The Datacenter Needs an Operating System

Matei Zaharia, Benjamin Hindman, Andy Konwinski, Ali Ghodsi,
Anthony D. Joseph, Randy Katz, Scott Shenker, Ion Stoica
University of California, Berkeley

1 Introduction

Clusters of commodity servers have become a major computing platform, powering not only some of today's most popular consumer applications—Internet services such as search and social networks—but also a growing number of scientific and enterprise workloads [2]. This rise in cluster computing has even led some to declare that "the datacenter is the new computer" [16, 24]. However, the tools for managing and programming this new computer are still immature. This paper argues that, due to the growing diversity of cluster applications and users, the datacenter increasingly needs an operating system.[1]

We take a broad view of an operating system as both a software layer that manages and abstracts hardware and a package of tools, such as programming languages and debuggers, that facilitate the use of a computer. Traditional OSes for a single computer play several key roles. First, they enable *resource sharing* between programs and users, allowing a computer to be used for multiple tasks and to respond interactively to several users. Second, they enable *data sharing* between programs, through abstractions such as pipes and files, so that users can combine independently written applications to solve problems. Third, they provide *programming abstractions* that simplify software development. Finally, OSes include system-wide *debugging and monitoring* facilities. Together, these features have allowed single-computer operating systems to foster a rich ecosystem of interoperable applications that we now take for granted.

Much like the first computers, the first datacenters ran a small number of applications written by expert users. This situation is changing now, and we believe that datacenters need an OS-like layer for the same reason computers did: a rising diversity of applications and users.

On the application side, there is an increasing number of cluster computing frameworks (*e.g.,* MapReduce [10], Dryad [17], Pregel [21]) and storage systems [12, 9, 14], each suited for different use cases. Ideally, an organization should be able to dynamically share resources between these applications, and to easily exchange data between them (*e.g.,* have a job that combines MapReduce and Pregel steps). However, this is currently difficult because applications are written independently, with no common interfaces for accessing resources and data.

In addition, clusters are serving increasing numbers of concurrent users, which require responsive time-sharing. For example, while MapReduce was initially used for a small set of batch jobs, organizations like Facebook are now using it to build data warehouses where hundreds of users run near-interactive ad-hoc queries [29].

Finally, programming and debugging cluster applications remains difficult even for experts, and is even more challenging for the growing number of non-expert users (*e.g.,* scientists) starting to leverage cloud computing.

While cluster users are well-aware of these problems, current solutions are often ad-hoc. For example, in the Hadoop stack [3], MapReduce acts as a common execution layer on which higher-level programming interfaces like Pig and Hive [22, 4] are built. The MapReduce scheduler provides resource sharing, but this means that only jobs that compile down to MapReduce can run on the cluster. Unfortunately, MapReduce is too high level of an abstraction to support all applications efficiently.[2] Similarly, many open source cluster storage systems provide a plugin to let their data be read by MapReduce, but this is done through a Java API specific to Hadoop.

Our goal in this paper, therefore, is to encourage researchers to attack these problems from an operating system perspective. To this end, we discuss the main challenges that need to be addressed by a datacenter OS (Section 2), and how research in this area can complement industry work (Section 3). Not only are the problems we highlight relevant today, but we believe that work on a datacenter OS can significantly impact the future cloud software stack. If successful, a datacenter OS should enable the development of a rich ecosystem of interoperable cluster applications, much like the desktop software ecosystem, and greatly benefit cluster users.

2 Datacenter OS Functionality

This section discusses four key areas of functionality that a datacenter OS needs to provide: resource sharing, data sharing, programming abstractions, and debugging. To avoid producing a laundry list of problems, we focus

[1] By datacenter OS, we mean a software stack providing functionality for the overall datacenter that is analogous to what a traditional OS provides on one machine. We are not calling for a new host OS to be run in datacenters, though such a change may also prove beneficial.

[2] Indeed, the recently announced "next-generation Hadoop" design from Yahoo! aims to support running non-MapReduce jobs as well [5].

on *cross-application concerns*: issues involving interaction between applications, and useful shared abstractions. These are the core concerns that motivate a common, OS-like layer underlying all applications. We put these ideas together in Section 2.5 to sketch a set of goals that a successful datacenter OS would meet.

