Labeled RISC-V: A New Perspective on Software-Defined Architecture

Zihao Yu, Bowen Huang, Jiuyue Ma, Ninghui Sun, Yungang Bao
State Key Laboratory of Computer Architecture, ICT, CAS
{yuzihao,huangbowen,majiuyue,snh,baoyg}@ict.ac.cn

ABSTRACT

Traditional computer architectures are insufficient to convey important high-level requirements of applications to the hardware. These requirements include QoS and security, which are extremely important to data centers in the cloud era. To guarantee better QoS in data centers, we propose a new computer architecture LvNA (Labeled von Neumann Architecture) that leverages labeling mechanism and programmable label-based policies to enable computer hardware with more software-defined functionalities.

In this paper, we will present the motivations and design principles of LvNA as well as an FPGA-based prototype (i.e., labeled RISC-V). We demonstrate how to reconstruct the RISC-V's in-order core RocketChip to be a software-defined architecture with the concept of LvNA. Additionally, we will explore the change of software stack in data center to make better use of LvNA, including hypervisors, OS kernels and cluster management systems, compilers, and runtime libraries.

CCS CONCEPTS

• **Computer systems organization** → *Cloud computing*;

KEYWORDS

label, software-defined, QoS

ACM Reference Format:
Zihao Yu, Bowen Huang, Jiuyue Ma, Ninghui Sun, Yungang Bao. 2017. Labeled RISC-V: A New Perspective on Software-Defined Architecture. In *Proceedings of Computer Architecture Research with RISC-V, Boston, MA, USA, October 2017 (CARRV 2017),* 7 pages.
https://doi.org/10.475/123_4

1 INTRODUCTION

Nowadays datacenters have become the infrastructure of information technology. According to some studies [35], most of requests of online applications will spend nearly half of their lifetime in datacenters to be processed. Meanwhile, the cost of datacenters is very high. It is revealed that it costs 15 billions dollars for Microsoft to build a new datacenter [15].

In fact, datacenters confront the conflict of high utilization and high Quality of Service(QoS). On one hand, to achieve high utilization, one straightforward approach is to co-locate workloads or virtual machines (VM) on each physical server. In this manner, However, co-location induces contention for various shared hardware resources (such as CPU cores, caches, memory bandwidth, and network switches [14]), as well as shared software resources (e.g., page caches, socket buffers, and multiple layers of queueing buffers [14, 21]). Such contention causes unpredictable performance variability [28, 29, 31] that is amplified at data center scales [14] where online services involve tens to hundreds of servers in processing even a single user request [32]. Moreover, such performance variability occurs frequently due to the unpredictability of frequent, short-running workloads. A month of profiling data from a 12,000-server Google data center shows that more than a million jobs ran for only a few minutes [36].

On the other hand, to guarantee QoS of latency-critical online services, data center operators or developers tend to avoid sharing by either dedicating resources or exaggerating reservations for online services in shared environments. In fact, without co-location industry-wide utilization is even lower — only between 6% [20] to 12% [2]. For shared environments, Reiss et al. [36] verify that developers indeed exaggerate resource requirements: in their 12,000-server data center, actual CPU and memory utilizations are only 20% and 40% respectively, while the average reservations are 75% and 60%.

It is true that industry are struggling with this conflict. Google's batch-workload data centers achieve 75% CPU utilization, on average [12]. But typical online-service datacenters exhibit only about 30% CPU utilization, on average, much lower than batch-workload data centers [12]. Amazon's cloud service tries to improve utilization, without the guarantee of QoS. But DropBox announced that they dropped Amazon's cloud to have better overall user experience [9].

In this paper, we ask the following question: Can we obtain a guaranteed QoS for a critical application, meanwhile trying to improve the utilization of the datacenter? Specifically, co-location is an effective way to improve utilization. However, this will lead to interference over different resources, especially for those hardware resources without effective management. Therefore, we should improve the control ability for traditional hardware. In face, Dick Sites, an expert of datacenter in Google, has called for innovation of hardware support to solve this conflict. The community white paper **21st Century Computer Architecture** [10] also suggests that *"new, higher-level interfaces are needed to encapsulate and convey programmer and compiler knowledge to the hardware, resulting in major efficiency gains and valuable new functionality."*

To improve the control ability of hardware, there are two challenge.

How to give hardware the ability of identifying requests from critical applications? It is possible for hardware to give

CARRV 2017, October 2017, Boston, MA, USA

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

high priority to requests from critical application, only after hardware can identify them. To solve this challenge, we propose a new architecture: Labeled von Neumann Architecture(LvNA) [11]. LvNA is inspired from Software-Defined Network. There are four features in LvNA. (1) Fine-grain Object - attaching a label to each memory and I/O request. (2) Semantic-Gap - correlating labels with VM/Proc/Thread/Var. (3) Propagation - propagating labels in a whole machine. (4) DiffServ - providing differentiated services based on different label-indexed rules. With these features, it is easy for hardware to identify the software source of a request.

