Nobody ever got fired for buying a cluster

Raja Appuswamy[1]
Microsoft Research, Cambridge, UK

Christos Gkantsidis
Microsoft Research, Cambridge, UK

Dushyanth Narayanan
Microsoft Research, Cambridge, UK

Orion Hodson
Microsoft Research, Cambridge, UK

Antony Rowstron
Microsoft Research, Cambridge, UK

Technical Report
MSR-TR-2013-2

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
http://www.research.microsoft.com

[1]Work done while on internship from Vrije Universiteit Amsterdam

Abstract

In the last decade we have seen a huge deployment of cheap clusters to run data analytics workloads. The conventional wisdom in industry and academia is that scaling out using a cluster is better for these workloads than scaling up by adding more resources to a single server. Popular analytics infrastructures such as Hadoop are aimed at such a cluster scale-out environment, and in today's world nobody gets fired for adopting a cluster solution.

Is this the right approach? Our measurements as well as other recent work shows that the majority of real-world analytic jobs process less than 100 GB of input, but popular infrastructures such as Hadoop/MapReduce were originally designed for petascale processing. We claim that a single "scale-up" server can process each of these jobs and do as well or better than a cluster in terms of performance, cost, power, and server density. Is it time to consider the "common case" for "big data" analytics to be the single-server rather than the cluster case? If so, this has implications for data center hardware as well as software architectures.

Unfortunately widely used platforms such as Hadoop perform poorly in a scale-up configuration. We describe several modifications to the Hadoop runtime to address this problem. These changes are transparent, do not require any changes to application code, and do not compromise scale-out performance. However they do significantly improve Hadoop's scale-up performance. We present a broad evaluation across 11 representative Hadoop jobs that shows scale-up to be competitive in all cases and significantly better in some cases, than scale-out. Our evaluation considers raw performance, as well as performance per dollar and per watt.

1 Introduction

Data analytics, and in particular, MapReduce [8] and Hadoop [1] have become synonymous with the use of cheap commodity clusters using a distributed file system that utilizes cheap unreliable local disks. This is the standard scale-out thinking that has underpinned the infrastructure of many companies. Clearly large clusters of commodity servers are the most cost-effective way to process exabytes, petabytes, or multi-terabytes of data. Nevertheless, we ask: is it time to reconsider the scale-out versus scale-up question?

First, evidence suggests that the *majority* of analytics jobs do not process huge data sets. For example, at least two analytics production clusters (at Microsoft and Yahoo) have median job input sizes under 14 GB [10, 17], and 90% of jobs on a Facebook cluster have input sizes under 100 GB [4].

Second, many algorithms are non-trivial to scale out efficiently. For example, converting iterative-machine learning algorithms to MapReduce is typically done as a series of MapReduce rounds. This leads to significant data transfer across the network between the map and reduce phases and across rounds. Reducing this data transfer by changing the algorithm is expensive in terms of human engineering, may not be possible at all, and even if possible results in an approximate result.

Third, hardware price trends are beginning to change performance points. Today's servers can affordably hold 100s of GB of DRAM and 32 cores on a quad socket motherboard with multiple high-bandwidth memory channels per socket. DRAM is now very cheap, with 16 GB DIMMs costing around $220, meaning 192 GB costs less than half the price of a dual-socket server and 512 GB costs 20% the price of a high-end quad-socket server. Storage bottlenecks can be removed by using SSDs or with a scalable storage back-end such as Amazon S3 [3] or Azure Storage [6, 5]. The commoditization of SSDs means that $2,000 can build a storage array with multiple GB/s of throughput. Thus a scale-up server can now have substantial CPU, memory, and storage I/O resources and at the same time avoid the communication overheads of a scale-out solution. Moore's law continues to improve many of these technologies, at least for the immediate future.

Is it better to scale up using a well-provisioned single server or to scale out using a commodity cluster? For the world of analytics in general and Hadoop MapReduce in particular, this is an important question. Today the default assumption for Hadoop jobs is that scale-out is the only configuration that matters. Scale-up performance is ignored and in fact Hadoop performs poorly in a scale-up scenario. In this paper we re-examine this question across a range of analytic workloads and using four metrics: *performance*, *cost*, *energy*, and *server density*.

This leads to a second question: how best to achieve good scale-up performance. One approach is to use a shared-memory, multi-threaded programming model and to re-implement algorithms to use this model. This is the easiest way to show good performance as the implementation can be tuned to the algorithm. However, it is intensive in terms of human work. It also means the work is wasted if the job then needs to scale in the future beyond the limits of a single server.

A second approach is to provide the same MapReduce API as is done for scale-out but to optimize the infrastructure for the scale-up case. In fact, ideally the switch between scale-out and scale-up would be completely transparent to the Hadoop programmer. This is the approach we take: how can we get all the benefits of scale-up within a commonly used framework like Hadoop. Our solution is based on transparent optimiza-

tions to Hadoop that improve scale-up performance without compromising the ability to scale out.

While vanilla Hadoop performs poorly in a scale-up configurations, a series of optimizations makes it competitive with scale-out. Broadly, we remove the initial data load bottleneck by showing that it is cost-effective to replace disk by SSDs for local storage. We then show that simple tuning of memory heap sizes results in dramatic improvements in performance. Finally, we show several small optimizations that eliminate the "shuffle bottleneck".

This paper makes two contributions. First, it shows through an analysis of real-world job sizes as well as an evaluation on a range of jobs, that scale-up is a competitive option for the majority of Hadoop MapReduce jobs. Of course, this is not true for petascale or multi-terabyte scale jobs. However, there is a large number of jobs, in fact the majority, that are sub-terabyte in size. For these jobs we claim that processing them on clusters of 10s or even 100s of commodity machines, as is commonly done today, is sub-optimal. Our second contribution is a set of transparent optimizations to Hadoop that enable good scale-up performance. Our results show that with these optimizations, raw performance on a single scale-up server is better than scale-out on an 8-node cluster for 9 out of 11 jobs, and within 5% for the other 2. Larger cluster sizes give better performance but incur other costs. Compared to a 16-node cluster, a scale-up server provides better performance per dollar for all jobs. When power and server density are considered, scale-up performance per watt and per rack unit are significantly better for all jobs compared to either size of cluster.

Our results have implications both for data center provisioning and for software infrastructures. Broadly, we believe it is cost-effective for providers supporting "big data" analytic workloads to provision "big memory" servers (or a mix of big and small servers) with a view to running jobs entirely within a single server. Second, it is then important that the Hadoop infrastructure support both scale-up and scale-out efficiently and transparently to provide good performance for both scenarios.

