To Compress or Not To Compress - Compute vs. IO tradeoffs for MapReduce Energy Efficiency

Yanpei Chen
Archana Sulochana Ganapathi
Randy H. Katz

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2010-36
http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-36.html

March 29, 2010

To Compress or Not To Compress
Compute vs. IO tradeoffs for MapReduce Energy Efficiency

Yanpei Chen, Archana Ganapathi, Randy H. Katz
RAD Lab, EECS Department, UC Berkeley
{ychen2, archanag, randy}@eecs.berkeley.edu
Paper 7, 6 pages

ABSTRACT

Compression enables us to shift the computation load from IO to CPU. In modern datacenters where energy efficiency is a growing concern, the benefits of using compression have not been completely exploited. We develop a decision algorithm that helps MapReduce users identify when and where to use compression. For some jobs, using compression gives energy savings of up to 60%. As MapReduce represents a common computation framework for Internet datacenters, we believe our findings will provide signficant impact on improving datacenter energy efficiency.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms

Measurement, Performance, Design

1. INTRODUCTION

Internet datacenters increasingly rely on frameworks such as MapReduce for business critical computations. MapReduce hides the complexity of using a cluster of commodity machines. Users see a simple programming interface, but still need to optimize cluster configurations. One set of configurations controls compression on a per-job basis. Compression shifts the computation load from IO to CPU. Using compression improves both time and energy efficiency if the IO path is the latency and energy bottleneck. This idea already received much attention in prior work on wireless encodings, where radio transmission represents the energy bottleneck.

In MapReduce and datacenters, current uses of compression remain ad-hoc. The recent Yahoo! petasort implementation used MapReduce with compression [7]. From our RAD Lab industry partners, we heard that some companies use compression almost always, while other companies never use compression. We believe that it is suboptimal to have a blanket rule to always or never use compression. The decision about whether compression makes sense depends on job-specific data properties and IO patterns.

We perform a systematic evaluation of the compute vs. IO tradeoffs in using compression for MapReduce. For read-heavy text data, compression gives 35-60% energy savings. For highly compressible data, the savings are even higher. The data properties and IO patterns for each job will determine the amount of savings. Our key technical contribution is a decision algorithm that answers the "to compress or not to compress" question per job. The algorithm facilitates a selective compression policy for multi-job workloads, and we present some initial thoughts on workload driven evaluations. We believe that an effective compression decision algorithm provides another tool for datacenter operators to reason about workloads and improve energy efficiency.

The remainder of this paper is as follows. Section 2 gives an overview of compression mechanisms in MapReduce and related work on energy efficiency. Section 3 outlines the parameters we consider, our energy measurement method and cluster setup. Section 4 analyzes our results and quantifies compute vs. IO tradeoffs. Section 5 presents our compression decision algorithm. Section 6 highlights topics for future work.

2. BACKGROUND

2.1 Compression in MapReduce

We use the open source Hadoop implementation of MapReduce. At its core, MapReduce has two user-defined functions. The Map function takes in a key-value pair, and generates a set of intermediate key-value pairs. The intermediate key-value pairs are *shuffled* over the network to Reduce nodes. The Reduce function emits a final set of key-value pairs. For Hadoop, the input and output data reside in the Hadoop distributed file system (HDFS). The intermediate key-value pairs stage to local disks before being shuffled.

Compression offers a way to decrease IO demands (compressed data has smaller size) through increased CPU work (required for compression and decompression). Hadoop exposes various configuration settings to control three aspects of compression.

What to compress: Hadoop allows users to compress output data, intermediate data, or both. Hadoop checks whether input data is in a compressed format and decompresses the data as needed.

Compression codec: We use Hadoop 0.18.2, which includes two lossless codecs. By default, Hadoop uses the gzip codec. It implements the DEFLATE algorithm, a combination of Lempel-Ziv 1977 (LZ77) and Huffman encoding. The other codec implements the Lempel-Ziv-Oberhumer (LZO) algorithm, a variant of LZ77 optimized for decompression speed.

Compression unit: Hadoop allows both per-record and per-block compression. Thus, the record or block size affects the compressibility of the data.

2.2 Related Work

Our study builds on prior work in energy benchmarking as well as MapReduce energy evaluation.