How to mitigate the conflict of high QoS and high utilization with such an ability? We propose Programmable Architecture for Resourcing-on-Demand(PARD) [27], a mechanism leveraging labeling to perform request-level control with software/hardware co-design. To solve this problem, there are further four steps to go. (1) To deploy DiffServ into each shared resources in hardware, PARD uses parameter tables to store label-based control policies. (2) To know whether the critical application is interfered, PARD uses statistics tables to monitor the usage of each shared resources. (3) To protect the critical application from future interference, PARD uses trigger tables to adjust control policies once interference occurs. (4) To manage these programmable tables in a unified manner, PARD uses adapt platform resource manager(PRM) to abstract these tables as a device file tree.

With the concept of LvNA, we reconstruct RocketChip [5] to a software-defined architecture, called labeled RISC-V, and implement some control logics of PARD over it. We verify our work on Xilinx VC709 evaluation board. With LvNA, we can simultaneously boot unmodified OS in multiple VMs, without the help of software hypervisors. Besides, we implement label-based token bucket mechanism, to achieve precise control over memory bandwidth allocation. Evaluation results show that, we can dynamically adjust the allocation of memory bandwidth in a fine-grained manner. By this mean, we can achieve the goal that improving memory bandwidth utilization, while satisfying different QoS targets. Finally, the resource overhead of our mechanism is about than 7.83%, while we do not introduce any performance overhead.

In summary, we propose: (1) LvNA, a novel computer architecture, which conveys software semantic information to hardware by label, to equip hardware with the ability of identifying software sources. (2) PARD, a case to control hardware at request level over LvNA, to achieve guarantee of QoS and high utilization. (3) Labeled RISC-V, a prototype proof of LvNA and PARD.

2 BACKGROUND

We first describe the problems of traditional architecture about why it can not achieve guaranteeing QoS under sharing. Then we describe the background of DiffServ as well as SDN to conclude how they solve similar problem in the network area.

2.1 Unmanaged sharing: The Problem

In the traditional multi-core architecture, multiple applications can run over it simultaneously. They share the hardware resources including last-level cache(LLC), memory bandwidth, as well as I/O bandwidth. This makes it possible to improve the resource utilization with multi-core architecture. When an application can not fully utilize the hardware resources, running another application at the same time can make good use of the remaining resources.

However, when the resources are with heavy contention, it may introduce performance interference. There are a lot of studies about this "unmanaged sharing" phenomenon. When two cache-sensitive applications run separately, they both run good. However, when they run together on the same machine, they both suffer from unpredictable performance degradation up to 3.3 times [41]. Some researches over Google datacenters show that, this interference exists over all shared resources, including hyper-threading, LLC, DRAM, network [26]. To mitigate the interference, the mainstream of current solutions adapted by industry utilize software technique to schedule those applications with severe contentions to different physical machines.

The root cause of unmanaged sharing is that, the low-level hardware lacks of software semantic information. The hardware controllers of shared resources can not tell which source a request comes from. This results that the controllers can not determine how critical a request is, which is important for the controllers to guarantee QoS of critical applications. Therefore, we need a way to convey the software semantic information down to hardware. Only this mean can make it possible to mitigate resource contentions at the hardware level.

2.2 DiffServ and SDN: The Inspiration

Historically, Internet confronted with similar dilemma in the 1990s when a lot of network applications emerged, such as streaming video, e-commerce, email and file transfers, which exhibited varying QoS requirements. To deliver end-to-end QoS on an IP-network, the Internet Engineering Task Force (IETF) proposed Differentiated Services (DiffServ) [37] in 1998. The key idea of DiffServ is defining an 8-bit Differentiated Services field (DS field) in the IP header to designate an application's requirements so that routers can leverage the DS field to manage each packet for applications' different requirements.

The advent of software-defined networking (SDN) [6] further facilitates network management and traffic engineering. There is one prevalent interpretation of SDN's key principles:

- Each network packet is attached with a label like *flowid* in OpenFlow [4].
- The label contains application information.
- The label is propagated in the whole network.
- SDN devices will process packets differentiately according to their lables.

