On the Energy (In)efficiency of Hadoop Clusters

Jacob Leverich, Christos Kozyrakis
Computer Systems Laboratory
Stanford University
{leverich, kozyraki}@stanford.edu

ABSTRACT

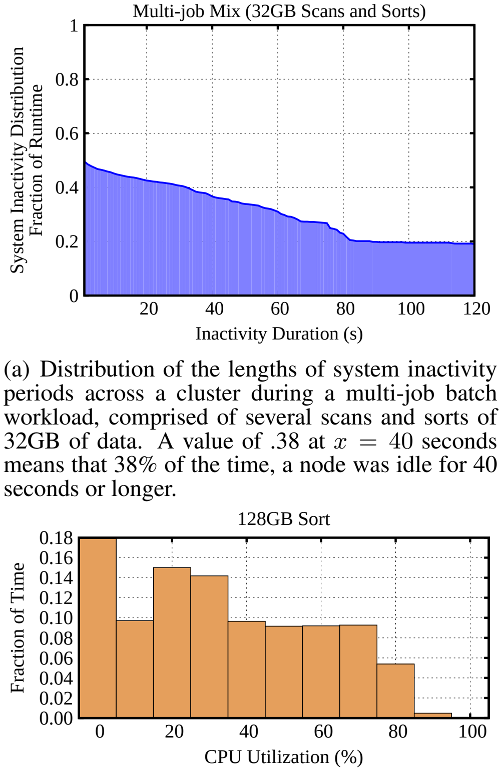
Distributed processing frameworks, such as Yahoo!'s Hadoop and Google's MapReduce, have been successful at harnessing expansive datacenter resources for large-scale data analysis. However, their effect on datacenter energy efficiency has not been scrutinized. Moreover, the filesystem component of these frameworks effectively precludes *scale-down* of clusters deploying these frameworks (i.e. operating at reduced capacity). This paper presents our early work on modifying Hadoop to allow scale-down of operational clusters. We find that running Hadoop clusters in fractional configurations can save between 9% and 50% of energy consumption, and that there is a tradeoff between performance energy consumption. We also outline further research into the energy-efficiency of these frameworks.

1. INTRODUCTION

Energy consumption and cooling are now large components of the operational cost of datacenters and pose significant limitations in terms of scalability and reliability [3]. A growing segment of datacenter workloads is managed with MapReduce-style frameworks, whether by privately managed instances of Yahoo!'s Hadoop [2], by Amazon's Elastic MapReduce [12], or ubiquitously at Google by their archetypal implementation [5]. Therefore, it is important to understand the energy efficiency of this emerging workload.

The energy efficiency of a cluster can be improved in two ways: by matching the number of active nodes to the current needs of the workload, placing the remaining nodes in low-power standby modes; by engineering the compute and storage features of each node to match its workload and avoid energy waste on oversized components. Unfortunately, MapReduce frameworks have many characteristics that complicate both options.

First, MapReduce frameworks implement a distributed data-store comprised of the disks in each node, which enables affordable storage for multi-petabyte datasets with good performance and reliability. Associating each node with such a large amount of state renders state-of-the-art techniques that manage the number of active nodes, such as VMWare's VMotion [13], impractical. Even idle nodes remain powered on to ensure data availability [3]. To illustrate the waste, Figure 1(a) depicts the distribution of the lengths of system inactivity periods across a cluster during a multi-job Hadoop workload, comprised of several scans and sorts of 32GB of data. We define inactivity as the absence of activity in all of the CPU, disk, and network. While significant periods of inactivity are observed, the need for data availability prohibits the shutting down of idle nodes.



(a) Distribution of the lengths of system inactivity periods across a cluster during a multi-job batch workload, comprised of several scans and sorts of 32GB of data. A value of .38 at $x = 40$ seconds means that 38% of the time, a node was idle for 40 seconds or longer.

(b) Average CPU utilization across the cluster when sorting 128GB of data.

Figure 1: Opportunities for improved efficiency.

Second, MapReduce frameworks are typically deployed on thousands of commodity nodes, such as low-cost 1U servers. The node configuration is a compromise between the compute/data-storage requirements of MapReduce, as well as the requirements of other workloads hosted on the same cluster (e.g., front-end web

HotPower '09, Copyright 2009 ACM

serving). This implies a mismatch between the hardware and any workload, leading to energy waste on idling components. For instance, Figure 1(b) shows that for an I/O limited workload, the CPU utilization is quite low and most of the energy consumed by the CPU is wasted.

Finally, given the (un)reliability of commodity hardware, MapReduce frameworks incorporate mechanisms to mitigate hardware and software failures and load imbalance [5]. Such mechanisms may negatively impact energy efficiency.

