vStarCloud: An Operating System Architecture for Cloud Computing

Zhang Lufei, Chen Zuoning
Jiangnan Institute of Computing Technology
Wuxi, China
e-mail: zhanglf04@126.com, chenzuoning@vip.163.com

***Abstract*—In recent years, cloud computing is increasingly abstracting away the operating system, allowing developers to focus higher up the stack on applications, not infrastructure. In this position paper, we analyzed the deficiencies of the traditional operating systems in the cloud environment and the features of existing cloud operating systems. We have proposed an operating system architecture for cloud computing which is called vStarCloud, and implemented a prototype to show that our approach is promising. We describe how common problems for the cloud environment can be effectively solved by POSIV (Portable Operating System Interface of vStarCloud) interface. The prototype of vStarCloud provides the ability of virtual machine management and MapReduce job management. Then we discussed how to cope with the challenges of the new cloud operating systems.**

Keywords-cloud computing; operating system; microkernel; container

I. INTRODUCTION

With rapid development of the cloud computing system, widespread interest in cloud operating system (Cloud OS) is a relatively recent phenomenon. In fact, a lot of companies have been managing cloud infrastructure at scale and providing cloud service for more than ten years and built many different cloud-management software in that time. The definition of Cloud OS from PC Magazine Encyclopedia is: a server-side software platform that provides the complete infrastructure for setting up cloud computing services. Cloud OS is the "core" of the cloud-management software, and does what a traditional OS does: bridges hardware (called bare metal) and upper application, but at the scope and scale of cloud computing.

The hardware of cloud is evolving into a multi-dimensional space that includes many-core processors, virtualized clusters and software-defined networks. There are different business models of cloud computing, such as IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service). So there are different levels of API (Application Programming Interface) for cloud services and a variety of network middleware, such as message queue, distributed lock, distributed database and distributed storage, shielding the natural distribution of the large system. A key challenge for architects and designers of future clouds is to provide the right set of abstractions so that application builders can fully leverage these new opportunities, while avoiding the ossification that often arises when such a large diversity of factors is involved.

Most of cloud computing systems use vertical integration approach and customize the underlying OS and network middleware for efficiency, even simply remove much of the functionality in a general purpose OS. If the underlying OS is based monolithic kernel, which is inconvenient for on-demand dynamic reconfiguration and hot-plugging services, the cloud-management software could not accommodate various type of network computing applications, such as the computation-intensive, data-intensive and network-intensive. Meanwhile, the form of service delivery is going through an evolution, with VMs being joined by containers and unikernels as packing and deployment mechanisms.

So, the next-generation Cloud OS as the unified framework for versatility of cloud-management software ecosystem is very significant. It means the network computing application can shift to more efficiently managing datacenter resources as a whole, including networking, storage and compute. The end-user will be able to deliver powerful applications and data on virtually any device to boost productivity, while maintaining security and compliance.

II. RELATED WORKS

Several operating systems are similar to Cloud OS in characteristic. They are distributed operating system, virtual machine management software, container management software and cloud service provider. Some research institutes and business companies also presented their so-called Cloud OS.

There are some logical components in vStarCloud architecture, such as Cloud task (vTask), Cloud storage (vStore), Cloud kernel and Cloud system call. The logic unit called vTask uses cloud resources, which can be scheduled by cloud OS. Applications and their overall operating environment is packaged into vTask for scheduling, such as virtual appliance, Java applications and Erlang applications. The entity called vStore provides vTask with persistent storage capabilities which usually built on top of the distributed file system such as HDFS (Hadoop Distributed File System) or distributed storage, aggregating node storage capacity, and providing different functional forms for different vTasks. Cloud kernel provides cloud resource management (such as access control, monitoring, acquiring, releasing), vTask lifecycle management (such as creating, deploying, scheduling, destruction), as well as vStore support. Cloud system call is the basic interface providing for the end user by the Cloud kernel, usually interacting with kernel for

978-1-5090-4499-3/17/$31.00 ©2017 IEEE

acquiring and releasing resources. Similar with standard interface of traditional OS such as POSIX (Portable Operating System Interface of UNIX), vStarCloud also defines a set of interfaces (system calls / basic functional library), called POSIV (Portable Operating System Interface of vStarCloud).

