From Cloud Computing to Sky Computing

Ion Stoica and Scott Shenker

Abstract

We consider the future of cloud computing and ask how we might guide it towards a more coherent service we call sky computing. The barriers are more economic than technical, and we propose reciprocal peering as a key enabling step.

1 Introduction

In 1961, John McCarthy outlined what he saw as the future of computing (we regretfully note the gendered language):

> *"computation may someday be organized as a public utility, just as the telephone system is a public utility. We can envisage computer service companies whose subscribers are connected to them [...]. Each subscriber needs to pay only for the capacity that he actually uses, but he has access to all programming languages characteristic of a very large system."*

As a prediction about technology, McCarthy's vision was remarkably prescient. His short description accurately depicts what we now call cloud computing, where users have access to massive amounts of computation and storage and are charged only for the resources they use. On the contrary, his prediction about economics was far off the mark; today, in the United States even telephone service is no longer a public utility but delivered instead through a competing set of providers. However, while not a utility, phone service is largely a commodity, providing a uniform user experience: i.e., no matter what provider you use, you can reach anyone, and switching providers is relatively easy; you can even keep your number when switching.

Returning to McCarthy's prediction, what concerns us here is not that there is no single public computing utility, but that cloud computing is not an undifferentiated commodity (like phone service). In contrast, the cloud computing market has evolved away from commoditization, with cloud providers striving to differentiate themselves through proprietary services. In this paper we suggest steps we can take to overcome this differentiation and help create a more commoditized version of cloud computing, which we call the Sky computing. Before doing so, we briefly summarize the history of cloud computing to provide context for the rest of the paper.

2 Historical Context

The NSF high performance computing (HPC) initiative in the 1980s provided an early glimpse of the utility computing vision, though limited to the HPC community. However, the advent of personal computers delayed this vision by putting the focus on smaller-scale computing available to all, rather than high-performance computing for a smaller community. The personal computer industry was started by hobbyists that wanted their own machines to tinker with, and was fuelled by Moore's Law that enabled personal computers to keep up with the users' rapidly increasing demands.

The emergence of the Internet led quickly to several popular services being accessible world wide, including e-mail, bulletin board systems, and games. The advent of the World Wide Web ignited an explosion of new services such as search, on-line retail, and eventually social media. Coping with the resulting scale required these services to build datacenters and design their own complex distributed systems. This path was followed by many companies – such as Yahoo!, Google, eBay, and Amazon – but the onus of creating large-scale service-specific infrastructures became a barrier to entry in the Internet services market because these steps required huge investments which few could afford.

This changed in 2006 when Amazon launched S3 and EC2, kicking off the age of cloud computing by democratizing access to compute/storage and promoting the "pay-as-you-go" business model. This, together with the end of Moore's Law (which made it quite expensive to build and scale services in on-prem clusters), led to a renewed push towards building what could have become utility computing. However, commercial trends have pushed us in a different direction.

In the early years when Amazon was dominant in the cloud, they set the de facto standard for cloud computing. However, in the past decade several competitors have emerged in this market. According to a recent report [5], AWS owns now just 32% of the market, followed by Microsoft 19%, Google 7%, Alibaba 6%, and the other clouds (such as IBM and Oracle) splitting the rest of 37%. This competition has led to lower prices and an increasing array of products and services. For instance, AWS alone offers more than 175 products and services [7]. However, many of these services are proprietary, and these proprietary services are one of the main ways cloud providers differentiate themselves. For example, each cloud has its own APIs for managing clusters, its own version of object store, its own version of a data warehouse, its own serverless offering, and so on. Applications developed on one cloud often do not work on a different cloud without substantial changes, the same way an application written for Microsoft Windows requires substantial changes to work on Mac OS. Thus, competition in the cloud market has led us away from the vision of utility computing.

Of course there have been many calls for standardizing cloud computing [25, 31], but these have had little impact on the increasing drive towards cloud differentiation. The

business models of current cloud providers are built around luring, and then retaining, customers, and this goes against offering a purely commoditized service. The question we address, then, is how do we make progress towards the goal of utility computing?