The rest of this paper is organized as follows. Section 2 shows an analysis of job sizes from real-world MapReduce deployments that demonstrates that most jobs are under 100 GB in size. It then describes 11 example Hadoop jobs across a range of application domains that we use as concrete examples in this paper. Section 3 then briefly describes the optimizations and tuning required to deliver good scale-up performance on Hadoop. Section 4 compares scale-up and scale-out for Hadoop for the 11 jobs on several metrics: performance, cost, power, and server density. Section 5 discusses some implications for analytics in the cloud as well as the crossover point between scale-up and scale-out. Section 6 describes related work, and Section 7 concludes the paper.

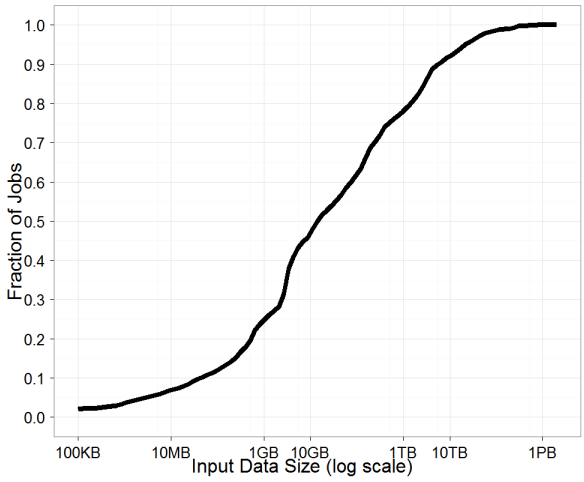


Figure 1: Distribution of input job sizes for a large analytics cluster

2 Job sizes and example jobs

A key claim of this paper is that the majority of real-world analytic jobs can fit into a single "scale-up" server with up to 512 GB of memory. We analyzed 174,000 jobs submitted to a production analytics cluster in Microsoft in a single month in 2011 and recorded the size of their input data sets. Figure 1 shows the CDF of input data sizes across these jobs. The median job input data set size was less than 14 GB, and 80% of the jobs had an input size under 1 TB. Thus although there are multi-terabyte and petabyte-scale jobs which would require a scale-out cluster, these are the minority.

Of course, these job sizes are from a single cluster running a MapReduce like framework. However we believe our broad conclusions on job sizes are valid for MapReduce installations in general and Hadoop installations in particular. For example, Elmeleegy [10] analyzes the Hadoop jobs run on the production clusters at Yahoo. Unfortunately, the median input data set size is not given but, from the information in the paper we can estimate that the median job input size is less than 12.5 GB.[1] Ananthanarayanan et al. [4] show that Facebook jobs follow a power-law distribution with small jobs dominating; from their graphs it appears that at least 90% of the jobs have input sizes under 100 GB. Chen et al. [7] present a detailed study of Hadoop workloads for Facebook as

[1] The paper states that input block sizes are usually 64 or 128 MB with one map task per block, that over 80% of the jobs finish in 10 minutes or less, and that 70% of *these* jobs very clearly use 100 or fewer mappers (Figure 2 in [10]). Therefore conservatively assuming 128 MB per block, 56% of the jobs have an input data set size of under 12.5 GB.

Job	Input	Shuffle	Output
Log crunching			
FindUserUsage	41 GB	4 MB	36 KB
ComputeIOVolumes	94 GB	157 MB	30 MB
Analytic queries			
Select Task[1]	41 GB	0 KB	0 KB
Aggregate Task	70 GB	5 GB	51 MB
Join Task[2]	113 GB	10 GB	4.3 GB
TeraSort			
10 GB TeraSort	11 GB	11 GB	11 GB
50 GB TeraSort	54 GB	54 GB	54 GB
Pig Cogroup	5 GB	7 GB	131 GB
Mahout[3]			
k-Means	72 MB	N/A	N/A
Wikipedia	432 MB	N/A	N/A
Indexing	10 GB	34 GB	26 GB

[1] Select produces negligible intermediate and output data.
[2] Sum of input, output, and shuffle bytes across three stages.
[3] Mahout jobs iterate over many map-reduce stages and hence we only measure input data size.

Table 1: Summary of jobs used with input, shuffle, and output data sizes.

well as 5 Cloudera customers. Their graphs also show that a very small minority of jobs achieves terabyte scale or larger and the paper claims explicitly that "most jobs have input, shuffle, and output sizes in the MB to GB range."

These data are extracted from jobs running in production clusters: we are not able to compare their performance directly on scale-up versus scale-out infrastructures. In order to do so, we have collected a smaller set of jobs from a variety of sources. We describe these jobs in the rest of the section, and use them as benchmarks in our evaluation in Section 4. Broadly the jobs we examine can be classified by application domain into log analysis, query-based data analytics, machine learning, and searching/indexing. Table 1 shows the amount of input, shuffle, and output data for each job.

2.1 Log crunching

One of the most common uses of Hadoop and MapReduce is to process text logs. We use two such jobs from a real-world compute platform consisting of tens of thousands of servers. Users issue tasks to the system that spawn processes on multiple servers which consume resources on each server. The system writes logs that capture statistics such as CPU, I/O and other resource utilization on these servers. Administrators can process these logs to extract both fine- and coarse-grained information about resource usage. In this paper we use two such jobs. The *FindUserUsage* job processes one such log to aggregate resource. The *ComputeIOVolumes* job processes two log files to combine process-level and task-level information, and hence find the amount of input and intermediate data I/O done by each task.

2.2 Analytic queries

We use three analytical queries from a benchmark that was created for comparing the performance of Hadoop against a parallel DBMS [13]. These tasks mimic query processing on a cluster of HTML documents gathered by a web crawl. The input data to these tasks consists of two tables, each sharded across a large number of files. Both data sets consist of random records generated by a custom data generator.

Records in the *PageRank* table associate each unique URL with a page rank. The *UserVisits* table contains one record per user visit, which contains the source IP, the date of visit, ad revenue generated, user agent, search word used, URL visited, country code, and language code. We use three queries (converted into Hadoop jobs) based on these two tables in our benchmark suite.

The *Select* task finds the top 1% of URLs by page rank from the PageRank table. The *Aggregate* task calculates the total ad revenue per source IP address, from the UserVisits table. The *Join* takes both data sets as input. It finds the source IP addresses that generates the most revenue within a particular date range and then computes the average page rank of all the pages visited by those IP addresses in that interval. It is computed in three phases, each of which is a MapReduce computation. We converted each phase into a separate Hadoop job in our benchmark suite.