Although vTask has some characteristics like the virtual machine and container such as concurrency, interoperability, independence, vTask as a schedulable entity is more like a group of associated "microservices" in a physical node. In vStarCloud, services are hierarchical and coupled tightly. vStarCloud puts most parts of traditional OS functions in a set of services, which are called meta services. vStarCloud is an extensible system built on a microkernel-based architecture, cleanly separating logical model from the physical level of objects and APIs. The distributed meta service management needs flexible service plug-in mechanism. Meta service management provides semantic service description mechanism, allowing the upper application to customize the service strategy.

According to the design principle of separating strategies from mechanisms, vStarCloud Kernel mainly composed of two modules, vStarCloud Kernel Worker and vStarCloud Kernel Manager. Manager is responsible for high-level strategy formulation in a global view. Workers are deployed on each physical node as specific actors. The primary functional modules of Manager are Resource Manager, Storage Manager, Task Manager, Security Manager, Distribution Manager.

A. Distributed Operating System

The feature of distributed operating systems is distribution transparent abstractions, both at the system level and the application level. At the systems level, distributed virtual memory and process migration hide distribution from the higher levels of the operating system software. At the application level, remote procedure calls and inter-process messaging hide distribution from applications.

The traditional distributed operating systems include Amoeba [1], Sprite [2] and Clouds [3], but never caught on commercially. One reason was that in the 1990's, processor performance was scaling rapidly while networking performance was lagging.

The distributed scalable microkernel architecture parallelize OS data structure based microkernel technique, such as Tornado [4] and K42 [5]. Fos [6] distributes a service to run in parallel on several servers. Making full use of the space parallelism of the multicore processor, fos provides the common operating system functions through the distributed services. However, these microkernel operating systems only have simple services coupled tightly with microkernel, without enough flexibility.

B. Virtual Machine Management Software

The approach taken by IaaS puts OS construction in the context of cloud computing within a virtual machine. The basic premise of the movement to virtualization is to create an enormous increase in overall resource utilization along with corresponding reductions in both time-to-value and in the repetitive work required to deliver services.

The most significant breakthrough of the virtual machine management software like OpenStack and CloudStack is to enable resource utilization as small as a single core out of a physical computer, available on demand. But in many types of cloud service, users don't need virtualize an entire machine when they want to further subdivide compute resources. And the virtual machine cannot package the application as a singularly addressable, registry-stored, fast-booted service to simplify the deployment.

C. Container Management Software

Light-weight container design will lead to high resource utilization, much improved DevOps agility. Containers and give the user almost bare metal performance. With containers, cloud service provider can use efficient packaging formats to enable unprecedented workload portability across a hybrid cloud.

With container management software like Docker [7], Borg [8] and Omega [9], users can develop their microservice using components from the depository like the Docker Hub, and easily deploy into the cloud. And Kubernetes [10] has created a layer of abstraction to improve the composability of application systems, which allows the developer and administrator to work collectively on improving the behavior and performance of the desired service, rather than any of its individual component containers or infrastructure resources.

D. Cloud Service Provider

Typical IaaS system, such as Amazon EC2, provides computing resources through Virtual Machine and the Linux kernel image. Typical PaaS system, such as Google App Engine and Microsoft Azure, provides API for the development of application, and aims to particular programming language which supports automatic scaling and fault tolerance. Cloud service providers develop and host web applications in commercial managed data centers, and always just serve for their business models. So generic Cloud OS should try to provide the above functions independent of application modes and programming languages.