3 Lessons from the Internet

Despite the fact that cloud computing and the Internet differ along many dimensions, we think the Internet provides a useful set of historical lessons. In the early 1960s several groups were developing packet switching technology. These early networks worked well, but were incompatible with each other. The community faced a choice: should it standardize on a single networking technology, or could it find a way to accommodate diversity? In 1972 Robert Kahn proposed open-architecture networking [27], which advocated a universal interoperability layer that could allow any two networks to interconnect. This became the Internetworking Protocol (IP). IP and the protocols above it were all that was needed in the ARPAnet, which was a single coherent network. However, as the ARPAnet grew into the Internet, and the Internet grew to include separate Autonomous Systems (ASes, which are independently run networks), the Internet needed some way for packets in one network to reach another. This was not about low-level compatibility in network technologies, this was a question of routing: how can the network successfully direct packets over a set of independently run networks.

The Border Gateway Protocol (BGP) was invented to solve this problem, and is now the "glue" that makes the set of independently run networks appears as a coherent whole to users. This was a technical solution, but an economic problem remained: when networks interconnect, does money change hands, and if so, who pays whom? A set of business practices have emerged for these "peering" or interconnection agreements. For instance, customers pay providers, but two networks of similar size and capability might do payment-free peering. These peering arrangements are technically straightforward, but bring up thorny policy questions such as: can provider A charge Internet service B for carrying B's traffic, even if B is not directly connected to A. Thus, issues of unfair competition arise in these agreements (and are related to questions of network neutrality).

Thus, there were three key design decisions that allowed the Internet to provide a uniform interface to a huge infrastructure made out of heterogeneous technologies (from ethernet to ATM to wireless) and competing companies. The first is a "compatibility" layer that masks technological heterogeneity. The second is interdomain routing that glues the Internet together, making it appear as one network to end users. The third is a set of economic agreements, forming what we will call a "peering" layer, that allow competing networks to collaborate in creating a uniform network.

What lessons does this teach us about the cloud? To fulfil the vision of utility computing, applications should be able to run on any cloud provider (i.e., write-once, run-anywhere). Moreover, users should not have to manage the deployments on individual clouds, or face significant impediments moving from one cloud to another. In short, it should be as easy for a developer to build a multi-cloud application, as it is to build an application running on a single cloud. We call this Sky computing. We use this term because utility computing implies that the infrastructure is a public utility, whereas Sky computing refers to building the illusion of utility computing on an infrastructure consisting of multiple and heterogeneous competing commercial cloud providers.

We contend that the three design issues the the Internet had to address are exactly the pieces needed to create Sky computing out of our current set of clouds. We need a compatibility layer to mask low-level technical differences, an intercloud layer to route jobs to the right cloud, and a peering layer that allows clouds to have agreements with each other about how to exchange services. In the next three sections we describe these layers in more detail. We then conclude by speculating about the future.

4 Compatibility Layer

The first step towards achieving the Sky computing vision is to provide a cloud compatibility layer; i.e., a layer that abstracts away the services provided by a cloud and allows an application developed on top of this layer to run on different clouds without change. In short, the compatibility layer is a set of interfaces or APIs that applications can be built on; this compatibility layer can then be ported to each cloud using the cloud's set of (perhaps proprietary) interfaces.

In our Internet analogy, this is similar to the IP layer which enables routers using different underlying (L2) communication technologies to handle IP packets. However, unlike IP which is a narrow waist, the cloud compatibility layer is significantly broader and less well defined since clouds expose a rich (and growing) set of services to the applications, including computation, storage, data transfers, and much more. Thus, the cloud compatibility layer is more similar in spirit to an operating system (e.g., Linux) that manages the computer's resources and exposes an API to applications.

How do we build such a cloud compatibility layer? While every cloud offers a set of proprietary low-level interfaces, most users today interact mostly with higher level management and service interfaces. Some of these are proprietary, but a growing number of them are supported by open source software (OSS).

