Shaready: a Resource-Isolated Workload Co-location System

Shiqing Xue[12], Chunming Hu[12], Jianyong Zhu[12], Renyu Yang[34*]
[1]*SKLSDE Lab, Beihang University, China*
[2]*Beijing Advanced Innovation Center for Big Data and Brain Computing (BDBC), China*
[3]*School of Computing, University of Leeds, UK*
[4]*Edgetic Ltd., UK*
{*xuesq, hucm, zhujy*}*@act.buaa.edu.cn ; r.yang1@leeds.ac.uk*

Abstract—**Over a decade, cloud and subsequent joint cloud computing has been evolving into one of biggest disruptive technologies in modern digital age. The rapidly maturing cloud service and system management still heavily relies on virtualization which underpins Infrastructure as a Service (IaaS) to offer on-demand and low-cost computing services. Nevertheless datacenters still suffer from low utilization and resource imbalance. IaaS systems and their workloads, as legacy estates, are intricate to be migrated or re-planned, thereby increasing the complexity of utilization improvement. Arguably workload co-location of long-running applications encapsulated in virtual machines and latency-insensitive batch jobs is an alternative to improve overall resource utilization. However, guaranteeing the quality of long-running services is still challenging. In this context, we proposed an isolation-based cluster resource sharing system *Shaready* to enable workload co-residences. By means of global resource quota configuration and multi-resource isolation, long-running services in virtual machines can be prioritized with maximized resource provisioning. We implemented and validated it based on Openstack and Yarn clusters, and experiments demonstrate that system CPU and memory utilization can be improved by roughly 50% and 16.67% respectively on average with at most 7% performance degradation.**

Index Terms—**Cluster management, Co-location workloads, Resource isolation, Quality of service**

1. Introduction

The next generation of information technology represented by cloud computing and big data has become the mainstream of today's IT infrastructure. The rapid development of cloud computing and big data has also become the core growth driver of the new generation Internet data center (IDC). At the same time, IDC's virtualization-based approach to resource allocation has many problems. Amazon, one of the world's largest cloud service providers, pointed out in 2015 that cloud server clusters are mostly idle, resulting in an average physical utilization of clusters below 20% [1]. Another study [2] observes that a fraction of Amazon EC2 virtual machines (VMs) show an average CPU utilization of 7% over one week. To improve resource utilization, data centers co-locate different workloads and it has become a common practice [3] [4]. Long-running services in virtual machines have high quality of service requirements. So we are willing to use workload co-location technique to improve resource utilization whilst guanrantee QoS of long-running services in the shared cluster.

In this paper we propose *Shaready*, a resource-isolated workload co-location system. An improved resource utilization is fulfilled by co-locating batch jobs, while QoS of long-running services will be primarily prioritized. We leverage multi-dimensional resource isolation and real time resource monitoring to achieve resource re-usability and QoS control. Generic APIs are developed and provided to enable and simplify the integration of heterogeneous platforms. It is noteworthy that *Shaready* configures resources for co-located workloads dynamically by changing the resource quota for batch jobs during workloads execute. Experimental results show that compared with not using the system, *Shareday* can improved CPU and memory utilization by roughly 50% and 16.67% respectively on average with at most 7% performance degradation. The main contributions of this work are:

- We designed a resource-isolated workload co-location system and it can minimize the performance impact of batch jobs on long-running services. The management system can be easily deployed over multi-platforms and configures resources dynamically.
- The dynamic quota configuration is implemented through *Shaready* controller and resource isolation, by which we can dynamically change the resource quota for batch jobs during the workloads execute. Here, we used cgroup to manage cpu resources availability for batch jobs. As for memory resources isolation, we update the maximum amount of memory resources available to batch jobs through existing heartbeat mechanism, while reusing the dynamic thread monitoring technology.

. This work is supported by the National Key Research and Development Program (2016YFB1000503) and NSFC(61421003). Renyu Yang is the corresponding author

Organization. We firstly outline the background and related work in Section 2. We present the objectives and core design overview in Section 3 before describing the involved key techniques in Section 4. The experiments are given in Section 5. We conclude our work in Section 6.