This paper makes a first effort towards improving the energy efficiency of MapReduce frameworks like Hadoop. First, we argue that Hadoop has the global knowledge necessary to manage the transition of nodes to and from low-power modes. Hence, Hadoop should be, or cooperate with, the energy controller for a cluster. Second, we show it is possible to recast the data layout and task distribution of Hadoop to enable significant portions of a cluster to be powered down while still fully operational. We report our initial findings pertaining to the performance and energy efficiency trade-offs of such techniques. Our results show that energy can be conserved at the expense of performance, such that there is a trade-off between the two. Finally, we establish a research agenda for a broad energy efficient Hadoop, detailing node architecture, data layout, availability, reliability, scheduling, and applications.

2. IMPROVING HADOOP'S ENERGY-EFFICIENCY

Our first efforts focus on the impact of data layout. While this discussion is specifically targeted at Hadoop, many of our observations apply to the Google Filesystem [8] and similar cluster filesystems [1].

2.1 Data Layout Overview

Hadoop's filesystem (HDFS) spreads data across the disks of a cluster to take advantage of the aggregate I/O, and improve the data-locality of computation. While beneficial in terms of performance, this design principle complicates power-management. With data distributed across all nodes, any node may be participating in the reading, writing, or computation of a data-block at any time. This makes it difficult to determine when it is safe to turn a node or component (e.g. disk) off.

Tangentially, Hadoop must also handle the case of node failures, which can be frequent in large clusters. To address this problem, it implements a data-block replication and placement strategy to mitigate the effect of certain classes of common failures; namely, single-node failures and whole-rack failures. When data is stored in a HDFS, the user specifies a *block replication factor*. A replication factor of n instructs HDFS to ensure that n identical copies of any data-block are stored across a cluster (by default $n = 3$). Whilst replicating blocks, Hadoop maintains two invariants: (1) no two replicas of a data-block are stored on any one node, and (2) replicas of a data-block must be found on at least two racks.

2.2 A Replication Invariant for Energy

The fact that Hadoop maintains replicas of all data affords an opportunity to save energy on inactive nodes. That is, there is an expectation that if an inactive node is turned off, the data it stores will be found somewhere else on the cluster. However, this is only true up to a point. Should the n nodes that hold the n replicas of a single block be selected for deactivation, that piece of data is no longer available to the cluster. In fact, we have found through examination of our own Hadoop cluster that when configured as a single rack, removing *any* n nodes from the cluster (where n is the replication factor) will render some data unavailable. Thus the largest number of nodes we could disable without impacting data availability is $n - 1$, or merely two nodes when $n = 3$. While Hadoop's autonomous re-replication feature can, over time, allow additional nodes to be disabled, this comes with severe storage capacity and resource penalties (i.e. significant amounts of data must be transferred over the network and condensed onto the disks of the remaining nodes). Hadoop's rack-aware replication strategy mitigates this effect only moderately; at best a single rack can be disabled before data begins to become unavailable.

To address this short-fall in Hadoop's data-layout strategy, we propose a new invariant for use during block replication: at least one replica of a data-block must be stored in a subset of nodes we refer to as a *covering subset*. The premise behind a covering subset is that it contains a sufficient set of nodes to ensure the immediate availability of data, even were all nodes not in the covering subset to be disabled.

This invariant leaves the specific designation of the covering subset as a matter of policy. The purpose in establishing a covering subset and utilizing this storage invariant is so that large numbers of nodes can be gracefully removed from a cluster (i.e. turned off) without affecting the availability of data or interrupting the normal operation of the cluster; thus, it should be a minority portion of the cluster. On the other hand, it cannot be too small, or else it would limit storage capacity or even become an I/O bottleneck. As such, a covering subset would best be sized as a moderate fraction (10% to 30%) of a whole cluster, to balance these concerns.[1]

Just as replication factors can be specified by users on a file by file basis, covering subsets should be established and specified for files by users (or cluster admin-

[1] This discussion is limited to simply "turning off" a node in order to save power. In Section 4, we revisit this assumption.

istrators). In large clusters (thousands of nodes), this allows covering subsets to be intelligently managed as current activity dictates, rather than as a compromise between several potentially active users or applications. Thus, if a particular user or application vacates a cluster for some period of time, the nodes of its associated covering subset can be turned off without affecting the availability of resident users and applications.