III. vSTARCLOUD ARCHITECTURE

There are some logical components in vStarCloud architecture, such as Cloud task (vTask), Cloud storage (vStore), Cloud kernel and Cloud system call. The logic unit called vTask uses cloud resources, which can be scheduled by cloud OS. Applications and their overall operating environment is packaged into vTask for scheduling, such as virtual appliance, Java applications and Erlang applications. The entity called vStore provides vTask with persistent storage capabilities which usually built on top of the distributed file system such as HDFS (Hadoop Distributed File System) or distributed storage, aggregating node storage capacity, and providing different functional forms for different vTasks. Cloud kernel provides cloud resource management (such as access control, monitoring, acquiring, releasing), vTask lifecycle management (such as creating,

deploying, scheduling, destruction), as well as vStore support. Cloud system call is the basic interface providing for the end user by the Cloud kernel, usually interacting with kernel for acquiring and releasing resources. Similar with standard interface of traditional OS such as POSIX (Portable Operating System Interface of UNIX), vStarCloud also defines a set of interfaces (system calls / basic functional library), called POSIV (Portable Operating System Interface of vStarCloud).

Although vTask has some characteristics such as concurrency, interoperability, independence like the virtual machine and container, vTask as a schedulable entity is more like a group of associated "microservices" in a physical node. In vStarCloud, services are hierarchical and coupled tightly. vStarCloud puts most parts of traditional OS functions in a set of services, which are called meta services. vStarCloud is an extensible system built on a microkernel-based architecture, cleanly separating logical model from the physical level of objects and APIs. The distributed meta service management needs flexible service plug-in mechanism. Meta service management provides semantic service description mechanism, allowing the upper application to customize the service strategy.

According to the design principle of separating strategies from mechanisms, vStarCloud Kernel mainly composed of two modules, vStarCloud Kernel Worker and vStarCloud Kernel Manager. Manager is responsible for high-level strategy formulation in a global view. Workers are deployed on each physical node as specific actors. The primary functional modules of Manager are Resource Manager, Storage Manager, Task Manager, Security Manager, Distribution Manager.

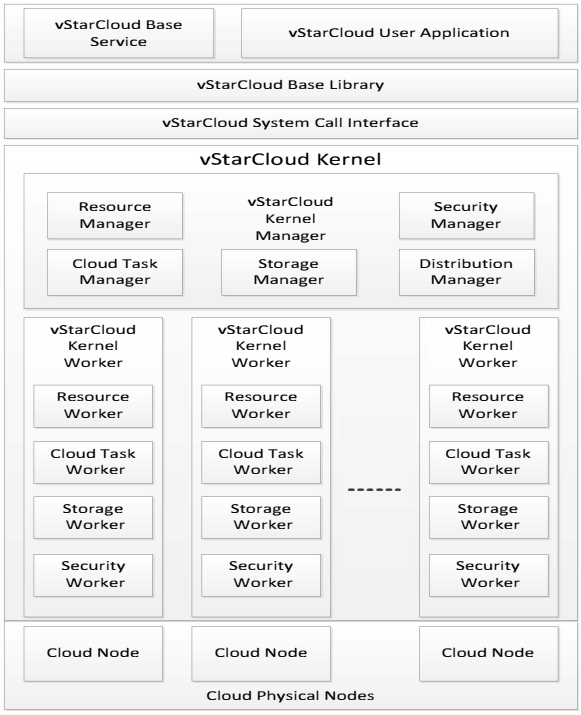

Figure 1. The vStarCloud architecture.

With the constant development of the vStarCloud Kernel, vStarCloud could provide users more rich Cloud OS services. Currently the function of vStarCloud Kernel is only creating vTask, scheduling vTask, vTask communication. vStarCloud base library can provide services such as data replication consistency and availability for vTask. When developers implement a new function, appropriate entities will be defined as a basic vTask and vStore. vStarCloud Kernel Manager chooses the scheduling strategy automatically.