2. Background and Related Work

The rapid advancement of cloud computing brought unprecedented progresses by on-demand and elastic computing and business models. The shared resource pool is commonly adopted by cloud vendors to reduce monetary costs and lift the management flexibility. However, large-scale IaaS systems severely suffer from low utilization and resource imbalance. For example, 30% of servers on average in the world are in a state where hardware resources are not being used effectively [5]. In general, the resources required for different types of workloads often differ. Different types of task loads, such as long-running applications, and batch jobs for massive data, have different characteristics and are typically implemented on different platforms. To a certain extent, physical cluster resources are not effectively utilized. Besides, different types of task loads have different degrees of dependence on different kinds of resources (CPU, memory, IO, etc.). As a result, traditional IDCs are underutilized with resources, which further aggravate such problems. From the trend of green IT, improvement for optimized computing power of IDCs, guaranteed quality of service and increased resource utilization is imperative. To wrap up, resource imbalance and low utilization are still challenging problems for cluster management systems in data centers, even in the joint cloud [6] and fog computing [7] environments.

Resource management systems have emerged to improve cluster resource allocation. Mesos [8], Yarn [9], Fuxi [10] and so on are the current mainstream resource management framework, and also the representative of the two-tier scheduling system, they decouple the resource management module and the task scheduling module. Similar work includes: Sparrow [11], ROSE [12], etc. Unfortunately, most existing batch job schedulers are not well compatible with online systems running VM tasks and cannot flexibly isolate and limit task resources. For Borg [13], most long-running applications run in containers instead of virtual machines. Facebook proposed Bistro [14] to replace the previously used Hadoop. Bistro is a scheduler that allows batch jobs to share clusters with online customer-facing workloads without harming the performance of either. Bistro treats large-scale data nodes as resource trees and organizes cluster resources hierarchically. In the production cluster resource configuration, Bistro adopts manual static configuration, which satisfies Facebook's demand for large-scale data processing tasks in production clusters. Static resource allocation cannot cope with the rapid change of resource usage, which does not meet our objectives. Besides, some IaaS systems, as legacy estates, are hard to be re-planned by new cluster management systems. Therefore, this paper we propose *Shaready*, a resource-isolated workload co-location system.

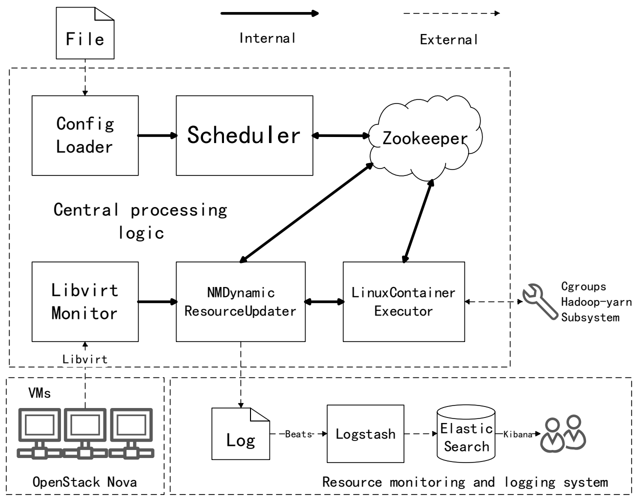


Figure 1. System architecture

3. System Design

3.1. Objectives

The main purposes of *Shaready* can be summarized as follows:

Flexible deployment over existing platforms. IaaS systems are mostly intricate to be re-planned. Due to this reason, Shaready is designed to adapt and incorporate different platforms via generic service invocation interfaces without huge modifications of existing systems.

Dynamic resource configuration for improved resource utilization. The majority of existing systems adopt static and manual configuration, indicating the difficulties of dynamic resource tuning among different loads. Since resource usage may sharply fluctuate during the creation and migration of virtual machines, it is potential for other batch jobs to temporarily reuse the idle resources. The key thing for elastic resource sharing is the over-subscribed resource should be timely revoked without impacting the expected resource planning.

