A Modular Approach to Design an Experimental Framework for Resource Management Research

Lucia Pons (lupones@disca.upv.es)
Universitat Politècnica de València
Salvador Petit
Universitat Politècnica de València
Julio Pons
Universitat Politècnica de València
María E. Gómez
Universitat Politècnica de València
Julio Sahuquillo
Universitat Politècnica de València

Research Article

Keywords: Cloud computing, high-performance computing, resource management, virtualization, experimental framework

Posted Date: October 13th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-3400308/v1

Additional Declarations: No competing interests reported.

A Modular Approach to Design an Experimental Framework for Resource Management Research

Lucia Pons[1*], Salvador Petit[1], Julio Pons[1], María E. Gómez[1], Julio Sahuquillo[1]

[1]Universitat Politècnica de València, Spain.

*Corresponding author(s). E-mail(s): lupones@disca.upv.es;
Contributing authors: spetit@disca.upv.es; jpons@disca.upv.es;
megomez@disca.upv.es; jsahuqui@disca.upv.es;

Abstract

Research on resource management focuses on optimizing system performance and energy efficiency by distributing shared resources like processor cores, caches, and main memory among competing applications. This research spans a wide range of applications, including those from high-performance computing, machine learning, and mobile computing. Existing research frameworks often simplify research by concentrating on specific characteristics, such as the architecture of the computing nodes, resource monitoring, and representative workloads. For instance, this is typically the case with cloud systems, which introduce additional complexity regarding hardware and software requirements. To avoid this complexity during research, experimental frameworks are being developed. Nevertheless, proposed frameworks often fail regarding the types of nodes included, virtualization support, and management of critical shared resources.

This paper presents Stratus, an experimental framework that overcomes these limitations. Stratus includes different types of nodes, a comprehensive virtualization stack, and the ability to partition the major shared resources of the system. Even though Stratus was originally conceived to perform cloud research, its modular design allows Stratus to be extended, broadening its research use on different computing domains and platforms, matching the complexity of modern cloud environments.

Keywords: Cloud computing, high-performance computing, resource management, virtualization, experimental framework

1 Introduction

Resource management is an active research domain in both academic and industrial contexts. This area of research concentrates on enhancing system performance and reducing energy consumption by distributing shared resources, including processor cores, cache, main memory, and more. These goals are pursued across a broad spectrum of applications, encompassing high-performance computing (HPC), machine learning, latency-sensitive tasks, mobile computing, and various others.

With the aim of simplifying research efforts and improving flexibility, existing research frameworks often incorporate just a subset of characteristics found in real deployed systems. These characteristics typically involve node types, resource monitoring capabilities, and supported workloads. In fact, a predominant focus in the literature is on single multicore processor server platforms, emphasizing processor-centric workloads.

In contemporary computing scenarios, applications demanding substantial computational and storage resources, such as HPC and machine learning applications, are typically deployed in cloud environments [1–3]. Cloud platforms include a diverse array of computing nodes, each composed of a set of shared resources, such as cores, main memory, and storage. Virtual machines (VMs) [4] are employed to provide isolation and privacy for tenant applications hosted in public cloud environments, further complicating these systems. This complexity extends beyond hardware considerations such as software stack requirements related to virtualization, resource efficiency, Service Level Agreement (SLA) [5] compliance, and multi-tenancy support.

In this context, to conduct resource management research, many public cloud organizations develop their own controlled experimental frameworks as they cannot control user workloads. These frameworks offer the advantage of reduced complexity and enhanced flexibility, as the experimental platform and the workload can be tightly controlled. For instance, they enable the assessment of SLA compliance when applying resource management policies, facilitating obtaining conclusions before implementing such policies on real production systems.

The development of a representative experimental framework is challenging since such a platform must provide three key attributes: i) it should include the main types of hardware nodes (server, client, and storage), ii) it should offer VM-based isolation for tenant applications, and iii) it should support the partitioning of major shared resources, including main memory, last level cache (LLC), network components, etc. However, most frameworks in existing research works present shortcomings such as: i) being composed of a single machine [6–10], ii) lack of virtualization support [6–8], and iii) neglecting the management of important components such as the network infrastructure [6–8] and the remote storage [7, 8, 11–13].