IV. PROTYPE

The first implementation of vStarCloud architecture is developed in the cloud computing operating system project [11]. In the prototype, the vTask management system is the main part, message between which is realized through Thrift and ZeroMQ together.

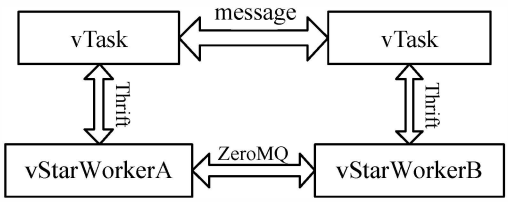

Figure 2. The message implementation between vTasks.

A. Implementation

vStarCloud prototype uses master-slave architecture. The master vTask implements distribution control logic, interacts with vStore, supports dynamic creation, scheduling and termination of slave vTasks. The status monitoring and message passing between vTasks can tolerate some faults. Users can implement dynamic control of vTask, process vTasks according to their status.

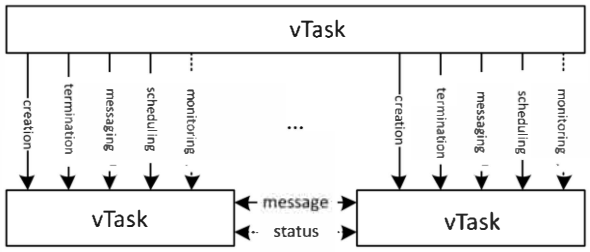

Figure 3. vTask framework.

vStarCloud support multi-language POSIV interface, such as in python and Erlang.

TABLE I. VTASK INTERFACE

Interface	Description
vTask creation	*vtid=app.vtask_new(Name, vStorePath, cmd)*
vTask scheduling	*app.vtask_sched(vtid)*
vTask termination	*app.vtask_kill(vtid)*
register synchronized message	*app.vtask_register_rpc(rpc_identity,rpc_handler)*

Interface	Description
deliver synchronized message	*app.vtask_rpc(vtid,rpc_identity,timeout,*args)*
register asynchronous message	*app.vtask_register_msg(msg_identity,msg_handler)*
deliver asynchronous message	*app.vtask_msg(vtid,msg_identity,Msg,timeout)*
monitor start status	*app.vtask_on_start(vtid,startcallback)*
monitor exit status	*app.vtask_on_exit(vtid,exitcallback)*
monitor die status	*app.vtask_on_die(vtid,diecallback)*

B. Use Case: Virtual Machine Management

In the virtual machine (VM) management system, each vTask corresponds to a virtual machine agent. There is a master vTask to manage and monitor the entire cloud environment. To create a virtual machine, for example, user sends the parameters of virtual machine to the master vTask. The master vTask will parse the command, store the information of virtual machine into the database, select a vStarWorker, and then send a synchronized message of VM creation. Once the slave vTask is created on the specified vStarWorker, it will download a virtual machine template according to the information stored, then perform the VM creation action, and finally sent the response of success to the master vTask. In the end, the master vTask feed back to the user of VM information. Like the VM creation, all actions such as status monitoring, load balancing, fault tolerance and migration in the VM's entire life cycle can utilize POSIV interface.

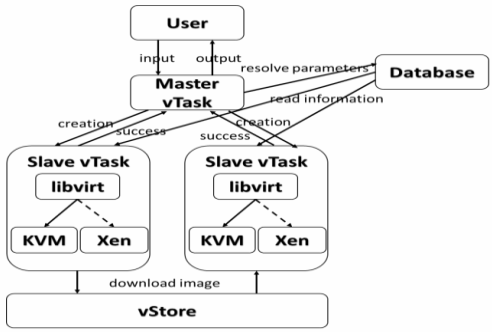

Figure 4. Implementation of VM management software.

