Bao YG, Wang S. Labeled von Neumann architecture for software-defined cloud. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 32(2): 219–223 Mar. 2017. DOI 10.1007/s11390-017-1716-0

Labeled von Neumann Architecture for Software-Defined Cloud

Yun-Gang Bao, *Member, CCF, ACM, IEEE*, and Sa Wang, *Member, CCF*

State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China

University of Chinese Academy of Sciences, Beijing 100049, China

E-mail: {baoyg, wangsa}@ict.ac.cn

Received November 25, 2016; revised February 15, 2017.

Abstract As cloud computing is moving forward rapidly, cloud providers have been encountering great challenges: long tail latency, low utilization, and high interference. They intend to co-locate multiple workloads on a single server to improve the resource utilization. But the co-located applications suffer from severe performance interference and long tail latency, which lead to unpredictable user experience. To meet these challenges, software-defined cloud has been proposed to facilitate tighter coordination among application, operating system and hardware. Users' quality of service (QoS) requirements could be propagated all the way down to the hardware with differential management mechanisms. However, there is little hardware support to maintain and guarantee users' QoS requirements. To this end, this paper proposes Labeled von Neumann Architecture (LvNA), which introduces a labelling mechanism to convey more software's semantic information such as QoS and security to the underlying hardware. LvNA is able to correlate labels with various entities, e.g., virtual machine, process and thread, and propagate labels in the whole machine and program differentiated services based on rules. We consider LvNA to be a fundamental hardware support to the software-defined cloud.

Keywords software-defined cloud, von Neumann architecture, tail latency, performance interference

1 Introduction

It has been a few years since cloud computing[1] emerged and hit the IT industry. At the beginning, cloud computing referred to the new term for the long-held dream of computing as a utility. With the help of virtualization, cloud providers like Amazon, could abstract their underlying hardware into a shared pool of configurable computing resources. Customers (also called tenants) just need purchase a bulk of instances (virtual machines) to deploy their applications without concerning about underlying hardware resource management. Computing resources in cloud can be elastically provisioned and released. We consider this is the first generation of cloud, denoted as virtualized cloud. Virtualized cloud just provides best-effort service with unpredictable performance.

As the data on Internet massively increased, parallel data processing framework, e.g., Hadoop[2], Spark[3], PowerGraph[4], became another kind of mainstream applications in cloud. In order to prevent performance interference caused by virtualization and maintain the ease of operations, different applications were separated into isolated clusters. One physical cluster serves only one type of workload. This is the second generation of cloud, denoted as partitioned cloud. As a matter of price, the resource utilization of partitioned cloud is inevitably low.

As the workload grew up and cloud computing moved forward rapidly, the huge market demand urged IT companies to keep ramping up their investment in data center. On the other hand, the resource utilization of these data centers is much lower than expected. According to reports from Gartner① and Kaplan *et*

Short Paper

Special Section on MOST Cloud and Big Data

This work was supported by the National Key Research and Development Program of China under Grant No. 2016YFB1000200 and the National Natural Science Foundation of China under Grant No. 61420106013.

①http://www.gartner.com/newsroom/id/1472714, Feb. 2017.

al.[5], CPU usage of most data centers in the world was around 6%∼12%. Even for the world biggest public cloud, Amazon EC2, their CPU usage was merely 7%∼17%[6], which was far from efficiency and resulted in a great loss to the IT companies. Recently, industry and academia both attempted to develop the third generation of cloud, shared cloud, which co-locates different workload onto the same underlying hardware to improve the resource utilization while guaranteeing user experience. Shared cloud poses great challenges to cloud providers.

• *Long Tail Latency.* Tail latency was first proposed by Google researchers[7] and instantly attracted much attention. They found that rare high latency in individual components may come to dominate the whole service performance at large scale. Variability in the latency distribution of individual components was amplified in service level by scale. In order to guarantee user experience in shared cloud, the first challenge is to curb latency variability. However latency variability can arise for many reasons, including concurrent locks inside application, disordered resource sharing in cache and memory bandwidth, queuing in various resource levels, and so on.

• *Low Utilization.* To reduce latency variability, cloud providers either employ experienced engineers and take every effort to optimize their cloud applications carefully, or leverage over-provisioning to guarantee the resource utilization of applications. Both methods bring huge cost and little effect. Meanwhile, over-provisioning can make the resource utilization of data centers rather low. Shared cloud encounters this crucial trade-off between low utilization and tail latency.