2.3 TeraSort

TeraSort is a standard benchmark for data processing platforms. We use the Hadoop version of TeraSort. Despite its name, TeraSort can be used to sort different sizes of input data. In this paper we consider input sizes of 10 GB, 50 GB, and 100 GB.

2.4 Mahout machine learning tasks

In recent years, Mahout has emerged as a popular framework to simplify the task of implementing scalable machine learning algorithms by building on the Hadoop ecosystem. We used two standard machine learning algorithms implemented in Mahout in our benchmark suite.

The *Clustering* benchmark is based on k-means clustering, provided as part of the Mahout machine learning library [12]. This is an iterative machine-learning al-

gorithm implemented as a series of map-reduce rounds. The input to the algorithm is a set of points represented as d-dimensional vectors, and an initial set of cluster centroids (usually chosen randomly). In each round, the mappers map each vector to the nearest centroid, and the reducers recompute the cluster centroid as the average of the vectors currently in the cluster. We use the Mahout k-means implementation to implement a tag suggestion feature for Last.fm, a popular Internet radio site. The basic idea is to group related tags together so that the website can assist users in tagging items by suggesting relevant tags. We use the raw tag counts for the 100 most frequently occurring tags generated by Last.fm users as our input data set. The data set contains 950,000 records accounting for a total of 7 million tags assigned to over 20,000 artists.

The *Recommendation* benchmark is based on the recommendation mining algorithm also provided with Mahout. It is also iterative with multiple map-reduce rounds. We use this algorithm on the publicly available Wikipedia data set to compute articles that are similar to a given one. The output of the recommendation algorithm is a list of related articles that have similar outlinks as the given article and hence, are related to it and should possibly be linked with it. The Wikipedia input data set contains 130 million links, with 5 million sources and 3 million sinks.

2.5 Pig

Apache Pig is another platform that has gained widespread popularity in recent years as a platform that fosters the use of a high level language to perform large scale data analytics. We use a "co-group" query from a standard list of queries used as performance tests for Pig [18][2]. This takes two tables as input and does a "co-group" that effectively computes the cross-product of the two tables. It is interesting in that the final output is much larger than either the input or shuffle data, since the cross-product is computed at the reducers.

2.6 Indexing

Building an inverted index from text data was part of the original motivation for MapReduce [8], and inverted indices are commonly used by many search engines. We implemented a simple indexer as a Hadoop MapReduce job that takes a set of text files as input and outputs a full index that maps each unique word to a list of all occurrences of that word (specified as file name + byte offset).

[2]Other queries on this list essentially replicate the selection, aggregation, and projection tasks already included in our benchmark suite.

Optimization	Scale-up	Scale-out
SSD storage	Yes	Yes
Local FS for input	Yes	No
Optimize concurrency	Yes	Yes
Suppress OOB heartbeat	Yes	No
Optimize heap size	Yes	Yes
Local FS for intermediate data	Yes	No
Unrestricted shuffle	Yes	No
RAMdisk for intermediate data	Yes	No

Table 2: Summary of Hadoop optimizations grouped as storage, concurrency, network, memory, and reduce-phase optimizations. All the optimizations apply to scale-up but only some to scale-out.

3 Optimizing for scale-up

In order to evaluate the relative merits of scale-up and scale-out for the jobs described in Section 2, we needed a software platform to run these jobs. The obvious candidate for scale-out is Hadoop; we decided to use Hadoop as our platform on a single scale-up server as well, for three reasons. First, there is a large number of applications and infrastructures built on Hadoop, and thus there is a big incentive to stay within this ecosystem. Second, any scale-up platform will have to scale out as well, for applications that need to scale beyond a single big-memory machine. Finally, using the same base platform allows us an apples-to-apples performance comparison between the two configurations.

We first tuned and optimized Hadoop for good performance in the baseline, i.e., cluster scale-out, scenario. We then further optimized it to take advantage of features of the scale-up configuration, such as local file system storage and local RAMdisks. In this section we describe these optimizations. Table 2 lists the optimizations applied to Hadoop divided into five categories: storage, concurrency, network, memory, and reduce-phase optimizations. The storage, concurrency, and memory optimizations apply to both scale-up and scale-out configurations whereas the network and reduce-phase optimizations are only meaningful for the scale-up configuration. The concurrency and memory optimizations require tuning Hadoop parameters for the specific workload; all the other optimizations are workload-independent.

Here we describe each optimization. Section 4.5 has an experimental evaluation of the performance benefit of each optimization for the scale-up case.

3.1 Storage

Our first step was to remove the storage bottleneck for both scale-up and scale-out configurations. In a default configuration with disks, Hadoop is I/O-bound and has low CPU utilization. We do not believe that this is a re-

alistic configuration in a modern data center for either a scale-up or a scale-out system. Storage bottlenecks can easily be removed either by using SSDs or by using one of many scalable back-end solutions (e.g. SAN or NAS in the enterprise scenario, or Amazon S3 / Windows Azure in the cloud scenario). In our experimental setup which is a small cluster we use SSDs for both the scale-up and the scale-out machines.

Even with SSDs, there is still substantial I/O overhead due to the use of HDFS. While HDFS provides scalable and reliable data storage in a distributed Hadoop installation, we doubt its utility in single-node, scale-up installations for several reasons. A scalable storage back end can easily saturate the data ingest capacity of a single compute node. Thus using the compute node itself to serve files via HDFS is unnecessary overhead. Alternatively, if data is stored locally on SSDs, modern local file systems like ZFS and BTRFS already provide equivalent functionality to HDFS (e.g. check-summing) but without the associated overheads, and can be retrofitted to work with a single-node Hadoop setup.

Hence for our scale-up configuration we store the inputs on SSDs and access them via the local file system. For the scale-out configuration, we also store the inputs on SSDs but we access them via HDFS.

3.2 Concurrency

The next optimization adjusts the number of map and reduce tasks to be optimal for each job. The default way to run a Hadoop job on a single machine is to use the "pseudo-distributed" mode, which assigns a separate JVM to each task. We implemented a multi-threaded extension to Hadoop which allows us to run multiple map and reduce tasks as multiple threads within a single JVM. However, we found that when the number of tasks as well as the heap size for each tasks are well-tuned, there is no performance difference between the multi-threaded and pseudo-distributed mode.

Since our goal is to avoid unnecessary changes to Hadoop, we use the pseudo-distributed mode for the scale-up configuration and the normal distributed mode for the scale-out configuration. In both cases we tune the number of mappers and reducers for optimal performance for each job.