C. Use Case: MapReduce Job Management

Implementation of MapReduce framework consists of the following components: Master vTask, Map Worker and Reduce Worker. Map Worker and Reduce Worker are composed of a set of slave vTasks. The master vTask deploy and manage the entire MapReduce environment, and also coordinate with vStore to complete the MapReduce process. Users need to upload files to the vStore, then inform the master vTask some parameters such as file path, file size, split size, number of map tasks and reduce tasks. After that, map vTask is created, based on the incoming <key, value> pairs, searching the appropriate files in vStore. After processed by map vTasks, the intermediate <key,value> data will be sent to reduce vTasks. Meanwhile, map vTasks send messages of completion to the master vTask, informing that resources could be released. After reduce vTask receiving the incoming <key,value> pairs from map vTasks, according to user-defined reduce function, the data will be processed and sent to the master vTask. The master vTask combines all the results as soon as received integral reduce messages.

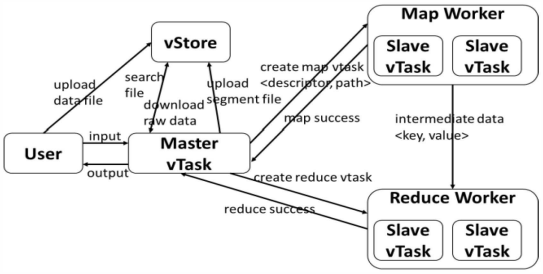

Figure 5. Implementation of mapreduce framework.

D. Evaluation

A test-bed was built for evaluation of self-made VM management software and MapReduce framework. It consisted of 4 servers (HP DL380 G7 with 8 cores, 32GB memory and 1TB disk) which were connected with Gigabit Ethernet switch.

First of all, vStarCloud was compared with OpenStack (Release Mitaka) for VM management capability. Different amount of VMs (from 200 to 1,000) were created on the test-bed in batch, and the total time for creating all VMs was gathered. Each VM consumes 1 VCPU, 64MB memory and 2GB disk. Over 99% of VMs were successfully created. So the key comparison was efficiency of VM management software

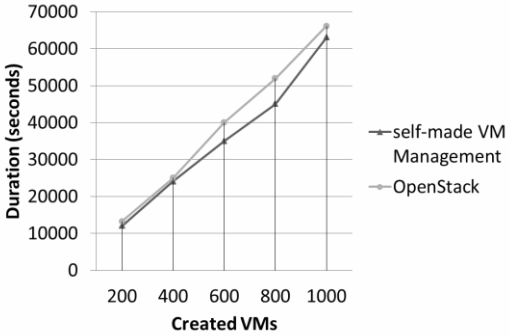

Figure 6. Comparison of VM management software.

Secondly, vStarCloud was compared with Hadoop (Release 2.4.0) for data processing capability. WordCount benchmark was implemented by MapReduce framework. Different amount of data (from 1GB to 5GB) was processed on the test-bed, and the total time was gathered. The results of WordCount on 2 frameworks are the same. So the key comparison was performance of MapReduce jobs.

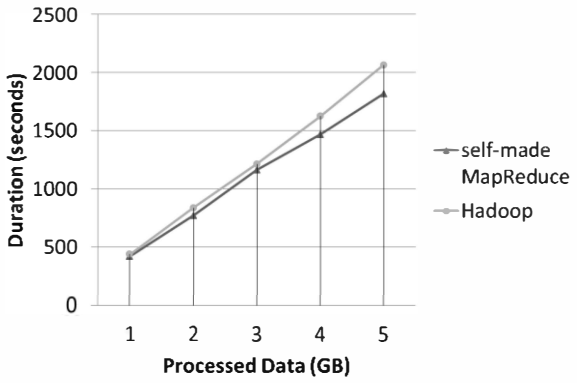

Figure 7. Comparison of mapreduce framework.