These OSS projects exist all levels of the software stack, including operating systems (Linux), cluster resource managers (Kubernetes [11], Apache Mesos [26]), application packaging (Docker [9]), databases (MySQL [14], Postgres [15]), big data execution engines (Apache Spark [34], Apache Hadoop [33]), streaming engines (Apache Flink [23], Apache Spark [34], Apache Kafka [4]), distributed query engines and databases (Cassandra [3], MongoDB [13], Presto [16], SparkSQL [20],

Redis [17]), machine learning libraries (PyTorch [30], Tensorflow [21], MXNet [24], MLFlow [12], Horovod [32], Ray RLlib [28]), and general distributed frameworks (Ray [29], Erlang [22], Akka [1]).

Furthermore, a plethora of companies founded by OSS creators have emerged to provide hosted services on multiple clouds. Example are Cloudera (Apache Hadoop), Confluent (Apache Kafka), MongoDB, Redis Labs, HashiCorp (Terraform, Consul), Datastax (Cassandra), and Databricks (Apache Spark, MLFlow, and Delta). This makes it relatively easy for enterprises to switch from one cloud to another if their applications are using one of these multi-cloud OSS-based offerings.

The compatibility layer could be constructed out of some set of these OSS solutions. Indeed, there are already efforts underway to consolidate different OSS components in a single coherent platform. One example is Cloud Foundry [8], an open source multi-cloud application platform that supports all major cloud providers, as well as on-premise clusters.

While OSS provides solutions at most layers in the software stack, the one glaring gap is the storage layer. Every cloud provider has its own version of proprietary highly-scalable storage. Examples are AWS' S3 [2], Microsoft's Azure Blob Storage [6] and Google's Cloud Storage [10]. This being said, there are already several solutions providing S3 compatibility APIs for Azure's Blob Storage and Google's Cloud Storage, such as S3Proxy [18] and Scality [19]. Some cloud providers offer their own S3 compatibility APIs to help customers transition from AWS to their own cloud.

Thus, we think achieving a widely usable compatibility layer is, on purely technical grounds, easily achievable. The problem is whether the market will support such an effort because, while the compatibility layer has clear benefits for users, it naturally leads to the commoditization of the cloud providers, which may not be in their interests. We will discuss the incentives issues in Section 7.

5 Intercloud Layer

The compatibility layer is just the first step in realising the Sky vision. Even though the compatibility layer allows the user to run an application on different clouds without change, the user still needs to decide on which cloud to run the application. Thus, the user is still responsible for making the performance/cost tradeoffs across different clouds. This is akin to an Internet user explicitly selecting the AS paths for its interdomain traffic, which would be an onerous task.

To address this problem, the Internet employs BGP to make AS-level routing decisions, decisions that are transparent to the application. Similarly, the Sky architecture should implement an intercloud layer that abstracts away the cloud providers from the user; that is, the user should not be aware of which cloud the application is running on (unless they explicitly want to know). The intercloud layer is implemented on top of the compatibility layer, as seen in Figure 1.

The intercloud layer must allow users to specify policies about where their jobs should run, but not require the user to make low-level decisions about job placement (but would allow the user to do so if they desired). These policies would allow a user to express their preferences about the tradeoff between performance, availability, and cost. In addition, a user might want to avoid their application running on a datacenter located in an unfriendly country, or stay within certain countries to obey relevant privacy regulations. To make this more precise, a user might specify that this is a Tensorflow job, it involves data that cannot leave Germany, and must be finished within the next two hours for under a certain cost.[1]

We believe there are no fundamental technical limitations in implementing the intercloud layer on top of the cloud compatibility layer. This is simply because, from performance perspective, moving jobs across clouds is very similar to moving jobs within the same cloud across datacenters. Once an application has been design to run across multiple datacenters, remaining cross-cloud issues can be address by the following three functionalities:

1. A uniform naming scheme for OSS services.
2. A directory service which allows cloud providers to register their services, and applications to select a service based on their preferences.
3. An accounting and charging mechanism across clouds.

We now discuss each of these functionalities in turn.

Service Naming Scheme In order to identify a service instance running on a particular cloud we need a naming scheme to identify that instance. There are many possible schemes, and it is not our goal here to advocate for one. A natural possibility would be to leverage DNS for naming these service instances. In addition, we need to associate metadata with each such service instance. Such metadata should contain how the service should be invoked, the name of the cloud provider, location, software or API version, hardware type, etc. Furthermore, we might also want to add dynamic information like pricing and load or availability. All this information can be stored in json format or another standard format.