This paper introduces Stratus, an experimental framework that overcomes all three aforementioned shortcomings and currently serves as a valuable resource for conducting resource management research within realistic environments, such as those found in cloud computing scenarios [14, 15]. Stratus is based on a hardware infrastructure implementing three main node types: a server node hosting tenant applications, a client node generating requests to the server, and a storage node providing remote storage capabilities to the server. Regarding the system software, Stratus features a

full software stack implementation, including technologies such as KVM-QEMU and Libvirt, to ensure isolation for tenant applications through VMs. Furthermore, the selection of the hardware technologies managed by Stratus enables the partitioning of critical shared resources, including the LLC, the memory bandwidth, and so on.

A key component of Stratus is its manager, which incorporates three main functionalities: management and control of the execution of VMs and running applications, monitoring hardware performance counters and system resource utilization, and partitioning the shared system resources through the available technologies in the nodes. Moreover, Stratus supports the execution of several workloads, including client-server workloads (e.g., TailBench [16], CloudSuite [17]), as well as best-effort or batch workloads (e.g., stressor microbenchmarks [18, 19] or SPEC CPU workloads [20]).

This paper extends the previous work [21] in the following ways. It offers rules for extending Stratus (Section 3) to different computing platforms and details its applicability across diverse computing domains, such as HPC and cloud computing. It presents new experiments (Section 5) assessing Stratus' performance on various computing platforms (ARM and Intel), including the results of workloads combining tail latency applications with background jobs. Finally, the paper discusses the potential research topics (Section 6) that will be opened by future Stratus platform to match complex cloud system deployments and the required software extensions.

2 Related Work

HPC experimental frameworks include only a small subset of the features present in cloud frameworks, including the type of nodes, resource monitoring capabilities, and the supported workload. In particular, most HPC works focus on a single multicore processor server platform, and the stress is mainly put on the processor side; thus, workloads stressing this system component are employed. This means that a relatively simple *ad-hoc* framework is used and not made publicly available. On the contrary, the high complexity of cloud systems has prompted researchers to create new testbeds to conduct their research. Thus, this section focuses on cloud system experimental platforms.

At a large scale, testbeds such as Grid'500 [22], Cloudlab [23], and Chamaleon [24] have been built to allow researchers to carry out experiments on distributed systems deployed in multiple sites spread geographically. Under these testbeds, users request the desired resources for a limited amount of time using a reservation system and then configure such resources for their use (e.g., deploy a custom software stack).

This paper focuses on built-in experimental testbeds for small-scale research that users deploy in their research facilities without relying on external systems. Table 1 summarizes, for a representative subset of testbeds recently proposed, how they fulfill four main features: i) if they support virtualization with VMs, ii) the supported workloads, iii) the type of nodes the platform includes, and iv) the resources that can be monitored or managed. For comparison purposes, the bottom row of the table includes our experimental platform, Stratus. The table shows that, to the best of our knowledge, no existing work proposes a controlled experimental platform that includes all the features in Stratus.

Table 1: Summary of the experimental infrastructure used in resource-oriented works.

Paper	Year	VMs	Workload	Type of Node			Resource Monitoring/Partitioning				
				Server	Client	Storage	CPU	LLC	Mem.BW	Disk	Net.
ServerMore [6]	2021	✓	LC	✓	×	✓	✓	✓	✓	×	×
Skynet [25]	2021	×, containers	LC	✓	✓	✓	✓	×	×	✓	✓
Alita [7]	2020	✓	LC, TO, micro	✓	×	×	✓	✓	✓	×	×
CLITE [8]	2020	×	LC, TO	✓	×	×	✓	✓	✓	✓	×
PARTIES [11]	2019	×, containers	LC	✓	✓	×	✓	✓	✓	✓	✓
Scavenger [12]	2018	✓	LC, TO, micro	✓	✓	×	✓	✓	×	×	✓
Vertical Elasticity [13]	2018	×, containers	LC, TO	✓	✓	×	✓	✓	×	✓	✓
Stratus [21]	2023	✓	LC, TO, micro	✓	✓	✓	✓	✓	✓	✓	✓

Stratus deploys the three types of nodes, uses VMs to allocate tenant applications, and provides monitoring and partitioning capabilities of the main system resources. As it can be seen in the table, only the testbed used to evaluate Skynet [25] includes the three types of nodes that Stratus deploys. All the approaches support the typical latency-critical (LC) workloads that run in cloud systems (e.g., Tailbench [16]) and follow the client-server model. Some approaches [6] run server and clients on the same machine or use single-node experimental platforms [7, 8], obviating the network interference.