E. Summary

In the evaluation, the virtual machine management system based on vStarCloud architecture can realize most of functions of VM management, such as creating, starting, suspending, resuming, terminating and achieves better efficiency of VM creation compared to OpenStack. And the self-made MapReduce framework using mechanisms of vTask achieves better performance compared to Hadoop in the same experimental conditions.

Although vStarCloud prototype lacks of efficient scheduling policies and needs the performance tuning, it demonstrates the usability of vStarCloud which supports rapid vTask deployment, helps developers to achieve smart scheduling, failover and load balancing of vTask and vStore, disregarding distributed underlying OS details, but making focus on their business logic.

V. CONCLUSION

VM Induced Technology helps cloud computing system evolve more quickly especially for today's container technology. Most of containers run as applications on top of the Linux-hypervisor stack. But they are not designed with the current server hardware in mind. Today's servers are distributed system in miniature, with low latency data center network and hyper-converged storage. Lots of complex abstractions are needed and, more importantly, are often exposed to the programmer. If infrastructure is defined in software the programmers will need to deal with programming infrastructure. So the underlying OS itself technology should enhance modularization, and make system reduction and expansion easier.

Cloud OS is the best way to increase performance and reduce complexity in the next generation of cloud system software. In this paper, we presented an operating system architecture for cloud computing which is called vStarCloud, which can facilitate the cloud computing system, to meet the mentioned challenges. Two ideas are highlighted: (1) Vertical integration combined with the hierarchical architecture. (2) Separation of distributed control strategies and mechanisms. By implementation and use cases of vStarCloud architecture, it can be seen that the same mechanism in vStarCloud supports both the typical IaaS and PaaS applications.

ACKNOWLEDGMENT

This research is supported by National Key Research & Development Program under grant of 2016YFB1000500.

REFERENCES

[1] A. S. Tanenbaum, S. J. Mullender, and R. van Renesse. Using sparse capabilities in a distributed operating system. In Proceedings of the International Conference on Distributed Computing Systems, pages 558–563, May 1986.

[2] J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson, and B. B. Welch. The Sprite network operating system. IEEE Computer, 21(2):23–36, Feb. 1988.

[3] P. Dasgupta, R. Chen, S. Menon, M. Pearson, R. Ananthanarayanan, U. Ramachandran, M. Ahamad, R. J. LeBlanc, W. Applebe, J. M. Bernabeu-Auban, P. Hutto, M. Khalidi, and C. J. Wileknloh. The design and implementation of the Clouds distributed operating system. USENIX Computing Systems Journal, 3(1):11–46, 1990.

[4] B. Gamsa, O. Krieger, J. Appavoo, and M. Stumm. Tornado: Maximizing locality and concurrency in a shared memory multiprocessor operating system. In Proceedings of the Symposium on Operating Systems Design and Implementation, pages 87–100, Feb. 1999.

[5] J. Appavoo, M. Auslander, M. Burtico, D. M. da Silva, O. Krieger, M. F. Mergen, M. Ostrowski, B. Rosenburg, R. W. Wisniewski, and J. Xenidis. K42: an open-source linux-compatible scalable operating system kernel. IBM Systems Journal, 44(2):427–440, 2005.

[6] D. Wentzla and A. Agarwal. Factored operating systems (fos): The case for a scalable operating system for multicores. SIGOPS Oper. Syst. Rev., 43(2):76–85, 2009.

[7] 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 22–25, 2016.

[8] Verma A, Pedrosa L, Korupolu M et al. Large-scale cluster management at Google with Borg. In Proc. European Conference on Computer Systems, Apr, 2015, p. 18.

[9] 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.

[10] Burns B, Grant B, Oppenheimer D et al. Borg, Omega, and Kubernetes, ACM Queue, 2016, vol.14, pp.70-93

[11] Zuoning C, Hongwu Y, Lufei Z et al. vStarCloud: A New Network Computing Operating System for Multicore and Clouds. In Proc. International Symposium on Parallel Architecures Algorithms and Programming, December 9–11, 2011.