Decoupling Storage and Computation in Hadoop with SuperDataNodes

George Porter
UC San Diego
La Jolla, CA 92093
gmporter@cs.ucsd.edu

Abstract

The rise of ad-hoc data-intensive computing has led to the development of data-parallel programming systems such as Map/Reduce and Hadoop, which achieve scalability by tightly coupling storage and computation. This can be limiting when the ratio of computation to storage is not known in advance, or changes over time. In this work, we examine decoupling storage and computation in Hadoop through *SuperDataNode*s, which are servers that contain an order of magnitude more disks than traditional Hadoop nodes. We found that SuperDataNodes are not only capable of supporting workloads with high storage-to-processing workloads, but in some cases can outperform traditional Hadoop deployments through better management of a large centralized pool of disks.

1 Introduction

Recently, there has been a rapid growth in ad-hoc *data intensive computing*, which is computing over very large, unstructured datasets. Several novel data-parallel programming systems have been developed to attack this problem, including Hadoop[2], Map/Reduce[3], and DryadLINQ[12]. Hadoop scales by allocating and moving computation efficiently near the data. Hadoop storage nodes run *DataNode* processes, which store portions of the Hadoop Distributed FileSystem (HDFS) on their local disks. A single *NameNode* manages the HDFS metadata, and determines which data blocks are located on each DataNode. Similarly, each worker node hosts a *TaskTracker* process that executes part of each Hadoop job. A single *JobTracker* coordinates processing across the TaskTrackers. Each Hadoop node serves both as a storage node and worker node, and the Hadoop scheduler tries to ensure that tasks run on a node with the storage it needs.

Thus, Hadoop achieves scale by harnessing many nodes that contribute both storage and computation. There are a variety of benefits to this approach. First, Hadoop is able to process a portion of the data in parallel on each node, leading to very high scalability. Second, scaling independent nodes provides for storage and computing fault tolerance and resiliency, since higher-layer replication can mask individual node failures. Third, Hadoop relies on commodity server nodes and networking fabrics,

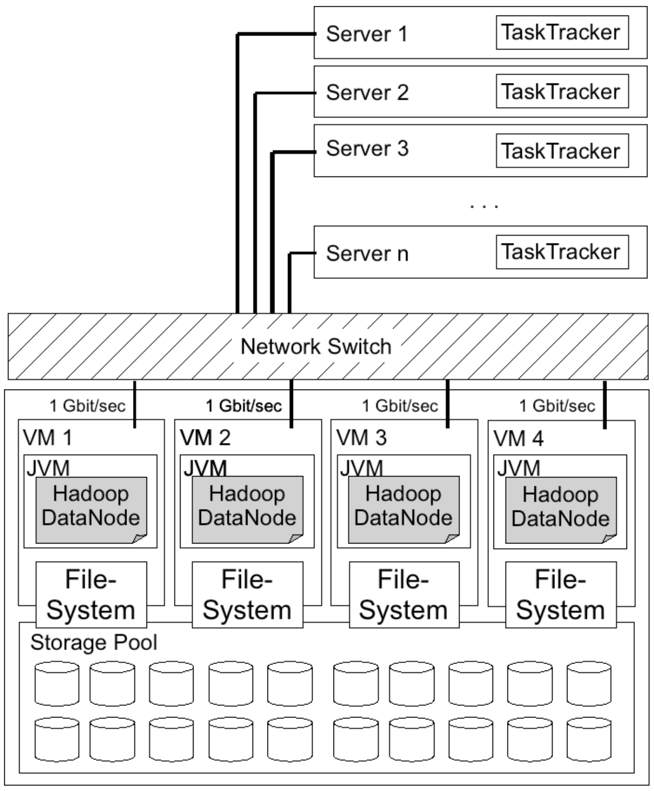

Figure 1: A SuperDataNode consists of several dozen disks forming a single storage pool. From this pool, several filesystems are built, each one imported into a virtual machine running an unmodified copy of the Hadoop DataNode process. Each VM is assigned its own network interface (if 1 Gbit/sec links are used), or a portion of a network interface (if 10 Gbit/sec links are used), and its own IP address.