Power proportionality has been proposed as a worthy system design goal [4]. Power *proportionality* differs from energy *efficiency* in that proportionality implies low energy consumption at low system utilization, and efficiency implies low energy usage to complete a certain compute job. Non-power proportional machines have small dynamic power ranges. This makes energy efficiency equivalent to time efficiency, since energy is the product of power and time. We will see this equivalence later in our results.

JouleSort is a software benchmark that measures the energy required to perform an external sort [8]. Recent work in MapReduce energy efficiency used the JouleSort benchmark for clusters of tens of machines sorting 10-100GB data [5, 6]. Unrelated to energy efficiency, Yahoo! used Hadoop to sort petabyte scale data in reasonable time [7]. The Yahoo! petasort effort used LZO compression for intermediate data.

In [6], the authors seek to improve HDFS energy efficiency by "sleeping" nodes during periods of low load. The evaluation focuses only on default Hadoop configurations and assumes sleeping nodes have zero power. The zero power assumption would not be true if the nodes still participate in data replication. This work offered us many lessons in developing our methodology.

A direct predecessor to our work, [5] explores MapReduce energy consumption for a variety of jobs, stressing each part of the MapReduce data path. It examined design choices including cluster size, configuration parameters, and input sizes, to name a few. We use several MapReduce jobs from [5] for evaluating compression.

3. METHODOLOGY

3.1 MapReduce Jobs and Parameters

Hadoop is a complex system with many parameters. A production system imposes additional complexities that make a full scan of the parameter space impractical. We simplify the problem by focusing on key parameters and a few demonstrative Hadoop jobs.

We look at four Hadoop jobs - HDFS write, HDFS read, shuffle, and sort. The first three jobs stress one part of the Hadoop IO pipeline at a time. Sort represents a 1-1-1 combination of the three different IO stages. We implement these jobs by modifying the `randomwriter` and `randomtextwriter` examples that are prepackaged with every Hadoop distribution.

Data compressibility is a key factor. Without a way to finely control data compressibility, we used four algorithms that generate data of different compressibility:

1. `randomwriter`: Generates random bits in terasort format. Has gzip block compression ratio of roughly 1.1, i.e., the ratio between compressed data size and uncompressed data size is 1.1. The compression ratio is greater than 1 due to gzip prefix tree overhead.
2. `randomtextwriter`: Samples from a random selection of 1000 words from the Linux dictionary. Has gzip block compression ratio of roughly 0.3. The gridmix pseudo-benchmark uses this job. The algorithm favors infrequent words, and the small words list leads to an artificially low compression ratio.
3. `randomshakespeare`: Samples from the English Wikipedia entry for "Shakespeare". Represents English text more accurately than `randomtextwriter`, and gives a gzip block compression ratio of roughly 0.4. We used the March 2010 Wikipedia snapshot for Shakespeare.
4. `repeatshakespeare`: Repeats the English Wikipedia entry for "Shakespeare" in the correct word order. Represents highly compressible data. Gzip picks up the repeated common substrings, with the block compression ratio being roughly 0.004.

Other key parameters include HDFS block size (for per block compression) and record size (for per record compression). Memory allocation is another factor, since gzip stores the compression prefix tree in memory. The number of maps and reduces is also a factor, especially for sort and shuffle. More data per map decreases parallelism, since Hadoop stores all intermediate key-value pairs before performing compression on the intermediate data. More data per reduce would amplify network bottlenecks, and emphasize compression benefits.

Clearly, it is impractical to sweep the entire parameter space. Our strategy is to look at the full combination of input types and job types for only default Hadoop parameters. For non-default parameter values, we look at one input type only.

We defer several topics for future work: e.g., the choice of compression codecs; jobs with input-shuffle-output data ratios that are not 1-0-0, 0-1-0, 0-0-1, or 1-1-1; and the impact of different compute functions for map and reduce (our map and reduce functions are essentially identify functions). Section 5.2 offers a glimpse of the more expanded problem space.

Our hypothesis is that compression would yield the greatest benefit for highly compressible data, and lead to considerably smaller performance variation for all data. For uncompressible data, compression would create significant overhead. A priori, we have no estimation of what compressibility level would reverse the tradeoff between IO and CPU, or whether any non-default parameter values would affect the tradeoff.

3.2 Energy Measurement

We measure energy at the wall power socket using a power meter. Thus, our measurements capture the holistic system performance, including any idle components drawing wasted power. We use a Brand Electronics Model 21-1850/CI power meter, with 1W power resolution and 1Hz sampling rate.