3.3 Heartbeats

In a cluster configuration, Hadoop tracks task liveness through periodic "out-of-band" heartbeats. This is an unnecessary overhead for a scale-up configuration where all tasks run on a single machine. For the scale-up configuration we disable these heartbeats.

3.4 Heap size

By default each Hadoop map and reduce task is run in a JVM with a 200 MB heap within which they allocate buffers for in-memory data. When the buffers are full, data is spilled to storage, adding overheads. We note that 200 MB per task leaves substantial amounts of memory unused on modern servers. By increasing the heap size for each JVM (and hence the working memory for each task), we improve performance. However too large a heap size causes garbage collection overheads, and wastes memory that could be used for other purposes (such as a RAMdisk). For the scale-out configurations, we found the optimal heap size for each job through trial and error. For the scale-up configuration we set a heap size of 4 GB per mapper/reducer task (where the maximum number of tasks is set to the number of processors) for all jobs.

3.5 Shuffle optimizations

The next three optimizations speed up the shuffle (transferring data from mappers to reducers) on the scale-up configuration: they do not apply to scale-out configurations. First, we modified Hadoop so that shuffle data is transferred by writing and reading the local file system; the default is for reducers to copy the data from the mappers via http. However, we found that this still leads to underutilized storage bandwidth during the shuffle phase, due to a restriction on the number of concurrent copies that is allowed. In a cluster, this is a reasonable throttling scheme to avoid a single node getting overloaded by copy requests. However it is unnecessary in a scale-up configuration with a fast local file system. Removing this limit substantially improves shuffle performance. Finally, we observed that the scale-up machine has substantial excess memory after configuring for optimal concurrency level and heap size. We use this excess memory as a RAMdisk to store intermediate data rather than using an SSD or disk based file system.

4 Evaluation

In this section, we will use the benchmarks and MapReduce jobs we described in Section 2 to perform an in-depth analysis of the performance of scale-up and scale-out Hadoop.

In order to understand the pros and cons of scaling up as opposed to scaling out, one needs to compare optimized implementations of both architectures side by side using several metrics (such as performance, performance/unit cost, and performance/unit energy) under a wide range of benchmarks. In this section we first describe our experimental setup. We then compare scale-

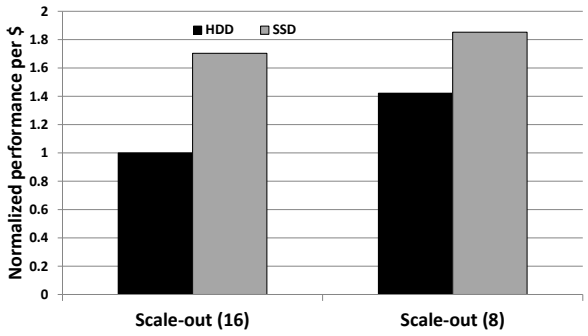


Figure 2: Performance/$ for scale-out running TeraSort with HDDs and SSDs, normalized to performance/$ of 16-node cluster with HDDs.

up and scale-out across all our entire benchmark suite on several metrics. These results are based on applying all the optimizations described in Section 3 to all the configurations.

We then look more closely at two of the jobs, FindUserUsage and TeraSort. For these two jobs we measure the individual phases — map, shuffle, and reduce — to understand better the scale-up/scale-out tradeoffs. We also measure the individual contributions of each of our optimization techniques on scale-up performance.

4.1 Hardware

The commodity scale-out cluster we used in this work consists of 16 data nodes and one name node. Each node is a Dell Precision T3500 Essential workstation with a 2.3 GHz E5520 Xeon quad-core processor (8 hyperthreads), 12 GB of memory, a 160 GB Western Digital Caviar Blue HDD, and a 32 GB Intel X25-E SSD. We used this cluster in two configurations: with 8 data nodes and with all 16 data nodes.

The scale-up machine is a 4-socket Dell PowerEdge 910 server with 4 8-core 2 GHz Intel Xeon E7-4820 processors for a total of 32 cores (64 hyperthreads). The server is also equipped with two RAID controllers. To this base configuration we added 512 GB of DRAM, two Hitachi UltraStar C10K600 HDDs in a RAID-1 configuration, and 8 240 GB Intel 520 Series SSDs in a RAID-0 configuration. The HDDs are used primarily for the OS image and the SSDs for Hadoop job data. Table 3 summarizes the hardware configurations and their prices. The scale-up configuration costs around $25k[3], whereas the 8-machine and 16-machine scale-out configurations cost $19k and $38k respectively, without counting the cost of the network switch that would be required in the scale-out case.

[3]Since the scale-up machine we used in our study is nearing its end-of-life period, we use as-on-date pricing of an equivalent configuration to approximate its acquisition cost.

We use SSDs for both the scale-up and the scale-out server. This is based on our observation that without SSDs, many Hadoop jobs become disk-bound. Thus although SSDs add 13% to the base cost of the workstation-class machine in Table 3, they improve the overall performance/price ratio of the cluster. We verified this by running the 10 GB TeraSort benchmark on the 8- and 16-node configurations with and without SSDs. Figure 2 shows the performance/price ratio for the four configurations, normalized so that the 16-node cluster using HDDs has a value of 1. We see that SSDs do improve performance/$ for the scale-out configuration, and we use SSDs consistently for both scale-up and scale-out in our experiments.

Table 3 also shows the power draw under load of the two hardware platforms. We measured the power consumption offline using a stress test to ensure consistent results. We used the 10 GB TeraSort benchmark to generate load, since it uniformly stresses the resources of all the servers. We used a *Watts up? Pro* power meter which logs power usage statistics at a one second granularity in its internal memory during the job run, and derive the average power from this log. We ran this test both on the scale-up and the scale-out configuration. In the scale-out case, TeraSort loads all servers equally and thus we only measure power on a single server. We measured the power in both the 8-node and 16-node configurations and found no significant difference.

All configurations have all relevant optimizations enabled (Section 3). Thus all three configurations use SSD storage, and have concurrency and heap sizes optimized for each job. The scale-up configuration also has heartbeats disabled and the shuffle optimizations described in Table 2 enabled.

4.2 Workloads

The benchmarks used in this evaluation are those described in Section 2. Both the scale-up and the larger (16-node) scale-out configuration were able to run all the jobs. However the 8-node scale-out configuration was not able to run three of the jobs at their full data size. In order to have a consistent comparison, we reduced the data set for these three jobs as described below.

