Chen ZN, Chen K, Jiang JL *et al.* Evolution of cloud operating system: From technology to ecosystem. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 32(2): 224–241 Mar. 2017. DOI 10.1007/s11390-017-1717-z

Evolution of Cloud Operating System: From Technology to Ecosystem

Zuo-Ning Chen[1], *Fellow, CCF*, Kang Chen[1], Jin-Lei Jiang[1], *Member, CCF, ACM, IEEE*, Lu-Fei Zhang[2]
Song Wu[3], *Member, CCF, IEEE*, Zheng-Wei Qi[4], *Member, CCF, ACM, IEEE*
Chun-Ming Hu[5], *Member, CCF, IEEE*, Yong-Wei Wu[1], *Senior Member, CCF, IEEE*
Yu-Zhong Sun[6], *Member, CCF, IEEE*, Hong Tang[7], Ao-Bing Sun[8], and Zi-Lu Kang[9]

[1] *Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China*

[2] *Jiangnan Institute of Computing Technology, Wuxi 214083, China*

[3] *School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*

[4] *School of Software, Shanghai Jiao Tong University, Shanghai 200240, China*

[5] *School of Computer Science and Engineering, Beihang University, Beijing 100191, China*

[6] *Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China*

[7] *Alibaba Cloud Computing Inc., Hangzhou 310024, China*

[8] *G-Cloud Technology Inc., Dongguan 523808, China*

[9] *Institute of Technology of Internet of Things, Information Science Academy of China Electronics Technology Group Corporation, Beijing 100081, China*

E-mail: chenzuoning@vip.163.com; {chenkang, jjlei}@tsinghua.edu.cn; zhanglf04@126.com
wusong@hust.edu.cn; qizhwei@sjtu.edu.cn; hucm@buaa.edu.cn; wuyw@tsinghua.edu.cn
yuzhongsun@ict.ac.cn; hongtang@alibaba-inc.com; sunab@g-cloud.com.cn; zlkang@189.cn

Received November 14, 2016; revised December 16, 2016.

Abstract The cloud operating system (cloud OS) is used for managing the cloud resources such that they can be used effectively and efficiently. And also it is the duty of cloud OS to provide convenient interface for users and applications. However, these two goals are often conflicting because convenient abstraction usually needs more computing resources. Thus, the cloud OS has its own characteristics of resource management and task scheduling for supporting various kinds of cloud applications. The evolution of cloud OS is in fact driven by these two often conflicting goals and finding the right tradeoff between them makes each phase of the evolution happen. In this paper, we have investigated the ways of cloud OS evolution from three different aspects: enabling technology evolution, OS architecture evolution and cloud ecosystem evolution. We show that finding the appropriate APIs (application programming interfaces) is critical for the next phase of cloud OS evolution. Convenient interfaces need to be provided without scarifying efficiency when APIs are chosen. We present an API-driven cloud OS practice, showing the great capability of APIs for developing a better cloud OS and helping build and run the cloud ecosystem healthily.

Keywords cloud computing, operating system, architecture evolution, virtualization, cloud ecosystem

1 Introduction

1.1 Cloud Operating System

In recent years, cloud computing systems have been becoming more and more prevalent both abroad[1]~[3] and domestically[4]. Many traditional applications have been migrated to the cloud systems. For developers and service providers, services and applications are hosted without the concern about infrastructure construction, application deployment, hardware updating

Survey

Special Section on MOST Cloud and Big Data

The work is supported by the National Key Research and Development Program of China under Grant No. 2016YFB1000500.

[1] https://aws.amazon.com/cn/documentation/, Nov. 2016.

[2] https://www.azure.cn/documentation/, Nov. 2016.

[3] https://cloud.google.com/, Nov. 2016.

[4] https://develop.aliyun.com/, Nov. 2016.

or data center maintenance. Now, the platform components running inside the cloud providers are considered as cloud operating systems (OS). The cloud OS[1], similar to traditional OS managing bare metal hardware[2-6] and the software resources, goes through a serial of evolution and now enters the next stage of evolution. We start our discussion with the functions that can be provided by cloud OS[1]. And towards the end of this paper, we hope to provide a guideline for the next stage of evolution.

Cloud computing systems[1,7-9] are often built from the existing single machine OS, usually Linux. Now, the other operating systems like Windows have been used in the data center infrastructure. Most of the cloud computing system components are constructed as user applications[2-3] from the traditional OS point of view. However, the platform for running cloud applications can still be called an OS as the platform serves the same two critical goals as a traditional OS. On the one hand, the cloud OS is used for managing the large-scale distributed computing resources, similar to the traditional OS managing hardware in a single machine. On the other hand, the cloud OS provides abstraction for running user applications. APIs (application programming interfaces) are provided for programmers to develop cloud applications. This is similar to the case that a local OS provides system calls for servicing applications. For example, file systems are used for providing storage APIs instead of exposing the block operation directly from disks in local OS. Cloud OS provides similar APIs to a distributed file system. Besides programming APIs, some other facilities will be provided for running the system smoothly in the cloud. Job scheduler is such a service for scheduling jobs[10-12]. Local OS schedules processes while cloud OS schedules distributed jobs. Data management is also a very important component in the cloud OS[13-18]. Users and applications need to track the data flow inside their applications running in the cloud environment. Data processing applications especially need the thorough understanding of their data flow among different components in the cloud OS. Fig.1 shows the cloud OS API levels as well as various services supported. Core APIs are relatively stable and used for managing the underlying infrastructure and hiding resources heterogeneity and distribution. On top of core APIs are other APIs that provide more functionalities, easing the burden of building more higher levels of cloud services and applications. Through this way, the cloud ecosystem can be built and cloud services and applications can proliferate.

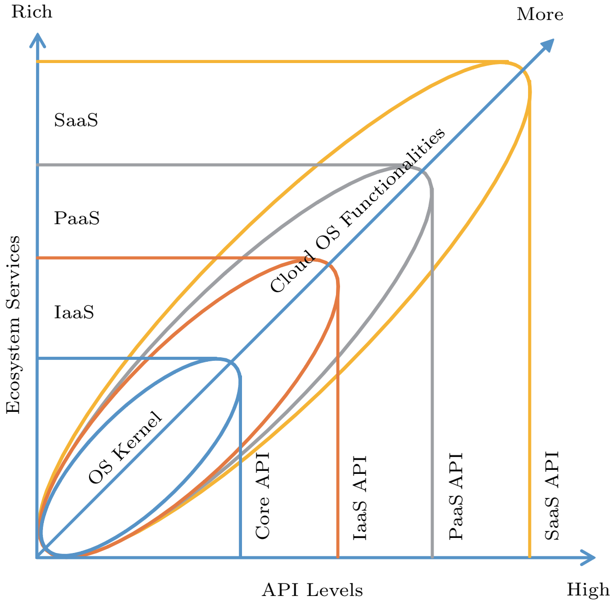


Fig.1. Cloud OS API levels and the ecosystem services.

1.2 Requirements of Cloud OS

In a single computer, it is quite clear that there needs an OS abstracting the bare metal hardware for facilitating the applications as well as users to use the computing resources conveniently. However, most of the components in the cloud OS are implemented as user-level programs. The fundamental necessity of building a cloud OS needs further discussion as cloud applications can always be built directly on the current operating systems without concerning the management of hardware in the kernel mode. The true necessity relies on the programming APIs and runtime support in the cloud environment[13,18-21]. Local OSes mainly focus on a single machine. They usually do not consider any cloud application logically as a whole unified application spanning over a large number of machines. Thus traditional OSes just provide the basic facilities for communications. In addition, cloud applications have different forms such as micro services for building web applications, batch processing for big data analysis[19-21], query-based data analytics[18,22-23], and real time processing of streaming data[24-26]. All the applications have their own internal logical organizations of multiple components running on multiple, or even thousands of machines. Exposing very simple communication interfaces is just not enough. Developers need higher levels of abstractions for building their applications without struggling with details related to the complex communication patterns and machine architecture. Exposing APIs that can run on top of multiple machines will be very helpful. This is a very common case for the current cloud OS practitioners such as Google GAE[5], Amazon AWS[6], Microsoft Azure[7] and Alibaba Cloud[8]. Cloud application developers can get the programming interfaces without considering the physical resources. For example, developers can store objects in the cloud without knowing the final disks storing the data[14]. Programming is always about abstractions and we need the cloud OS to provide the cloud programming abstraction for building cloud applications. The other reason why a cloud OS is necessary is that running cloud application is different from running applications in a single machine. For each single machine OS, a task scheduler will be used for managing the processes created in the system. However, the runtime characteristics are quite different for running different types of cloud applications such as batch processing, steam processing or graph processing. Cloud operating systems should be built for managing the computing resources related to thousands, or even tens of thousands of machines[27]. And they should provide different management schemas for different applications. The coordination[28-30] and interference of different applications should be taken into consideration while building such a job scheduler in the cloud environment. Thus, the administrators do not need to manage each individual machine. Based on the above observation, cloud OS is quite necessary and important.

1.3 State-of-the-Art Cloud OS Practices

Currently, there exist multiple cloud providers. The famous ones include Google, Amazon, Microsoft and Alibaba Cloud. The services they provide are called public clouds because they help to build cloud applications publicly available to Internet users. While facing the huge amount of users, these providers have to build their infrastructures to be scalable with high performance. The technologies used behind the infrastructure are so important that open source communities have adopted them and help build the open source versions of cloud infrastructure such as OpenStack and Hadoop. We can identify the infrastructures and their open source incarnations as cloud operating systems.