The prevalence of DiffServ and SDN motivate us to investigate whether it is feasible to apply SDN's principles to computer architecture. We find a key observation that **a computer is inherently a network**. As shown in Figure 1, a computer can be viewed as an intra-computer network[1]; hardware components communicate with each other via different types of packets, such as network-on-chip (NoC), QuickPath Interconnect (QPI) [17] and PCIe packet; apart from processing packets, the controllers of hardware components actually play the role of network routers, i.e., forwarding

[1]The BarrelFish OS work [13] makes a similar observation that a computer is already a distributed system.

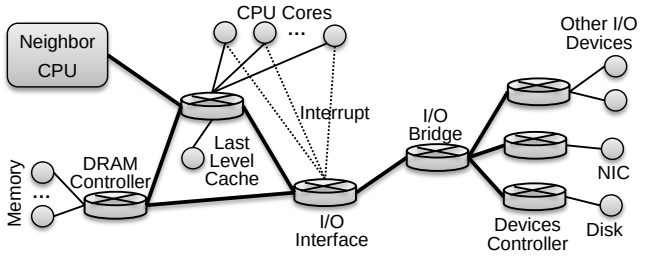

Figure 1: A traditional computer can be viewed as a network.

packets to a next hop. Therefore, it should be possible to apply similar technique to computer architecture.

3 LVNA: A NOVEL ARCHITECTURE

To convey the software semantic information down to hardware, we propose a novel computer architecture, Labeled von Neumann Architecture(LvNA). LvNA is inspired by SDN. It has the following four features.

Fine-grained Object. Every hardware requests will be attached with a label. Specifically, we add a DiffServ ID (DS-id) label register into all sources that generate requests, including each CPU core and every I/O device. These registers are used for labeling cache-access, memory-access, DMA.

Semantic-Gap. The labels are correlated with software semantic information. Specifically, labels can be used to identify different entities, including virtual machines, processes, threads, even more find-grained entities such as functions and variables. For the sake of simplicity, we first discuss virtual machine level labeling. That is, requests with different labels mean they come from different virtual machines.

Propagation. When a DS-id label is attached to a request at source-end, the label will travel along with the request during its whole lifetime until the request is completed. To achieve this, we can benefit from bus. After adding serval bits on the bus to present labels, the bus will transfer the labels automatically. However, we should pay attention to LLC. Usually a write operation is divided into multiple phases due to caching. For example, at the LLC level, the first phase of a write request to DRAM only stores data in LLC and marks the data block as dirty. For the second phase, the dirty data block is selected to be evicted when a cache miss occurs, and then written back to DRAM. During these latter phases, we need to determine which label should be assigned to a writeback request. To resolve this issue, We find that it is necessary to store DS-id if there are multi-phase writeback requests, otherwise resulting in inaccuracy. For example, assume that the current request is from VM1 and attached with DS-id1, the cache block to be written back belongs to VM2 with DS-id2. So the writeback request issued to DRAM has to use the DS-id1. Consequently, the memory controller may believe it is a request coming from VM1, which is not the case. Therefore, for current LLC design, once a piece of requested data is filled into LLC, its DS-id is stored into the tag array of LLC and is marked as owner DS-id. When the data is to be written back to DRAM, the owner DS-id will be assigned to the writeback request.

DiffServ. Hardware should provide mechanisms to process requests with different labels by different policies. Since each request contains a label, we introduce a programmable control logic into every shared resource to make use of the label. Once receiving a request, a control logic first serves the request according to its DS-id and then generates a new request attached with the same DS-id to next component.

4 PARD: A CASE OF LVNA

LvNA defines the features which an architecture should have to convey the software semantic information down to hardware. But LvNA does not define how the hardware should use this information. Different label-based policies will lead to different usages of LvNA. To mitigate the conflict between guarantee of QoS and high utilization in datacenters, we propose Programmable Architecture for Resourcing-on-Demand(PARD). PARD is a case of LvNA to improve the controllability of datapath in the traditional architecture. Other usages of LvNA, for example, security, will not be discussed here.

4.1 overview

Figure 2 shows an overview of PARD based on LvNA. There are various hardware components that behave differently and use DS-id labels in different ways as well, e.g., LLC using DS-id for capacity allocation or memory controller using DS-id for bandwidth allocation. We devise a basic control logic structure for a variety of components. Specifically, this structure consists of three DS-id indexed tables, i.e., a *parameter table* storing resource allocation policies, a *statistics table* storing resource usage information and a *trigger table* storing performance triggers. Besides, the structure includes a programming interface and an interrupt line. All control logics are connected to a centralized platform resource manager. This control logic structure can be easily instantiated and integrated into LLC, memory and I/O controllers. More details will be described below.