• *High Interference.* The basic idea of shared cloud is locating different workload together on the same physical servers. Different applications sharing the same hardware resource can cause performance interference to each other. With the development of distributed processing framework, the execution time of single task reduces to hundreds of milliseconds, which makes the performance interference more severely and hard to locate.

To overcome these challenges, software-defined cloud[8] has emerged, which borrows the wisdom of software-defined network and facilitates tighter coordination among application, operating system, virtualization and hardware. The QoS requirement and identification of upper applications could be propagated all the way down to the underlying hardware effectively. When the performance of certain application violates its QoS requirement, the whole system could coordinate to identify, protect or accelerate the application, such as assigning more cache capacity to corresponding applications, throttling other applications with high CPU utilization. Industry and research communities have taken a lot of effort to optimize the full software stack, from virtualization, operating system to distributed system architecture, as illustrated in Table 1. After 10 years' effort, Dick Sites, Google senior engineer, admitted it was really hard to enforce performance isolation among different applications in shared environments and advocated that more hardware support is needed②.

Table 1. Contention Identified in Prior Work

Software Layer	Contention Point
Data center	Global file system[7]
Application	Background deamon[7], maintenance activities[7], backup jobs[9]
Network stack	Small packets triggered Nagle's algorithm[9], limited buffers[9], delayed ACK results in RTO[9], TCP congestion control[10-11], packet scheduling[12-15], kernel sockets[16]
Kernel	Lock contention[17-20], context switch[16], kernel scheduling overhead[16,21-23], SMT load imbalance[16,24-27], IRQ imbalance[16,28]
Power	C-State[16], DVFS[16]
Hypervisor	Virtual machine scheduling[9,21,29-33], network isolation[29-30,34-35]

2 Background and Motivation

The von Neumann architecture is a classical model for a stored-program digital computer, which was proposed in 1945 by the mathematician and physicist John von Neumann. As illustrated in Fig.1, it consists of a processing unit containing an arithmetic logic unit and processor registers, a control unit containing an instruction register and a program counter, a memory to store both data and instructions, external mass storage, and input and output mechanisms.

The von Neumann architecture is elegant, but far from perfect. In John Backus's 1977 ACM Turing Award lecture, he pointed out that the data transfer between CPU and memory would become the very

②https://www.cs.wisc.edu/events/1887, Feb. 2017.

bottleneck of von Neumann architecture. As John Backus predicted, the performance gap between CPU and memory became larger and larger due to Moore's law, resulting in the "Memory Wall" problem. To deal with the gap, architects had to add more cache layers, as shown in Fig.2.

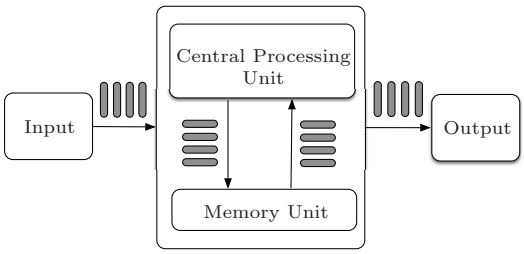


Fig.1. von Neumann architecture.

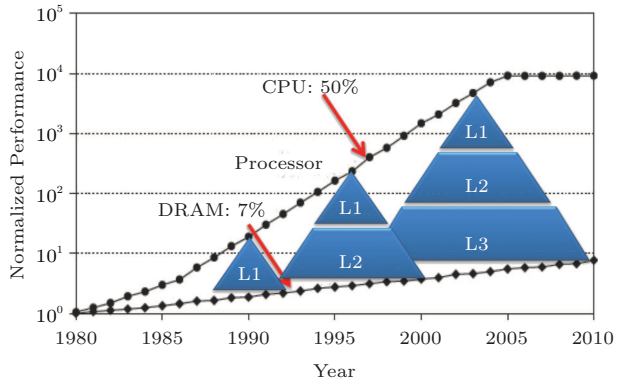


Fig.2. Increasing memory hierarchy.