reducing the cost of deployment considerably. Despite the advantages of Hadoop, its tight coupling of storage and computation has some limitations, of which we highlight two as motivation for this work. The first limitation is that the ratio of computation to storage might change over time, or might not be known in advance. This can result in having more data than the storage was initially provisioned for. A special case of this is "archival Hadoop," which is when part of the provisioned storage is infrequently accessed, but must still be available for processing. The second limitation is that when the workload varies, it might be desirable to power down or re-purpose some of the Hadoop nodes for other applications. Unfortunately, Hadoop spreads data over all of the nodes, and so that data would first have to be migrated elsewhere, which can be very time consuming. Offloading a single terabyte off of a typical disk over a gigabit link takes approximately three hours. This second limitation is particularly relevant for Cloud computing environments, where if data is put directly on the nodes, they can not be released during periods of low demand. On the other hand, accessing data from infrastructure services like Amazon's S3[4] means losing data locality entirely. We would prefer a third approach that decouples storage from computation while maintaining some data locality.

To address these limitations, we propose a new type of Hadoop node called a SuperDataNode. A SuperDataNode is simply a node with an order of magnitude more disks (and thus storage) than a traditional Hadoop DataNode. Using this storage are a number of virtual machines, each running unmodified Hadoop DataNode processes that efficiently utilizes the vast amount of storage present in a way that is fully backwards compatible with traditional Hadoop, and can effectively take advantage of high bandwidth network interfaces present on the SuperDataNode. SuperDataNodes have several advantages: they decouple the amount of storage from the number of nodes, they support processing archival data, they lead to more uniformity for job scheduling and datablock placement, and they ease the management of clusters hosting both Hadoop and non-Hadoop applications. SuperDataNodes do have some limitations, including a need for high-bandwidth connectivity within a datacenter rack, a possible reduction in fault tolerance due to consolidation of storage into a single node, and a higher cost than traditional Hadoop nodes. However, we will show that these limitations are surmountable. We have deployed a prototype of a SuperDataNode in a Hadoop cluster, and found that SuperDataNodes are not only capable of supporting workloads with high storage-to-processing workloads, but in some cases actually outperform traditional Hadoop deployments through better management of a large centralized pool of disks. For example, compared to traditional Hadoop, the use of a SuperDataNode reduced total job execution time of a Sort workload by 17%, and a Grep workload by 54%.

2 A Case for SuperDataNodes

We now present the design of our storage-rich Hadoop SuperDataNodes. We discuss the advantages of adopting this alternative architecture for Hadoop storage as well as its limitations. SuperDataNodes represent a fundamentally new unit of scaling different from traditional Hadoop, and so in Section 3 we discuss how large-scale datacenter environments can scalably incorporate SuperDataNodes.

2.1 Design

Figure 1 shows the design of a SuperDataNode. A SuperDataNode is a standard server with two exceptions: a much richer than average storage layer consisting of an order of magnitude more disks (and thus storage) than is typically present in a standard DataNode, and a large amount of aggregate bandwidth to the network. For example, in our deployment, each SuperDataNode has 48 disks organized across six disk interfaces, and four gigabit network links providing 4 Gbit/sec total bandwidth to the top-of-rack network switch. The type and size of each disk is comparable to a traditional Hadoop node, e.g., a 1 TB SATA disk. Each SuperDataNode also hosts a set of virtual machines, one for each network interface (with its own IP address). Executing in the VM is a standard, unmodified copy of the Hadoop DataNode process. We chose to host multiple Hadoop DataNodes in their own VMs, rather than executing a single large DataNode process to provide fine-grained control over the large disk pool and available memory.