4.2 Control Logic Design

To support differentiated services for different requests, PARD uses parameter tables to store label-based rules in each control logics. For example, in the cache control logic, we can add label-indexed waymask column in the parameter table. When requests with different labels arrive, cache controller will use their labels to retrieve their waymask information stored in the parameter table, then determine which ways for each request can go to. In the memory controller, parameter table can contain policies about address mapping and the priorities of memory accessing. Requests with different labels will also be mapped into different memory segments. Memory controller can also schedule them according to their different priorities.

To know whether the critical application is interfered, PARD uses statistic tables to monitor the usage of each shared resources. In fact, statistic tables act as per-label performance counters. By attaching labels with each requests, it is easy to count hardware events separately, such as per-label cache hit/miss, per-label memory accessing.

To protect the critical application from future interference, PARD introduces trigger-action mechanism. The trigger is a condition referring statistics table, such as cache miss rate greater than 30%.

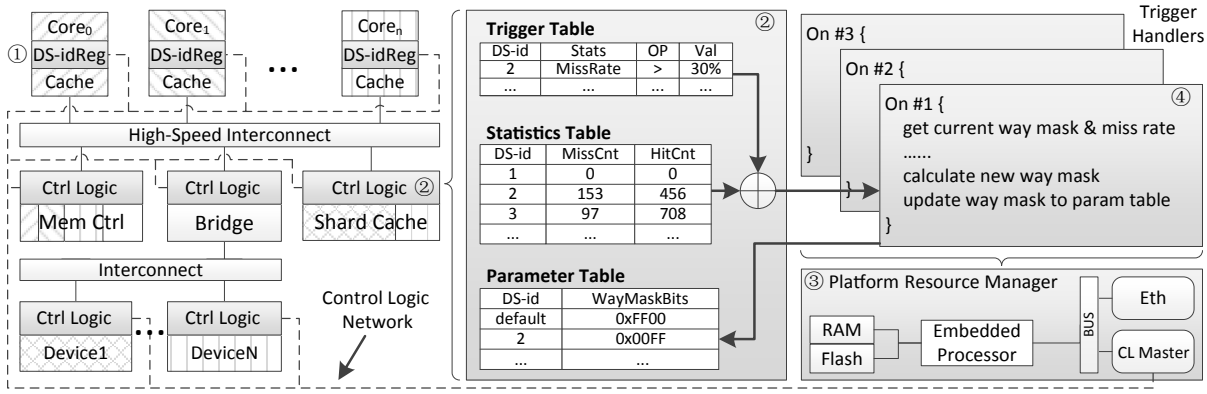

Figure 2: PARD Architecture Overview. The grey boxes represent PARD components.

The action runs in the firmware to adjust parameter table. For example, we can adjust the way partition configuration when the miss rate trigger condition is satisfied, allocating more ways to the application with high priority. The trigger condition and action signal are stored in trigger table.

Finally, each control logic contains three tables, a programming interface and an interrupt line. These three tables are all accessible though the programming interface. When a condition in the trigger table is satisfied, an interrupt will be sent out with pre-defined notify message.

4.3 Platform Resource Manager

To manage the programmable tables above in a unified manner, PARD includes a per-computer centralized platform resource manager(PRM). PRM acts similar as IPMI [3] in conventional servers. It connects all control logics and label registers (see the dash lines in Figure 2). PRM is essentially an embedded system-on-chip (SoC) that consists of an embedded processor, RAM, flash, a local bus, an Ethernet adaptor and the control logic master.

A Linux-based firmware running on PRM abstracts all control logics as a device file tree that is logically centralized. The firmware provides a uniform file based programming interface to access different control logics, along with a more advanced trigger-action programming methodology (see below) for operators to create and deploy resource management policies.

4.4 Trigger-Action Programming Methodology

To facilitate operators to define resource management policies and program control logics, we propose a trigger-action programming methodology.

Operators can define several DS-id label-based trigger-action rules, each of which targets a set of hardware resources. Particularly, *triggers* are based on performance metrics such as LLC miss rate and are stored in trigger tables; an *action* (a.k.a., trigger handler) can be written in any languages as long as they support file primitives. Operator-defined rules are installed in the device file tree of the firmware. It is worth noting that thank to the centralized PRM, *trigger* and *action* can be designated to different resources. For instance, if a trigger is created to monitor memory bandwidth, its action can be defined to adjust LLC capacity because LLC miss rate is strongly correlated with both memory bandwidth and LLC capacity.

Data center operators define *actions* and trigger-action rules to represent different resource management policies that are correlated to service-level agreements (SLA). Therefore, users can choose suitable SLA according to their QoS requirements.

5 IMPLEMENTATION

We reconstructed RocketChip, an open-sourced RISC-V implementation, into a software-defined architecture, called Labeled RISC-V, with the concept of LvNA. To verify PARD can mitigate the conflict between guarantee QoS and high utilization. then we implement some control logics of PARD over it. We perform the above evaluation on a Xilinx VC709 FPGA development board.