However, each cloud OS practitioner started their own cloud OS under different considerations. Amazon started with virtual machine (VM) instances for developers and administrators to start building their own systems, releasing them from the maintenance of the underlying hardware. At the same time, Google started its cloud OS considering large amount of data processing. These are the two main pioneers building the cloud OS. The services provided by these two companies are quite different. The followers, such as Microsoft, started with the similar services with different interfaces exposed. For business reasons, the interfaces are proprietary, leading to open source versions to provide similar but slightly different interfaces. Thus, the cloud application developers have to face the problem of vendor lock-in which means that applications built for one specific provider will not run on the other platforms. Even for building the private cloud environments using open source software, the developers have to rely on the specific open source implementation of specific version. This will impede the construction of cloud ecosystem or make it difficult to adopt cloud computing technologies. For the next phase of cloud OS evolution, well-defined APIs will be critical.

1.4 Impetus of Cloud OS Evolution

Like traditional local OS, the cloud OS is not static. It is evolving from generation to generation. The impetus of cloud OS evolution is mainly due to two considered conflicting goals. One is to improve the efficiency and the other is to improve the users (and developers) experiences. The former includes the improvement of performance and the efficiency of computing resources used. This goal is quite straightforward — never wasting any computing power provided. The later is related to the better programming interfaces, more convenient management facilities and user-friendly interfaces. However, these two goals are often considered conflicting. The reason is that better user experiences require more convenient interfaces which often need additional layers of abstractions. More computation overhead will be introduced for such abstractions. Exposing lower level interfaces is more performance-friendly but implies more work of developers, hurting the cloud ecosystem. Thus, the evolution of cloud OS is mainly to find a better tradeoff between these two conflicting

[5] https://cloud.google.com/, Nov. 2016.

[6] https://aws.amazon.com/cn/documentation/, Nov. 2016.

[7] https://www.azure.cn/documentation/, Nov. 2016.

[8] https://develop.aliyun.com/, Nov. 2016.

goals. The tradeoff drives the technology evolution, architecture evolution and ecosystem evolution. We will discuss each of them in detail in Section 2, Section 3, and Section 4 respectively.

1.5 Importance of Cloud OS APIs

For supporting more cloud applications, more application programming interfaces are added to the API set of cloud operating systems. However, some fundamental programming interfaces (called core APIs) are becoming more and more stable. These core APIs are quite important and almost all the cloud providers make them available to developers and users though their implementations may be different. This is key to building the cloud application ecosystem. The applications built with the same set of APIs can benefit from the same construction template. This can make the application running with similar technology proliferated. In addition, with core APIs, the applications can do the communication and make it easier to build larger and more complex cloud applications. Just like the cases in traditional operating systems, choosing appropriate APIs is quite important for the users of cloud operating systems. For further evolution, we need to make clear API definitions and their levels and categories. This tries to make the evolution in the right direction. For such a goal, this paper analyzes the current evolution of cloud OS from different angles including the technology trends, OS architecture improvement, and the cloud OS ecosystem.

2 Enabling Technology Evolution

As the definition of APIs is tightly related to the development of technologies, we will firstly study the evolution of the enabling technologies for cloud computing. Since the evolution is based on the impetus of appropriate tradeoff between the two goals, namely the higher efficiency and the more convenient user experiences, the analysis will also focus on technologies dealing with these two aspects. Virtualization and programming interface evolutions are quite related to bringing better user experience of cloud OS. Hardware and large-sale computing are related to improving the computing efficiency.

2.1 VM-Induced Technology Evolution

The first technology evolution is quite related to the introduction of virtual machines[31-32]. In the early stage of cloud computing, VM is even considered as synonym for cloud computing. One could say that if there were no VMs, there would be no cloud computing. Virtual machines are just like physical machines but run instructions using simulated hardware rather than physical hardware. This is mostly convenient for the users who want to deploy their systems on the Internet without purchasing, deploying and maintaining hardware. Amazon is the first company to provide VM services to the public. This is especially useful for start-ups providing Internet services. VM instances can be rented based on the current user scale and can be expanded on demand. This is a new model of infrastructure provisioning which is usually considered to be difficult and time-consuming[33]. However, the VM technology introduces huge amount of performance overhead. A lot of efforts have been conducted to improve the performance[34-38]. Despite the hardware improvement discussed in Section 3, many lightweight technologies have been developed to provide similar capability but with improved performance. One is to use the lightweight containers to replace the heavyweight virtual machines[39-41]. Containers are the virtualization inside OS kernel. It provides a higher level of virtualization instead of instruction-level virtualization used by VMs. Thus, a lot of simulation overhead can be eliminated. It is a typical trend that one improves the user experience with overhead introduced. The tradeoff requires to improve the performance of the next generation platform. Here, it does not mean that all the VMs will be replaced by containers. VMs have their own virtue and are used for the applications needing more isolation rather than higher performance. Some applications can benefit from containers for higher performance.

Despite the performance improvement of containers over VMs, cloud computing wants to further improve performance. Towards this goal, some other forms of computation models have emerged in recent years to push computation to the end user. Edge computing[42-43] is among the most promising technologies in this trend. While CDN (content delivery network) is used to put data near to the end user, edge computing tries to push the computing, instead of data, near to the end user. This is another way to improve the performance of virtualized infrastructure other than using containers. In edge computing, developers have to divide their computing into different parts and offload some of the computation to the edge servers instead of the centralized ones.

2.2 Evolution of Hardware Enhancement

Performance is always the main concern in any computing evolution. How to improve the cloud OS performance is no exception. There are mainly two ways to improve the performance of applications in the cloud environment.

One way is to use the newly developed hardware and the other is through software improvement. Hardware evolution is always accompanied by some corresponding software. During cloud system evolution, there are many hardware advances mainly for improving the performance. As mentioned before, VM introduces huge amounts of computing overhead because of the extra level of indirection. Intel has made great efforts to improve the performance of hardware virtualization by, for example, providing VT-x for CPU virtualization and memory virtualization, and VT-d for device virtualization. With the help of hardware virtualization, the performance overhead of current VM is negligible, usually less than 5%. Besides the improvement of system virtualization, there are a lot of other improvements using new hardware features. RDMA (remote direct memory access) was introduced one decade ago for improving the network performance and now becomes prevalent in cloud computing data centers[44-46]. Also, the network topology of a data center is quite important for cloud computing performance[47-48]. For performance of a single machine, FPGA[49-53] and GPGPU[54] were introduced to improve the performance of specific applications. They are now widely used in the cloud environment for specialized cloud applications. SSD (solid state drive) and NVM (non-volatile memory) devices will become more and more important for improving the storage performance.

For the software side, the system software has to adapt to the new hardware features. Legacy operating systems were designed for single CPU single core machines. Now, multi-core machines are ubiquitous. The system must adapt to the change to provide real multiple tasks capability[55-57] instead of time-sharing multitasks. The operating systems have provided multiprocess architecture for running concurrent tasks[58-59]. Threads will be used as multi-tasks inside a single process. Nowadays, new operating components and programming library have to support new features like SIMD instructions[60-63], GPGPU parallel processing and FPGA for performance accelerating[64]. These are all the system adaptation to embrace the evolution of underlying hardware. And also, it will be quite clear that future cloud OS will support the new hardware development. For example, huge-scale VM has emerged to support cloud applications such as Internet gaming that requires huge amounts of memory and a large number of CPU cores. Thus, for the next generation of cloud computing development, one should obviously take serious consideration of hardware evolution. On the other hand, cloud applications and cloud OS can also propel the hardware enhancement.

2.3 Evolution of Software Scalability

Despite the hardware development and its software supporting modules, software platform has its own adaptation for supporting higher performance cloud applications. There are mainly two ways for improvement. One is called scale-up, focusing on the performance in a single machine and the other is scale-out, focusing on the performance of multiple machines. Traditional OS mainly focused on the performance scale-up. Multiple tasks and multiple threading are the common technologies in OS to improve the throughput. And programming libraries are used for harnessing the power of SIMD, GPGPU and FPGA as mentioned in Subsection 2.2.

However, despite the great efforts of improving the single machine performance, the performance of a single machine will never be enough for many cloud applications requiring a huge amount of processing as in distributed operating systems[65]. With the development of big data applications, this will be an unavoidable case in many areas. Cloud OS has done great work to improve the performance and the programmability of the platform. This is usually called a scale-out scheme to deal with more machines. For example, distributed file systems are built for storing a large amount of unstructured data while distributed databases are built for storing structured data. Computation frameworks such as MapReduce and Spark have been built for the construction of cloud applications. Another improvement that cannot happen in a single machine is that the task management and scheduling in the cloud environment is quite different from that in a single machine. The cloud OS has to manage thousands of machines of a cluster, or manage multiple clusters in a data center, or even manage multiple data centers all over the world. The management infrastructure is surely quite comprehensive and complex. The cloud OS has to provide the scalable, fault-tolerant management function to deal with a huge number of nodes and the severe situations like machine failure, network failure and even data center failure. In addition to the scale of cloud computing, the cloud OS has to consider various types of applications. The task scheduler in the cloud environment has to face different granularities of tasks[66]. For example, VMs and containers can be used as units for scheduling[67]. This is quite different from the task scheduling in traditional OS[68-69]. VM live migration can help to migrate the workload from one machine to another[70-71], thus improving the inter-operability of different components. Such a case will never happen in a single machine.

