1 Introduction

Large clusters of commodity servers have become an important computing platform in both industry and academia. They power not only some of the most popular consumer applications in use today—Internet services such as search, social networks and e-commerce—but also a growing number of data-intensive scientific and enterprise applications [2, 5].

The use of thousands of computers to run a single, massively parallel application has led some to declare that “the datacenter is the new computer” [16, 22]. 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 multiple programs and multiple 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 debugging and monitoring facilities that makes it easier to troubleshoot applications.

We argue that the same need for sharing and abstraction that motivated single-computer OSes is arising in the datacenter. Resource sharing and data sharing are increasingly important because there is a growing number of parallel computing frameworks (e.g., MapReduce [10], Dryad [17], Pregel [19]) and storage systems (e.g., BigTable [9], Dynamo [14]) for clusters, 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 simultaneous users. For example, while MapReduce was initially used for a small set of batch jobs (e.g., web indexing), organizations like Facebook are now using it to build data warehouses where hundreds of users run near-interactive ad-hoc queries [27]. There is a need for responsive time-sharing between these users.

Programming and debugging cluster applications remains difficult even for experts, and is even more challenging for the increasing number of non-expert cluster users (e.g., scientists). Thus, there is a need for high-level abstractions and system-wide debugging facilities to make datacenters easier to use. If successful, a datacenter OS should allow programmers to develop a rich ecosystem of interoperable cluster applications that users can combine to solve problems.

To some extent, an OS-like software stack for clusters is already emerging. For example, in the open source Hadoop stack [3], MapReduce acts as a common execution layer upon which higher-level programming interfaces like Pig [20] and Hive [4] are built. The MapReduce scheduler performs resource multiplexing, although, unfortunately, this means that only jobs that compile down to MapReduce may run on the cluster.² In addition, most open source key-value stores and cluster filesystems provide a Hadoop plugin to let their data be read by MapReduce, but this is done by implementing a Java interface specific to Hadoop. When we asked a group of industry colleagues whether the datacenter currently has an operating system, the consensus was that it does, but that the OS it has is not the one they want.

Given that the problems motivating a cluster OS are real and that current solutions are ad-hoc, we believe that the systems community is well-positioned to influence the datacenter software stack by approaching these problems from an OS perspective. Our main goal in this paper is to encourage researchers to take this perspective.

¹By datacenter OS we mean a software stack that provides functionality for the overall datacenter that is analogous to what a traditional OS provides on a single machine. We are not calling for a new host OS to be run in datacenters. However, for reasons other than what we discuss here, such a change might also prove beneficial.

²Indeed, our contacts at Yahoo! report that users who wish to run MPI jobs are launching them inside Hadoop tasks just to be able to run on the company’s large Hadoop-managed clusters. This is problematic because the Hadoop scheduler is unaware of the requirements of MPI.
In the rest of this paper, we discuss the major challenges that need to be addressed by a datacenter OS (in Section 2): resource sharing, data sharing, programming abstractions, and debugging. We then discuss how researchers can work on these problems in a manner that complements industry in Section 3. Finally, we survey related work in Section 4 and conclude in Section 5.

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. We list research challenges in each area, as well as existing work on these topics from our group and others.

With a broad enough definition of “operating system,” such a discussion risks turning into a laundry list covering all areas of datacenter research. To avoid this, we focus on cross-application concerns: interactions between applications and useful shared abstractions. These are the core concerns that motivate an OS-like layer (a common software layer underlying all applications).

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 [27]. 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 is inefficient as the number of applications grows and their demand becomes increasingly 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, 27]. 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, at least for analytics applications. We are not claiming here that this is necessarily how a datacenter OS should allocate resources, we merely cite it as an example of cross-application resource sharing.