For the Recommendation task, the input data set is 1 GB in size and contains 130 million links from 5 million articles to 3 million other articles. The 8-machine cluster ran out of storage space as the amount of intermediate data produced between iterative phases was too large. So we halved the data set by limiting the maximum number of outlinks from any given page to 10. The resulting data set had 26 million links (compared to the original 130 million) for a total of 500 MB

For the Indexing task, the input data set consists of

Machine	Base spec	Base cost	SSD cost	DRAM cost	Total cost	Power
Workstation[1]	4 x 2.3 GHz, 12 GB, 1 HDD	$2130	$270[2]	none	$2400	154 W
Server[3]	32 x 2.0 GHz, 0 GB, 2 HDD	$17380	$2160[2]	$5440[4]	$24980	877 W

[1] Dell Precision Fixed Workstation T3500 Essential, http://configure.euro.dell.com/dellstore/config.aspx?oc=w08t3502&model_id=precision-t3500&c=uk&l=en&s=bsd&cs=ukbsdt1

[2] 240 GB Intel 520 Series SSD, http://www.newegg.com/Product/Product.aspx?Item=N82E16820167086

[3] Dell PowerEdge R910, no DRAM, http://configure.euro.dell.com/dellstore/config.aspx?oc=per910&model_id=poweredge-r910&c=uk&l=en&s=bsd&cs=ukbsdt1

[4] 16 GB 240-Pin DDR3 1333 SDRAM, http://www.newegg.com/Product/Product.aspx?Item=N82E16820239169

Table 3: Comparison of workstation (used for scale-out) and server (scale-up) configurations. All prices converted to US$ as of 8 August 2012.

28,800 text files from the Gutenberg project. Since each file is relatively small compared to a HDFS block, and Hadoop is not capable of handling such data sizes well, we merged the text files into 60 MB chunks such that each file group fit comfortably within a single HDFS block. The resulting data set was 9.54 GB in size. The 8-node cluster fails to complete this job due to insufficient memory. There is a large amount of shuffle data, and reducers try to merge all the shuffle data in-memory to avoid spilling to disk. We determined empirically that trimming the data set to 8.9 GB allowed the 8-node cluster to complete without failures, and so we use this slightly smaller data set.

Finally, we found that the 8-node cluster could not complete a TeraSort of 100 GB due to limited storage space. Hence we use the 10 GB and 50 GB TeraSort for comparison.

4.3 Scale-up vs. scale-out

We ran all 11 jobs in all three configurations and measured their throughput, i.e. the inverse of job execution time. All results are the means of at least 4 runs of each job on each configuration.Figure 3(a) shows the results normalized so that the scale-up performance is always 1. We see that scale-up performs surprisingly well: better than the 8-machine cluster for all but two jobs and within 5% for those two.

When we double the cluster size from 8 to 16, scale-out performs better than scale-up for 6 jobs but scale-up is still significantly better for the other 5 jobs. In general, we find that scale-out works better for CPU-intensive tasks since there are more cores and more aggregate memory bandwidth. Scale-up works better for shuffle-intensive tasks since it has fast intermediate storage and no network bottleneck. Note that Pig Cogroup is CPU-intensive: a small amount of data is shuffled but a large cross-product is generated by the reducers.

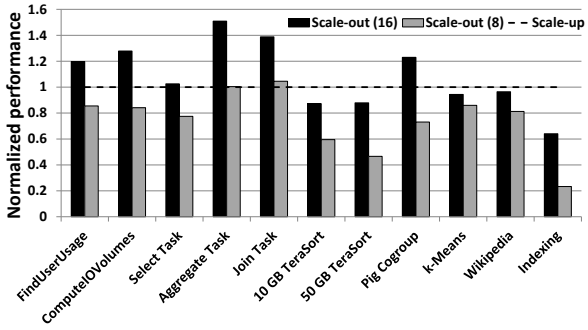
Clearly adding more machines does improve performance; but at what cost? It is important to consider not just raw performance but also performance per dollar. We derive performance per dollar by dividing raw performance by the capital/acquisition cost of the hardware (Table 3). In order to keep the analysis simple, here we do not consider other expenses like administration costs or cooling costs. In general we would expect these to be lower for a single machine.

Figure 3(b) shows the performance per dollar for the three configurations across all jobs, again normalized to set the scale-up platform at 1. Interestingly now the 8-node cluster does uniformly better than the 16-node cluster, showing that there are diminishing performance returns for scaling out even for these small clusters. It is worth noting that it is currently common practice to run jobs of these sizes (sub 100 GB) over even larger numbers of Hadoop nodes; we believe our results show that this is sub-optimal. Scale-up is again competitive (though slightly worse) for map-intensive tasks and significantly better for the shuffle-intensive tasks, than either scale-out configuration.

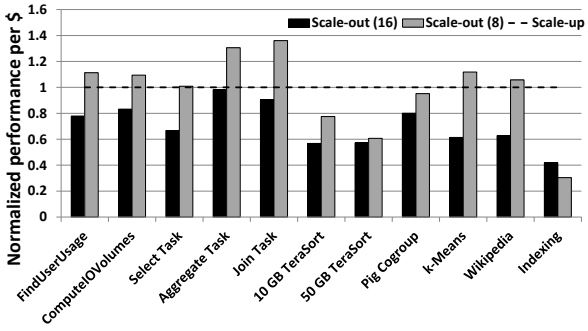
Interestingly we find that for k-means and Wikipedia, 8 nodes offer better performance/price than 16. This is because, although the core computation is CPU-bound, the speed-up from 8 nodes to 16 is sublinear due to the overheads of frequent task management: every iteration of the algorithm involves setting up a new MapReduce round, which impacts the performance substantially.

Figure 3(c) shows normalized performance per watt, based on the power measurements reported in Table 3. This is an important metric as data centers are often power-limited; hence more performance per Watt means more computation for the same power budget. On this metric we see that scale-up is significantly better than either scale-out configuration across all jobs. We believe the results are pessimistic: they underestimate scale-out power by omitting the power consumed by a top-of-rack switch, and overestimate scale-up power by including a redundant power supply.