In summary, the software evolution related to performance improvement tries to improve the efficiency by the ways of scale-up and scale-out. For future development, the organization of OS has to adapt to the new cloud applications. Single granularity and uniformed interface will not be enough to build the cloud ecosystem. Developers have to access various granularities of infrastructure abstraction which should be provided by the cloud operating system.

2.4 Programmability Evolution

In addition to the VM evolution, hardware and platform performance improvement, the other quite important evolution is the programming interface evolution. In the early days of cloud computing, there were quite a small number of programming interfaces. Most of the interfaces were related to the underlying infrastructure. For example, the AWS firstly provided programming interfaces for manipulating VM instances and for the communication among VM instances. These are considered as low-level APIs and suitable for the construction of virtual infrastructure. These APIs are not for any application such as business logic or scientific analysis. Though they can benefit the system developers or administrators, application developers cannot get too much help from these infrastructure-level programming interfaces. Application developers have to resort to the traditional OS APIs especially networking APIs and/or cluster middleware to build their cloud applications.

Using existing OS APIs or middleware is not enough for building cloud applications for several reasons. The designers of the existing environment did not have an appropriate consideration of the features of the current cloud systems. They have never seen an ultra-scale system like cloud computing. Cloud computing APIs need redesign and have better support for applications running in the cloud environment. That is why the programming interface evolution began to support cloud applications. And early programming interfaces for applications tried to mimic the traditional interfaces from traditional OS but were adjusted to the cloud environment. A distributed file system is a typical example that extends a local file system. Applications can use the similar programming interfaces to store data objects in the distributed file systems. Similar things happen in cloud OS with SQL (structured query language) APIs[24].

With the further improvement and application flourishing, the cloud ecosystems have extracted more common programming interfaces for building new cloud applications. Unlike previous general APIs, these new sets of APIs have their own application level purposes. For example, messaging APIs are used for sending and receiving messages. Distributed authentication APIs are used for user identification and verification. Distributed logging APIs help developers to do the programming logging which is quite important for application development and monitoring. These APIs are quite necessary in the cloud environment but have no counterpart in traditional operating systems.

It is obvious that different sets of APIs are and will be constructed to facilitate the development of different cloud applications. For example, we have seen the cloud computing APIs supporting batching processing, streaming processing[24-25], and graph processing[72-79]. Cloud APIs for deep learning and artificial intelligence[80] are now under fast development. The computation patterns have switched from the structured data computation to the combination of both structured and unstructured data computation for supporting more big data applications. Thus, future cloud OS will have more programming APIs supporting new cloud applications. The platform and the exposed APIs must support various applications with complex logic dealing with a large amount of data.

In summary, we have investigated the evolution of system virtualization, hardware enhancement, and performance and programming interfaces. It is no doubt that more and more programming APIs will be provided by the cloud OS in the future. Some of the APIs are not purely functional and they will be used to detect the application-level runtime characteristics so as to improve the efficiency. In addition, the core APIs will be stable because the stable APIs are required to build and run cloud ecosystems. Based on the core APIs, a higher level of APIs can be defined and the whole cloud ecosystem can be constructed.

3 Cloud OS Architecture Evolution

With the development of computing infrastructure, new network applications proliferate in the cloud. The cloud OS and the single node OS have the evolution of their internal architectures for fitting such development trend. The cloud applications have made more APIs available for programmers, thus enriching the cloud OS ecosystem.

3.1 Architecture Evolution of Single Node OS

At an early stage, the computing resources like mainframes were quite expensive, and time sharing OS was used as the fundamental building block to manage the underlying hardware. This was the first time that OS came into being. The OS abstracted the interaction between the programs and machines and made it possible to share computing resources among multiple users. Based on the computing resources available at that time, the OS was designed to have a monolithic kernel (such as UNIX and Solaris) with a single namespace and large-scale binary covering process management, memory management and file system. Such architecture can have high performance but with complex internal structure. Later, micro-kernel came into being like Mach and QNX. Such kernels only contain necessary and critical components such as isolation and inter procedure communication. Other modules and system functions run as independent components on top of the kernel. This can simplify the development of operating systems and make it easier to stabilize the whole system. By using the micro-kernel, a new service can be added by using a new process without modifying the kernel itself. Micro-kernel has better scalability but less performance. It is the performance that prevents the preference of micro-kernel. Although the debate between these two architectures will continue, there exist de-facto standard APIs. POSIX (portable operating system interface for UNIX) is such a quite important programming standard and now supported by most operating systems available. This makes it possible to run applications on different platforms.

Later, the computing resources became more powerful, the network connects every computing facility in the world, and the intension of OS has been extended. The model of interaction between machines and humans is also changed and the graphical user interfaces (GUIs) are made to be a must. Meanwhile, Windows OS became ubiquitous, providing the developers with capabilities of creating windows, drawing on windows and services for using various kinds of graphical devices.

Recently, more computing power has been put in the server-side computing, and CPUs are becoming multi-core and heterogeneous. Traditional operating systems lack the support of heterogeneous architectures and applications can interfere in performance. In addition, the shared memory model and the cache coherence protocol based on synchronization and mutual exclusion have limited the scalability of the system[81]. The virtualization technology can be used to deal with the processor heterogeneity and to perform isolation. In addition to the higher resource utilization, virtualization can increase the mobility of applications, reduce the management cost and enable disaster recovery. In large-scale system maintenance, applications with the underlying OS are packed in a single virtual image. This is a new form of software delivery called virtual appliance, which can decouple the application implementation from the underlying hardware platform. Many programming interfaces like libvirt[82] are proposed for manipulating VMs.

3.2 Architecture Evolution of Cloud OS

Cloud computing can be considered as a successfully commercialized distributed computing paradigm. Traditional distributed operating systems such as Amoeba[83] try to make the distributed environment transparent to the OS layer and the application layer. In the OS layer, distributed shared memory and process migration can achieve the goal. In the application layer, remote procedure call and inter process message communication can be used to achieve the same goal. The transparency of distributed environment is valid only in the situation of enough low communication latency. Now the computing resources are more flexible, ubiquitous, and heterogeneous. Together with the newly developed cloud applications like MapReduce, all the factors improve the development of cloud system software. MapReduce[13], proposed by Google first, provides a loosely coupled framework dealing with a large number of heterogeneous and unreliable nodes and is used to build data processing applications.

The current architecture of cloud OS is based on the platform software. The software components are just simply stacked together to provide the functions needed. The architecture is simply the OS combined with network middleware. The network components are typically the distributed database, and distributed storage and middleware for message communication. They are used to hide the distribution of the underlying computing resources. Sometimes, the infrastructure can do the optimization based on different types of applications. The virtualized underlying platform makes the cloud OS transparent by using VM live migration. However, VMs can introduce great overhead, leading to lower efficiency. The supporting of network data transmission is not enough either[31].

In recent years, the emerging and the proliferation of containers have improved the development of cloud OS. Containers use the isolation capability from the existing OS. This can reduce the number of abstractions, thus improving the efficiency. By using containers as the basic building blocks for cloud OS, the unified management framework can support extremely large-scale computing systems and storage systems. The jobs supported can be a mixture of various types like servicing and batch processing. Complex and comprehensive resource description language can be used for task assignment, scheduling and fault tolerance[10-12,84-86]. In the cloud OS software stack, the application containers are suited for hosting micro services. Now, the management tool like Docker[87] becomes very popular in practice. With its standard build file and the flexible RESTful APIs, the management tool can greatly help improve the automation of software packaging, testing and deployment. However, the current implementation of containers lacks performance isolation and security isolation[88]. Thus, the containers are usually run inside VMs for performance isolation. This obvious redundancy brings some new opportunities to bring the containers one level lower in the OS by using the micro kernel or customized kernel to remove the problem of performance isolation[89-91].

The architecture of traditional loosely coupled cloud OS makes it complex to use and introduces a lot of redundant work. It lacks application workload perception and the policy cannot notify the underlying component effectively. Thus, the architecture should consider the vertical integration of OS to reduce the levels of abstraction. More perception points can be put in the OS which can consider the scheduling policies in the distributed and cloud environment. This can help to build a unified and flat management framework for cloud computing platform. Such architecture is beneficiary for function convergence of each component, workload perception based optimization, and orderly evolution. Traditional monolithic single node OS is not easy to decouple the policy from mechanism. Single node OS needs the enhancement of internal structure to improve the capability of application workload perception. The mechanism can be implemented in the small kernel and provide users with more customizable policy interfaces. And more advanced research includes the data distribution, function distribution and space reuse to replace time reuse in OS. This can make the single node OS more scalable and flexible[92-95].