QoS guarantee for long-running services. Virtual machine services are representative of long-running and resource-sensitive workloads in cloud environment. Higher degree of resource sharing implies increased resource contention and performance interference. Therefore, guaranteeing the QoS of long-running services is the priority.

3.2. System Overview

Figure 1 discribes the system architechture, and each part of the system works asynchronously. In the central processing logic, the central resource scheduling module can sense the resource changes of the cluster and communicate with the global resource view maintained by Zookeeper in real time. The configuration loader avoids hard coding, facilitates the deployment of the system in different environments, and is able to set multiple operational strategies flexibly. Libvirt Monitor module is used to obtain the resource usage of the virtual machine in real time. Since the Libvirt API supports multiple virtualization technologies such as KVM, Xen, and Qemu, we use it to implement the module. Zookeeper maintains a global forest resource view to update the resource usage information uploaded by each node of the cluster. When the module gets the latest global resource view, it synchronizes information with the scheduler. The scheduler updates the scheduling result of each node to Zookeeper in real time. Each node obtains the target value from Zookeeper and adjusts its resource allocation. We use Logstash and ElasticSearch to collect resource logs. Kibana is responsible for displaying resource information in real time. Users can observe the resource changes of the cluster in real time through the Web UI. Restrictions on the use of multidimensional resources are implemented through cgroups technology.

4. Key Techniques

This section discusses the implementation details of the system. The core components include: 1) global configuration loader and resource scheduling; 2) dynamic resource isolation modules; 3) resource monitoring module. The global configuration gives the available resource information of nodes in a cluster and the control strategy for dynamic resource isolation. Once a deployment scheme is determined, the resource monitoring and isolation will work to guarantee the QoS of long-running services while there are batch jobs running at the same time. To ensure the consistency and high availability of clusters, we designed a resource forest module for distributed resource awareness, which will be introduced in the last part in this section.

4.1. Global Configuration and Resource Partition

A global configuration ensure the co-residence of diverse frameworks (e.g., different legacy systems) properly. We adopt a two-tiered architecture: 1) A global scheduler (Level-0 controller) that manages the allocation quota among different computing frameworks and coordinates multiple resource scheduler at framework-specific level (Level-1 scheduler); 2) Framework level scheduler is responsible for the application-specific logics – managing the granted resources according to the global quota share and mapping waiting tasks to particular resources.

The sharing proportion is determined by the pre-defined policy and resource ratio that specifies the allowance that a specific framework can own. Different policies mainly represent different priorities of long-running services. Theoretically, the higher priority we set for long-running services, the stricter resource restrictions will be enforced on batch jobs. We also update the latest available resources by leveraging information provided by agents that are deployed on different machines.

4.2. Resource Isolation

By resource isolation technique, we can change the resource quota for batch job tasks dynamically during the workloads execute. For memory, resource reconfiguration signal is implemented based on heartbeat mechanism. Once the resources are insufficient, batch jobs will immediately trigger the event and be killed immediately. Nevertheless, CPU is an elastic resource which is more flexible, so we implemented CPU isolation mainly based on cgroups technology.

Memory Isolation. In order to avoid the interference of batch jobs on the operation of long-running services, the dynamic resource isolation module needs to limit the available resources of batch jobs in real time. In Yarn, the resource manager(RM) needs to determine whether the node's resources are sufficient to allocate the next container. The default resource calculation class *DefaultResourceCalculator* only consider the memory. That is, the only resource considered when the RM allocates a container to the node manager(NM) is whether the memory size is sufficient. Hadoop Yarn sets the maximum amount of physical memory resources of a node by changing the configuration file. The disadvantage of this configuration is that if you want to limit the resource consumption of batch jobs, you must restart the entire Hadoop platform. In order to achieve dynamic resource isolation, we implemented a dynamic adjustment module for memory resources in Hadoop Yarn.