The disks are organized into a single storage pool, and each VM mounts its storage from that pool. The shared storage pool serves as a central point for operating system optimization of the concurrent and interleaved disk operations issued by the virtual DataNodes. As we will see in Section 4, this provides an opportunity for optimizations across DataNode disk accesses that is not possible in traditional, shared-nothing Hadoop deployments. Relying on virtual DataNodes means that the switch does not require any support for Ethernet-level link aggregation– each virtual DataNode will appear to the rest of the Hadoop infrastructure on its own IP address. Sharing the same switch as the SuperDataNode are a set of standard worker nodes, each running unmodified TaskTracker processes (though not any DataNode processes). The TaskTrackers running on these nodes execute map and reduce tasks on behalf of jobs as usual. When the HDFS client running within those tasks requires block access, it will contact the NameNode to receive a list of (in our case virtual) DataNodes responsible for those blocks. The client will then connect to the appropriately chosen virtual DataNode to obtain the block. Our approach balances datablock storage and retrieval traffic over the network interfaces in the SuperDataNode by simply relying on HDFS's random block placement algorithm.

2.2 Effect on replication

Hadoop relies on block-level replication for fault tolerance both at the disk- and node-level, resulting in an N-times inflation for a replication factor of N. Alternative approaches to masking disk failure, such as RAID coding of blocks across disks, are not feasible across DataNodes due to network latency. By consolidating disks from numerous virtual DataNodes into one physical system, such coding techniques are not only feasible, but can be done at a level below the virtual machine monitor, remaining transparent to Hadoop itself. This should reduce the storage requirements due to fault-tolerance. SuperDataNodes are still susceptible to node failures, and so block-level replication will still be required, however the level of replication will be less than simple N-level replication.

2.3 Advantages

Adopting a SuperDataNode approach to storage in Hadoop provides several benefits:

1. SuperDataNodes decouple the *amount of storage* in HDFS from the *number of nodes* making up the HDFS deployment.

Traditionally HDFS deployments providing N bytes of capacity require $M = \lceil N/c \rceil$ nodes, where c is the average amount of storage in each DataNode. These M nodes must run at all times, even when not used, since although HDFS is resilient to individual node and switch failures, powering down or removing a large number of nodes can result in data loss unless the replication factor is set unusually high. Furthermore, HDFS reacts to node loss or departure by attempting to re-replicate the blocks that node was responsible for elsewhere, making dynamic adjustments to the number of HDFS nodes impractical on short timescales. This impedes the ability to power off nodes or re-purpose them to run other applications during periods of low utilization. With our approach, a SuperDataNode can stay running while all of the TaskTracker nodes are powered down or retasked for other applications.

2. Support for archival data

Traditional Hadoop aims to provide high bandwidth access to all of the data stored within it. However, over time some deployments may access some data more infrequently than the rest. SuperDataNodes are a good fit for consolidating that archival data. TaskTrackers can perform both archival and non-archival jobs: the only difference is whether they retrieve datablocks locally from their disks, or remotely from a SuperDataNode. We envision that new support will need to be added to the Hadoop NameNode to allow users to signify (or for it to learn by introspection) that some data should be marked as archival and migrated to a SuperDataNode.

3. Increased uniformity for Job scheduling and datablock placement

One of the challenges to scheduling in Hadoop is choosing appropriate nodes to execute tasks on based on data locality. With SuperDataNodes, any rack-local TaskTracker is an equally good candidate for scheduling a given task. Furthermore, since the virtual DataNode processes running in the SuperDataNode share the same underlying storage pool, the storage pool has more flexibility to centrally manage disk requests from different TaskTrackers that would have otherwise been on different nodes.

4. Ease of management

Consolidating storage into SuperDataNodes provides several possible improvements to system management. The first is that since the TaskTracker nodes are no longer data-bound, they can be provisioned on much smaller timescales than traditional Hadoop, leading to better support for deployments with both Hadoop and non-Hadoop applications. Additionally, small non-Hadoop clusters can be extended to support Hadoop by adding a SuperDataNode.