As a conclusion, the cloud OS should vertically integrate single node OS and network middleware for application workload perception. At the bottom layer, the virtualization technology and containers can be used to support multi-core and heterogeneous processors. Also, the decoupling of application development and hardware should be supported as high efficient sandbox. Micro-kernel technologies can be used to improve the isolation of containers and the flexibility of single node OS. At the higher layer, the unified resource management framework can achieve the goals of mixed job deployment, global scheduling optimization and application workload perception[96-99]. Service mashup can be implemented by using the application package management. For each stage of cloud OS evolution, new APIs are introduced. With the new APIs and their standardization, new applications can push the next stage of OS architecture evolution, expanding cloud technologies adoption.

4 Cloud Ecosystem Evolution

For the future development of cloud OS, the main goal is to support the cloud ecosystem and build more applications. This section will study the evolution of cloud ecosystem. The evolution of the cloud systems is tightly related to the technology used and the change of the OS architecture as discussed above. This section will show how technologies and OS architecture fit in the cloud OS evolution. Based on the observation of different cloud providers and their open source counterparts, the evolution of cloud ecosystem can be studied through two different perspectives. The first is called the evolution of layers, and it is quite related to the technology development and driven by the gradually emerging applications in each phase of cloud OS evolution. The second is related to the adoption of cloud technologies.

Before diving into the detailed discussion, we first briefly sketch the importance of APIs and their role in improving the ecosystem. Table 1 shows several aspects of how APIs can improve the underlying cloud OS and the applications running on top of them. APIs can clearly define the boundary between the system platform and the applications. That is critical to build applications without worrying about the quick evolution of the underlying technologies. Cloud ecosystem and OS cannot be built without the introduction and evolution of APIs.

Table 1. Role of APIs for Cloud Ecosystem

Aspect	Description
Improved productivity	Different components of cloud OS and applications can be developed independently by different people
Enhanced usability	Implementation details are hidden and users only need to understand the information exposed
Functionality reuse	More advanced functions can be developed using existing APIs; such a way also improves productivity
Better interoperability	Applications developed with standardized APIs can run on any system that supports these APIs
Healthy growth	Larger scale ecosystem can be formed by utilizing existing APIs, developing and delivering new APIs

4.1 Evolution of Layered Cloud Ecosystem

The cloud ecosystems seem quite proliferated recently. And it is clearer that the core APIs are now relatively stable. However, until recently, cloud ecosystems were not very clear. Different cloud providers have their own definition of cloud ecosystems. The cloud ecosystem evolution is quite driven by the applications on its top. At first, cloud providers just wanted to provide services over Internet, releasing users and administrators from deployment and maintenance. The cloud OS at that time had very vague appearance in the ecosystem. Thus, many of the pioneer cloud practitioners tried their best to hold the services from developers for end users. For example, Google AppEngine helps the developers to run their customer services inside the Google cloud infrastructure. Thus the start-ups do not need to worry about the whole underlying infrastructure. They only need to worry about the application logic which can attract end users. Amazon did the same thing with a much lower level of programming interfaces. However, in this stage of cloud OS evolution, there is no clear methodology to define the core APIs and the application level APIs. The earlier cloud OS efforts just provided the services that were already inside the providers. Anyway, this stage at least revealed some about the later huge evolution of cloud OS.

With the development of cloud ecosystem, and especially because of the applications running on top of the cloud infrastructure, the underlying fundamental services became quite important. All the practitioners now have a clearer picture of what the ecosystem should look like. People realize that there are at least three levels for many components in any cloud ecosystem. Each level services different purposes. In the very first level, the underlying computing resources can be virtualized. The reason is quite simple that we have to use some forms of indirection to hide the physical resources. And then the physical resources can evolve on their own without disturbing the software stacks running on top. In this level of ecosystem, we can see virtualized hardware components including VM, virtual networking[100], and virtualized storage. This level also has its own evolution by introducing containers and edge computing servers. Decoupling this level from higher levels is quite important for the ecosystem evolution, for the developers now do not struggle with time-consuming and error-prone details of hardware deployment and management. The first level of virtualization mainly makes very convenient interface for system administration and application deployment. We call this layer the virtualization layer.

On top of the virtualization layer, the cloud OS has to provide supports for developers. In fact, this layer has its own independent development history from the virtualization layer. Some of the cloud applications in fact do not need to run in the virtualized environment for the reason of performance overhead. For example, the batch data processing or the stream data processing needs some computation frameworks to get the programs built more productively. And also, for hosting web services in the cloud environment without considering virtual or physical computing resources, the cloud OS has to expose the necessary interfaces without revealing the virtual or physical servers. To support such cloud applications, cloud providers have built the distributed file system and various computing frameworks for developers. The programming interfaces have to be abstract so that the programs do not need to notice the location of hosting. With the evolution of cloud OS, the storage services are now considered to be in the virtualization layer. And the programming frameworks can be considered in a higher level of ecosystem. This layer can be called service supporting layer. There is another very important component in this layer, namely the task management. As for supporting running cloud applications, task management will be used for scheduling various kinds of workloads. The workloads include Web services, batch processing, real-time monitoring, and other kinds of applications. As the scale of cloud computing system is often quite large, the task management is quite important to guarantee the efficient use of computing resources and improve the overall system performance. The service support layer is considered as the fundamental application service in the cloud ecosystem.

The application service layer is the final layer sitting on top of the service supporting layer. The application layer provides services to end users. Usually applications in traditional OS do not expose any programming interface for programmers. However, most of the cloud applications do so because through this way, the application developers can mash up multiple services to provide a new service for end users. It is quite common in the cloud environment that services can be connected together to construct more complex and higher level of applications. And some of the services are so common and important that they can form another level of core APIs. For example, user authentication and messaging services can be considered as application layer APIs for almost all applications.

From the above analysis, one can come to a conclusion that the architecture of the cloud OS should be and now is quite similar to that of the local OS. The system should be divided into different layers and each layer has its own purpose. The local OS has the hardware layer, OS and runtime layer, and user application layer. Cloud OS has each counterpart. For the future development of cloud OS, the layered service is unavoidable. The critical work that has to be done here is to separate each layer clearly and define the interface between layers precisely. This is common system practice and can reduce the whole system complexity. In addition, each layer can have its own evolution without disturbing the other layers too much. For example, the containers and edge computing can be adopted in the virtualization layer. Because of the clearly layered system, the introduction of new technologies can be fitted into the whole framework quickly. In addition, such introduction might have some other benefits. For example, with the introduction of containers, the cloud OS can mask the heterogeneous hardware computing resources without too much overhead like VMs. Thus, the service management APIs and the resource management APIs can be decoupled by using the containers. The resource management APIs can handle the heterogeneity of different computing resources and expose the computing resources as containers. The service management APIs can then manage and schedule these containers. The containers can be scheduled to appropriate computing resources to achieve high performance and efficiency.

Although the separation of these three layers is quite clear nowadays, there is no clear boundary between them. And often, it is not the lower layer that helps improve the upper layers. In contrast, the applications layer often plays the role of ecosystem impeller. With more and more cloud applications becoming prevalent in the services for ender users, the infrastructure and the platform have to adapt to such trends and make optimization for running the application more efficiently.

As a summary, the current cloud ecosystem works in the similar way to traditional operating systems. By exposing multiple layers of interfaces, application developers as well as system developers can all make contributions to the ecosystem. The impetus is still the tradeoff between the user experience improvement in the application level and the efficiency improvement in the system level. Thus, for the future evolution of cloud ecosystem, the layered model of cloud OS is still valid. The interface will proliferate and more application specific programming and management interfaces will be added to the ecosystem. The core APIs related to the infrastructure and resource management will be much more stable in the ecosystem. With the layered model decoupling the different functions in the cloud OS, the evolution of each layer can be relatively independent.

4.2 Evolution of Cloud Technology Adoption

Despite the public cloud, the technologies used in the cloud system have drawn much attention from the traditional industry. The ease of use, large-scale capability of data processing and the fault tolerance feature can benefit the current IT infrastructure of many institutes. In fact, the adoption of cloud technologies was slow in early stages. This is mainly because the ecosystem even for the public cloud was not mature. As the enabling technology implementation is quite difficult, there was no public reference implementation at that time. Besides the difficulties, one of the main reasons for the availability issue of public implementation is the lack of standardized interfaces, that is, APIs.

Later, with the development of public cloud ecosystem, the programming interfaces for the cloud OS become more and more mature. All the public cloud providers expose similar programming interfaces covering infrastructure management, and task management and some critical application-level programming interfaces. And a lot of efforts have been done to implement the relatively mature and stable core APIs. Two very important open source projects, Hadoop and OpenStack, started to make great influence on the adoption of cloud technologies. They both can be considered as the incarnation of public cloud operating systems. OpenStack, which is quite similar to the infrastructure virtualization in Amazon's cloud, can be used to create the virtualized infrastructure including the computing virtualization, networking virtualization and storage virtualization. Hadoop has implemented a bunch of data processing infrastructure for improving the performance, programmability and the fault tolerance of large-scale clusters. These two open source projects have a large number of components, enabling the users to adopt the cloud technology more easily. It is quite clear that eventually stabled programming interfaces have made it possible to expose the internally used cloud infrastructure to be publicly available.