Directory service Given a service name, an application must find a service instance that satisfies its requirements and preferences. This calls for a directory service. Each cloud provider will publish its services to this directory by providing its name and the metadata information. Furthermore, each cloud provider should periodically update their dynamic metadata, such as the load and spot pricing. In turn, the applications should request a particular service expressing its preferences and requirements. Upon receiving such request, the directory service would return an instance satisfying these requirements and preferences. The request schema and the resolution algorithms are outside the scope of this paper.

[1] We assume that the storage API provided by the compatibility layer allows reading data across clouds.

Accounting and charging With Sky computing, a user's application can run on one of many clouds or even on several clouds at the same time, and they will each account for the resources used. If the charging was done by each cloud, each user would need to have accounts on every cloud. We propose an alternative where each user signs up with a single cloud (or a third-party broker) who has an account on all the clouds, and accumulates the charges and then distributes payments from each of their users back to the various clouds.

While many details remain to be worked out, there do not appear to be any insurmountable technical barriers to achieving a reasonably effective intercloud layer. As with the compatibility layer, the issue is whether the market will produce one.

6 Peering Between Clouds

The intercloud layer is designed to run jobs on the cloud that best meets their needs. If the job involves large datasets, as many of the common cloud workloads do, this will require moving the data to the cloud where the computation will occur. Today, most clouds have pricing policies where moving data into a cloud is much cheaper than moving it out. For instance, transferring data out of AWS can cost as much as 0.09$/GB, the cost of storing a GB of data for several months[2]! In contrast, getting the data into AWS is free.

We will call this form of pricing "data gravity" pricing, and it creates a strong incentive for users to process the data in the same cloud. Of course, moving data from one cloud to another can still be the most cost-effective option, especially for jobs where the computation resources are much more expensive than the data transfer costs. For example, consider ImageNet training which involve a 150GB dataset. It costs about $13 to transfer it out of AWS, but, according to the DAWNBench[3], it costs over $40 to train ResNet50 on ImageNet on AWS compared to about $20 to train the same model on Azure. Given these numbers, it would be cheaper to move the data from AWS and perform training on Azure, instead of performing the training in AWS where the data is. Thus, while data gravity pricing does inhibit moving jobs, in some cases moving jobs is still worthwhile.

In addition, the incentives against job movement are most acute if the data is dynamic (i.e., is updated as a result of the computation). If the data is static (e.g., training data), then the user could use an archival store on one cloud (such as AWS's Glacier), which is significantly cheaper than blob stores, and then import the data to any other cloud where they want the computation to run.

To our knowledge, current pricing policies for exporting data are independent of the cloud the data might be going to. One alternative that we have not seen explored to date is for clouds to enter into reciprocal peering arrangements, where

[2]See S3 prices at https://aws.amazon.com/s3/pricing/.

[3]See the entries for February 2019 and April 2019, respectively, at https://dawn.cs.stanford.edu/benchmark/ImageNet/train.html

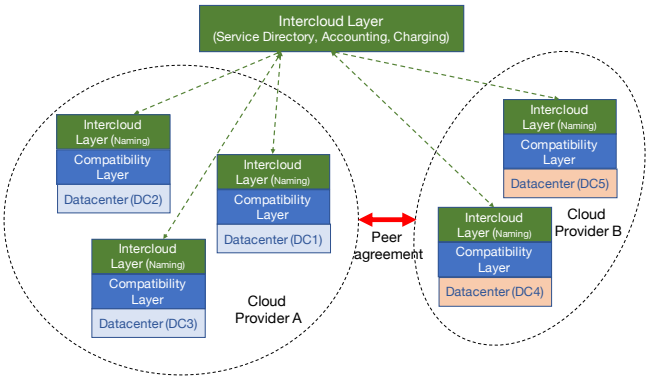

Figure 1: Possible Sky computing architecture.

they agree to allow free exporting of data to each other, and to connect with high-speed links (presumably at PoPs where they both have a presence). This would make data transfers both fast and free, lowering the data gravity between two peering clouds and enabling greater freedom in job movement. As we argue below, this may solve some of the underlying incentive problems inherent in creating the compatibility and intercloud layers.