The RM runs on a designated machine, allocating resources between competing applications. Multiple RMs can be launched for high availability, but only one is the primary RM. The NM periodically reports the status to the RM, which stores it as a cluster status. To achieve scalability, communication between RM and NM is based on a heartbeat mechanism. We put the NM resource update into every heartbeat. RM will change the available resource for each node after receiving its heartbeat, so that resources may not be allocated for new container for lack of resources. This process can be executed continuously while Hadoop Yarn still works. Once the resources are insufficient, the container will immediately trigger the event and be killed immediately.

CPU Isolation and Throtting. Unlike memory, the CPU is an elastic resource. That is, the task generally extends the makespan because of the limitation of the CPU, but it will not be killed immediately. In our system, the CPU limit for a specific task is implemented by cgroup, and its strategy can be expressed by the following formula.

$$N = CFS_{quota}/CFS_{period} \qquad (1)$$

N represents the number of cores the task can use. By adjusting the ratio of the formula, we can limit the overall CPU usage of the task. It should be noted that for multi-core CPUs, only such restrictions cannot prevent

Algorithm 1: getDomainMemoryStat

Input: *conn*

Output: RSS_{Memory}

```
RSS_Memory ← 0 , i ← 0
for id in listDomains(conn) do
    dom = domainLookupByID(id)
    MemStats[] = virDomainMemoryStat(VIR_DOMAIN_MEMORY_STAT_NR)
    for i in LengthOf(MemStats[])
        if MemStats[].tag = VIR_DOMAIN_MEMORY_STAT_RSS then
            RSS_Memory = RSS_Memory + MemStats[].value
    end
end
return RSS_Memory
```

Figure 2. Pseudo code for obtaining VM memory resources

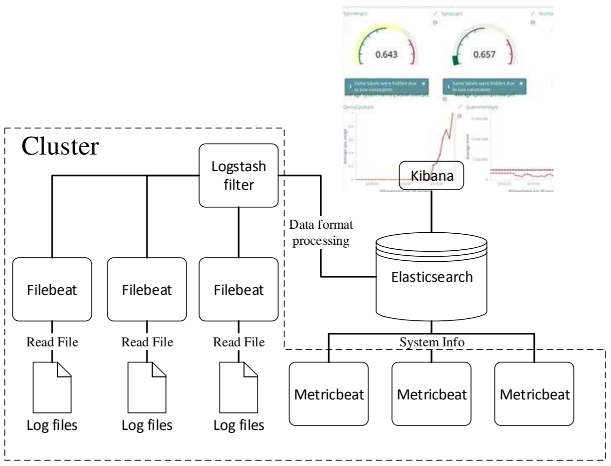


Figure 3. Logging and Monitoring

the load balancing operation of the Linux kernel. In this case, processes will run on all processor cores. Cgroups can provide kernel bindings to avoid the load balancing process caused by frequent multi-core scheduling. In our implementation, specific tasks are not bound to specific CPU cores by default for better performance.

4.3. Dynamic Resource Monitoring

This section corresponds to the Libvirt Monitor component depicted in Figure 1. To prioritize the performance of long-running applications, we need to recap the upper bound of available resource that Hadoop Yarn can use according to the resources occupied by virtual machines. Therefore, it is a key challenge to timely and accurately monitor occupied resources by virtual machines, particularly when a new virtual machine is launched or load bursting manifests (often accompanied with an non-negligible increase of resource utilization).

In the remaining section, we take memory and CPU resource collection as an example to demonstrate the relevant core idea. Due to libvirt can be adopted into different hypervisors, the proposed monitoring mechanism can be easily generalized.

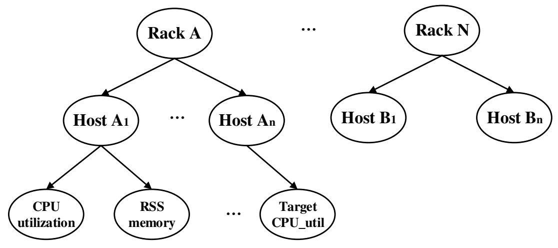


Figure 4. Global forest resource view

Minitoring. Figure 2 describes a pseudo code to obtain the memory resources. Specifically, the $conn$ represents the connection established with the hypervisor. The RSS_{Memory} represents the host memory usage. The basic flow of the program is to first get a list of virtual machine ids running on the entire machine, and then use the loop to read the RSS memory information of each virtual machine. By accumulating these values, we can get the total memory occupied by the virtual machine task on the current machine.