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

- **Sharing the network**: Isolating traffic from different applications remains a serious concern of datacenter operators [13]. Operators would like to be able to collocate user-facing web applications and batch analytics jobs in the same cluster, for example, but current network management mechanisms are not well-suited to the datacenter environment.
- **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 schemes assume that applications are independent, but in these cases it would be useful to take into account the dependency to prevent pathologies such as priority inversion.
- **Optimal scheduling**: There are hard modeling and algorithmic challenges in determining the “best” schedule for a set of cluster applications with various resource requirements and placement constraints. There are also multiple optimization criteria, including throughput, response time, and energy efficiency. While the community has designed cluster schedulers that optimize for fairness and data locality [18, 27], there are currently no well-defined models that let us measure 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 paral-
lel 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 MapReduce 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) [28]. 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 in a manner that allows efficient reconstruction of lost blocks. As a result, they can be stored in memory in the common case, without any need for disk writes or replication. We believe that RDDs are powerful enough to express MapReduce, SQL and Pregel computations, and could thus allow users 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 they are an existence proof of how a new data abstraction can support a right solution for a datacenter OS, but they are an existence proof of how a new data abstraction can support a right solution for a datacenter OS.

For disk writes or replication. We believe that RDDs are powerful enough to express MapReduce, SQL and Pregel computations, and could thus allow users 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 they are an existence proof of how a new data abstraction can support a broad range of cluster 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 [25] 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 [23], 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 programming frameworks [10, 17, 19, 23]. However, now that several different cluster computing frameworks have been implemented from scratch, we believe it is also time to look for common abstractions to simplify systems programming. Ideally, a datacenter OS should provide the right primitives to implement the next MapReduce, Pregel or BigTable in a matter of days.

Some useful common abstractions 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 many forms of distributed joins. In our experience, these communication patterns can rapidly become bottlenecks in parallel applications. A datacenter OS is well suited to provide efficient implementations of common patterns.

- **Fault-tolerant distributed data structures** such as the RDD abstraction discussed in the previous section.

- **Coordination primitives** such as Chubby [8].

In addition, research on programming languages for distributed systems is also very relevant in the datacenter.

### 2.4 Debugging and Monitoring

Figuring out what a massively parallel application is doing remains one of the hardest challenges in the data-
center space. Horror stories abound about how small 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 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 on an HBase table 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? Alternatively, would it be possible to build a useful replay debugger for datacenters if deterministic OSes 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.

3 Role of the Research Community

Given that 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 that researchers should pay attention to, and that there are several ways in which researchers can complement the work going on in industry:

1. 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 time to think about given their focus on shorter-term business goals.

2. Explore clean-slate approaches: Some problems in datacenters, such as 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, because much 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.

3. 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 applications. 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, there are three challenges that differentiate the modern datacenter environment from these settings. First, failures occur regularly in large commodity clusters and must be tolerated by OS components and applications [16]. Second, there is a strong focus on data-intensive parallel applications instead of compute-intensive ones [10]. Lastly, datacenters host a 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 [24] and Sprite [21]. These systems were designed to unify collections of workstations and other computing resources by providing a single system image in which processes, files and other objects are distributed transparently across machines. Their requirements differed from those of the datacenter in several ways. First, distributed OSes were mainly designed to run user 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

---

3HBase is an open source BigTable-like key-value store.
parallel programming abstractions beyond distributed file systems, threads and RPC. Second, the larger scale of datacenters makes failures more common, so OS abstractions for the datacenter need to be fault tolerant. Lastly, because task placement and fault recovery are so important in datacenters, we believe that transparency is less suitable in a datacenter OS. For example, in Mesos, we give applications control over their scheduling.

More recently, fos [26] was proposed as an operating system for clouds and multicore processors 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 resource abstractions, data sharing abstractions, programming interfaces and resource scheduling schemes for the datacenter.

Lastly, software platforms such as the Hadoop stack [3] (discussed in the introduction), LAMP, 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, significant effort was put into Hadoop scheduling for multi-user environments), 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 platforms 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 widely used Internet services but also a growing number of scientific and enterprise applications. Due to stalling processor speeds, increasing data volumes, and the continued growth of the Internet, we expect cluster computing to continue to grow in importance. We have argued that 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 starting to emerge in an ad-hoc manner, but now is the right time for the systems community to take a principled approach to these problems and have a lasting impact on the software infrastructure for this new computing platform.

References