After the 2000s, thank to the advent of multicore processors, the memory hierarchy was gradually partitioned into two categories: on-core and un-core. The on-core resource, like L1 and L2 cache, is considered as private resource to each core's own; the un-core resource including last level cache (LLC) and memory is shared by multiple cores. In this scenario, if two cores running two separate applications with different latency sensitivities both issue a request to shared LLC, the LLC cannot figure out which request is more critical and should be handled first. Therefore, the requests could be handled in an indeterminate order. We consider this phenomenon as unmanaged sharing problem. All shared memory hierarchies suffer from unmanaged sharing problem.

Unmanaged sharing is the root cause of performance interference, which leads to the "Noisy Neighbor" problem in the cloud. In a multicore processor, if an application is running alone, it could load all of its working set into LLC. However, if it is running with other applications, they could compete with each other in the LLC instead. Experimental results on Intel Xeon processor show that the interference caused by unmanaged sharing reduces the performance by more than three times. Therefore, new architectural mechanisms are needed to deal with these unmanaged chaos.

3 Labeled von Neumann Architecture

In this work, we propose a new computer architecture, Labled von Neumann Architecture (LvNA), which is a traditional von Neumann architecture enhanced with fine-grained hardware resource management mechanism as illustrated in Fig.3. Like labeled networking, LvNA could correlate labels with virtual machine, process and thread, etc., and propagate labels within a whole machine. We also integrate a programmable center control plane to provide differentiated services based on different label-indexed rules.

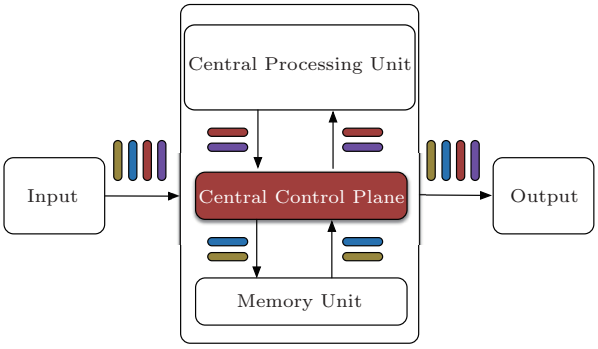


Fig.3. Labeled von Neumann Architecture.

In theory, we can simplify LvNA into a queuing model with multiple priorities. According to queuing theory, the service time of high priority request will not be disturbed by other requests. Based on the LvNA, the data center can achieve both resource efficiency and user experience. The programmable center control plane exposed to the software level can provide more flexibility for software-defined cloud.

However, the LvNA still leaves several open problems.

• *Theory.* What is the impact of LvNA on RAM, PRAM, LogP models? To elaborate, theoretically we can throttle all of the requests with low priority to provide absolute QoS guarantee; however, this will hurt utilization and overall throughput. Queuing and priorities must be carefully considered to achieve balance between tasks.

• *Hardware/Architecture*. How to implement LvNA in CPU, memory, storage, networking? The LvNA requires precise label control over the flow of all of the requests inside the whole system; however, industry designs pervasively use buffers, out-of-order processing on CPU, cache memory hierarchy and a line of other subsystems. It would be disastrous if all those requests are mis-labeled.

• *Programing Model and Compilers*. How to express users' requirements and propagate to the hardware via labels? How to make compilers support labels? Compilers have semantic information that is hard to retrieve on hardware. Compilers can pass important performance and usage pattern hints to hardware, which would be highly valuable to guide hardware resources allocation.

• *OS/Hypervisor*. How to correlate labels with VMs, containers, processors, threads? How to abstract programming interfaces for labels? OS/hypervisor does task scheduling. A label mechanism that permits the storage of related information of ready task as well as fast label context switching between running tasks is necessary to keep label information persistent during task scheduling cycles.

• *Distributed Systems*. How to correlate labels with distributed resources? How to manage distributed systems with label mechanisms? A distributed task with hundreds or even thousands of threads running on a line of servers surely poses many challenges on how to propagate and manage labels.

• *Measurement/Audit*. How to leverage labels to gauge and audit resource usages? For example, the labels flow on system memory can be discontinuous since system memory has to serve requests from multiple sources. Besides that, labels' origin can also be vague on system memory, since "write to memory" requests from cache are largely caused by cache evictions of dirty blocks, and the eviction may be invoked by cache read miss requests with different labels. Therefore one sometimes would be forced to gauge and audit resource usage on such a difficult scenario.