However, the current technology adoption is still limited to general cloud technologies. Some cloud application might need special considerations. For example, many of the current high performance computing applications do not benefit from the flexibility of cloud infrastructure. Thus, the general technologies currently implemented might not be appropriate. In the future evolution, the cloud technology needs to be integrated into the applications. The infrastructure should percept the applications and make appropriate adaptation to run the applications more efficiently. We have seen the cloud technology improving the IT service for many existing applications. The cloud OS will make more layered abstractions of underlying services. With the improved programming interfaces and their implementation, the cloud technologies will be more convenient for building specialized cloud services. Application developers and the IT specialist can use the publicly available cloud OS software stack to construct in-house cloud while the public cloud service cannot meet their demands.

5 Methodology of Cloud OS API Abstraction

Based on the study above, APIs are critical to propel the evolution of future cloud OS. API abstraction will be the first step for implementing any cloud OS and also it is the foundation of building cloud applications. There are a few principles for doing cloud API abstraction.

1) The APIs abstraction must be compatible with the current cloud practice. This principle shows the respect for the efforts currently done and can fit the users' expectation. A lot of applications have shown the validity of the current cloud OS.

2) Core APIs should be stable. We have seen many cloud applications requiring different underlying supports. However, the core APIs should be stable. This will help to build the cloud ecosystem. Developers have the relatively solid foundation for building their applications. Otherwise, if the APIs are changing swiftly, learnt knowledge will soon get obsolete and hurt the existing cloud applications.

3) APIs should be layered and cover enough demands. As analyzed before, the ecosystem needs layered API abstraction. Each layer needs to service different purposes. In fact, each layer can have its own ecosystem and can make evolution by itself. Through this way, the APIs can be made prolific to cover enough demands without too much overhead.

4) Although the APIs, especially the core APIs, must be kept relatively stable, they can and should have necessary evolution to embrace the technology improvement as well as the new requirements raised by cloud applications. New hardware will be introduced to the cloud ecosystem, and the cloud OS must support such hardware for running applications more efficiently.

Based on the above principles, it is now clear how we can define the cloud OS APIs. For compatibility with the current cloud operating systems, the APIs can be gotten from the current cloud system practitioners including the public clouds and their open source counterparts. Usually the APIs defined by these two communities are similar because the open source versions are derived from the public services. The open source versions often evolve more quickly and new features can be merged to the open source implementation fast. Cloud providers with other considerations cannot do the change that quickly. All such APIs are the source that a new cloud OS can learn from. The APIs will be divided into different categories based on their functions. Also, the categories should be put into multiple layers based on their distances from the underlying computing resources.

After the categorization and layering work, one should take very careful considerations of the relative stableness of all APIs as well as their functions. The core APIs should not be tied to any specific application and should be stable. Thus, two categories of APIs should always be considered as core APIs. One is related to the infrastructure virtualization and resource management such as VM, containers, virtual storage, and virtual networking. The other is related to the task management such as workload monitoring, different granularity of job management, task scheduler based on different workloads, and so on. Some programming frameworks should also be taken into consideration such as batch processing, streaming processing, graph analytics[101-102] and distributed query processing, because they are so widely used that they are amongst the most important applications, including the application framework that can help to build the cloud ecosystem. APIs supporting latest hardware should be included as they are required by applications and can benefit the adoption of cloud technologies while building specialized in-house cloud infrastructure.

In summary, one can define the core APIs of cloud OS as well as the necessary APIs for building applications from the existing public clouds and their open source counterparts. The ecosystem can then be built from these APIs. The APIs should be standardized so that applications can be developed more quickly without too much concern about API implementation. In the next section, we will give some exemplar cloud practice that first defines the APIs and then builds an ecosystem around it.

6 API-Driven Cloud OS Practice

By API-driven, we mean APIs are defined first and then a cloud OS implementing these APIs is developed. Such a way is possible because nearly a decade has passed since the advent of cloud computing and people have gained enough experience about the concept and the real-world applications. In addition, such a way can produce APIs of the most convenience and systems that can efficiently support them.

6.1 Core APIs Definition

In Section 5, we have pointed out some principles to define cloud OS APIs. According to these criteria and the practice of public cloud services and open source cloud projects, we define five categories of APIs as follows.

- Container-related APIs include Create, Start, Kill, Delete as defined by the open container initiative and other self-defining ones such as List (for listing all containers of a certain user), Watch (for getting the status of a certain container), Migrate, and Stop. These APIs make it possible for users to create and manipulate container-based applications.
- Virtual machine related APIs are supplied for users to create and control virtual machines. Virtual machines provide a way to utilize resources more flexibly and efficiently.
- Scheduling-related APIs are provided for users to submit jobs and monitor the execution process. It is the duty of scheduling to guarantee the desired quality of job execution.
- Storage-related APIs are used for users to save their contents. The storage types supported include object storage, file storage, and block storage.
- Operation management related APIs deliver functionalities such as resources deployment and management, configuration management, system monitoring, system auditing, and security management. These functionalities are essential for proper system operations.

6.2 Cloud OS Architecture

As a reference, we design a cloud OS that implements the above-mentioned APIs under the sponsor of National Key Research and Development Program of China. The architecture of our cloud OS is shown in Fig.2.

The OS consists of multiple layers, namely physical resources, OS kernel, system services, eco-system support, and applications. The physical resources layer provides various kinds of resources, including CPU, memory, storage, network, and so on. OS kernel, which locates between the physical resources layer and the system services layer, fulfills the task of resources abstraction and services provisioning. The system services layer provides such services and the corresponding APIs as networking services, large-scale in-memory computing, data storage, elastic computing, and so on based on the APIs shipped by the OS kernel. On the top of system services are other facilities (e.g., account management, user authorization, billing, container services, resources orchestration, and VM/container repository) that are necessary to build an eco-system. These facilities also provide APIs of their own. The most upper layer is the applications layer that ships such capabilities as big data processing, scientific computing, graph computing, deep learning, and so on. Please note the application here is called from the perspective of OS and it might not involve business logic.

In our system, we mainly focus on the key functions

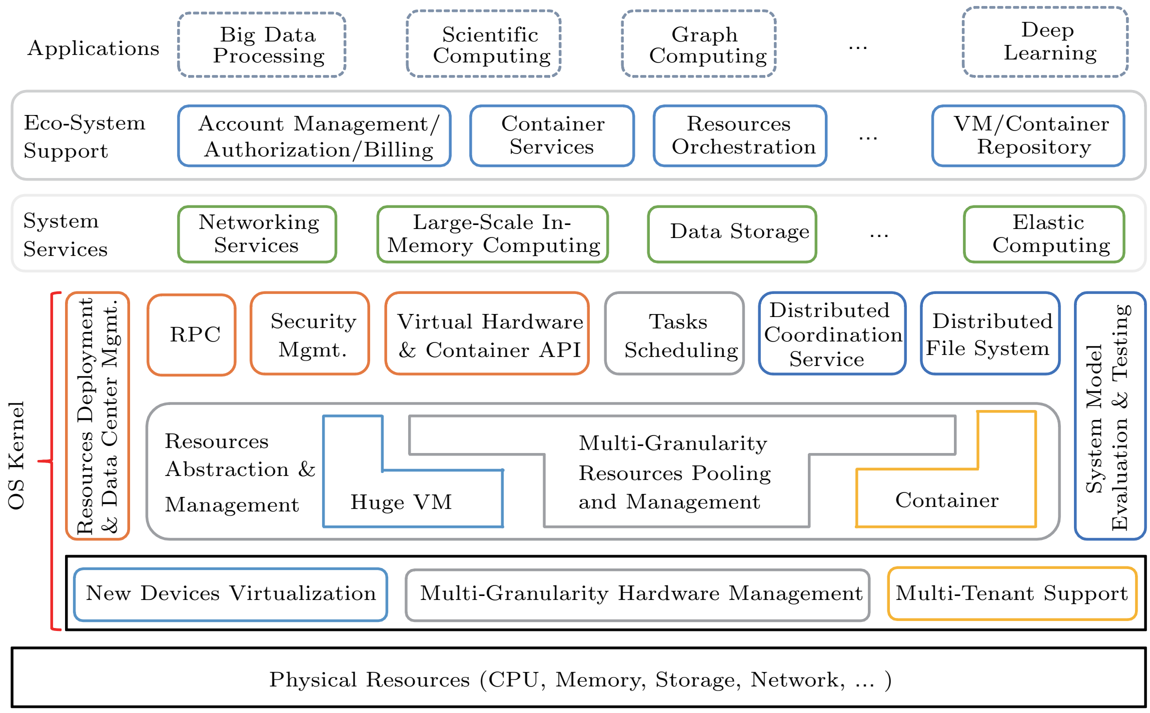

Applications
Big Data Processing
Scientific Computing
Graph Computing
...
Deep Learning
Eco-System Support
Account Management/ Authorization/Billing
Container Services
Resources Orchestration
...
VM/Container Repository
System Services
Networking Services
Large-Scale In-Memory Computing
Data Storage
...
Elastic Computing
OS Kernel
Resources Deployment & Data Center Mgmt.
RPC
Security Mgmt.
Virtual Hardware & Container API
Tasks Scheduling
Distributed Coordination Service
Distributed File System
System Model Evaluation & Testing
Resources Abstraction & Management
Huge VM
Multi-Granularity Resources Pooling and Management
Container
New Devices Virtualization
Multi-Granularity Hardware Management
Multi-Tenant Support
Physical Resources (CPU, Memory, Storage, Network, ...)

Fig.2. Overall architecture of the cloud OS developed. Mgmt means management.

in the OS kernel. Details of them will be explained in the following subsection.