Regarding virtualization, only three of the listed works use VMs to contain the tenant applications. Two of these approaches [6, 12] also use Linux KVM to deploy VMs. Some approaches use containers, which allow better performance at the expense of worse isolation. Software isolation tools (e.g., cgroups) indeed enable resource isolation in containers. However, containers are forced to use the same kernel as the host; thus, isolation is not possible at the kernel level.

Concerning the management of the shared resources, only PARTIES [11] considers all the shared resources similar to Stratus, but notice that this platform lacks a storage node and uses containers instead of VMs. From the listed resources, the CPU (which embraces hardware performance counters, CPU utilization monitoring, and allocation of CPU cores) is the only resource considered in all works. On the other hand, the network and disk are the shared resources least considered among the studied proposals. To induce interference, some works [7, 8, 12, 13] employ throughput-oriented (TO) workloads like PARSEC [26] and microbenchmarks (micro).

Apart from the platforms analyzed in Table 1, other research works make use of experimental platforms that focus on a single specific shared resource. Less Provisioning [27] and Twig [10] focus on CPU resource allocation by dynamically adjusting the CPU resources based on resource utilization and hardware performance counters, respectively. ReTail [9] also focuses on CPU resources but manages these resources by adjusting the CPU frequency. QWin [28] was devised to guarantee the tail latency SLO of distributed storage servers by partitioning cores of storage servers among tenant applications. Finally, LIBRA [29] proposes a dynamic memory bandwidth management framework.

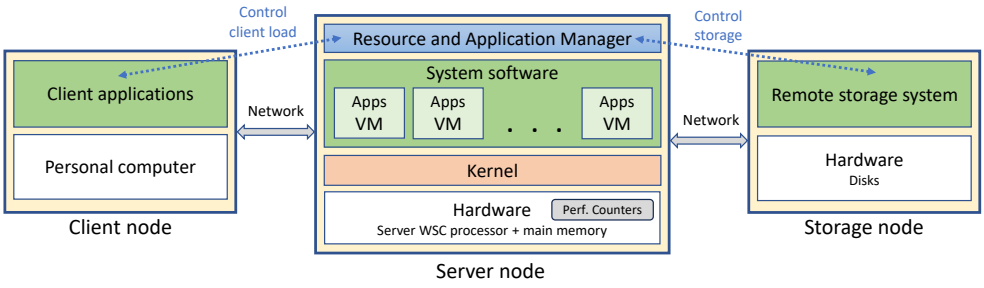


Fig. 1: High-level architecture of Stratus.

3 Implementing a Modular Experimental Infrastructure

Stratus was initially conceived to perform cloud research, but its modular design allows the use of Stratus to perform research beyond this scope. This section first presents the specific features of Stratus used to perform cloud research, and then it focuses on how the framework implemented in Stratus can be adapted to perform HPC research. Finally, it discusses how the modular approach allows for the framework to be deployed in machines with different architectures.

3.1 Cloud Specific Features

To carry out cloud research, four main design issues are considered:

1. The expected workloads to be run,
2. The main system shared resources to be monitored and partitioned among the tenant applications.
3. The target system (e.g., computing, storage) capabilities from an experimental perspective.
4. The software stack (e.g., virtualization).

The design of an experimental system is highly dependent on the expected workloads to be run and the major shared resources that must be considered. Regarding the expected workloads, cloud systems run many types of workloads. Among these, latency-critical applications are of great importance as they are increasingly common in data centers. These applications typically support online interactive services (e.g., web search) and must respond to the client requests within certain latency bounds to guarantee QoS (e.g., the 95^{th} or 99^{th} percentile latency) to provide a satisfactory user experience. Therefore, cloud systems must support the client-server (CS) architecture. Regarding the main system shared resources, cloud systems cannot obviate any system resource, but all of them must be considered together. This implies that client and server applications must be hosted in different nodes to take into consideration the network. In addition, these systems must provide remote storage capabilities.