Implementation — We made the following changes to Hadoop 0.20.0 to incorporate the covering subset invariant. HDFS's `ReplicationTargetChooser` was modified to first allocate a replica on the local node generating the data, then allocate a replica in the covering subset, and finally allocate a replica in any node not in the first node's rack. We changed Hadoop's strategy in choosing "excess" replicas for invalidation (for instance, when temporarily faulty nodes rejoin a cluster) to refrain from deleting replicas from a covering subset unless there is at least one other replica in the subset. To prevent Hadoop from autonomously re-replicating blocks for nodes which have been intentionally disabled, we have added a hook to Hadoop's HDFS `NameNode` (i.e. master node) which behaves similarly to Hadoop's *decommissioning* of a live node. For each block a node stores, our routine removes the node from Hadoop's internal list of storage locations for that block and stores it in a separate list of offline storage locations. In contrast to node decommissioning, replications are not scheduled for each block that goes offline. Furthermore, we adjust all queries regarding the number of live replicas of a block to account for offline replicas, so that extraneous activity does not trigger superfluous block replications (e.g. should a separate node legitimately fail, Hadoop will globally evaluate how many replicas it should generate for each block on that failed node).

Disabled nodes can gracefully rejoin a cluster after either a short period of stand-by, or even after a complete operating system reboot, by sending a heartbeat or node registration request to the `NameNode`.

At present, nodes are disabled and enabled manually. Data serving and task execution are disabled/enabled in unison. The implementation of a dynamic power manager is ongoing work. We have also added hooks to Hadoop to manage on-demand enabling of nodes and to take arbitrary action to revive nodes (i.e. IPMI [9] commands), but we do not evaluate this capability here.

3. EVALUATION

Methodology — We evaluate our changes to Hadoop through experiments on a 36-node cluster coupled with an energy model based linearly on CPU utilization. Each node is an HP ProLiant DL140G3 with 8 cores (2 sockets), 32GB of RAM, a gigabit NIC, and two disks. The nodes are connected by a 48-port HP ProCurve 2810-48G switch (48 Gbps bisection bandwidth). This cluster was also used for the results in Section 1. We made every effort to optimize the raw performance of all workloads we executed and our Hadoop environment before experimenting with energy-efficiency. To more accurately model Hadoop's rack-based data layout and task assignment, we arbitrarily divided the cluster into 4 "racks" of 9 nodes each. We selected one of these racks to be the covering subset for our input datasets.

In our experiments, we statically disable a number of nodes before running a Hadoop job and observe the impact on performance, energy, power consumption, and system inactivity. We assume the power consumption of a disabled node to be nil, and it contributes neither to the energy consumption nor performance of an experiment. Since the covering set for our input dataset is comprised of 9 nodes, we can gracefully disable up to 27 nodes.

Power models based on a linear interpolation of CPU utilization have been shown to be accurate with I/O workloads ($< 5\%$ mean error) for this class of server [11], since network and disk activity contribute negligibly to dynamic power consumption. Our nodes consume 223 W at idle and 368 W at peak utilization. Future evaluation will include large-scale experiments deployed on Amazon's EC2 service [12], for which we will develop a similar power model. Moreover, the use of a power model enables us to evaluate hypothetical hardware characteristics or capabilities.

Our primary workloads are webdata_sort and webdata_scan from Hadoop's own "gridmix" batch-throughput benchmark. We generate several datasets (from 16GB to 128GB) using the same methodology as gridmix. We additionally generate batch-compute traces of several sorts and scans issued randomly over a 30 minute period (also seen in Section 1). We schedule enough jobs to occupy 75% of the 30 minute period, had the jobs been issued sequentially. This workload is loosely inspired by SPECpower_jbb2008 [4]. Note that this workload more closely mimics a multi-user/tenant *throughput-sensitive* environment, rather than a single-user/tenant *latency-sensitive* environment.

Results — Figure 2 depicts performance, energy consumption, and power consumption over the duration of a Sort and Scan of a 32GB dataset, given different static configurations of the cluster. "0 sleeping nodes" is a baseline result, where all nodes are active. The other configurations show the effect of disabling an increasing number of nodes. We first observe that our mechanism works, and that a significant fraction of the cluster can be disabled with no impact on data availability, contrary to the position of [3].