7 Speculations About The Future

As noted above, while there are few technical barriers to the compatibility layer, it would make cloud computing resemble a commodity. As such, we expect incumbent cloud providers would strongly resist the emergence of a compatibility layer. However, unlike some other arenas (e.g., packet formats in networking) where standardization requires universal agreement, the software for a compatibility layer can be ported to any cloud, even if the cloud does not wish to officially support it. Thus, even users of a large incumbent cloud provider which does not officially support the compatibility layer might opt to use the compatibility interface rather than the cloud provider's lower-level interfaces just to preserve portability.

In addition, while large incumbents might not be happy about a compatibility layer, we expect smaller cloud providers will. For smaller cloud providers, offering proprietary interfaces may not be preferable to adopting a more widely supported standard. By doing so, these smaller providers would have access to a larger market share (i.e., users who have adopted the compatibility layer as their set of APIs) and could compete on price, performance, or various forms of customer service. We are already seeing this in the market, where Google has recently released Anthos, an application management platform based on Kubernetes that supports "write once, run anywhere" with a consistent development and operations experience for cloud and on-premise applications. Anthos is already running on Google Compute Cloud (GCP) and AWS, with Azure to follow shortly.

Once a compatibility layer has gained traction, an intercloud layer can be developed. The question is: who would

benefit from its development? Here we must speculate, since we are far from having these two layers in place.

We think that once a compatibility layer and an intercloud layer are in place, cloud providers will fall into two categories. There will be stand-alone cloud providers who try to lock customers in with proprietary interfaces and data export fees. These providers will typically be large enough so that they have the resources to innovate, allowing them to offer a variety of proprietary services. However, in contrast to today, we think there will also be commodity cloud providers who directly support the compatibility layer and agree to reciprocal peering with other commodity cloud providers. These commodity providers, taken together as a whole, form the Sky, which offers a unified interface to a set of heterogeneous and competing cloud providers.

Why do we believe the Sky will happen? It rests on the nature of innovation in the two classes of providers. In a competitive market, the stand-alone providers compete with each other, and with the Sky. The commodity providers also compete with each other within the Sky, and collectively compete with the stand-alone providers. In terms of tradeoffs, the stand-alone providers have higher margins (because their customers have exit barriers) but must innovate across the board to retain advantages for their proprietary interfaces.

In contrast, the commodity providers have lower margins, but can innovate more narrowly. That is, a commodity provider might specialize in supporting one or more services; jobs in the Sky that could benefit from these specialized services migrate there. For example, Oracle could provide a database-optimized cloud, while a company like EMC can provide a storage-optimized cloud. In addition, hardware manufacturers could directly participate in the cloud economy. For example, Samsung might be able to provide the best price-performance cloud storage, while Nvidia can provide hardware-assisted ML services. More excitingly, a company like Cerebras Systems, which builds a wafer-scale accelerator for AI workloads, can offer a service based on its chips. To do so it just needs to host its machines in one or more colocation datacenters like Equinox and port popular ML frameworks like TensorFlow, PyTorch, MXNet — thus providing a compatibility layer — onto Cerebras-powered servers. Cerebras only needs to provide processing service; all the other services required by the customers like data storage, data processing, etc, can run in existing cloud providers. In contrast, today, a company like Cerebras has only two choices: get one of the big cloud providers like AWS, Azure, or GCP to deploy its hardware, or build its own fully-featured cloud. Both are daunting propositions.

However, such a dynamic will only be effective if the intercloud layer can find these accelerated services, since most individual users won't be up-to-date on the relative performance of various services on the Sky. Thus, to make Sky computing work, we need all three layers to be effective: the compatibility layer to hide any differences in implementation between clouds; the intercloud layer to automatically find the best price/performance/properties for various services; and reciprocal peering to make data movements free and fast.

To be clear, we are not predicting the demise of proprietary clouds; we think that in the long-term we will continue to have both kinds of providers. The stand-alone providers will cater to those customers who need more assistance and where price and performance are not the critical factors. This might mean servicing the needs of smaller users who have less expertise in managing the cloud. However, we think the bulk of computation-intense workloads, particularly for sophisticated users, will migrate to the Sky, because of the greater access to innovation.