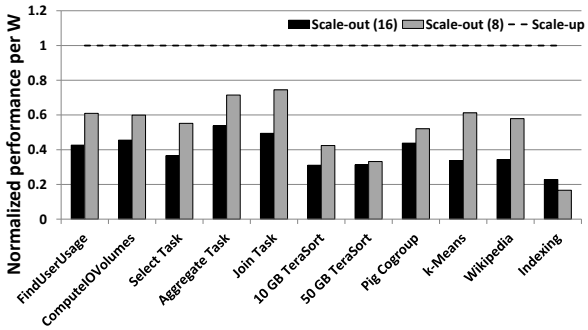
Finally, we look at server density. Like power, this is also an important consideration, as a higher server density means more computation for a given space budget. The scale-up machine is a "3U" form factor, i.e. it uses three rack units. Although our scale-out machines are in a workstation form factor, for this analysis we consider them to have a best-case 1U form factor. Figure 3(d) shows the performance per rack unit (RU). As



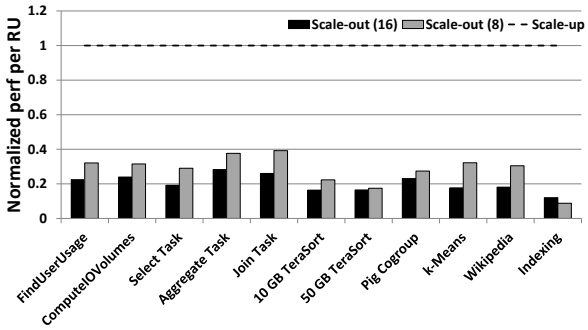
(a) Throughput



(b) Throughput per $

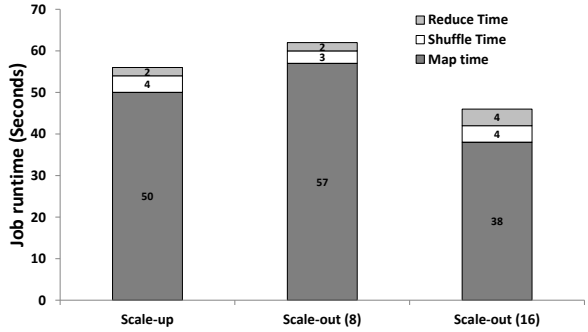


(c) Throughput per watt

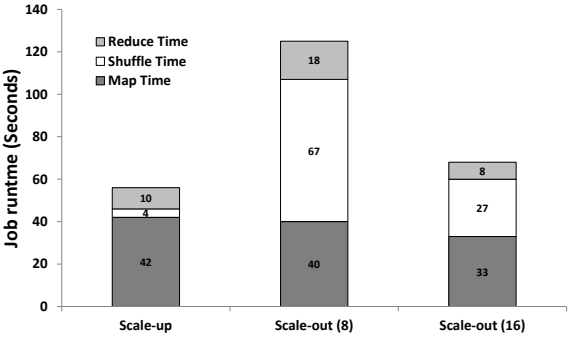


(d) Throughput per rack unit

Figure 3: Scale-out performance on different metrics, normalized to scale-up performance for each of 11 jobs.



(a) FindUserUsage



(b) 10 GB TeraSort

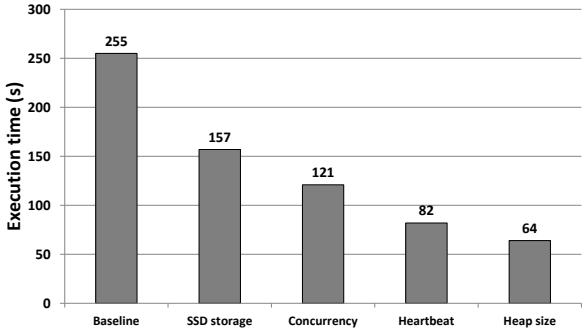
Figure 4: Runtime for different phases with FindUserUsage and 10 GB TeraSort

with power, we see clearly that the scale-up machine outperforms scale-out across all jobs.

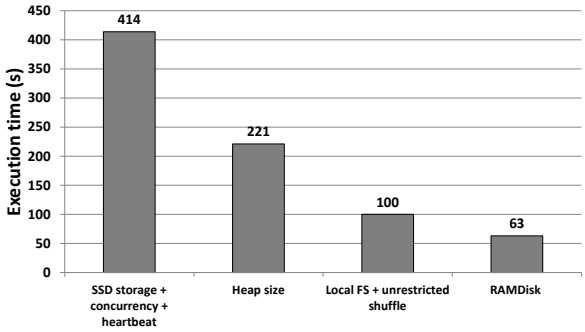
4.4 Phase-by-phase analysis

Broadly speaking, we expect map-intensive jobs to do relatively well for scale-out, and shuffle-intensive jobs to do well on scale-up. To validate this assumption, we choose two jobs: FindUserUsage, which is map-intensive, and the 10 GB TeraSort, which is shuffle-intensive. We then separated out the job execution times into map, shuffle, and reduce times. Since these can overlap we approximate them as follows: the map time is the time from the job start to the completion of the last map task; the shuffle time is the time interval from then to the completion of the last shuffle task; and the reduce time is the remaining time until the job is completed.

Figure 4(a) shows the results for FindUserUsage and Figure 4(b) the results for TeraSort. We see that as expected, runtime FindUserUsage is completely dominated by map time. Thus scale-up and 8-node scale-out have similar performance, since they both have 32 cores. The 16-node scale-out on the other hand benefits from twice as many cores. The scaling is not linear as Hadoop jobs also have task startup costs: "map time" is overlapped



(a) Effect of input storage, concurrency, heartbeat, and heap optimizations on FindUserUsage



(b) Effect of memory and shuffle optimizations on 10 GB TeraSort

Figure 5: Effect of different optimizations on FindUserUsage and 10 GB TeraSort

both with startup and with shuffle and reduce.

TeraSort is clearly shuffle-dominated on the 8-node cluster. For the 16-node cluster shuffle and map time appear approximately equal; however this is an artifact of our methodology where the overlap between the two is counted as "map time". In reality both shuffle and map time are reduced as the larger cluster has more cores as well as more bisection bandwidth, but the overall runtime remains shuffle-dominated. In the scale-up case we see that the runtime is clearly map-dominated, and the shuffle phase is extremely efficient.

4.5 Effect of optimizations

The results in the previous section used Hadoop setups that were fully optimized, i.e. all the relevant optimizations described in Section 3 were applied. In this section we look at the effect of each optimization individually. The aim is to understand the effect of the optimizations on scale-up performance, compared to a vanilla Hadoop running in pseudo-distributed mode.

We examine two jobs from our benchmark suite. FindUserUsage is map-intensive with little intermediate data and a small reduce phase. We use it to measure all optimizations except those that improve the shuffle phase, since the shuffle phase in FindUserUsage is very small. TeraSort is memory- and shuffle- intensive. Thus we use it to measure the effect of the memory and shuffle-phase optimizations. In all cases we use the scale-up server described previously.