Quantitatively, we find that disabling nodes in all cases leads to energy savings, from 9% with Sort to 51% with Scan. On the other hand, with the exception

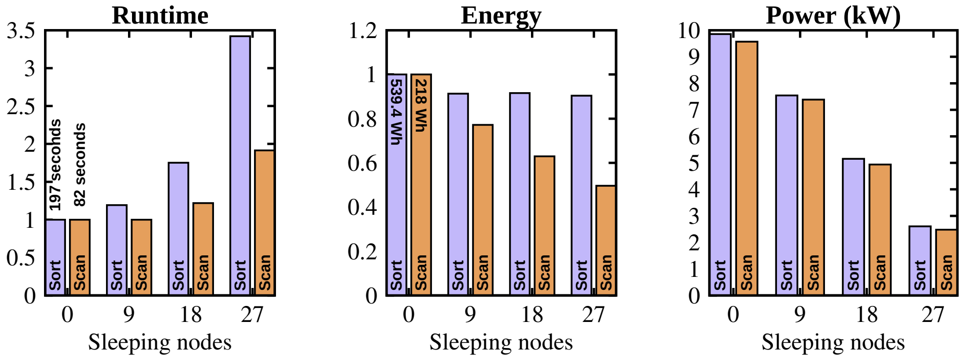


Figure 2: Runtime, Energy Consumption, and Average Power Consumption for the 32GB Sort and 32GB Scan workloads as nodes are disabled. Runtime and Energy are normalized to when all nodes are active.

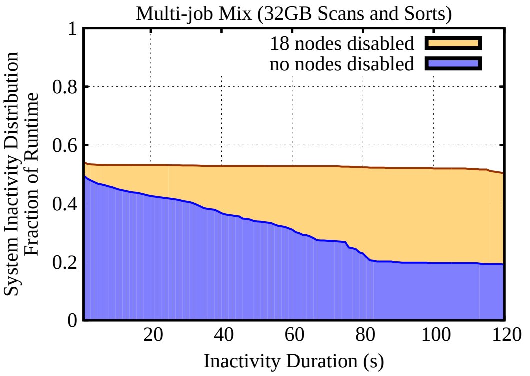


Figure 3: System Inactivity Distribution of the Multi-job workload when all nodes are active and when 18 nodes are disabled.

of disabling 9 nodes during the Scan, there is deleterious impact to performance as nodes are disabled (up to 71% when 27 nodes are disabled with Sort). Although in some cases severe, the penalty does not show a perfect slowdown. Overall, nodes tend to contribute less to performance than they do to energy consumption. Dramatic reduction is seen in the aggregate power consumption of the cluster as nodes are disabled. This clearly demonstrates that this mechanism can be used to implement cluster-level power-capping [7] (with a commensurate reduction in service-level).

A similar large-scale experiment would process perhaps 1 TB of data, occupying a thousand node cluster. However, it would exhibit the same result as our small-scale experiment: at some point, nodes contribute less to performance than they do to energy consumption. Moreover, we would expect to observe bottlenecks in such a large-scale experiment that we don't observe in our cluster, such as contention at the network's root-switch/router or individual rack uplinks. Such bottlenecks would further soften the performance penalty of disabling nodes and further improve energy-efficiency.

To study the impact of disabling nodes on *fractional-utilization* workloads, we performed experiments with batches of Hadoop jobs, described earlier. Figure 3 depicts the distribution of system inactivity periods of the job trace when 18 nodes are disabled, and the distribution when all nodes are active is overlaid for comparison. Disabling 18 nodes significantly increases the length of time spent idle for more than 82 seconds (by 154%), and moderately increases the length of time spent idle for more than 10 seconds (18%). The increase at 82 seconds represents more opportunity to utilize brute-force energy management knobs, such as fully shutting a machine down, to reduce energy consumption. This is in contrast to the improvement at 10 seconds, which represents time potentially available to use less aggressive modes, such as spinning down disks and putting the platform and power supply into a standby mode. Note that while the increase at 10 seconds is modest, a significant fraction of the total time spent there is now contributed by *intentionally inactive* nodes, and it is trivially permissible to disable them. This is not the case for the run where all nodes are enabled, as most of this inactive time is due to *spontaneously inactive* nodes, for whom it is harder to determine whether or not it is permissible to disable. Note that since this is a job trace over 30 minutes, both experiments complete in the same amount of time, even though the constituent jobs may run for different lengths of time. The run with 18 nodes disabled consumes 44% less energy than the run with no nodes disabled.

4. TOWARDS AN ENERGY-EFFICIENCY HADOOP

This work to date has just scratched the surface of research into optimizing the energy-efficiency of Hadoop and similar distributed processing frameworks. Our future work is focused on large-scale systems and addressing the issues we present below.

Data Layout — Section 2 described our implementation of an energy-aware replication invariant, but it is easy to imagine a multitude of alternatives. Certain types of data may exhibit temporal locality, which could be taken into account in choosing whether or not to keep those data in a covering subset. Next, the covering subset of a cluster should rotate among datanodes to manage durability, about which we discuss below. Finally, should a covering subset become a write bottleneck, a lazy replication strategy could be adopted (first create a replica wherever convenient or performant, and later move it to the covering subset).