For the sake of simplicity, we configure RocketChip into dual-core. However, the state-of-the-art implementation of RocketChip does not have an LLC. And due to some implementation issues, we have not made the PCI-e devices fully work. Therefore, we can only show some evaluation results on memory control logic to verify the basic idea of LvNA as well as PARD.

We show how to reconstruct RocketChip into Labeled RISC-V architecture. First, we add DS-id label registers after each core tiles. Each request comes out of a core tail will be attached the label stored in the corresponding label register. Then we associate the labels with their virtual machines running on the core tiles. To propagate labels in the whole uncore network, we add a new member *dsid* to the bundle of TileLink2. As the TileLink2 masters, core tiles will assign this new member with the value of label. Then the TileLink2 implementation will help us to propagate labels automatically. However, there are still peripherals outside the RocketChip. To propagate labels to them, we do some small modifications to the TileLink2/AXI4 Bridges. These modifications will support converting the *dsid* member in TileLink2 into the *user* signals of the AXI protocol, and vise versa.

Currently we have only implemented the memory control logic over Labeled RISC-V. The memory control logic supports memory address mapping. The address mapping is implemented by a label-indexed memory base column. When a request arrives, memory

control logic will retrieve the memory base by the label attached in the request, and add this memory base to the address field of the request. This function help to achieve fully hardware-supported virtualization, which will be introduced in § 6.1.

We also implement label-based token bucket mechanism in the memory control logic. We add three columns into the parameter table. They represent the bucket size, token addition frequency and token addition amount, respectively. The addition frequency is in unit of cycles. Combining the addition frequency with addition amount, we can represent any token rate in the token bucket mechanism. This makes it possible for memory control logic to allocate dedicated memory bandwidth to any applications, with the help of labels.

6 EVALUATION

6.1 Fully Hardware-Supported Virtualization

This experiment demonstrates the effectiveness of fully hardware-supported virtualization enabled by LvNA. We partitioned the dual-core RocketChip into two VMs that share hardware resources. We launch the unmodified Linux in each VM, then we run some SPEC CPU2006 workloads in each VM.

RISC-V currently does not have any hypervisor supports. However, with the help of LvNA, hardware is able to distinguish requests from different VMs, and perform resource isolation over them. This means that in LvNA, hardware can replace some basic functionalities of software hypervisor to manage VMs. Specifically, in our implementation, memory control logic is responsible for isolating memory regions which VMs access. By the address mapping mentioned above, memory control logic is easy to map memory accessing requests of different VMs into different physical memory region.

6.2 Label-based token bucket

Based on the VM isolated by LvNA above, we run stream benchmark [7] in one VM. When stream runs alone, it consumes about 16.7 MB/s of memory bandwidth. However, it is difficult to generate interference over memory bandwidth by only running applications on RocketChip. This is because the frequency of CPU on FPGA is only 100MHz, but the frequency of memory controller is 200MHz, much higher than CPU frequency.

To address this issue, we add a memory traffic generator in hardware to utilize memory bandwidth and introduce contention. With the generator enabled, the memory bandwidth drops to about 9.1 MB/s. By setting the parameters of label-based token bucket to restrict the access rate of the generator, the memory bandwidth of stream can restore to about 16.3 MB/s. With token bucket enabled, the total memory bandwidth is nearly fully utilized by co-locating stream and the generator, while the actual performance of stream is not hurt too much.

6.3 Overhead

Code complexity. To add the lable feature into RocketChip, we only add 16 line of scala codes. Thanks to the abstraction of Chisel [1], these 16 line of codes already implement the first three features of LvNA, including fine-grained object, semantic-gap and propagation.

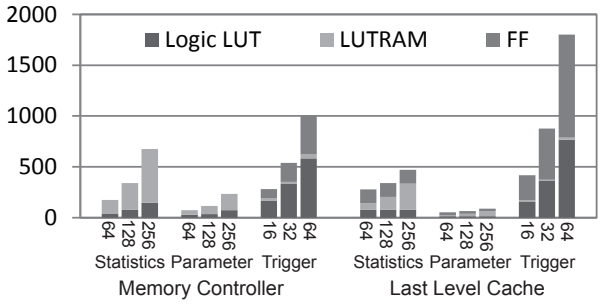

Figure 3: FPGA Resource Usage of two control logics for memory controller and last level cache. For reference, an original memory controller and 768KB 12-way LLC (w/ tag array only) consume 15178 and 75032 LUT/FF respectively.

To implement programmable control logics, we add another 800 line of scala codes.