4 Conclusions

In this paper, we first discussed the major issue in today's data center, resource efficiency. The CPU usage of most data center hovers around 6%∼12%. Increasing the resource utilization will cause user experience degraded severely. Then we recalled the development of computer architecture and found the root cause was unmanaged sharing among memory hierarchy. Finally, We borrowed the wisdom in labeled networking and presented LvNA, a fine-grained hardware resource management mechanism. We correlated labels with resource and propagated labels in the whole machine. LvNA still left several open problems to be overcome.

References

[1] Mell P M, Grance T. SP 800-145, the NIST definition of cloud computing. Tech. Rep., Gaithersburg, MD, United States, 2011. http:///www.nist.gov/node/568586, Feb. 2017.

[2] Shvachko K, Kuang H, Radia S, Chansler R. The Hadoop distributed file system. In *Proc. the 26th IEEE MSST*, May 2010.

[3] Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin M J, Shenker S, Stoica I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proc. the 9th USENIX NSDT*, April 2012, pp.15-28.

[4] Gonzalez J E, Low Y, Gu H, Bickson D, Guestrin C. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proc. the 10th USENIX OSDI*, Oct. 2012.

[5] Kaplan J M, Forrest W, Kindle N. Revolutionizing data center energy efficiency. Tech. Rep., McKinsey & Company, July 2008. http://pdfsr.com/pdf/revolutionizing-data-center-energy-efficiency, Feb. 2017.

[6] Liu H. A measurement study of server utilization in public clouds. In *Proc. the 9th IEEE International Conference on Dependable, Autonomic and Secure Computing*, Dec. 2011, pp.435-442.

[7] Dean J, Barroso L A. The tail at scale. *Commun. ACM*, 2013, 56(2): 74-80.

[8] Grandl R, Chen Y, Khalid J, Yang S, Anand A, Benson T, Akella A. Harmony: Coordinating network, compute, and storage in software-defined clouds. In *Proc. the 4th Annual Symposium on Cloud Computing*, Oct. 2013, pp.53:1–53:2.

[9] Xu Y, Musgrave Z, Noble B, Bailey M. Bobtail: Avoiding long tails in the cloud. In *Proc. the 10th USENIX Conference on Networked Systems Design and Implementation*, Apr. 2013, pp.329-342.

[10] Alizadeh M, Greenberg A, Maltz D A, Padhye J, Patel P, Prabhakar B, Sengupta S, Sridharan M. Data Center TCP (DCTCP). In *Proc. ACM SIGCOMM*, Aug.30-Sept.3, 2010.

[11] Alizadeh M, Kabbani A, Edsall T, Prabhakar B, Vahdat A, Yasuda M. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In *Proc. the 9th USENIX NSDI*, April 2012, pp.253-266.

[12] Vamanan B, Hasan J, Vijaykumar T. Deadline-aware datacenter TCP (D2tcp). In *Proc. the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, Aug. 2012, pp.115-126.

[13] Wilson C, Ballani H, Karagiannis T, Rowtron A. Better never than late: Meeting deadlines in datacenter networks. In *Proc. the ACM SIGCOMM*, Aug. 2011, pp.50-61.

[14] Zats D, Das T, Mohan P, Borthakur D, Katz R. DeTail: Reducing the flow completion time tail in datacenter networks. *ACM SIGCOMM Comput. Commun. Rev.*, 2012, 42: 139-150.

[15] Hong C Y, Caesar M, Godfrey P B. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM Comput. Commun. Rev.*, 2012, 42: 127-138.

[16] Leverich J, Kozyrakis C. Reconciling high server utilization and sub-millisecond quality-of-service. In *Proc. the 9th European Conference on Computer Systems*, Apr. 2014, pp.4:1-4:14.

[17] Kapoor R, Porter G, Tewari M, Voelker G M, Vahdat A. Chronos: Predictable low latency for data center applications. In *Proc. the 3rd ACM Symposium on Cloud Computing*, Oct. 2012, pp.9:1-9:14.

[18] Sukwong O, Kim H S. Is co-scheduling too expensive for SMP VMs? In *Proc. the 6th Conference on Computer Systems*, Apr. 2011, pp.257-272.

[19] Uhlig V, LeVasseur J, Skoglund E, Dannowski U. Towards scalable multiprocessor virtual machines. In *Proc. the 3rd Conference on Virtual Machine Research and Technology Symposium — Volume 3*, May 2004, pp.43-56.