Libvirt does not provide a function to calculate the CPU usage of the virtual machine, so here we have to calculate it indirectly by the relationship between CPU time and CPU usage.

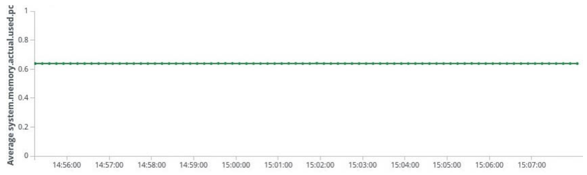
$$cpu_{usage} = \frac{cputime_{end} - cputime_{begin}}{realtime_{end} - realtime_{begin}} \times 100\% \quad (2)$$

In order to get a time difference, we generally need to let the program wait for a cycle time, after which we can get the CPU utilization according to formula 2. *Cputime* refers to the CPU time occupied by a virtual machine process, which is an absolute value. We need to calculate the difference and divide by the difference between the real time, which is the CPU usage of the virtual machine during this time. It should be noted that for multi-core CPUs, the value calculated at this time will be greater than 1.

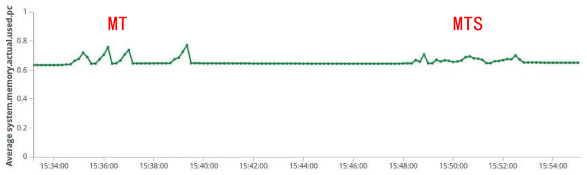
Logging System and Resource Visualization. The logging system and resource visualization module is shown in figure 3. We need to fetch information from two sources. One is the system metric information, such as the overall usage of the CPU and memory, using Metricbeat to collect data. The other is information from log files, such as monitoring information about virtual machine resources. Information from files needs to be formatted by Logstash. All information is then aggregated into the ES database and resource visualization is done through Kibana.

4.4. Global Forest Resource View

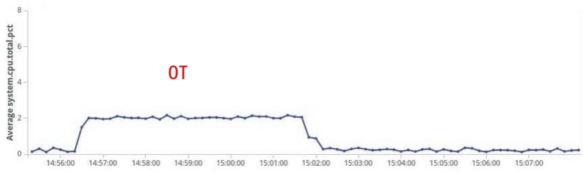
The cluster resource information stores in log files by resource monitoring module. In order to obtain detailed resource information of each node and complete resource isolation in real time. We design the global forest resource



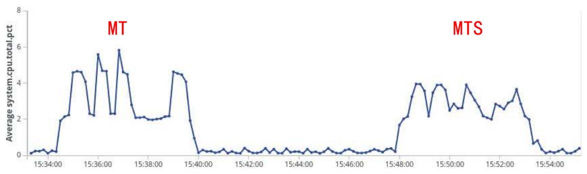
(a) Memory resource usage percent in OT case



(b) Memory resource usage percent in MT/MTS cases



(c) CPU usage percent in OT case(100% for one core)



(d) CPU usage percent in MT/MTS case(100% for one core)

Figure 5. Overall resource utilization in OT/MT/MTS cases

view, by which node status can be read and written at high speed and has high availability in clusters.

The global resource view of the cluster is established through Zookeeper. Figure 4 shows how the resource tree is built. The resource view presents a tree structure that records information including the rack number, host name, and resource information. The advantage of using Zookeeper is that high availability effectively avoids single points of failure, consistent performance of resource information and good read and write performance. The information of the child nodes of each host includes resource information of multiple dimensions on the current node and target resource values written by the scheduling module. The daemon running on the node completes the information interaction with the global resource view.