2.4 Limitations

The use of SuperDataNodes imposes some limitations that we now highlight:

1. Storage bandwidth between SuperDataNodes and TaskTrackers is a scarce resource

The performance of TaskTrackers in Hadoop is dominated by their ability to obtain high-throughput access to storage. With SuperDataNodes, TaskTrackers compete for network bandwidth between each other (to transfer intermediate results) and between themselves and the SuperDataNode. A single gigabit network link (with approximately 100 MB/sec capability) can support the equivalent of $\lfloor 100/M \rfloor$ local disks if each operates at M MB/sec on average. Thus, a SuperDataNode with N gigabit links can support the equivalent of $N\lfloor 100/M \rfloor$ local disks. Thus, if the SuperDataNode has a 10 Gbit/sec interface, it will be limited to approximately twenty local disks worth of bandwidth if each disk supports 50 MB/sec average throughput. As we discuss in Section 3, we expect that the bandwidth within a single rack will grow faster than inter-rack bandwidth, and so scheduling jobs to be rack-local with SuperDataNodes they access will help overcome this bandwidth limitation.

2. Effect on fault tolerance

One of the benefits of HDFS is its fault tolerance in the presence of individual node failures. Consolidating many nodes' worth of storage into a single SuperDataNode means that if it fails, the result is significantly worse than a traditional Hadoop DataNode failure. In fact, since each SuperDataNode relies on virtualization to export multiple traditional DataNode servers, a failure of one SuperDataNode will introduce correlated failures into HDFS. This is a situation that we need to more extensively test against.

A feasible way to overcome this limitation is to rely on the use of disk-level redundancy within the SuperDataNode. A sublinear coding approach, e.g., RAID-5, can be used in a SuperDataNode since that redundancy can be amortized over a large number of centrally located disks. With traditional DataNodes, redundancy must take the form of replicating entire blocks to different nodes, since any scheme based on computing disk parity is infeasible when the disks are in different DataNodes. We still require at least one block replica outside of the rack to mask SuperDataNode and switch failures. However, a replication factor of 2 (one off rack, and one on the RAID storage in the SuperDataNode) should mask both disk and switch failures using fewer disks than traditional Hadoop.

3. Cost of SuperDataNodes

One advantage of traditional Hadoop is that its scale relies on increasing the number of inexpensive commodity servers. SuperDataNodes cost significantly more than traditional nodes because of their larger memory and disk footprint, however each one is built from otherwise commodity components and operating systems. This increased cost could be offset in two ways. First, the introduction of SuperDataNodes could enable already deployed, smaller clusters to run Hadoop workloads without buying new equipment. Second, the ability to turn off or re-provision TaskTracker nodes without disrupting the underlying HDFS filesystem might provide opportunities for power savings.

3 Scaling SuperDataNodes in the Cloud

Scaling Hadoop with SuperDataNodes in dynamic datacenter environments such as Cloud Computing deployments operates differently than scaling traditional Hadoop deployments. First, since we anticipate much greater intra-rack verses inter-rack bandwidth, it is important that the SuperDataNode share a rack with the TaskTrackers that will operate on it. Furthermore, the ratio of TaskTrackers to each SuperDataNode must be limited based on the equivalent disk bandwidth that can be achieved over the network. Thus, between a 10-to-1 or 20-to-1 ratio of SuperDataNodes to TaskTrackers seems ideal. In the case of archival Hadoop, it may be desirable to oversubscribe that ratio, trading off lower bandwidth to each TaskTracker with the fact that the data will be infrequently accessed.