Since our Labeled RISC-V implementation has not fully supported PARD features yet, for reference, here we show some overhead data of our previous implementation of PARD over the MicroBlaze platform.

Latency. We find that the LLC control logic does not introduce extra latency. This is because the processing logic of the LLC control logic can be hidden into the pipeline of the LLC controller. In fact, pipelining is a typical design for modern CPUs' LLC. Furthermore, synthesis data verify that the logic of the LLC control logic is not on the critical path.

For the memory control logic, cycle accurate simulation results show that address mapping can be done by combination logic without any cycle delay. On the other hand, synthesis results show that this logic is not on the critical path, either. We can conclude that the address mapping function does not introduce any actual delay. This is also the case with our Labeled RISC-V implementation. For the label-base token bucket, latency will be introduced for non-critical applications. The actual latency dependents on the parameters of token bucket configured in the parameter table. Therefore, it is not easy to give a specific number.

FPGA Resources. The required FPGA resources of a control logic mainly depend on the number of table entries of the three tables.

Figure 3 shows synthesis data reported by Xilinx Vivado Design Suite [8]. Specifically, the 256-entry parameter and statistics tables of the memory control logic require 220 LUT for logic and 688 LUTRAM for storage. The 64-entry trigger table consumes more logic resources (582 LUT + 387 FF) than storages (40 LUTRAM) because it requires many comparators to implement triggers. Two priority queues with 16-depth require only 324 LUT + 30 FF. The total required FPGA resources (1189 LUT/FF) accounts for 7.83% of the original Xilinx MIGv7 memory controller (15178 LUT/FF).

The LLC control logic exhibits similar resources consumption. The 256-entry parameter and statistics tables and the 64-entry trigger table require 2359 LUT/FF, introducing only 3.1% extra FPGA resources compared with the original LLC controller (not counting data array). Additionally, storing *Owner_DS-id* in tag array adds 50% more of blockRAMs (from 12 to 18). This is because the

original tag array stores 28 bits for each cache block and the DS-id in our implementation is 8 bits. In fact, the tag array consumes very little FPGA resources compared with the whole cache, thus these extra six blockRAMs are negligible.

7 DISCUSSION

In this section, we will discuss some open problems brought by LvNA [11]:

Theory: What is the impact of LvNA on RAM, PRAM, LogP models? To elaborate, theoretically we can throttle all of the low priorities requests 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/Arch: 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 do 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 info 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 also can be vague on system memory, since "write to memory" requests from cache is largely caused by cache evictions of dirty blocks, and the eviction may be invoked by cache read miss requests with different labels. Therefor one sometimes would be forced to gauge and audit resource usage on such a difficult scenario.

8 RELATED WORK

Hardware based techniques. Kasture and Sanchez propose Ubik [22], a cache partitioning policy that characterizes and leverages the transient behavior of latency-critical applications to maintain their target tail latency. Vantage [39] implements fine-grained cache partitioning using the statistical properties of Zcaches [38]. Utility-based cache partitioning (UCP) [33] strictly partitions the shared cache depending on the benefit of allocating different number of ways to each application. Muralidhara *et al.* propose an application-aware memory channel partitioning (MCP) [30] to reduce memory system interference. However, these work usually focuses on only one type of resource while PARD is able to simultaneously manage all shared hardware resources within a server.

Similar to LvNA, NoHype [23] removes the virtualization layer and makes use of hardware virtualization extensions to partition a server into multiple submachines. However, this partitioning is static. In contrast, LvNA allows operators to dynamically partition a physical server into multiple VMs. Furthermore, PARD based on LvNA supports trigger-action mechanism to deploy resourcing-on-demand resource management policies.

Architectural support for QoS. Rafique *et al.* [34] propose a OS-driven hardware cache partitioning mechanism that tags cache requests and allows OS to adjust cache quota according to the tag. Sharifi *et al.* [40] further propose a feedback-based control architecture for end-to-end on-chip resource management. Iyer *et al.* make substantial contributions in architectural support for QoS [16, 18, 19, 24, 25]. The closest work to PARD is class-of-service based QoS architecture (CoQoS) [24, 25], which assigns a priority tag to each on-chip request and allows cache/DRAM/NoC to schedule the request according to the priority tag.

LvNA and PARD differ from these prior proposals in the following aspects: (1) Prior work primarily focuses on on-chip resources such as cache, NoC and memory bandwidth (managed by on-chip memory controller) while LvNA is able to manage not only on-chip resources but also I/O resources. (2) We design programmable control logics for PARD and a uniform programming interface while prior work does not support programmability. (3) PARD includes a centralized platform resource manager and a Linux-based firmware to facilitate operators' management. (4) LvNA supports not only QoS but also NoHype-like virtualization that can partition one server into multiple VMs. (5) PARD allows users to install trigger-action rules while prior work lacks this useful mechanism.