5. Evaluation

Environment Setup. To evaluate the performance of our system, we validated it on a minicluster with eight physical nodes. Each node has a 8 cores CPU @2.20GHz, 16GB memory and a 100GB disk running Ubuntu 14.04.5 LTS 64-bit system. Clusters are deployed with Openstack and Hadoop.

Metrics. The experiments will be performed upon three cases: 1) **OT** (Only long-running applications). Only a certain number of virtual machines are running in the cluster, and the virtual machine runs a typical long-running workload. 2) **MT** (Co-located mixed workloads without Shaready). We mix the virtual machine submissions and MapReduce batch jobs without running the system and same load inside the virtual machine. 3) **MTS** (Co-located mixed workloads with Shaready). Enable the system while co-located workloads are running.

Workloads. In our environment, three virtual machine instances are running on this node. Each instance has 512M of memory, a virtual core and 5GB disks, and is launched with *Ubuntu 12.04 cloud amd64 image*. Two of the three instances performed standard tests of *Tpcc-mysql*, simulating the actual access and database operations as a long-running application in the experimental scenario. During this time we submitted MapReduce batch jobs to the cluster. The batch jobs include WordCount and Monte Carlo methods for calculating pi.

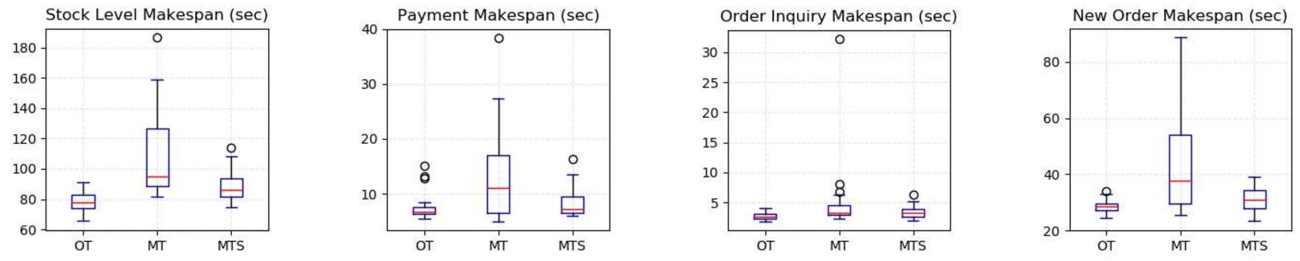
5.1. Efficiency: Resource Utilization

Firstly, we collected the CPU and memory in three cases to validate the effective improvement of resource utilization of *Shaready*. Figure 5 (a) and (b) are the real-time usage curves of node memory, and (c) (d) correspond to CPU usage. As shown in figure 5 (a), memory utilization kept stably at about 60% in OT case. We evaluated memory utilization in MT and MTS cases as shown in Figure 5 (b). In MT case, peak utilization can reach 80% while memory utilization fluctuates greatly. In MTS case, the average utilization can reach about 70% with minor fluctuations because of the dynamic resource isolation. Figure 5 (c) is the CPU usage in OT case, in which CPU utilization kept stably at about 20%. In figure 5 (d), it is observable that CPU utilization can reach 30% on average and 40% at certain time in MTS case. However, in MT case, CPU utilization can even reach higher, which means a better resource utilization, but the fluctuations will seriously affect the quality of long-running services. We will discuss this case in section 5.2.

In summary, *Shaready* increased the memory utilization by about 16.67% (60% to 70%). For CPU usage, *Shaready* was able to achieve a 50% CPU increment, with the average cluster CPU utilization increasing from 20% to 30% compared with not using co-location technique. Peak value can even achieve a 100% increment (20% to 40%).

5.2. Effectiveness: Performance

Both MT and MTS can bring an increase in resource utilization, which is in line with our expectations, but the impact of MT and MTS on the quality of service for long-running services varies widely. TPCC-mysql defines multiple types of services, and we measure the quality of long-running services through makespan. At the same time, we compare the performance score of the machine in the case of MT/MTS to measure the system's performance.