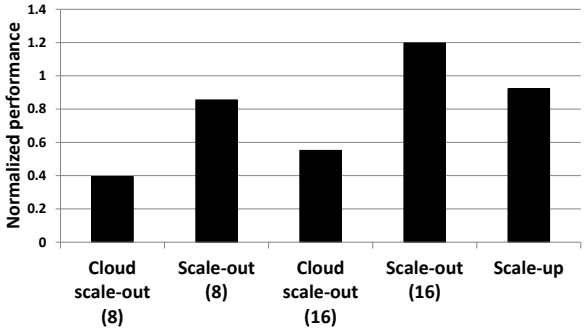


Figure 6: Relative performance of cloud and single-node scale-up with a scalable storage back-end

Figure 5(a) show the effect on FindUserUsage's execution time of successively moving from disk-based HDFS to an SSD-based local file system; of optimizing the number of mappers and reducers; of removing out-of-band heartbeats; and of optimizing the heap size. Each has a significant impact, with a total performance improvement of 4x.

Figure 5(b) shows the effect on execution time of 10 GB TeraSort starting from a baseline where the storage, concurrency, and heartbeat optimizations have already been applied. The heap size optimization has a significant effect, as input and intermediate data buffers are spilled less frequently to the file system. Moving to the file system based rather than http based shuffle and unthrottling the shuffle has an even bigger effect. Finally, using a RAMdisk for that intermediate data improves performance even further. For TeraSort, heap size optimization improves performance by 2x and shuffle-phase optimizations by almost 3.5x, for a total performance improvement of 7x.

5 Discussion

In the previous section we evaluated 11 jobs to show that, for real-world jobs and job sizes, scale-up is competitive on performance and performance/$, and superior on performance per watt and per rack unit. Here we consider implications for cloud computing and then the limits of scale-up.

5.1 Scale up vs. scale-out in the cloud

Our analysis so far was based on a private cluster and scale-up machine. As many analytic jobs including Hadoop jobs are now moving to the cloud, it is worth asking, how will our results apply in the cloud scenario? The key difference from a private cluster is that in the cloud, HDFS is unlikely to be used for input data and the compute nodes, at least today, are unlikely to use SSD storage. Instead, input data is read over the network from a scalable storage back end such as Amazon S3 or Azure Storage. Intermediate data will be stored in memory and/or local disk.

We re-ran the FindUserUsage job using Azure compute nodes to scale out and the Azure Storage back end to hold input data. We used extra-large instances (8 cores, 16 GB of RAM) as being the closest to our scale-out workstation machines. This is the largest size offered by Azure, and still only comparable to our scale-out rather than our scale-up machine. Figure 6 shows the performance of both the cloud and the private cluster scale-out configurations, normalized to that of the standalone (SSD-based) scale-up configuration. Performance in the cloud is significantly worse than in the private cluster; however even in the best case, we would expect it to match but not exceed that of the private cluster.

More importantly, the scale-up configuration when reading data from the network comes close (92%) to that of a scale-up configuration when using SSDs. Hence we believe that with a 10 Gbps NIC, and a scalable storage back-end that can deliver high bandwidth, our results from the private cluster also hold for the cloud.

As high-bandwidth networking hardware becomes cheaper (40 Gbps is already on the market and 100 Gbps on the horizon) it will be more cost-effective to add bandwidth to a single scale-up node than to each of many scale-out nodes. We thus expect that cloud "scale-up" instances (which are already being offered for applications such as HPC and large in-memory databases) will be a cost-effective way to run the majority of Hadoop jobs with our optimizations to Hadoop.

5.2 Limitations to scale-up

Our results show that, contrary to conventional wisdom, scale-up hardware is more cost-effective for many real-world jobs which today use scale-out. However, clearly there is a job size beyond which scale-out becomes a better option. This "cross-over" point is job-specific. In order to get some understanding of this crossover point, we use TeraSort as an example since its job size is easily parametrized.

Figure 7 shows the results, again normalized to set the values for scale-up at 1. All three of our test configurations are able to sort up to 50 GB of data, with the scale-up configuration providing by far the best performance per dollar. However the 8-node cluster cannot sort 100 GB of data without running out of storage, and the scale-up machine cannot sort significantly more than 100 GB without running out of memory. At 100 GB, scale-up still provides the best performance/$, but the 16-node cluster is close at 88% of the performance/$ of scale-up.

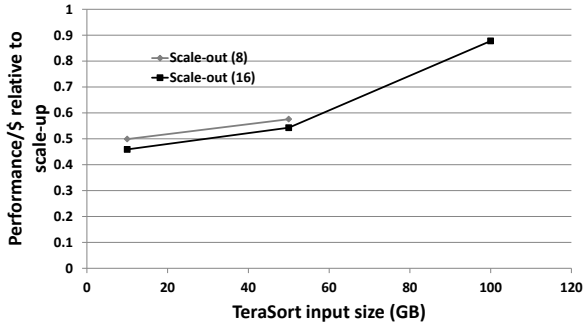


Figure 7: TeraSort performance/$ normalized to that of scale-up as a function of input size

These results tell us two things. First, even with "big memory", the scale-up configuration can become memory-bound for large jobs. However we would expect this point to shift upwards as DRAM prices continue to fall and multiple terabytes of DRAM per machine become feasible. Second, for TeraSort, scale-out begins to be competitive at around the 100 GB mark with current hardware.

Given that most job sizes are well under 100 GB and based on the results in Section 4 we believe that for many jobs, scale-up is a better option. However, some jobs still perform better with scale-out. Also, clearly Hadoop and MapReduce are also used for larger jobs for which scale-out is the best or the only option. Thus it is important to support both scale-up and scale-out transparently. While it is feasible to provision a data center with a mix of large and small machines, it is not desirable to maintain two versions of each application. By making all our changes transparently "under the hood" of Hadoop, we allow the decision of scale-up versus scale-out to be made at runtime.

We note that in theory it is also possible to scale out using large machines, i.e. combine scale-up and scale-out, and hence use fewer machines (e.g. 2 instead of 32). However this would sacrifice the 4x performance benefit of our shuffle optimizations (using the local FS, unrestricted shuffle, and a RAMdisk for intermediate data) while also losing the pricing advantage of the low-end machines. Thus at current price points it seems that the choice is effectively between scale-up on a large machine

versus scale-out on a large number of low-end machines.

The optimal choice depends both on job characteristics and input job sizes. This crossover point is job-specific and currently, the decision whether to scale out or up for a given job must be made manually at job start time. We are now working on ways to automate this decision based on profiling and static analysis.

6 Related Work

One of the motivations for this work was the observation that most analytic job sizes, including Hadoop job sizes, are well within the 512 GB that is feasible on a standard "scale-up" server today. We collected information about job sizes internally, where we found median job sizes to be less than 14 GB, but our conclusions are also supported by studies on a range of real-world Hadoop installations including Yahoo [10], Facebook [4, 7], and Cloudera [7].