Table 2: Node hardware specifications (processor and main memory).

Node	Processor Package		Main Memory
	Processor	**#Cores (#Threads)**	
Main	2x Intel Xeon Silver 4116	48 (96)	12 x DDR4-2666 16GB DIMMs
Client	Intel E5-2658A	12 (24)	1 x 32GB DIMM
Storage	Intel i5-9400F	6 (6)	2 x DDR4-2666 16GB DIMMs

These requirements have guided the design of Stratus. Figure 1 presents a high-level design of Stratus' experimental platform. On the hardware side, Stratus deploys server and client nodes to allow running CS workloads and a storage node to provide remote storage to the server applications. Notice that the specifications of the server and storage nodes should be representative of the nodes implemented in real cloud systems. Table 2 shows the specifications (processor package and main memory) for each of the nodes. The specifications of the server and storage nodes can be considered representative of the nodes implemented in real cloud systems. For example, Google Cloud CPU platforms [30], Amazon EC2 C6 and C5 instances [31] and Huawei Elastic Cloud servers [32] use Intel Xeon Scalable Processors including from tens to hundreds of gigabytes for main memory capacity.

An important feature of cloud systems is that they must support virtualization of the real hardware as tenant applications are hosted by virtual machines (VMs) [4] in the public cloud. Therefore, on the software side, Stratus includes the main components of current cloud software stacks [33]. Three main levels can be distinguished: the hypervisor, the virtual resource manager, and the guest OS and applications. The hypervisor refers to the OS installed in the host physical machine (PM). A wide set of both open-source and proprietary hypervisors are currently being used in the industry. Examples of open-source hypervisors are Linux with KVM [34] and Xen [35]. The former is one of the current industry trends, and it is being used by Amazon [36] and Google [37]. The latter is also supported by Amazon. The virtualization manager is the software platform that manages and distributes the PM hardware resources among VMs. One example of a virtualization manager is Libvirt [38]. Some virtualizers, like QEMU [39], also support both KVM and Xen. Finally, the guest OS and tenant applications run in the different VMs. Guest OS and applications can be either proprietary or open source (e.g., a Linux server distribution executing several Internet services). In addition, a virtual switch is used to interconnect the VMs with the physical network interface cards (NICs) of the server node. The virtual switch is set up with Open vSwitch (OvS) [40]. Emulated NICs at the VMs (i.e., virtio [41] NICs) and each physical NIC (both ports) in the server node are accessed from the virtual switch through the Data Plane Development Kit (DPDK) [42]. DPDK enables the direct transfer of packets between virtio NICs and physical NICs, bypassing the host OS kernel network stack. This setup boosts network performance compared to the default packet forwarding mechanism implemented in the Linux kernel. To provide remote storage, Ceph [43] is used, commonly installed in cloud environments.

On the highest level, we find the resource and application manager, which is the central axis of Stratus. It is in charge of controlling the operation of all the nodes,

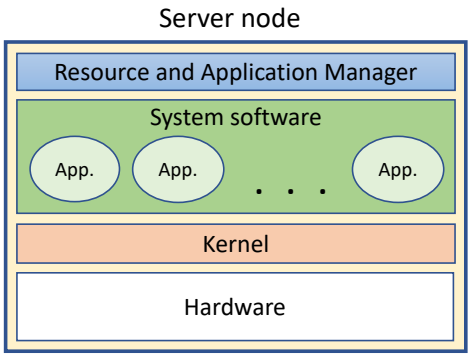


Fig. 2: High-level architecture of Stratus when tailored to HPC research.

which simplifies the administration. Firstly, it supports monitoring and partitioning the following main system shared resources: CPUs, last level cache (LLC) space, main memory, network, and disk bandwidth. In addition, it allows researchers to automate their daily work –launching experiments and taking results – by i) launching the workload: VM instances and guest applications in the client node, ii) measuring performance counters, iii) assigning partitions of shared resources to each VM instance, and iv) scheduling (stop and relaunch) VM instances and controlling physical machine (PM) cores. A detailed description of the specific tasks carried out is provided in Section 4.