[20] Ding X, Gibbons P B, Kozuch M A, Shan J. Gleaner: Mitigating the blocked-waiter wakeup problem for virtualized multicore applications. In *Proc. the USENIX Annual Technical Conference*, June 2014, pp.73-84.

[21] Kambadur M, Moseley T, Hank R, Kim M A. Measuring interference between live datacenter applications. In *Proc. the International Conference on High Performance Computing, Networking, Storage and Analysis*, Nov. 2012, pp.51:1-51:12.

[22] Blagodurov S, Zhuravlev S, Fedorova A, Kamali A. A case for NUMA-aware contention management on multicore systems. In *Proc. the 19th International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2010, pp.557-558.

[23] Tam D, Azimi R, Stumm M. Thread clustering: Sharing-aware scheduling on SMP-CMP-SMT multiprocessors. In *Proc. the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, Mar. 2007, pp.47-58.

[24] Zhang Y, Laurenzano M A, Mars J, Tang L. SMiTe: Precise QoS prediction on real-system SMT processors to improve utilization in warehouse scale computers. In *Proc. the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2014, pp.406-418.

[25] Cazorla F J, Ramírez A, Valero M, Fernández E. Dynamically controlled resource allocation in SMT processors. In *Proc. the 37th Annual International Symposium on Microarchitecture*, Dec. 2004, pp.171-182.

[26] Choi S, Yeung D. Learning-based SMT processor resource distribution via hill-climbing. In *Proc. the 33rd International Symposium on Computer Architecture*, June 2006, pp.239-251.

[27] Eyerman S, Eeckhout L. Per-thread cycle accounting in SMT processors. In *Proc. the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2009, pp.133-144.

[28] Cheng L, Wang C L. vBalance: Using interrupt load balance to improve I/O performance for SMP virtual machines. In *Proc. the 3rd ACM Symposium on Cloud Computing*, Oct. 2012, pp.2:1-2:14.

[29] Wang G, Ng T S E. The impact of virtualization on network performance of Amazon EC2 data center. In *Proc. the 29th Conference on Information Communications*, Mar. 2010, pp.1163-1171.

[30] Xu Y, Bailey M, Noble B, Jahanian F. Small is better: Avoiding latency traps in virtualized data centers. In *Proc. the 4th Annual Symposium on Cloud Computing*, Oct. 2013, pp.7:1-7:16.

[31] Chiang R C, Huang H H. TRACON: Interference-aware scheduling for data-intensive applications in virtualized environments. In *Proc. the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2011, pp.47:1-47:12.

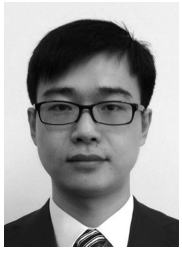
[32] Nathuji R, Kansal A, Ghaffarkhah A. Q-clouds: Managing performance interference effects for QoS-aware clouds. In *Proc. the 5th European Conference on Computer Systems*, Apr. 2010, pp.237-250.

[33] Mars J, Tang L, Hundt R, Skadron K, Soffa M L. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proc. the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2011, pp.248-259.

[34] Shieh A, Kandula S, Greenberg A, Kim C, Saha B. Sharing the data center network. In *Proc. the 8th USENIX Conference on Networked Systems Design and Implementation*, Mar.30-Apr.1, 2011.

[35] Cherkasova L, Gardner R. Measuring CPU overhead for I/O processing in the Xen virtual machine monitor. In *Proc. the USENIX Annual Technical Conference*, Apr. 2005, pp.387-390.

Yun-Gang Bao received his B.S. degree in computer science and technology from Nanjing University, Nanjing, in 2003, and Ph.D. degree in computer science from Chinese Academy of Sciences (CAS), Beijing, in 2008. He is a professor in Institute of Computing Technology, CAS, Beijing. From 2010 to 2012, he was a postdoctoral researcher in Department of Computer Science, Princeton University, New Jersey. His current research interests include computer architecture, operating system, system performance modeling and evaluation. He is a member of CCF, ACM, IEEE.



Sa Wang received his B.S. degree in computer science and technology from University of Science and Technology of China, Hefei, in 2009, and Ph.D. degree in computer science from Chinese Academy of Sciences (CAS), Beijing, in 2016. He is an assistant professor in Institute of Computing Technology, CAS, Beijing. His current research interests include operating system, system performance evaluation and optimization, and distributed system. He is a member of CCF.