6.3 Key Technologies

To achieve the purpose of high efficiency and ease of use, our cloud OS implementation pays attention to the following technologies.

Container Technology. Containers provide a lightweight way for users to develop and run distributed applications and are considered as a future direction. Though there are many containers available (e.g., Docker and LXC), they are far from perfect. Our OS will investigate ways to enhance container isolation and to do live migration without performance degradation. In addition, since container services will be consumed by many tenants in cloud environments and computing has expanded its scope from central data center to the edge, we will also study how to better support the multi-tenant feature and edge computing by containers.

Huge VM Technology. Traditional VM embodies the idea of resource partition, allowing users to manipulate resources in a more fine-grained way. Besides this paradigm, it is necessary to consider scenarios where large amounts of resources are combined and presented as a powerful virtual machine (called huge VM) along with the advent of new devices (e.g., GPU, RDMA, and FPGA) and applications (e.g., scientific computing, big data processing). The issues to study include how to do new devices virtualization so as to lay a good basis for huge VM construction, how to build a huge VM that can support more than 500 virtual computing cores and up to 2 TB memory, and how to do resources scheduling to improve performance.

Tasks Scheduling Technology. Tasks scheduling runs on top of multi-granularity resources pooling and management. It is the duty of tasks scheduling to guarantee both resources utilization and services quality. Since large-scale resources are involved and the tasks running are of different characteristics, it is not an easy task to do tasks scheduling. Here we focus on hybrid workloads scheduling, the scalability and fault tolerance of the scheduler itself, and the adaptation of typical computing frameworks with cloud computing environments.

Distributed File System Technology. Just like file system to traditional OS, distributed file systems play a key role in cloud computing, simplifying storage access. Since there are many distributed file systems (e.g., GFS/HDFS, Gluster, Ceph) available, we shift our focus from scalability to reliability, with an aim to achieve better reliability and storage efficiency. We utilize the large-scale erasure code to achieve the purpose. To guarantee read/write performance, parallel computing is introduced for data encoding and decoding.

Resources Deployment and Data Center Management Technology. For resources deployment, we put forward a method to identify the dependencies among multiple applications and design a Puppet-based and cache-enabled deployment tool, with which resources can be deployed quickly and energy efficiently. For data center management, we monitor system runtime states and devise a deep learning based algorithm to identify and locate failures and do security assessment. In this way, data center management becomes intelligent.

Besides the above key technologies, we also investigate the issue of system testing, with a purpose to give a full evaluation of the whole system in terms of API compliance and interface performance. The result is an interface semantic contract based automatic testing scheme that can shield the difference in interface formats and reduce testing cost.

In summary, our OS implementation shows the possibility to build an ecosystem around some APIs. In this implementation, we just focus on the core functions of OS kernel and the corresponding APIs. It can still be improved by, for example, introducing more advanced functions and/or services over the OS kernel to boost application development efficiency.

7 Conclusions

In this paper, we gave the brief study of the evolution of cloud OS related technologies and ecosystems. It is very clear that the APIs play a central role in the evolution of cloud OS. APIs are the abstraction of the underlying computing infrastructure, and implement the requirements of the cloud applications. With the internal management of jobs and resources, the cloud OS tries to run applications efficiently. Building the cloud ecosystems based on the carefully chosen OS APIs is critical to make the system productive and healthy.

References

[1] Armbrust M, Fox A, Griffith R *et al.* A view of cloud computing. *Communications of the ACM*, 2010, 53(4): 50-58.

[2] Tanenbaum A S, Woodhull A S. Operating Systems Design and Implementation (3rd edition). Pearson, 2006.

[3] Auslander M A, Larkin D C, Scherr A L. The evolution of the MVS operating system. *IBM Journal of Research and Development*, 1981, 25(5): 471-482.

[4] Deitel H M, Deitel P J, Choffnes D. Operating Systems. Pearson/Prentice Hall, 2004.

[5] Bic L F, Shaw A C. Operating Systems Principles. Prentice Hall, 2003.

[6] Silberschatz A, Galvin P B, Gagne G. Operating System Concepts. John Wiley & Sons Ltd., 2008.

[7] Hu T H. A Prehistory of the Cloud. MIT Press, 2016.

[8] Mell P, Grance T. SP800-145. The NIST definition of cloud computing. *Communications of the ACM*, 2010, 53(6): 50.

[9] Zheng W. An introduction to Tsinghua cloud. *Science China Information Sciences*, 2010, 53(7): 1481-1486.

[10] Hindman B, Konwinski A, Zaharia M *et al.* Mesos: A platform for fine-grained resource sharing in the data center. In *Pro. USENIX Conference on Networked Systems Design and Implementation*, Mar.31-Apr.1, 2013, pp.429-483.

[11] Schwarzkopf M, Konwinski A, Abd-El-Malek M *et al.* Omega: Flexible, scalable schedulers for large compute clusters. In *Proc. ACM European Conference on Computer Systems*, Apr. 2013, pp.351-364.

[12] Verma A, Pedrosa L, Korupolu M *et al.* Large-scale cluster management at Google with Borg. In *Proc. the 10th European Conference on Computer Systems*, Apr. 2015, pp.18:1-18:17.

[13] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. In *Proc. the 6th Symposium on Operating Systems Design & Implementation*, Dec. 2004, pp.137-150.

[14] Ghemawat S, Gobioff H, Leung S T. The Google file system. *ACM SIGOPS Operating Systems Review*, 2003, 37(5): 29-43.

[15] Chang F, Dean J, Ghemawat S *et al.* Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems* (*TOCS*), 2008, 26(2): 205-218.

[16] Baker J, Bond C, Corbett J *et al.* Megastore: Providing scalable, highly available storage for interactive services. In *Proc. the 5th Biennial Conference on Innovative Data Systems Research*, January 2011, pp.223-234.

[17] Corbett J C, Dean J, Epstein M *et al.* Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems* (*TOCS*), 2013, 31(3): 8:1-8:22.

[18] Yu Y, Isard M, Fetterly D *et al.* DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In *Proc. the 8th USENIX Symposium on Operating Systems Design and Implementation*, Dec. 2008, pp.1-14.

[19] Isard M, Budiu M, Yu Y *et al.* Dryad: Distributed data-parallel programs from sequential building blocks. *ACM SIGOPS Operating Systems Review,* 2007, 41(3): 59-72.

[20] Zaharia M, Chowdhury M, Das T *et al.* Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proc. the 9th USENIX Conference on Networked Systems Design and Implementation*, Apr. 2012, pp.141-146.

[21] Power R, Li J. Piccolo: Building fast, distributed programs with partitioned tables. In *Proc. the 9th USENIX Symposium on Operating Systems Design and Implementation*, October 2010, pp.293-306.

[22] Melnik S, Gubarev A, Long J J *et al.* Dremel: Interactive analysis of web-scale datasets. *Communications of the ACM*, 2011, 54(6): 114-123.

[23] Peng D, Dabek F. Large-scale incremental processing using distributed transactions and notifications. In *Proc. the 9th USENIX Symposium on Operating Systems Design and Implementation*, October 2010, pp.251-264.

[24] Neumeyer L, Robbins B, Nair A *et al.* S4: Distributed stream computing platform. In *Proc. the 10th IEEE International Conference on Data Mining Workshops*, Dec. 2010, pp.170-177.

[25] Viglas S, Naughton J F. Rate-based query optimization for streaming information sources. In *Proc. ACM SIGMOD International Conference on Management of Data*, Jun. 2002, pp.37-48.

[26] Shen H, Zhang Y. Improved approximate detection of duplicates for data streams over sliding windows. *Journal of Computer Science and Technology*, 2008, 23(6): 973-987.

[27] Li Y, Chen F H, Sun X *et al.* Self-adaptive resource management for large-scale shared clusters. *Journal of Computer Science and Technology*, 2010, 25(5): 945-957.

[28] Hunt P, Konar M, Junqueira F P *et al.* ZooKeeper: Wait-free coordination for Internet-scale systems. In *Proc. USENIX Annual Technical Conference*, Jun. 2010.

[29] Ongaro D, Ousterhout J. In search of an understandable consensus algorithm. In *Proc. USENIX Annual Technical Conference*, Jun. 2014, pp.305-319.

[30] Lamport L. Paxos made simple. *ACM SIGACT News*, 2001, 32(4): 18-25.

[31] Barham P, Dragovic B, Fraser K *et al.* Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 2003, 37(5): 164-177.

[32] Ben-Yehuda M, Day M D, Dubitzky Z *et al.* The turtles project: Design and implementation of nested virtualization. In *Proc. the 9th USENIX Conference on Operating Systems Design and Implementation*, Oct. 2010, pp.423-436.

[33] Xiao Z, Song W, Chen Q. Dynamic resource allocation using virtual machines for cloud computing environment. *IEEE Transactions on Parallel and Distributed Systems*, 2013, 24(6): 1107-1117.

[34] Kivity A, Laor D, Costa G *et al.* OSv — Optimizing the operating system for virtual machines. In *Proc. USENIX Annual Technical Conference*, June 2014, pp.61-72.

[35] Ren S, Tan L, Li C *et al.* Samsara: Efficient deterministic replay in multiprocessor environments with hardware virtualization extensions. In *Proc. USENIX Annual Technical Conference*, June 2016, pp.551-564.