One might doubt that the commodity providers would provide more innovation than the stand-alone providers, but the PC market offers such an example. In 1981, IBM released its Personal Computer which pushed the computing industry in overdrive. The IBM PC was based on an open architecture. The only proprietary component was the BIOS (Basic Input/Output System) firmware. However, one year later, the BIOS was reverse engineered and this opened the floodgates for PC clones. As expected, this led to much lower prices; however, it also led to more innovation. The clone manufacturers like Compaq, not IBM, were responsible for creating the first portable PC, and for leading the adoption of the new Intel processors (e.g., Intel's x386). Furthermore, this open architecture led to an industry of add-ons such as hardware accelerators and storage devices. By 1986, IBM PC compatible computers reached half of the entire PC market with the PC clones outselling IBM PCs by significant numbers.

Our line of reasoning is, of course, purely speculative. However, the lack of adoption of a compatibility layer in even the smaller cloud providers is because they do not yet see such a layer increasing their revenues (because they are each competing individually against the larger clouds). However, if such providers simultaneously agree on reciprocal peering, then collectively the Sky becomes a viable competitive counterbalance to the large proprietary clouds, and allows the commodity clouds to focus their innovation efforts more narrowly.

8 Conclusion

In this short paper we have described some challenges that must be overcome before cloud computing can transform into Sky computing, which would move us closer to McCarthy's notion of utility computing. Some of these challenges are purely technical, and are likely achievable. However, to make the economic incentives work, Sky computing requires a group of cloud providers to adopt reciprocal data peering so that user's jobs can easily migrate within this large and heterogeneous collection of commodity clouds. Our purpose in writing this paper was to identify this as a crucial step in how the cloud market could be organized, and hopefully create momentum towards the vision of a Sky full of computation, rather than isolated clouds.

References

[1] Akka. https://akka.io/.

[2] Amazon S3. https://aws.amazon.com/s3/.

[3] Apache Cassandra. https://cassandra.apache.org/.

[4] Apache Kafka. https://kafka.apache.org/.

[5] AWS vs Azure vs Google Cloud Market Share 2020: What the Latest Data Shows. https://www.parkmycloud.com/blog/aws-vs-azure-vs-google-cloud-market-share/.

[6] Azure Blob Storage. https://azure.microsoft.com/en-us/services/storage/blobs/.

[7] Cloud Computing with AWS. https://aws.amazon.com/what-is-aws/.

[8] Cloud Foundry. https://www.cloudfoundry.org/.

[9] Docker. https://github.com/docker.

[10] Google Cloud Storage. https://cloud.google.com/storage.

[11] Kubernetes. https://github.com/kubernetes/kubernetes.

[12] MLFlow. https://mlflow.org/.

[13] MongoDB. https://github.com/mongodb/mongo.

[14] MySQL. https://www.mysql.com/.

[15] PostgreSQL. https://www.postgresql.org/.

[16] Presto. https://github.com/prestodb/presto.

[17] Redis. https://github.com/redis/redis.

[18] S3Proxy. https://github.com/gaul/s3proxy.

[19] Scality. https://www.scality.com/.

[20] SparkSQL. https://spark.apache.org/sql/.

[21] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA*, 2016.

[22] Joe Armstrong. *Making reliable distributed systems in the presence of software errors*. PhD thesis, Mikroelektronik och informationsteknik, 2003.

[23] Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, and Kostas Tzoumas. State management in Apache Flink: Consistent stateful distributed stream processing. *Proc. VLDB Endow.*, 10(12):1718–1729, August 2017.

[24] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. In *NIPS Workshop on Machine Learning Systems (LearningSys'16)*, 2016.

[25] LLC. Cloud Strategy Partners. Ieee cloud standardization.

[26] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 295–308, Berkeley, CA, USA, 2011. USENIX Association.

[27] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts, and Stephen Wolff. A brief history of the internet. *SIGCOMM Comput. Commun. Rev.*, 39(5):22–31, October 2009.

[28] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2018.

[29] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, Carlsbad, CA, 2018. USENIX Association.

[30] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.

[31] Hiroshi Sakai. Standardization activities for cloud computing.

[32] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

[33] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[34] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.