4 Evaluation

We now describe our evaluation strategy, highlighting the performance impact of adopting SuperDataNodes. We found that SuperDataNodes decreased Hadoop job execution time up to 54% compared to traditional Hadoop

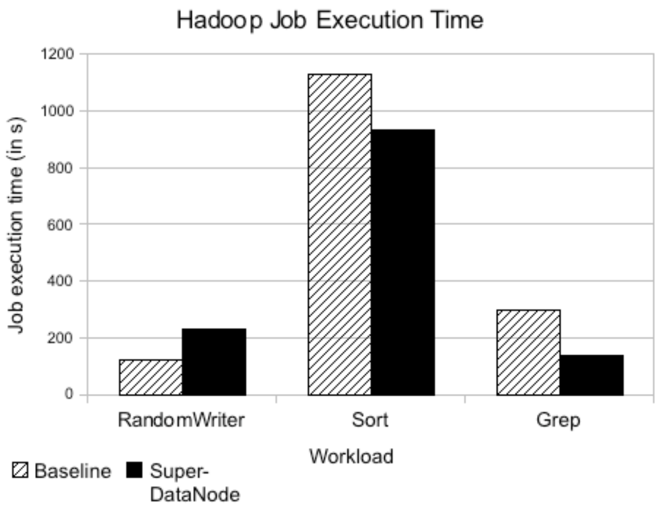

Figure 2: Comparison of the total job execution times of three canonical Hadoop workloads

for some workloads. We do not intend for these results to show that SuperDataNodes will perform better or worse than traditional Hadoop–only that the performance results are comparable given its other benefits. There are workloads for which the SuperDataNode performs worse, and we highlight those as well.

4.1 Experimental Setup

The TaskTrackers in our Hadoop deployment consist of ten SunFire^{TM}X4150 Servers running OpenSolaris^{TM}, each with 8 GB of memory and four 146GB SAS disk drives. One of the disks is dedicated to the operating system and for storing intermediate data from the TaskTrackers. For the baseline measurements, two of the drives are made available to a DataNode process running on each node. For the SuperDataNode measurements, those two disks are unused, and no DataNode process runs on the nodes. We deployed Hadoop 0.19 with 128MB blocks. For our SuperDataNode, we used a SunFire^{TM}X4540 Server (a successor to the "Thumper") configured with 64 GB of memory and 48 500GB SATA drives. The SuperDataNode has four gigabit network interfaces connected to the same switch as the rest of our Hadoop cluster. We used OpenSolaris^{TM}Zones for each DataNode VM, and configured only twenty of the disks as a single ZFS^{TM}storage pool, so that our baseline and SuperDataNode measurements would have the same number of disks allocated to HDFS for an equal comparison. Both our baseline and experimental results use the ZFS filesystem. Our SuperDataNode draws approximately 1,200 Watts of power.

4.2 Hadoop Performance Results

We began by deploying three different canonical Hadoop workloads on both the baseline and SuperDataNode deployments. The first workload is a *RandomWriter* job that generates 60GB of random input data into HDFS, and forms the basis of the second job which is the Sort example included with Hadoop. The third workload is a simple Grep job that searches for the keyword *Hamlet* in a 34GB text file of repeated Shakespeare plays. Figure 2 shows that in both the Sort and Grep workloads, the SuperDataNode-based deployment performed better than traditional Hadoop. With Sort, the execution time was 17% less, and with Grep it was 54% less than the baseline. In the case of the RandomWriter workload, which involves the least amount of computation (and thus is entirely weighted toward raw access to storage), the reverse was true, and the latency of the SuperDataNode approach was 92% larger than baseline. These results show that the performance impact of SuperDataNodes is workload dependent, and that achieving the benefits of SuperDataNodes does not necessarily have to incur a performance penalty.

4.3 Measuring storage pool latencies with X-Trace

Next, we wanted to see why using a SuperDataNode resulted in better performance for the Sort and Grep workloads. Our hypothesis was that consolidating all of the DataNode storage into a single node would give that node the ability to better schedule, manage, and optimize the performance of the aggregate storage workload, simply because it had visibility (in our case) to ten times as many disk operations.