2.1 Resource Sharing

Datacenters already host a diverse array of applications (storage systems, web applications, long-running services, and batch analytics), and as new cluster programming frameworks are developed, we expect the number of applications to grow. For example, Google has augmented its MapReduce framework with Pregel (a specialized framework for graph applications), Dremel (a low-latency system for interactive data mining), and Percolator (an incremental indexing system). At the same time, the number of cluster users is growing: for example, Facebook's Hadoop data warehouse runs near-interactive SQL queries from hundreds of users [29]. Consequently, it is crucial for datacenter operators to be able to multiplex resources efficiently both between users of an application and across applications.

Unfortunately, cluster applications are currently developed as standalone programs that get launched on some set of nodes and assume they have full control over those nodes for their duration. The only option for sharing resources between these applications is coarse-grained partitioning at the level of physical or virtual hosts. This approach becomes inefficient as the number of applications grows and their demand becomes more dynamic (*i.e.,* the load on each application changes over time). The solution is clear: resource sharing needs to happen at a finer granularity. Indeed, systems like Hadoop and Dryad already perform fine-grained sharing between their jobs, at the level of "tasks" within a job [18, 29]. However, there is no standard interface for fine-grained sharing across *different* applications, making it difficult for organizations to use multiple cluster computing frameworks.

We have undertaken some initial work in this area by designing Mesos [15], a system that enables fine-grained sharing across applications. Such a system faces several serious challenges: it must be flexible enough to support the placement and fault recovery needs of many applications, scalable to clusters running millions of tasks, and highly reliable. Mesos takes a minimalist approach to the problem by employing an application-controlled scheduling model called resource offers: Mesos decides which applications have priority for resources, but applications choose which resources to use and which tasks to launch on them. We found that this approach performs surprisingly well. We are not claiming that this is necessarily how a datacenter OS should allocate resources, we merely cite it as an example of cross-application sharing.

In addition to the issue of fine-grained sharing, we note several other questions that deserve attention:

- *Sharing the network:* Greenberg et al. identify isolating traffic from different datacenter applications as a serious concern [13]. Operators would like to be able to colocate user-facing web applications and batch analytics jobs in the same cluster, for example, but current network management mechanisms fall short.
- *Interdependent services:* Many web applications are composed of multiple interacting services. For example, a front-end server in a search engine might query a spell-checking service, a map service, etc. Most cluster scheduling systems assume that applications are independent and do not heed these dependencies.
- *Optimal scheduling:* There are hard modeling and algorithmic challenges in determining the "best" schedule for a set of cluster applications with various resource and placement requirements. There are also multiple optimization criteria, including throughput, response time, and energy efficiency. While the community has designed schedulers that improve fairness and data locality [18, 29], we currently lack models to let us gauge how close these schedulers are to optimal.
- *Role of virtualization:* The largest datacenter operators, including Google, Microsoft, and Yahoo!, do not appear to use virtualization due to concerns about overhead. However, as virtualization overhead goes down, it is natural to ask whether virtualization could simplify scheduling (*e.g.,* through VM migration).

2.2 Data Sharing

Datacenter applications need to share not only computing resources but also data. For example, it is natural to want to combine steps written in different parallel programming frameworks in a workflow (*e.g.,* build a graph using MapReduce and compute PageRank on it using Pregel), in the same way that Unix programs can be grouped into pipelines. Enabling data sharing requires two steps: finding the right *abstractions* for sharing data between cluster applications, and defining *standardized interfaces* to those abstractions that allow their implementations and clients to evolve independently.

Today, the most common abstraction used for data sharing in clusters is distributed filesystems. This approach is simple, but it is not always efficient due to the cost incurred by filesystems to achieve reliability (replicating data across nodes and checkpointing it to disk). For example, in iterative MapReduce computations, jobs often spend a significant amount of time reading and writing from the filesystem, even though each intermediate dataset is only used in the next job.