3.2 Using Stratus for HPC

Notice that the modular design of Stratus allows the experimental environment to be greatly simplified for HPC evaluation since, in these studies, only the server machine is typically evaluated, and there is no need to include virtualization capabilities. Figure 2 shows the design of Stratus adapted for this purpose. Below, the main modifications performed over Stratus' global architecture are discussed:

- **Hardware.** HPC systems focus on running compute- and memory-intensive workloads in one or multiple server nodes. Typical HPC experimental platforms used for resource management research comprise a single server machine based on a multicore processor with a high core count. In addition, some research works started recently to use multi-socket platforms.
- **Software.** HPC tasks that run on privately owned experimental platforms are executed natively on the machines without using virtualization. Thus, the complex software stack deployed in cloud systems is not required.
- **Resource and application manager.** Not using VMs to launch applications greatly simplifies the execution of experiments. However, HPC workloads may require additional features concerning the monitoring and managing of the processes or threads created at runtime by applications.

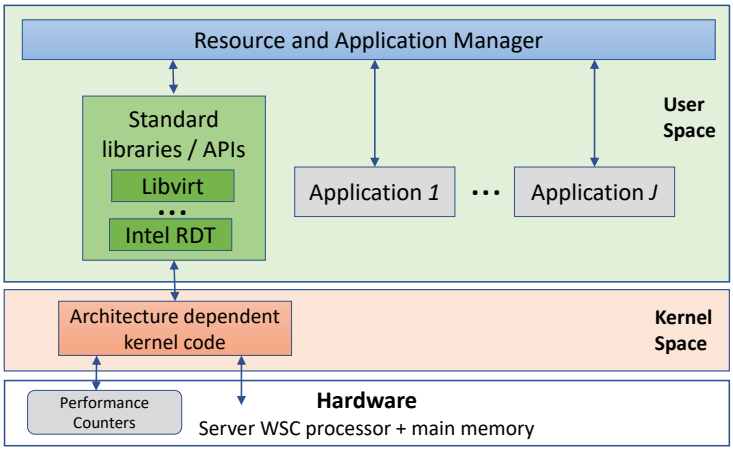


Fig. 3: Stratus resource and application manager software architecture.

3.3 Allowing Architecture Interoperability

The emergence of new architectures in the context of the server market has forced many performance evaluation studies to be conducted in order to assess performance and energy efficiency [44, 45], as well as architecture-specific features like SIMD instructions [46, 47]. For this purpose, it is important to have an experimental framework that can provide a common and homogeneous methodology to perform the experiments.

Figure 3 shows the software architecture of Stratus. Stratus resource and application manager is tailored to different libraries or APIs (e.g., libvirt) that access the OS kernel. Some of this code is architecture-dependent as it has to access specific hardware. This is the case, for instance, of performance counters. Current processors deploy dedicated performance counters to account for the number of processor cycles and instructions committed. Additional general-purpose counters can be programmed to monitor architecture-specific events like misses at a given cache level. Since the performance counter architecture varies among processor vendors, a framework developed for an Intel processor will not run on an Arm processor. In fact, consecutive generations or micro-architectures of the same processor manufacturer may have different or even incompatible performance counter architectures.

Any software that accesses processor-specific registers or architecture-dependent code must be tuned to the underlying hardware. In this regard, Stratus has been so far implemented in Intel and Arm processors. Stratus resource and application manager is linked to the Linux kernel source files in order to access specific hardware tools. This is the case for performance counters, which are accessed using Perf [48]. In the case of Intel processors, the manager uses Intel Resource Director Technology (RDT) [49] to perform last level cache (LLC) and memory bandwidth monitoring and partitioning. This allows Stratus to monitor and partition system resources without using any external programs.