To examine this hypothesis, we instrumented HDFS with X-Trace[6], a cross-layer datapath tracing framework. Our instrumentation API captures relevant events that span the TaskTracker, HDFS client, and DataNode process, including the latencies of disk block read and write operations. These latencies isolate the time required to fetch or store data onto the DataNode's filesystem, excluding network latency, TaskTracker think time, and non-DataNode disk subsystem latencies. Figures 3 and 4 show a cumulative distribution function of block read and write latencies for the Sort workload, and Figure 5 shows the read block latencies for the Grep workload (there were too few blocks written to generate a graph for the write portion).

We see that in the case of the Sort workload, the read latency within the SuperDataNode from the shared storage pool was larger than the aggregate read latency of individual DataNodes residing in the traditional cluster. However, the write performance was significantly better, as shown in Figure 4, leading to a reduced total job execution time. In the case of the Grep workload, the read performance for the SuperDataNode was significantly better

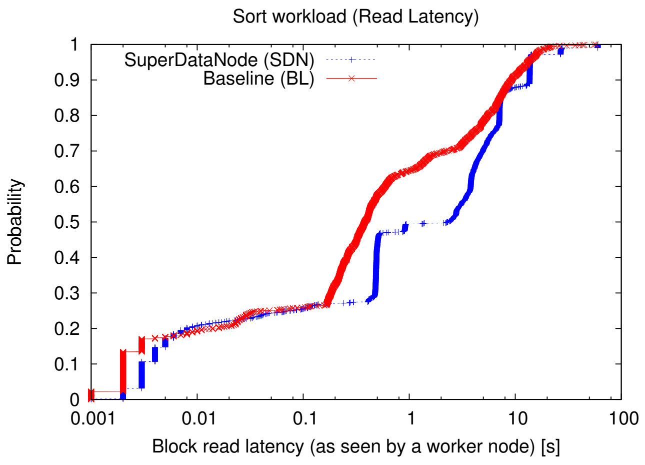

Figure 3: Read block latencies for the Sort workload

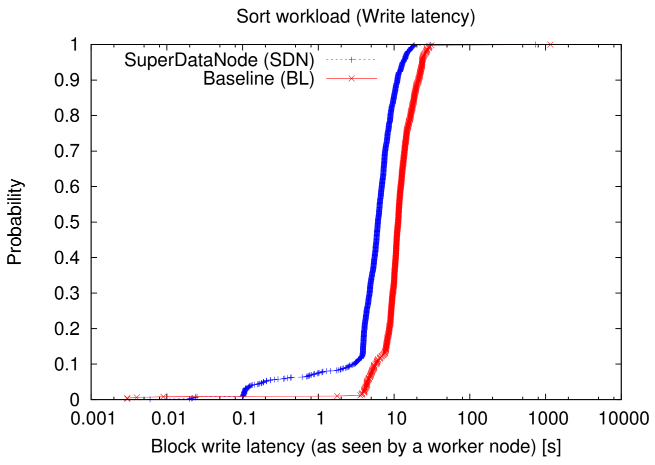

Figure 4: Write block latencies for the Sort workload

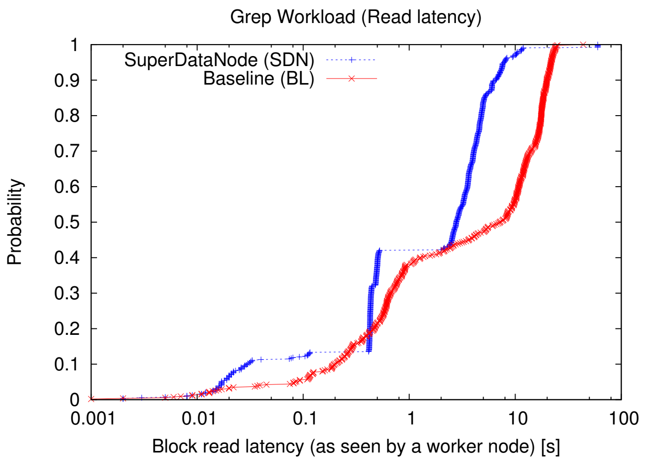

Figure 5: Read block latencies for the Grep workload

than traditional Hadoop, and since there was so few data blocks written, the total job execution time for the SuperDataNode case was nearly half that of the traditional case. We plan to better instrument the filesystem during read and write block operations to determine exactly what features of the underlying filesystem and storage pool led to these differences, however for now we can say that the benefit is workload dependent, and that our initial experience integrating SuperDataNodes into typical clusters is promising.