Piccolo [14] is an in-memory distributed key-value store, aimed at applications that need low-latency fine-grained random access to state. Resilient Distributed Datasets (RDDs) [19] similarly offer a distributed memory like abstraction in the Spark system, but are aimed at task-parallel jobs, especially iterative machine learning jobs such as the Mahout jobs considered in this paper. Both of these are "in-memory scale-out" solutions: they remove the disk I/O bottleneck by keeping data in memory. However they still suffer from the network bottleneck of fetching remote data or shuffling data in a MapReduce computation. Our contribution is to show that scale-up rather than scale-out is a competitive option even for task-parallel jobs (both iterative and non-iterative) and can be done with transparent optimizations that maintain app-compatibility with Hadoop.

Metis [11] and Phoenix [15] are in-memory multi-core (i.e. scale-up) optimized MapReduce libraries. They demonstrate that a carefully engineered MapReduce library can be competitive with a shared-memory multi-threaded implementation. In this paper we make a similar observation about multi-threaded vs. MapReduce in the Hadoop context. However our distinct contribution is that we provide good scale-up performance transparently for Hadoop jobs; and we evaluate the tradeoffs of scale-up vs. scale-out by looking at job sizes as well as performance, dollar cost, power, and server density.

In previous work [17] we showed that certain machine learning algorithms do not fit well within a MapReduce framework and hence both accuracy and performance were improved by running them as shared-memory programs on a single scale-up server. However this approach means that each algorithm be implemented once for a multi-threaded shared-memory model and again for MapReduce if scale-out is also desired. Hence in this paper we demonstrate how scale-up can be done transparently for Hadoop applications without sacrificing the potential for scale-out and without a custom shared-memory implementation. We believe that while custom multi-threaded implementations might be necessary for certain algorithms, they are expensive in terms of human effort and notoriously hard to implement correctly. Transparent scale-up using Hadoop is applicable for a much broader range of applications which are already written to use Hadoop MapReduce.

The tradeoff between low-power cores and a smaller number of server-grade cores was extensively studied by Reddi et al. [16] in the context of web search. Although this is a different context from that of our work, they reach similar conclusions: scaling to a larger number of less capable cores is not always cost-effective even for highly parallel applications. Similarly, recent work [2] shows that for TPC-H queries, a cluster of low-power Atom processors is not cost-effective compared to a traditional Xeon processor. In general, the scale-up versus scale-out tradeoff is well-known in the parallel database community [9]. A key observation is that the correct choice of scale-up versus scale-out is workload-specific. However in the MapReduce world the conventional wisdom is that scale-out is the only interesting option. We challenge this conventional wisdom by showing that scale-up is in fact competitive on performance and cost, and superior on power and density, for a range of MapReduce applications.

7 Conclusion

In this paper, we showed that, contrary to conventional wisdom, analytic jobs — in particular Hadoop MapReduce jobs — are often better served by a scale-up server than a scale-out cluster. We presented a series of transparent optimizations that allow Hadoop to deliver good scale-up performance, and evaluated our claims against a range of Hadoop jobs from different application domains.

Our results have implications for the way Hadoop and analytics clusters in general are provisioned, with scale-up servers being a better option for the majority of jobs. Providers who wish to support both scale-up and scale-out will need an automated way to predict the best architecture for a given job. We are currently working on such a predictive mechanism based on input job sizes and static analysis of the application code.

Our results also imply that software infrastructures such as Hadoop must in future be designed for good scale-up as well as scale-out performance. The optimizations presented in this paper provide a good initial starting point for improving scale-up performance. We are considering further optimizations: exploiting

shared memory for zero-copy data exchange between phases; changing the intermediate file format to avoid (de)serialization and checksumming; and using a customized grouping implementation for map outputs instead of the standard sort-based grouping.

References

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

[2] Wimpy Node Clusters: What About Non-Wimpy Workloads? In *Workshop on Data Management on New Hardware (DaMon)*, Indianapolis, IN, June 2010.

[3] Amazon Simple Storage Service (Amazon S3). `http://aws.amazon.com/s3/`. Accessed: 08/09/2011.

[4] G. Ananthanarayanan, A. Ghodsi, AndrewWang, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica. PACMan: Coordinated memory caching for parallel jobs. In *NSDI*, Apr. 2012.

[5] Windows Azure Storage. `http://www.microsoft.com/windowsazure/features/storage/`. Accessed: 08/09/2011.

[6] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. F. u. Haq, M. I. u. Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas. Windows Azure Storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 143–157, New York, NY, USA, 2011. ACM.

[7] Y. Chen, S. Alspaugh, and R. H. Katz. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. *PVLDB*, 5(12):1802–1813, 2012.

[8] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, 2004.

[9] D. DeWitt and J. Gray. Parallel database systems: The future of high performance database systems. *Communications of the ACM*, 35(6):85–98, June 1992.

[10] K. Elmeleegy. Piranha: Optimizing short jobs in Hadoop. In *under submission*, 2012.

[11] Y. Mao, R. Morris, and F. Kaashoek. Optimizing MapReduce for multicore architectures. Technical Report MIT-CSAIL-TR-2010-020, 2010.

[12] S. Owen, R. Anil, T. Dunning, and E. Friedman. *Mahout in Action*. Manning Publications Co., 2011.

[13] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data*, pages 165–178, New York, NY, USA, 2009. ACM.

[14] R. Power and J. Li. Piccolo: Building fast, distributed programs with partitioned tables. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Vancouver, BC, Oct. 2010.

[15] C. Ranger, R. Raghuraman, A. Penmetsa, G. R. Bradski, and C. Kozyrakis. Evaluating MapReduce for multi-core and multiprocessor systems. In *HPCA*, 2007.

[16] V. J. Reddi, B. C. Lee, T. M. Chilimbi, and K. Vaid. Web search using mobile cores: Quantifying and mitigating the price of efficiency. In *Proc. 37th International Symposium on Computer Architecture (37th ISCA'10)*, pages 314–325, Saint-Malo, France, June 2010.

[17] A. Rowstron, D. Narayanan, A. Donnelly, G. O'Shea, and A. Douglas. Nobody ever got fired for using Hadoop. In *Workshop on Hot Topics in Cloud Data Processing (HotCDP)*, Bern, Switzerland, Apr. 2012.

[18] A. P. Wiki. `http://wiki.apache.org/pig/PigPerformance`.

[19] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, Apr. 2012.