the other with sufficient resources to support I/O devices. Power-efficient operation at no additional cost in terms of DFS complexity is shown to hold for such a cluster. Our future research will explore platform-level power-efficiency in more detail, for both homogeneous and heterogeneous many-core platforms.

References

[1] Hadoop. http://hadoop.apache.org.

[2] Phidgets Inc. http://www.phidgets.com.

[3] ADILETTA, M., ROSENBLUTH, M., BERNSTEIN, D., WOLRICH, G., AND WILKINSON, H. The next generation of intel ixp network processors. *Intel Technology Journal 6* (2002).

[4] AGARWAL, Y., HODGES, S., SCOTT, J., CHANDRA, R., BAHL, P., AND GUPTA, R. Somniloquy: Augmenting Network Interfaces to Reduce PC Energy Usage. In *NSDI 09* (2009).

[5] AMUR, H., CIPAR, J., GUPTA, V., GANGER, G. R., KOZUCH, M. A., AND SCHWAN, K. Robust And Flexible Power-proportional Storage. In *SOCC '10: Proceedings of the First ACM Symposium on Cloud Computing* (2010).

[6] ANAGNOSTOPOULOU, V., BISWAS, S., SAVAGE, A., BIANCHINI, R., YANG, T., AND CHONG, F. T. Energy Conservation in Datacenters through Cluster Memory Management and Barely-Alive Memory Servers. In *WEED '09* (2009).

[7] ANDERSEN, D. G., FRANKLIN, J., KAMINSKY, M., PHANISHAYEE, A., TAN, L., AND VASUDEVAN, V. FAWN: A Fast Array of Wimpy Nodes. In *SOSP '09: Proceedings of the 22nd ACM Symposium on Operating Systems Principles* (2009).

[8] BARROSO, L. A., AND HÖLZLE, U. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. 2009.

[9] CHASE, J. S., ANDERSON, D. C., THAKAR, P. N., VAHDAT, A. M., AND DOYLE, R. P. Managing energy and server resources in hosting centers. *SIGOPS Oper. Syst. Rev.* (2001).

[10] GHEMAWAT, S., GOBIOFF, H., AND LEUNG, S. T. The Google File System. *SIGOPS Oper. Syst. Rev. 37*, 5 (2003), 29–43.

[11] HAMILTON, J., AND TREADWELL, D. Resource consumption shaping. 2008.

[12] LEVERICH, J., AND KOZYRAKIS, C. On the energy (in)efficiency of hadoop clusters. In *HotPower'09, co-located with SOSP'09* (2009).

[13] MEISNER, D., GOLD, B. T., AND WENISCH, T. F. Powernap: eliminating server idle power. In *ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems* (2009).

[14] PINHEIRO, E., WEBER, W. D., AND BARROSO, L. A. Failure trends in a large disk drive population. In *FAST '07: Proceedings of the 5th USENIX conference on File and Storage Technologies* (2007).

[15] THERESKA, E., DONNELLY, A., AND NARAYANAN, D. Sierra: a power-proportional, distributed storage system. Tech. rep., Microsoft Research, 2009.