5 Related work

Balancing storage and computation is not a new problem, nor specific to data-intensive computing. Jim Gray outlines the economics of tradeoffs between storage, networking, and computation, and highlights the benefits of moving computation near the data over which it operates[7]. One such balance was used by Yahoo on Jim Gray's sort benchmark[1], and required 3,800 nodes. An interesting counterpoint discussion to this configuration led by Joseph M. Hellerstein is available at [11].

Map/Reduce is quickly gaining adoption, and is increasingly being used in the cloud. Amazon now supports Map/Reduce as a service[9], and it is in fact the only application that they will provision directly for EC2 users. Our SuperDataNode proposal relies on high bandwidth within racks, and for TaskTrackers to prefer those data blocks over off-rack locations. Hadoop now supports this rack affinity[8]. The application of SuperDataNodes to supporting archival workloads is similar to that taken by the SAM/QFS storage system[10]. While we do not evaluate our approach to power savings, we note that power utilization is a key concern for datacenter operators, and typical approaches at large scale rely on maximally utilizing the provisioned resources, rather than powering down some of the datacenter during periods of low demand[5].

6 Conclusions

Hadoop and Map/Reduce represent an increasingly important approach to data-intensive computing. In this work, we explore the advantages and limitations of decoupling storage from computation in Hadoop through SuperDataNodes. SuperDataNodes consolidate a number of traditional DataNodes into a single, storage-rich node, providing more agility in terms of scaling the amount of computation applied to data. Although not intended to replace traditional, dedicated Hadoop clusters, two applications for which SuperDataNodes show promise are extending Hadoop into pre-existing clusters shared with non-Hadoop applications, and for supporting Map/Reduce over archival data.

Acknowledgments

We would like to thank Hsianglung Wu for providing and setting up the Hadoop testbed, and Matei Zaharia, Rodrigo Fonseca, Steve Heller, and Harsha Madhyastha for their invaluable feedback.

References

[1] Yahoo Developer Blog. `http://developer.yahoo.net/blogs/hadoop/2009/05/hadoop_sorts_a_petabyte%_in_162.html`.

[2] Hadoop Core. `http://hadoop.apache.org/core`.

[3] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, Berkeley, CA, USA, 2004. USENIX Association.

[4] Amazon EC2 and S3. `http://aws.amazon.com`.

[5] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 13–23, New York, NY, USA, 2007. ACM.

[6] Rodrigo Fonseca, George Porter, Randy H. Katz, Scott Shenker, and Ion Stoica. X-trace: A pervasive network tracing framework. In *NSDI*. USENIX Association, Cambridge, MA, 2007.

[7] Jim Gray. Distributed computing economics. *Queue*, 6(3):63–68, 2008.

[8] Rack Aware Placement JIRA Issue. `http://issues.apache.org/jira/browse/HADOOP-692`.

[9] Amazon Elastic Map/Reduce. `http://aws.amazon.com/elasticmapreduce`.

[10] The SAM/QFS Storage System. `http://www.opensolaris.org/os/project/samqfs`.

[11] Prof. Joseph M. Hellerstein *DataBeta* Blog. `http://databeta.wordpress.com/2009/05/14/bigdata-node-density`.

[12] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, lfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In Richard Draves and Robbert van Renesse, editors, *OSDI*, pages 1–14. USENIX Association, 2008.