We measure energy consumption for MapReduce workers only. Any fixed, per-cluster overhead such as the master or the network switch would be amortized across a large cluster. Also, for a homogenous cluster not optimized for rack locality, the behavior of one worker is statistically identical to the behavior of other workers. Thus, we monitor only one worker, and capture the variation between workers via repeated measurements.

For each configuration, we take 10 repeated readings. Single measurements are useless for comparisons, since Hadoop performance variation can be quite large [5]. We show measurement averages and 95% confidence intervals assuming a Gaussian (normal) distribution of the data. This assumption is statistically valid per the Central Limit Theorem and the Berry-Esseen Theorem.

For our energy measurements, each Hadoop job read, write, shuffle, or sort 10GB of data.

3.3 Cluster Setup

We use a 10-node cluster. Each node is a Sun Microsystems X2200 M22 machine running Linux 2.6.26-1-xen-amd64, with two dual-core AMD Opteron Processor at 2.2GHz, 3.80GB RAM, 250GB SATA Drive, and 1Gbps Ethernet through a single switch. Each machine has approximately 150W fully idle power and 250W fully active power [2]. The machines consume approximately 190W running Hadoop at idle, i.e., no active Hadoop jobs.

Although a 10-node cluster appears small for our experiments, a survey of production clusters suggests that around 70% of MapReduce clusters contain fewer than 50 machines [1]. Thus, our findings on 10 machines easily generalize to clusters at that scale.

We use Hadoop 0.18.2 with no virtualization. We decided against newer Hadoop distributions to ensure data comparability with our early experiments and the results in [5]. Unless otherwise noted, we use default configuration parameters for Hadoop.

We run experiments in a controlled environment. We use a shared cluster and closely monitor the CPU, disk, and network load during our experiments. When we detect any activity not due to Hadoop, we stop data collection and repeat the measurement at a later time.

4. RESULTS

We conducted several experiments to understand the time and energy efficiency impact of compression under various cluster configurations and input datasets.

4.1 Default Hadoop Configurations

Figure 1 shows the effect of using compression for the default Hadoop configuration. We used data generated by the `randomshakespeare` algorithm. Compression has a small cost for HDFS write and sort, a small benefit for HDFS read, and a significant cost for shuffle.

Both HDFS write and sort involve transmitting a lot of data over the network. Thus, the additional work of compression balances out the benefit of transmitting less data. For HDFS read, the benefit of reading less data outweighs the relatively small cost of decompression. For shuffle, transmission over the network begins only after each machine compresses all of its intermediate data. The additional work of compression outweighs the benefit of transmitting less data. Also, for sort, doing compression for only the shuffle step impose a smaller cost. We expect these tradeoffs to change for data with different compressibility.

The machines in our cluster are not power proportional. Thus, power consumption remains largely fixed (note the truncated vertical axis). For these machines, the duration and energy graphs are near identical, since the power is near constant and energy is the product of power and time. Thus, for subsequent results, we show the energy graph only.

4.2 Data Compressibility

Figure 2 shows the effect of different data compressibility levels. We organized the data by decreasing compressibility, i.e., `repeatshakespeare`, followed by `randomtextwriter`, `randomshakespeare`, and `randomwriter`. For jobs that use compression, decreasing compressibility leads to increased energy (and time). For uncompressible data, compression represents wasted work.

The compressibility of data determines whether compression is worthwhile. For `repeatshakespeare`, compression always yields clear benefits of 50-70%. For `randomtextwriter`, compression has a 60% benefit for read, 10% benefit for sort, 20% cost for write, and 90% cost for shuffle. For `randomshakespeare`, compression has a 35% benefit for read, and a clear disadvantage for other IO patterns. For `randomwriter`, compression has a clear disadvantage for all jobs. Clearly, a blanket policy on compression does not make sense. For read heavy jobs and jobs with highly compressible data, compression brings considerable benefits.

The data for `randomtextwriter` is noteworthy as compression results in relatively small cost or benefit for all stages of the IO pipeline except shuffle. This suggests that a compression ratio of roughly 0.3 is a turning point. Compression always helps for data with a compression ratio much less than 0.3, and always hurts for data with higher compression ratios. The further away from the turning point, the greater the benefit or cost.

Production data in binary, numeric, text, and image formats have different compressibility. Per-job compression decisions must account for this variation.