One example of a more efficient approach is to have the storage system remember how to recompute each block of data, in much the same way that a MapRe-

duce system knows how to re-run a map task if it loses its output. To this end, we have designed an abstraction called resilient distributed datasets (RDDs) [30]. RDDs are read-only partitioned collections of elements built by transforming data in stable storage through a limited set of operators, and they remember the transformations that went into building them to allow efficient reconstruction of lost blocks. As a result, they can be stored in memory in the common case, without requiring disk writes or replication. We believe that RDDs are powerful enough to express MapReduce, SQL and Pregel computations, and could thus be used to efficiently share data between these programming models. Again, we are not claiming that RDDs are necessarily the right solution for a datacenter OS, but are an example of how a new data abstraction can support a broad range of applications.

There are many other open questions in this area:

- *Standardized interfaces:* An important contribution of a datacenter OS would be standard interfaces for cluster storage abstractions, much like VFS. This is not trivial even for distributed file systems, because these systems are expected to not only provide access to data but also give applications hints about where to access it from to achieve the best performance. Contour [27] is an interesting step in this direction.
- *Streaming data:* Distributed filesystems and RDDs are best suited for "write-once" data, such as intermediate results in batch computations. It is still an open question to determine similar abstractions for streaming data. Two promising but very different approaches are distributed message queues [1], which are commonly used in enterprise systems, and Google's Percolator system [25], which is based on triggers on BigTable.
- *Performance isolation:* Performance guarantees are difficult to achieve in complex distributed storage systems such as key-value stores. As a result, many datacenter operators use separate storage systems for front-end web servers and back-end analytics, and copy data from the former to the latter periodically. The ability to safely run batch queries on live data would greatly improve the timeliness of analytics, but appears to require a far more careful design approach.

2.3 Programming Abstractions

One of the key roles of an OS is to provide abstractions that hide the intricacies of hardware and simplify application development. In the datacenter, the hardware is more complex (nodes can fail, perform poorly, etc), and applications are harder to develop. Programming abstractions for datacenters remain an important problem.

We differentiate between two classes of programmers: systems programmers that are writing low-level infrastructure such as MapReduce and BigTable, and productivity programmers that use this infrastructure to solve problems. So far, a lot of effort has been invested in productivity programming, in the form of parallel computing frameworks [10, 17, 21, 25]. However, now that several different computing frameworks have been built from scratch, we believe it is also time to look for common abstractions to simplify systems programming.

Systems abstractions appear to be necessary because numerous *specialized* cluster computing systems continue to be developed for problems where the general frameworks are not a good fit. For example, Percolator [25], Pregel [21] and GraphLab [20] are some recent specialized computing frameworks for web indexing, graph processing and machine learning respectively. Similarly, Facebook had to write a scalable and fault-tolerant backend from scratch for Facebook Chat [19], because a traditional three-tier web application architecture could not support this use case. Ideally, a datacenter OS should provide primitives to implement these systems significantly faster. Some useful primitives might include:

- *APIs for launching and monitoring tasks:* We found that even the minimal interface in Mesos, which allows an application to start tasks and get notified when they end, made it easier to prototype new programming models, because it obviated the need for each framework to implement a master and a slave daemon.
- *Communication primitives:* Many parallel applications have similar communication patterns. For example, the all-to-all shuffle pattern in MapReduce is also present in Pregel and in various distributed joins. In our experience, these parallel transfers often become bottlenecks. A datacenter OS is well suited to provide efficient implementations of common patterns.
- *Fault-tolerant distributed data structures*, such as the RDDs discussed earlier, to manage state (including control state for applications needing master failover).
- *Coordination primitives* such as Chubby [8].

2.4 Debugging and Monitoring

Figuring out what a massively parallel application is doing remains one of the hardest challenges in cluster computing. Horror stories abound about how minor programming errors, unusual load patterns, and bit flips brought down major systems. In general, debugging tools that work on a single machine are very difficult to use at scale due to the much larger volume of events.

In addition to correctness debugging, performance debugging is also critical in the datacenter. Much of the complexity in datacenter applications resides in control plane logic and data structures, such as the task scheduler in a computing framework or the metadata node(s) in a storage system. These components need to scale up to support large jobs, many concurrent users, and large numbers of objects. Often, the load on them also shifts as the application is is picked up for new use cases.

Finally, both correctness and performance debugging are becoming harder as the datacenter software stack grows in complexity. For example, when a Pig job running over data in HBase (a BigTable-like key-value store) performs poorly or outputs the wrong result, is the problem in the user's code, in Pig, in the MapReduce framework underlying Pig, in HBase, or in the HDFS file system that HBase runs over?