[36] Chen H, Wang X, Wang Z *et al.* DMM: A dynamic memory mapping model for virtual machines. *Science China Information Sciences*, 2010, 53(6): 1097-1108.

[37] Zhao X, Yin J, Chen Z *et al.* vSpec: Workload-adaptive operating system specialization for virtual machines in cloud computing. *Science China Information Sciences*, 2016, 59(9): 92-105.

[38] Wang X, Sun Y, Luo Y *et al.* Dynamic memory paravirtualization transparent to guest OS. *Science China Information Sciences*, 2010, 53(1): 77-88.

[39] Lu L, Zhang Y, Do T *et al.* Physical disentanglement in a container-based file system. In *Proc. the 11th USENIX Symposium on Operating Systems Design and Implementation*, Oct. 2014, pp.81-96.

[40] Arnautov S, Trach B, Gregor F *et al.* SCONE: Secure Linux containers with Intel SGX. In *Proc. USENIX Symposium on Operating Systems Design and Implementation*, Nov. 2016, pp.689-704.

[41] Banga G, Druschel P, Mogul J C. Resource containers: A new facility for resource management in server systems. In *Proc. USENIX Symposium on Operating Systems Design and Implementation*, Feb. 1999, pp.45-58.

[42] Pedro G L, Alberto M, Dick E *et al.* Edge-centric computing: Vision and challenges. *ACM SIGCOMM Computer Communication Review*, 2015, 45 (5): 37-42.

[43] Shi W, Cao J, Zhang Q *et al.* Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 2016, 3(5): 637-646.

[44] Dragojević A, Narayanan D, Castro M *et al.* FaRM: Fast remote memory. In *Proc. USENIX Symposium on Networked Systems Design and Implementation*, Apr. 2014, pp.401-414.

[45] Mitchell C, Geng Y, Li J. Using one-sided RDMA reads to build a fast, CPU-efficient key-value store. In *Proc. USENIX Annual Technical Conference*, June 2013, pp.103-114.

[46] Jose J, Subramoni H, Luo M *et al.* Memcached design on high performance RDMA capable interconnects. In *Proc. International Conference on Parallel Processing*, Sept. 2011, pp.743-752.

[47] Greenberg A, Hamilton J R, Jain N *et al.* VL2: A scalable and flexible data center network. *ACM SIGCOMM Computer Communication Review*, 2009, 39(6): 51-62.

[48] Paraiso F, Haderer N, Merle P *et al.* A federated multi-cloud PaaS infrastructure. In *Proc. the 5th IEEE International Conference on Cloud Computing*, Jun. 2012, pp.392-399.

[49] Eguro K, Venkatesan R. FPGAs for trusted cloud computing. In *Proc. the 22nd International Conference on Field Programmable Logic and Applications*, Aug. 2012, pp.63-70.

[50] Hutchings B L, Franklin R, Carver D. Assisting network intrusion detection with reconfigurable hardware. In *Proc. the 10th IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 2002, pp.111-120.

[51] Chalamalasetti S R, Lim K, Wright M *et al.* An FPGA Memcached appliance. In *Proc. ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Feb. 2013, pp.245-254.

[52] Huang M, Wu D, Yu C H *et al.* Programming and runtime support to blaze FPGA accelerator deployment at datacenter scale. In *Proc. ACM Symposium on Cloud Computing*, Oct. 2016, pp.456-469.

[53] Wang X M, Thota S. A resource-efficient communication architecture for chip multiprocessors on FPGAs. *Journal of Computer Science and Technology*, 2011, 26(3): 434-447.

[54] Dong Y, Xue M, Zheng X *et al.* Boosting GPU virtualization performance with hybrid shadow page tables. In *Proc. USENIX Annual Technical Conference*, July 2015, pp.517-528.

[55] Zhang K, Chen R, Chen H. NUMA-aware graph-structured analytics. In *Proc. the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Feb. 2005, pp.183-193.

[56] Mao Y, Kohler E, Morris R T. Cache craftiness for fast multicore key-value storage. In *Proc. ACM European conference on Computer Systems*, Apr. 2012, pp.183-196.

[57] Tu S, Zheng W, Kohler E *et al.* Speedy transactions in multicore in-memory databases. In *Proc. ACM Symposium on Operating Systems Principles*, Nov. 2013, pp.18-32.

[58] Zhang G, Horn W, Sanchez D. Exploiting commutativity to reduce the cost of updates to shared data in cache-coherent systems. In *Proc. IEEE/ACM International Symposium on Microarchitecture*, Dec. 2015, pp.13-25.

[59] Wang Z, Qian H, Li J *et al.* Using restricted transactional memory to build a scalable in-memory database. In *Proc. the 9th European Conference on Computer Systems*, Apr. 2014, Article No. 26.

[60] Russell R M. The CRAY-1 computer system. *Communications of the ACM*, 1978, 21(1): 63-72.

[61] Barik R, Zhao J, Sarkar V. Efficient selection of vector instructions using dynamic programming. In *Proc. IEEE/ACM International Symposium on Microarchitecture*, Dec. 2010, pp.201-212.

[62] Klimovitski A. Using SSE and SSE2: Misconceptions and reality. *Intel Developer Update Magazine*, Mar. 2001. http://saluc.engr.uconn.edu/refs/process/intel/sse_sse2.pdf, Feb.2017.

[63] Intel I. Intel® SSE4 Programming Reference, D91561-103, 2007. http://software.intel.com/sites/default/files/m/8/6/8/D9156103.pdf, Feb. 2017.

[64] Tian C, Zhou H, He Y *et al.* A dynamic Mapreduce scheduler for heterogeneous workloads. In *Proc. International Conference on Grid and Cooperative Computing*, Aug. 2009, pp.218-224.

[65] Sun N, Liu W, Liu H *et al.* Dawning-1000 PROOS distributed operating system. *Journal of Computer Science and Technology*, 1997, 12(2): 160-166

[66] Zhang L, Litton J, Cangialosi F *et al.* Picocenter: Supporting long-lived, mostly-idle applications in cloud environments. In *Proc. the 11th European Conference on Computer Systems*, Apr. 2016, pp.37:1-37:16.

[67] Canali C, Lancellotti R. Improving scalability of cloud monitoring through PCA-based clustering of virtual machines. *Journal of Computer Science and Technology*, 2014, 29(1): 38-52.

[68] Le K, Bianchini R, Zhang J *et al.* Reducing electricity cost through virtual machine placement in high performance computing clouds. In *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2011.

[69] Chun B G, Ihm S, Maniatis P *et al.* CloneCloud: Elastic execution between mobile device and cloud. In *Proc. the 6th European Conference on Computer Systems*, Apr. 2011, pp.301-314.

[70] Jin H, Deng L, Wu S *et al.* Live virtual machine migration with adaptive, memory compression. In *Proc. IEEE International Conference on Cluster Computing and Workshops*, Aug. 2009.

[71] Ye K, Jiang X, Huang D *et al.* Live migration of multiple virtual machines with resource reservation in cloud computing environments. In *Proc. IEEE International Conference on Cloud Computing*, Jul. 2011, pp.267-274.

[72] Malewicz G, Austern M H, Bik A J *et al.* Pregel: A system for large-scale graph processing. In *Proc. ACM SIGMOD International Conference on Management of Data*, Jun. 2010, pp.135-146.

[73] Kyrola A, Blelloch G, Guestrin C. GraphChi: Large-scale graph computation on just a PC. In *Proc. USENIX Symposium on Operating Systems Design and Implementation*, Oct. 2012, pp.31-46.

[74] Girod L, Mei Y, Newton R *et al.* XStream: A signal-oriented data stream management system. In *Proc. the 24th IEEE International Conference on Data Engineering*, Apr. 2008, pp.1180-1189.

[75] Low Y, Bickson D, Gonzalez J *et al.* Distributed GraphLab: A framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 2012, 5(8): 716-727.

[76] Chen R, Shi J, Chen Y *et al.* PowerLyra: Differentiated graph computation and partitioning on skewed graphs. In *Proc. European Conference on Computer Systems*, Apr. 2015.

[77] Zhang M, Wu Y, Chen K *et al.* Exploring the hidden dimension in graph processing. In *Proc. the 12th USENIX Symposium on Operating Systems Design and Implementation*, Nov. 2016, pp.285-300.

[78] Zhu X, Chen W, Zheng W *et al.* Gemini: A computation-centric distributed graph processing system. In *Proc. USENIX Symposium on Operating Systems Design and Implementation*, Nov. 2016, pp.301-316.

[79] Gonzalez J E, Xin R S, Dave A *et al.* GraphX: Graph processing in a distributed dataflow framework. In *Proc. USENIX Symposium on Operating Systems Design and Implementation*, Oct. 2014, pp.599-613.

[80] Abadi M, Barham P, Chen J *et al.* TensorFlow: A system for large-scale machine learning. In *Proc. the 12th USENIX Symposium on Operating Systems Design and Implementation*, Nov. 2016, pp.265-283.

[81] Nesbit K J, Moreto M, Cazorla F J *et al.* Multicore resource management. *IEEE Micro*, 2008, 28(3): 6-16.