(a) Stock Level (b) Payment (c) Order Inquiry (d) New Order

Figure 6. Makespan of different types of workload in OT/MT/MTS cases

TABLE 1. Instance Performance Scores

Instance ID	Tpmc scores		
	OT	***MT***	***MTS***
testb	389.400	294.600	361.000
testc	384.600	312.800	358.600

Figure 6 shows makespans of four typical businesses of TPCC-mysql: stock level, payment, order inquiry, and new orders. We collected more than 30 sets of sample data for each business.

It can be seen that in the case of MTS, the completion time of the business is significantly better than that of the MT. First reflected in the average time, followed by the control of outliers. It solves the problem that the task execution time is long and the execution time is unstable. For the longest completion time of the business (including outliers), MTS has an improvement of about 50% compared to MT, while relative OT (baseline), elapsed time can be controlled within 5% to 10%.

Meanwhile we compare the performance scores of virtual machines in different cases. The standard test for each TPCC also gives a standard indicator (TpmC), which implies how many new order transactions can be processed per minute when the system performs above standard business operations. Therefore TpmC can be used as a standard score to measure server performance. Table 1 shows the virtual machine performance scores for different situations.

The experimental results demonstrate that MTS increases the overall performance of MT by roughly 20%. Performance degradation is 7% lower compared against baselines that only run long-running application tasks (OT).

6. Conclusions

This paper proposed *Shaready*, a resource-isolated workload co-location system based on Openstack and Hadoop. The system can be easily deployed onto multi-platform with resources dynamically configured. We also designed a resource forest view to expose all system resources. By deploying the system, the overall utilization can be improved, and the quality of long-running services in virtual machines is guaranteed. The system has not been perfected. In the future, we are planning to reinforce the resource isolation of cache and network bandwidth. We are also experimenting based on a larger scale testbed to further validate the deployment effectiveness and efficiency.

References

[1] Aws news blog. [Online]. Available: https://aws.amazon.com/cn/blogs/aws/cloud-computing-server-utilization-the-environment/

[2] H. Liu, "A measurement study of server utilization in public clouds," in *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*. IEEE, 2011, pp. 435–442.

[3] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Queue*, vol. 14, no. 1, p. 10, 2016.

[4] M. Schwarzkopf, A. Konwinski *et al.*, "Omega: flexible, scalable schedulers for large compute clusters," in *ACM EuroSys*, 2013.

[5] M. Uddin, A. Shah, R. Alsaqour, J. Memon, and M. J. Saqour RAHASRAHA, "Measuring efficiency of tier level data centers to implement green energy efficient data centers," *Middle-East Journal of Scientific Research*, vol. 15, no. 2, pp. 200–207, 2013.

[6] H. Wang, P. Shi, and Y. Zhang, "Jointcloud: A cross-cloud cooperation architecture for integrated internet service customization," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 1846–1855.

[7] R. Yang, Z. Wen, D. McKee, T. Lin, J. Xu, and P. Garraghan, "Fog orchestration and simulation for iot services," 2018.

[8] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center." in *USENIX NSDI*, vol. 11, no. 2011, 2011, pp. 22–22.

[9] V. K. Vavilapalli, A. C. Murthy, C. Douglas *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *ACM SoCC*, 2013.

[10] Z. Zhang, C. Li, Y. Tao, R. Yang, H. Tang, and J. Xu, "Fuxi: a fault-tolerant resource management and job scheduling system at internet scale," *VLDB Endowment*, vol. 7, no. 13, pp. 1393–1404, 2014.

[11] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: distributed, low latency scheduling," in *ACM SOSP*, 2013.

[12] X. Sun, C. Hu, R. Yang, P. Garraghan, and C. Li, "Rose: Cluster scheduling through efficient resource overselling," in *ACM SOSP poster*, 2017.

[13] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *ACM EuroSys*, 2015, p. 18.

[14] A. Goder, A. Spiridonov, and Y. Wang, "Bistro: Scheduling data-parallel jobs against live production systems." in *USENIX Annual Technical Conference*, 2015, pp. 459–471.