We believe that debugging is an area where researchers should explore clean-slate approaches. For example, how much easier would debugging become if the entire software stack implemented a tracing interface like X-Trace [11]? Alternatively, would it be possible to build a useful replay debugger for datacenters if deterministic OSes [7, 6] or languages were used throughout? While it may seem that there is too much legacy datacenter software for this approach to have impact, datacenter operators are writing new software and reimplementing old systems quite frequently due to changing workloads. At the very least, this type of work would identify the limits of debuggability in large distributed systems and let us know how far we are from them with current tools.

2.5 Putting it All Together

Although we have discussed the functions of a datacenter OS in isolation so far, we emphasize that we ultimately envision a platform that *combines* these functions. Such a platform would greatly improve the usability and programmability of datacenters while laying the foundation for an ecosystem of interoperable cluster applications.

To make this vision concrete, a good first step towards a datacenter OS might strive to meet the following goals:

1. Support a software stack similar to today's open source MapReduce and storage systems, while gaining *cross-application* benefits that are difficult to achieve today, such as cross-stack replay debugging.
2. Enable the implementation of new programming frameworks within a matter of weeks.
3. Share data and resources efficiently between both the existing stack and new applications written by users.
4. Allow users to understand cluster behavior without ever having to log into a remote machine.

3 Role of the Research Community

Given that most systems researchers lack access to the largest datacenters, it will be harder for them to work on a datacenter OS than on OSes for other platforms. Nonetheless, we believe that datacenters are a sufficiently important platform for researchers to pay attention to, and that there are several ways in which researchers can complement the work going on in industry:

Focus on paradigms, not performance: While it is tempting to do research into improving the performance of cloud systems (due to ease of evaluation), the industry is already investing many resources into performance. Instead, researchers are better suited to identify the abstractions to put into a datacenter OS, which industry teams have less liberty to take a long-term view on given their need to focus on shorter-term business goals.

Explore clean-slate approaches: Some problems in datacenters, like software reliability and debugging, are nearly intractable under the current software stack. However, if researchers show that, for example, restricting the programming language makes it much easier to debug cluster applications or to make guarantees about their performance, there is a chance that practitioners will pay attention, as a lot of datacenter software is yet to be written (there is little legacy software) and these problems are very costly. Practitioners have already adopted a functional programming model (MapReduce) for its benefits.

Bring cluster computing to non-experts: One of the most exciting things about datacenter technology is that it is increasingly being applied to "big data" problems in the sciences. With cloud computing, scientists can readily acquire the hardware to run large parallel computations; the main thing missing is the right software. These non-expert cluster users have very different needs from those in large corporations: they are not backed by an operations team that will configure their systems and tune their programs. Instead, they need cluster software that configures itself correctly out of the box, rarely fails, and can be debugged without intimate knowledge of several interacting distributed systems. These are difficult but worthwhile challenges for the community to pursue.

4 Related Work

A datacenter OS can leverage many insights from distributed operating systems, high-performance computing, and grid computing. Nonetheless, several factors differentiate the modern datacenter environment from these settings. First, failures occur more regularly in commodity clusters than in most previous platforms [16]. Second, there is a strong focus on data-intensive parallel applications instead of compute-intensive ones [10]. Finally, datacenters host a far more heterogeneous mix of applications than many previous platforms, ranging from latency-sensitive web services to batch jobs. This makes resource sharing and isolation challenging [13].

The closest research to our vision is on distributed operating systems like Amoeba [26] and Sprite [23]. These systems were designed unify collections of workstations by providing a single system image in which processes, files and other objects are distributed transparently across machines. However, distributed OSes were mainly designed to run timesharing workloads consisting of single-process tasks such as email clients and compilers, rather

than data-intensive parallel applications. Although parallel applications were also supported, these OSes provided few parallel programming abstractions beyond distributed file systems, threads and RPC. In addition, because task placement and fault recovery are so important in datacenters, we believe that transparency is less needed in a datacenter OS. For example, in Mesos, we give applications control over their scheduling.