4.3 Miscellaneous Configuration Parameters

Figure 3 shows the effect of different HDFS block sizes and memory allocations. The default HDFS block size is 64MB. We increase it to twice the default value. For memory allocations, we configure the `io.sort.mb`, `fs.inmemory.size.mb`, `mapred.child.java.opts` parameters in `hadoop-site.xml`. The default for these parameters are respectively 100MB, 75MB, and 200MB. For the large memory configurations, we double the default values for all three parameters.

Figure 3 shows that different block sizes and memory allocations have minimal impact on the compression tradeoff. There is some variability between the configurations, but the tradeoffs are preserved.

Figure 3 also shows that using compression leads to smaller variability. Jobs that use compression have much smaller 95% confidence intervals. Other graphs also show this effect.

Figure 4 shows the effect of changing the data per map by assigning a different number of map tasks. More map tasks "smooth out" any cost and benefits. E.g., if compression has a benefit compared with no compression, then the benefit remains, but becomes smaller when there are more map tasks. Less data per map means compression contributes less to the overall finishing time and hence energy compared to the overhead of launching many waves of map tasks. In the extreme case where each map task deals with a trivial amount of data, the overhead of task launches dominate, and compression has no effect on the energy consumption.

5. DISCUSSION

Our results allow MapReduce users to identify the appropriate compression settings for their jobs. For an entire workload, the benefit that compression brings depends on the distribution of jobs in the workload. We discuss below initial ideas on both the decision algorithm and the challenges of workload driven evaluations.

5.1 Decision Algorithm for Compression

Our results in Figure 2 enable us to construct a per-job decision algorithm that indicates whether the job should compress output and/or intermediate pairs. Note that Hadoop cannot control compression at the input stage - it must reads data as required by the job. Figures 3 and 4 indicate that the decision algorithm remains fixed regardless of block sizes, memory allocations, or the number of map and reduce tasks.

We propose the following algorithm:

```
Input

  No decision required.

Output

  If compression ratio < 0.2, compress.
  If compression ratio > 0.4, do not compress.
  Else,
    If data will be frequently read, compress.
    Else, do not compress.

Shuffle

  If compression ratio < 0.2, compress.
  Else, do not compress.
```

The algorithm requires prior knowledge of output and intermediate data compressibility. Such knowledge is not always available. The intuition to assess compressibility is that the more "ordered" the data is, the more compressible the data would be as less information is required to represent it. Compression codecs try to find the optimal representation.

Truly random data (`randomwriter`) is uncompressible, and highly ordered data (`repeatshakespeare`) is extremely compressible. Textual data has compression ratio ranging from 0.3 (`randomtextwriter`, small dictionary) to 0.4 (`randomshakespeare`, larger dictionary). To provide more examples, twitter data feeds in JSON format have roughly 0.2 compression ratio. Hadoop code has roughly 0.4 compression ratio. Wind power logs from a wind farm have 0.15 compression ratio. Multimedia formats such as MP3 and JPEG have 0.8-0.9 compression ratio.

When the data represents an unknown format, we recommend performing compression at least once. This helps the algorithm make future decisions regarding the same format.

5.2 Towards Workload Driven Evaluations

To use this decision algorithm in production, we must consider multi-job workloads. As a result, we must address several additional challenges. First, we need to identify a representative workload. Through our RAD Lab industrial partners, we know that different organizations have vastly different workloads. Thus, workload driven evaluations are likely to be case-by-case studies.

We are currently pursuing a thorough evaluation of a Facebook production workload based on Hadoop log data. Figures 5 and 6 offer initial insights.

Figure 5 shows that most jobs have data sizes less

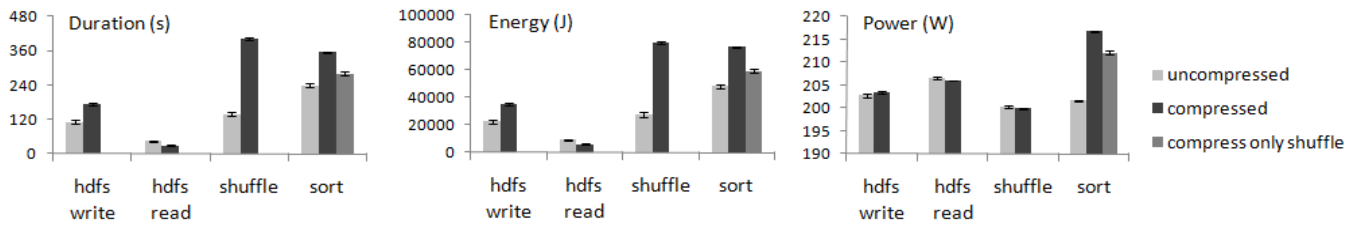

Figure 1: Duration, energy, and power for HDFS write, HDFS read, shuffle, and sort. Data from `randomshakespeare`.