Data Availability — We have assumed that it is not permissible for requested data to only reside on disabled nodes. The concept of a covering subset and the design of our proposed replication invariant ensure some minimum availability of all requested data. However, it is reasonable to imagine that nodes could be enabled *on-demand* to satisfy availability. The efficacy of this approach would depend on how quickly a node can be enabled, or how well the task scheduler copes with an unruly delay. Such a solution may permit an energy-efficient Hadoop with no changes to data layout.

In addition, we currently handle the failure of a covering subset node by enabling all nodes and reconstituting replicas of the data. A less dramatic solution should be sought, or else a very minor equipment failure can turn into a very major power spike.

Reliability and Durability — A supposition of this work is that the data stored on intentionally inactive nodes are indelible. The degree to which this is true should be bounded, or the degree on which this property is depended should be limited. At a minimum, it is prudent for sleeping nodes to be woken periodically so that the integrity of the data they contain can be verified.

Second, the fact that HDFS (and the Google Filesystem) materializes n replicas of all data is blindly accepted as the cost of reliability. However, spraying replicas throughout a cluster during a computation reduces the performance of the computation (by consuming disk and network bandwidth), which ultimately translates to wasted energy (by not spending energy on useful work). Design principles regarding reliability should be revisited, and the incorporation of some other data durability mechanism could be considered (e.g. an enterprise SAN). We seek to quantify the trade-off between reliability and energy consumption, such that users can make informed decisions.

Dynamic Scheduling Policies — A full implementation of an energy-efficient Hadoop should contain a dynamic power controller which is able to intelligently respond to changes in utilization of a Hadoop cluster. Different jobs may have disparate service levels requirement. Moreover, we must identify the best real-time signals to use for actuating power management activities, and how these impact job execution. As a most basic (but fundamental) example, inactive nodes that previously participated in a currently running job should not necessarily be disabled until after the job is complete; these nodes may contain transient output from Map tasks that will later be consumed by some separate Reduce task. There should be some intelligent cooperation between Hadoop's job scheduler and a power controller. Furthermore, the job scheduler has information that can improve the decisions made by the power controller. For example, all of the data blocks that a job will access are determined a priori, and this knowledge can be used to preemptively wake nodes or to hoist tasks with data on active nodes to the front of the task queue.

Node Architecture — Future technology, such as PowerNap [10] and PCRAM, may make short inactivity periods as useful as long inactivity periods in reducing energy consumption. In either case, this work is grounded in *increasing* the total inactive time available. Moreover, PCRAM and solid-state disks may not soon be economical for capacity-maximizing, cost-sensitive MapReduce clusters.

Workloads and Applications — While this paper considered Hadoop's MapReduce, note that HBase [2] and BigTable [6] build upon the same filesystem infrastructure as their sibling MapReduce frameworks. While they have similar requirements of the storage layer as MapReduce (massively scalable, distributed, robust), the nature of their workload and activity is far removed. MapReduce computations are characterized by long, predictable, streaming I/O, massive parallelization, and non-interactive performance. On the other hand, the aforementioned structured data services are often used in real-time data serving scenarios [6]. As such, quality-of-service and throughput become more important than job runtime. The energy management strategies discussed in this paper should be evaluated for these structured data services.

5. REFERENCES

[1] Lustre: A Scalable, High Performance File System. http://lustre.org/.

[2] Apache. Hadoop. http://hadoop.apache.org/.

[3] Luiz André Barroso and Urs Hölzle. The Case for Energy-Proportional Computing. *Computer*, 40(12), 2007.

[4] Standard Performance Evaluation Corporation. Specpower_ssj2008. http://www.spec.org/power_ssj2008/.

[5] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 2008.

[6] Chang Fay et al. Bigtable: A Distributed Storage System for Structured Data. OSDI, USENIX, 2006.

[7] Xiaobo Fan, W. Weber, and L. A. Barroso. Power Provisioning for a Warehouse-sized Computer. ISCA, ACM, 2007.

[8] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. *SIGOPS Oper. Syst. Rev.*, 2003.

[9] Intelligent Platform Management Interface. http://www.intel.com/design/servers/ipmi/.

[10] David Meisner, B. T. Gold, and T. F. Wenisch. PowerNap: Eliminating Server Idle Power. ASPLOS, ACM, 2009.

[11] Suzanne Rivoire, Parthasarathy Ranganathan, and Christos Kozyrakis. A Comparison of High-Level Full-System Power Models. HotPower, 2008.

[12] Amazon Web Services. http://aws.amazon.com/.

[13] VMotion. http://vmware.com/products/vi/vc/vmotion.html.