More recently, fos [28] was proposed as an operating system for clouds and multicore CPUs in which OS services are inherently distributed. Like distributed OSes, fos aims to provide a single system image. The main focus in fos has been on designing scalable OS services (*e.g.,* file systems or name servers) rather than exposing new abstractions to cluster applications. This work is complementary to the type of research we solicit. We encourage the community to go beyond scaling up existing OS services and design new programming primitives, data abstractions and resource schedulers for clusters.

Finally, software platforms such as the Hadoop stack, LAMP, Amazon Web Services, Windows Azure, and Google's GFS / BigTable / MapReduce stack [12, 9, 10] form today's *de facto* datacenter OS. These platforms are gradually evolving to cope with the increased diversity of datacenter users and workloads (for example, substantial effort was put into Hadoop scheduling for multi-user clusters), but datacenter applications are still generally hard to develop and do not interoperate easily. We envision a future software stack in which new cluster storage systems, data processing frameworks and services are significantly easier to build and can plug into an ecosystem of interoperable tools using standardized interfaces.

5 Conclusion

Datacenters have become a major computing platform, powering not only popular Internet services but also a growing number of scientific and enterprise applications. We argued that, as the use of this new platform grows, datacenters increasingly need an operating system-like software stack for the same reasons that single computers did: resource sharing between applications and users, data sharing, and abstraction. This kind of software stack is already emerging in an ad-hoc manner, but now is the right time for researchers to take a long-term approach to these problems and have a lasting impact on the software infrastructure for this new computing platform.

References

[1] ActiveMQ. http://activemq.apache.org.

[2] Amazon Web Services Case Studies. http://aws.amazon.com/solutions/case-studies.

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

[4] Apache Hive. http://hadoop.apache.org/hive.

[5] The next generation of Apache Hadoop MapReduce. http://developer.yahoo.com/blogs/hadoop/posts/2011/02/mapreduce-nextgen.

[6] A. Aviram, S.-C. Weng, S. Hu, and B. Ford. Efficient system-enforced deterministic parallelism. In *OSDI 2010*.

[7] T. Bergan, N. Hunt, L. Ceze, and S. D. Gribble. Deterministic process groups in dOS. In *OSDI 2010*.

[8] M. Burrows. The Chubby lock service for loosely-coupled distributed systems. In *OSDI 2006*.

[9] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. BigTable: a distributed storage system for structured data. In *OSDI 2006*.

[10] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[11] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica. X-trace: A pervasive network tracing framework. In *NSDI 2007*.

[12] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *SOSP 2003*.

[13] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, 2009.

[14] D. Hastorun, M. Jampani, G. Kakulapati, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *SOSP 2007*.

[15] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI 2011*.

[16] U. Hoelzle and L. A. Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition, 2009.

[17] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *EuroSys 2007*.

[18] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair scheduling for distributed computing clusters. In *SOSP 2009*.

[19] E. Letuchy. Facebook Chat. http://www.facebook.com/note.php?note_id=14218138919.

[20] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. GraphLab: A new parallel framework for machine learning. In *UAI*, 2010.

[21] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD 2010*.

[22] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD 2008*.

[23] J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson, and B. B. Welch. The Sprite network operating system. *Computer*, 21:23–36, February 1988.

[24] D. A. Patterson. Technical perspective: the data center is the computer. *Commun. ACM*, 51:105–105, January 2008.

[25] D. Peng and F. Dabek. Large-scale incremental processing using distributed transactions and notifications. In *OSDI 2010*.

[26] A. S. Tanenbaum, R. van Renesse, H. van Staveren, G. J. Sharp, and S. J. Mullender. Experiences with the Amoeba distributed operating system. *Commun. ACM*, 33:46–63, December 1990.

[27] B. Tiwana, M. Balakrishnan, M. K. Aguilera, H. Ballani, and Z. M. Mao. Location, location, location!: modeling data proximity in the cloud. In *HotNets 2010*.

[28] D. Wentzlaff, C. Gruenwald, III, N. Beckmann, K. Modzelewski, A. Belay, L. Youseff, J. Miller, and A. Agarwal. An operating system for multicore and clouds: mechanisms and implementation. In *SOCC 2010*.

[29] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *EuroSys 2010*.

[30] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *HotCloud 2010*.