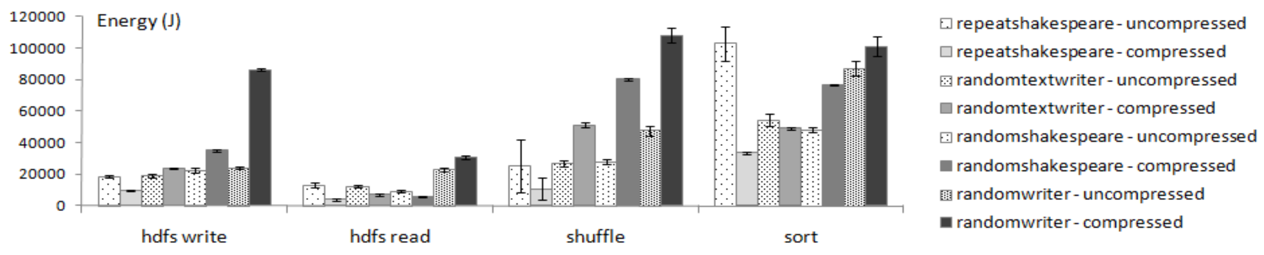

Figure 2: Effect of data with different compressibility. Showing energy only.

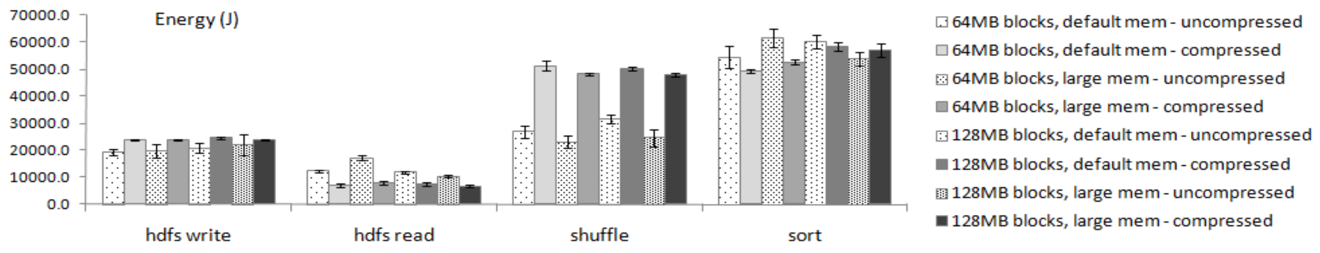

Figure 3: Effect of different block sizes and memory allocations. Showing energy only. Data from `randomtextwriter`.

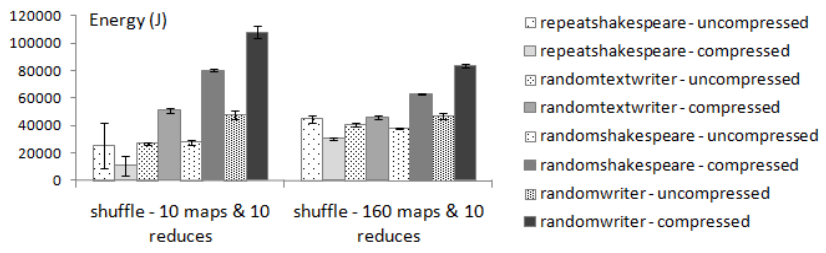

Figure 4: Effect of different amount of data per map. Showing energy for shuffle only. Data from `randomshakespeare`.

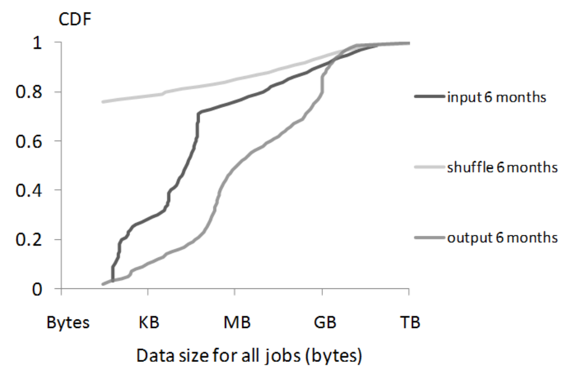

Figure 5: CDF of data sizes for 6 months of production Hadoop jobs on one Facebook cluster.