[82] Bolte M, Sievers M, Birkenheuer G *et al.* Non-intrusive virtualization management using libvirt. In *Proc. European Design and Automation Association Conference on Design, Automation and Test in Europe*, Mar. 2010, pp.574-579.

[83] Tanenbaum A S, Kaashoek M F, van Renesse R *et al.* The Amoeba distributed operating system — A status report. *Computer Communications*, 1991, 14(6): 324-335

[84] Vavilapalli V K, Murthy A C, Douglas C *et al.* Apache Hadoop YARN: Yet another resource negotiator. In *Proc. ACM Symposium on Cloud Computing*, Oct. 2013, pp.5:1-5:16.

[85] Burns B, Grant B, Oppenheimer D *et al.* Borg, Omega, and Kubernetes. *ACM Queue*, 2016, 14(1): 70-93

[86] Zhang Z, Li C, Tao Y *et al.* Fuxi: A fault-tolerant resource management and job scheduling system at Internet scale. *Proceedings of the VLDB Endowment*, 2014, 7(13): 1393-1404

[87] Harter T, Salmon B, Liu R *et al.* Slacker: Fast distribution with lazy docker containers. In *Proc. USENIX Conference on File and Storage Technologies*, February 2016.

[88] Singh B, Srinivasan V. Containers: Challenges with the memory resource controller and its performance. In *Proc. Ottawa Linux Symposium*, June 2007.

[89] Nikolaev R, Back G. VirtuOS: An operating system with kernel virtualization. In *Proc. ACM Symposium on Operating Systems Principles*, Nov. 2013, pp.116-132.

[90] Soltesz S, Pötzl H, Fiuczynski M E *et al.* Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. *ACM SIGOPS Operating Systems Review*, 2007, 41(3): 275-287.

[91] Steinberg U, Kauer B. NOVA: A microhypervisor-based secure virtualization architecture. In *Proc. European Conference on Computer Systems*, Apr. 2010, pp.209-222.

[92] Boyd-Wickizer S, Clements A T, Mao Y *et al.* An analysis of Linux scalability to many cores. In *Proc. USENIX Symposium on Operating Systems Design and Implementation*, Oct. 2010, pp.86-93.

[93] Colmenares J A, Bird S, Eads G *et al.* Tessellation operating system: Building a real-time, responsive, high-throughput client OS for many-core architectures. In *Proc. IEEE Hot Chips Symposium*, Aug. 2011.

[94] Baumann A, Peter S, Schüpbach A *et al.* Your computer is already a distributed system. Why isn't your OS? In *Proc. the 12th Conference on Hot Topics in Operating Systems*, May 2009.

[95] Wentzlaff D, Agarwal A. Factored operating systems (FOS): The case for a scalable operating system for multicores. *ACM SIGOPS Operating Systems Review*, 2009, 43(2): 76-85.

[96] Grandl R, Chowdhury M, Akella A *et al.* Altruistic scheduling in multi-resource clusters. In *Proc. USENIX Symposium on Operating Systems Design and Implementation*, Nov. 2016, pp.65-80.

[97] Grandl R, Kandula S, Rao S *et al.* GRAPHENE: Packing and dependency-aware scheduling for data-parallel clusters. In *Proc. USENIX Symposium on Operating Systems Design and Implementation*, Nov. 2016, pp.81-98.

[98] Gog I, Schwarzkopf M, Gleave A *et al.* Firmament: Fast, centralized cluster scheduling at scale. In *Proc. USENIX Symposium on Operating Systems Design and Implementation*, Nov. 2016, pp.99-115.

[99] Jyothi S A, Curino C, Menache I *et al.* Morpheus: Towards automated SLOs for enterprise clusters. In *Proc. USENIX Symposium on Operating Systems Design and Implementation*, Nov. 2016, pp.117-134.

[100] Zhou F F, Ma R H, Li J *et al.* Optimizations for high performance network virtualization. *Journal of Computer Science and Technology*, 2016, 31(1): 107-116.

[101] Tang H, Mu S, Huang J *et al.* Zip: An algorithm based on loser tree for common contacts searching in large graphs. *Journal of Computer Science and Technology*, 2015, 30(4): 799-809.

[102] Ma C, Yan D, Wang Y *et al.* Advanced graph model for tainted variable tracking. *Science China Information Sciences*, 2013, 56(11): 1-12.

Zuo-Ning Chen received her Master's degree in computer application technology from Zhejiang University, Hangzhou, in 1999. She is an adjunct professor in computer science and technology, Tsinghua University, Beijing, and an academician of the Chinese Academy of Engineering. Her current research interests include big data computing, cloud computing, and high performance computing. She has made important contributions in the field of computer software and high-end computers and received the Special and First Prizes of the National Science and Technology Progress Award of China.

Kang Chen received his Ph.D. degree in computer science and technology from Tsinghua University, Beijing, in 2004. Currently, he is an associate professor of computer science and technology at Tsinghua University, Beijing. His research interests include parallel computing, distributed processing, and cloud computing.

Jin-Lei Jiang received his Ph.D. degree in computer science and technology from Tsinghua University, Beijing, in 2004, with an honor of excellent dissertation. He is currently an associate professor of computer science and technology at Tsinghua University, Beijing. His research interests include distributed computing and systems, cloud computing, big data, and virtualization.

Lu-Fei Zhang received his B.S. degree from Tsinghua University, Beijing, in 2008. He is currently an engineer in Jiangnan Institute of Computing Technology, Wuxi. His research interests include cloud computing, big data, etc.

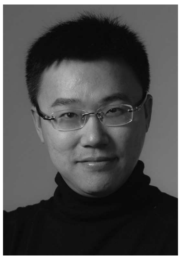
Song Wu is a professor of computer science at Huazhong University of Science and Technology (HUST), Wuhan. He received his Ph.D. degree in computer science from HUST in 2003. He is now served as the director of Parallel and Distributed Computing Institute, and the vice head of Service Computing Technology and System Laboratory (SCTS) of HUST. He has published more than one hundred papers and obtained over thirty patents in the area of parallel and distributed computing. His current research interests include cloud computing and virtualization.

Zheng-Wei Qi is a professor in School of Software, Shanghai Jiao Tong University (SJTU), Shanghai, and a member of the Shanghai Key Laboratory of Scalable Computing and Systems. He received his Ph.D. degree in computer science from SJTU with a thesis on the formal method in distributed systems. His research interests focus on trusted computer systems: virtual machines, program analysis, cloud computing, mobile computing, and GPU virtualizations. He received the Second Prize of the National Science and Technology Progress Award of China in 2014.

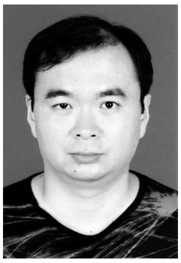
Chun-Ming Hu received his Ph.D. degree in computer science from Beihang University, Beijing, in 2006. He is an associate professor in School of Computer Science and Engineering, Beihang University, Beijing. His current research interests include distributed systems, system virtualization, data center resource management and scheduling, and large-scale data processing systems.

Yong-Wei Wu received his Ph.D. degree in applied mathematics from the Chinese Academy of Sciences, Beijing, in 2002. He is currently a professor in computer science and technology at Tsinghua University, Beijing. His research interests include parallel and distributed processing, mobile and distributed systems, cloud computing, and storage. He has published over 80 research publications and received two Best Paper Awards. He is currently on the editorial boards of IEEE Transactions on Cloud Computing, Journal of Grid Computing, IEEE Cloud Computing, and International Journal of Networked and Distributed Computing.

Yun-Zhong Sun received his Ph.D. degree in computer engineering from Institute of Computing Technology (ICT), Chinese Academy of Sciences, Beijing. He is a professor in the State Key Laboratory of Computer Architecture at ICT. His research interests focus on distributed system software and computing/programming models. He has authored and co-authored more than 50 publications, and has served in various academic conferences and journals. He is a member of CCF and IEEE, and the IEEE Computer Society.



Hong Tang received his B.S. degree in computer science from Zhejiang University, Hangzhou, in 1997, and his Ph.D. degree in computer science from University of California (UC), Santa Barbara, in 2003. He joined Alibaba Cloud Computing in 2010, and is currently the chief architect of Apsara Cloud Platform. Prior to Alibaba, he was a director of Search System Infrastructure at Ask.com. In 2008, he joined Yahoo (US)'s Cloud Computing team as a senior principal engineer, driving the research and development of Hadoop. Dr. Tang's research interests include high-performance computing and parallel systems, distributed computing and storage systems, large-scale Internet services, and cloud computing. He was elected as Innovative Talent in the National Recruitment Program of Global Experts in 2013.



Ao-Bing Sun received his Ph.D. degree in computer science from the School of Computer Science and Technology of Huazhong University of Science and Technology, Wuhan, in 2008, and worked as a postdoctoral researcher at Institute of Computing Technology (ICT), Chinese Academy of Sciences, Beijing, during 2010∼2012. He is now a vice president of G-Cloud Technology Inc., Dongguan. His research interests include cloud computing, grid computing, image processing and so on.

Zi-Lu Kang is the director of the Institute of Technology of Internet of Things, Information Science Academy of China Electronics Technology Group Corporation, Beijing. He has directed or participated in multiple engineering projects and product developments related to smart city and Internet of Things. He has also participated in the development of Internet of Things standards of International Telecommunication Union (ITU).