9 CONCLUSION

This paper presents a study of building a software-defined architecture. We first propose Labeled von Neumann Architecture(LvNA) to equip the hardware with the ability of identifying requests from different software. Then we propose programmable architecture for resourcing-on-demand(PARD), leveraging such an ability to improve the controllability of traditional hardware. We develop Labeled RISC-V, an FPGA implementation with the concept of LvNA and PARD. Our experiments demonstrated that PARD is able to address the trade-offs between high utilization and high QoS in datacenter environments.

We believe that LvNA is a promising architecture to enable hardware with more functionalities. PARD makes a case for this direction and provides new interfaces for users to interact with the hardware. To encourage researchers to explore more on LvNA and PARD, we release our Labeled RISC-V implementation at https://github.com/fsg-ict/labeled-RISC-V

10 ACKNOWLEDGEMENT

This work was supported by the National Key R&D Program of China under Grant No. 2016YFB1000201 and the National Natural Science Foundation of China under Grant No. 61420106013.

REFERENCES

[1] [n. d.]. Chisel. https://github.com/freechipsproject/chisel3. ([n. d.]).

[2] [n. d.]. Gartner says efficient data center design can lead to 300 percent capacity growth in 60 percent less space. http://www.gartner.com/newsroom/id/1472714. ([n. d.]).

[3] [n. d.]. Intelligent Platform Management Interface (IPMI). http://en.wikipedia.org/wiki/Intelligent_Platform_Management_Interface. ([n. d.]).

[4] [n. d.]. OpenFlow Switch Specification. https://www.opennetworking.org/sdn-resources/openflow/. ([n. d.]).

[5] [n. d.]. RocketChip. https://github.com/freechipsproject/rocket-chip. ([n. d.]).

[6] [n. d.]. Software-Defined Networking. https://www.opennetworking.org/sdn-resources/sdn-definition/. ([n. d.]).

[7] [n. d.]. STREAM: Sustainable Memory Bandwidth in High Performance Computers. http://www.cs.virginia.edu/stream/. ([n. d.]).

[8] [n. d.]. Vivado Design Suite. http://www.xilinx.com/products/design-tools/vivado/. ([n. d.]).

[9] [n. d.]. Why Dropbox dropped Amazon's cloud. http://www.networkworld.com/article/3045570/cloud-computing/why-dropbox-dropped-amazons-cloud.html. ([n. d.]).

[10] 2012. Computing Community Consortium (CCC). 21st Century Computer Architecture. *A community white paper* (2012). http://cra.org/ccc/docs/init/21stcenturyarchitecturewhitepaper.pdf

[11] Yun-Gang Bao and Sa Wang. 2017. Labeled von Neumann Architecture for Software-Defined Cloud. *Journal of Computer Science and Technology* 32, 2 (01 Mar 2017), 219–223. https://doi.org/10.1007/s11390-017-1716-0

[12] Luiz Andre Barroso, Jimmy Clidaras, and Urs Holzle. 2013. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. *Synthesis Lectures on Computer Architecture* 8, 3 (2013), 1–154.

[13] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. 2009. The Multikernel: A New OS Architecture for Scalable Multicore Systems. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP '09)*. New York, NY, USA, 29–44.

[14] Jeffrey Dean and Luiz Andre Barroso. 2013. The Tail at Scale. *Commun. ACM* 56, 2 (Feb. 2013), 74–80.

[15] L. Albert Greenberg. 2015. *SDN for the Cloud.* Technical Report. SIGCOMM Keynote.

[16] Andrew Herdrich, Ramesh Illikkal, Ravi Iyer, Don Newell, Vineet Chadha, and Jaideep Moses. 2009. Rate-based QoS techniques for cache/memory in CMP platforms. In *Proceedings of the 23rd international conference on Supercomputing*. 479–488.

[17] Intel. 2009. *An Introduction to the IntelÂő QuickPath Interconnect.*

[18] Ravi Iyer. 2004. CQoS: a framework for enabling QoS in shared caches of CMP platforms. In *Proceedings of the 18th annual international conference on Supercomputing*. 257–266.

[19] Ravi Iyer, Li Zhao, Fei Guo, Ramesh Illikkal, Srihari Makineni, Don Newell, Yan Solihin, Lisa Hsu, and Steve Reinhardt. 2007. QoS Policies and Architecture for Cache/Memory in CMP Platforms. In *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '07)*. New York, NY, USA, 25–36.

[20] James M Kaplan, William Forrest, and Noah Kindler. 2008. Revolutionizing Data Center Energy Efficiency. *Technical report, McKinsey & Company* (2008).