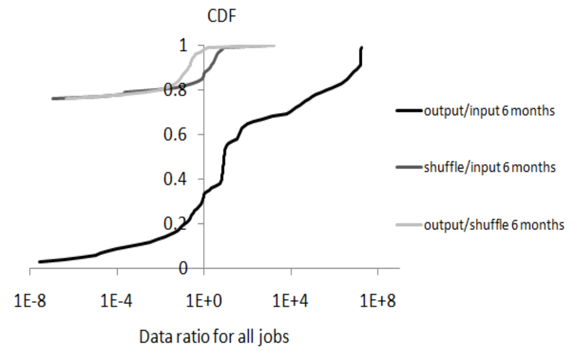

Figure 6: CDF of data ratios for 6 months of production Hadoop jobs on one Facebook cluster.

than 1GB, while GB scale jobs represent a small albeit non-negligible fraction. For jobs in the MB and KB data scale, Hadoop overhead dominates the finishing time and energy consumption. However, the GB scale jobs consume a large share of compute resources, validating our focus on GB scale datasets.

Approximately 80% of the jobs have no shuffle data. Thus, for this workload, output compression decisions outweight shuffle compression decisions.

Figure 6 shows that few jobs have input-shuffle-output data ratio that is 1-0-0, 0-1-0, 0-0-1, or 1-1-1. While our decision algorithm accomodates any data ratios, we would not have arrived at the prequisite insights without first studying simplified data ratios. This validates our initial focus on single, simplified jobs.

Combined, Figures 5 and 6 build a strong case for evaluation using multi-job workloads.

Many challenges remain. For example, reproducing data compressibility would be difficult without access to production input datasets. Reproducing the read and write patterns of each data item would also be a challenge. Even if we assume `randomshakespeare` format and reproducing only the data size and data ratios, we still need a workload replay mechanism. This mechanism would not be straightforward since we cannot faithfully reproduce the same cluster scale and configuration of Facebook's production cluster.

We are working to overcome these challenges. We developed some initial thoughts regarding the workload replay mechanism in [3].

6. CONCLUSION

We analyzed how compression can improve performance and energy efficiency for MapReduce workloads. Our results show that compression provides 35-60% energy savings for read heavy jobs as well as jobs with highly compressible data. Based on our measurements, we construct an algorithm that examines per-job data characteristics and IO patterns, and decides when and where to use compression.

The compression decision algorithm would contribute to a Hadoop workload manager that makes per-job optimizations without user involvement. The workload manager can encompass many concerns other than compression. However, evaluating such a workload manager requires knowledge of workload characteristics, as well as a workload generation or replay framework. This framework would also facilitate detailed exploration of several compression factors not examined in this paper, such as a range of data compressibility, different compression codecs, resource contention between compression and the compute function of maps and reduces, etc. Thus, we believe the workload evaluation framework should be the focus of future work.

7. REFERENCES

[1] Hadoop Power-By Page. http://wiki.apache.org/hadoop/PoweredBy.
[2] Sun Fire X2200 M2 Server power calculator. http://www.sun.com/servers/x64/x2200/calc/.
[3] A. Ganapathi et al. Statistics-driven workload modeling for the Cloud. In *SMDB '10: International Workshop on Self Managing Database Systems.*
[4] L. A. Barroso and U. Hölzle. The case for energy proportional computing. *Computer*, 40(12):33–37, 2007.
[5] Y. Chen, L. Keys, and R. Katz. Towards energy efficient MapReduce. Technical Report UCB/EECS-2009-109, UC Berkeley, 2009.
[6] J. Leverich and C. Kozyrakis. On the Energy (In)efficiency of Hadoop Clusters. In *HotPower '09: Workshop on Power Aware Computing and Systems.*
[7] O. O'Malley and A. C. Murthy. Winning a 60 second dash with a yellow elephant. Yahoo! Developer Network Blog. May 11, 2009, http://developer. yahoo.com/ blogs/hadoop/Yahoo 2009.pdf.
[8] S. Rivoire et al. Joulesort: a balanced energy-efficiency benchmark. In *SIGMOD '07: Proceedings of the ACM international conference on Management of data.*