[21] Rishi Kapoor, George Porter, Malveeka Tewari, Geoffrey M. Voelker, and Amin Vahdat. 2012. Chronos: Predictable Low Latency for Data Center Applications. In *Proceedings of the Third ACM Symposium on Cloud Computing (SoCC '12)*. New York, NY, USA, Article 9, 14 pages.

[22] Harshad Kasture and Daniel Sanchez. 2014. Ubik: Efficient Cache Sharing with Strict Qos for Latency-critical Workloads. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14)*. New York, NY, USA, 729–742.

[23] Eric Keller, Jakub Szefer, Jennifer Rexford, and Ruby B. Lee. 2010. NoHype: Virtualized Cloud Infrastructure Without the Virtualization. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA '10)*. New York, NY, USA, 350–361.

[24] Bin Li, Li Shiuan Peh, Li Zhao, and Ravi Iyer. 2012. Dynamic QoS Management for Chip Multiprocessors. *ACM Trans. Archit. Code Optim.* 9, 3 (Oct. 2012), 17:1–17:29.

[25] Bin Li, Li Zhao, Ravi Iyer, Li Shiuan Peh, Michael Leddige, Michael Espig, Seung Eun Lee, and Donald Newell. 2011. CoQoS: Coordinating QoS-aware shared resources in NoC-based SoCs. *J. Parallel and Distrib. Comput.* 71, 5 (2011), 700–713.

[26] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. 2015. Heracles: Improving Resource Efficiency at Scale. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture (ISCA '15)*. ACM, New York, NY, USA, 450–462. https://doi.org/10.1145/2749469.2749475

[27] Jiuyue Ma, Xiufeng Sui, Ninghui Sun, Yupeng Li, Zihao Yu, Bowen Huang, Tianni Xu, Zhicheng Yao, Yun Chen, Haibin Wang, Lixin Zhang, and Yungang Bao. 2015. Supporting Differentiated Services in Computers via Programmable Architecture for Resourcing-on-Demand (PARD). In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '15)*. ACM, New York, NY, USA, 131–143. https://doi.org/10.1145/2694344.2694382

[28] Jason Mars, Lingjia Tang, and Mary Lou Soffa. 2011. Directly Characterizing Cross Core Interference Through Contention Synthesis. In *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers (HiPEAC '11)*. New York, NY, USA, 167–176.

[29] Jason Mars, Neil Vachharajani, Robert Hundt, and Mary Lou Soffa. 2010. Contention Aware Execution: Online Contention Detection and Response. In *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO '10)*. New York, NY, USA, 257–265.

[30] Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda. 2011. Reducing memory interference in multicore systems via application-aware memory channel partitioning. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. 374–385.

[31] Onur Mutlu and Thomas Moscibroda. 2007. Stall-time fair memory access scheduling for chip multiprocessors. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*. 146–160.

[32] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, David Stafford, Tony Tung, and Venkateshwaran Venkataramani. 2013. Scaling Memcache at Facebook. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL, 385–398.

[33] Moinuddin K. Qureshi and Yale N. Patt. 2006. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. 423–432.

[34] Nauman Rafique, Won-Taek Lim, and Mithuna Thottethodi. 2006. Architectural Support for Operating System-driven CMP Cache Management. In *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques (PACT '06)*. New York, NY, USA, 2–12.

[35] Lenin Ravindranath, Jitendra Padhye, Ratul Mahajan, and Hari Balakrishnan. 2013. Timecard: Controlling User-perceived Delays in Server-based Mobile Applications. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13)*. ACM, New York, NY, USA, 85–100. https://doi.org/10.1145/2517349.2522717

[36] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. 2012. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing (SoCC '12)*. New York, NY, USA, Article 7, 13 pages.

[37] RFC2475. [n. d.]. An Architecture for Differentiated Services. http://tools.ietf.org/html/rfc2475. ([n. d.]).

[38] Daniel Sanchez and Christos Kozyrakis. 2010. The ZCache: Decoupling Ways and Associativity. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '43)*. Washington, DC, USA, 187–198.

[39] Daniel Sanchez and Christos Kozyrakis. 2011. Vantage: scalable and efficient fine-grain cache partitioning. In *ACM SIGARCH Computer Architecture News*, Vol. 39. 57–68.

[40] Akbar Sharifi, Shekhar Srikantaiah, Asit K. Mishra, Mahmut Kandemir, and Chita R. Das. 2011. METE: meeting end-to-end QoS in multicores through system-wide resource management. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*. 13–24.

[41] Christine Wang. 2014. Intel Xeon Processor E5-2600 v3 Product Family Performance & Platform Solutions. (2014).