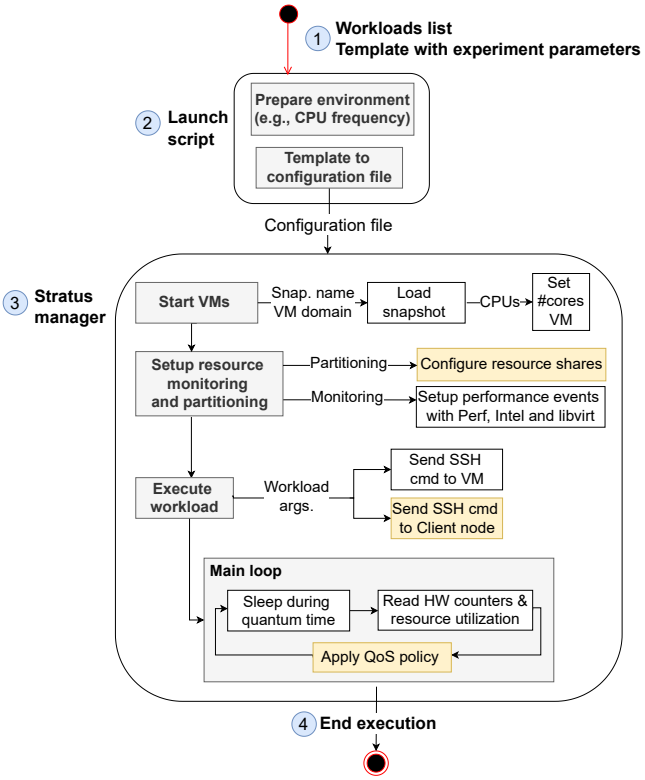


Fig. 4: Workflow followed by Stratus' resource and application manager to launch experiments with VMs. Yellow boxes represent actions that are optional.

4 Stratus Resource and Application Manager

As mentioned in the previous section, Stratus resource and application manager is a key element of the experimental framework. It provides a friendly interface to perform experiments and access to system resources.

This section first illustrates how Stratus performs experiments by describing the steps followed in an experiment where one or multiple VMs are launched. Notice that these steps would be similar to those performed if applications are launched natively in the machine (i.e., those steps concerning VM management would be skipped). Then, we describe how Stratus accesses each of the system's shared resources regarding both monitoring and partitioning.

4.1 Execution of Experiments

To illustrate how Stratus' resource and application manager performs experiments, Figure 4 shows a block diagram of the workflow when carrying out experiments using Stratus's manager. The diagram illustrates the main steps performed when launching one or more VMs together with the applications to be run on them (*VM-application*

pairs). Next, each of the steps is discussed in detail.

1) Define experiment workload and parameters. As a prior step, the workload (i.e., VMs and applications to be run on them) and experimental conditions must be defined. To ease this task, Stratus makes use of MAKO templates [50], which provides a simple and intuitive language to specify the parameters of the experiments: VMs and applications to be executed (domain name, workload, number of CPUs, etc.), VCPUs core pinning, performance events to be monitored, length of the quantum, etc. The template can also specify if VMs are only allowed to use a partition of a shared resource (LLC, memory bandwidth, network bandwidth, or disk bandwidth).

2) Execute *Launch* script to start the manager. To start running an experiment, the user executes the *launch* script. First, the script prepares the execution environment. For instance, fixing the processor frequency to avoid variability among experiments. Additionally, the server clocks of both the server and client machines are synchronized to ensure server- and client-collected metrics are aligned, using the Network Time Protocol (NTP) [51] with a known NTP time server (e.g. europe.pool.ntp.org). When the environment is ready, the configuration file is generated with the workloads to execute and all the experiment parameters from the MAKO template. Then, the manager starts to run.

3) Prepare VMs for execution. The first step the manager performs is setting up and starting the VMs. To reduce the start-up overhead, the manager uses the *snapshots* feature of Libvirt. A snapshot is a copy of the state of a VM, including the disk and main memory contents. This feature preserves a VM's actual state and data at a given time. Therefore, this state can be reverted at any moment. For each VM, we have taken a snapshot that has already performed the OS boot process and is ready to receive the command to launch the target benchmark. Once the VMs are started and the snapshots are loaded, the number of CPUs of each VM (i.e., VCPUs) can be modified in case a multi-threaded application is going to be executed, and more than one CPU is required.

4) Setup resource monitoring and partitioning. With QEMU, each VCPU is associated with a processor ID (PID) in the host OS. These PIDs are required to monitor hardware performance counters with Perf individually for each thread (i.e., VCPU) of the VM. Similarly, LLC and memory bandwidth monitoring is performed on a PID basis. The remaining resources, network, and disk bandwidth are monitored per VM. The manager also allows partitioning of the main system shared resources and assigns each VM a share of a given resource. Therefore, if specified in the template, a resource share is allocated to the VM.

5) Start running applications in the VMs. When the VMs are operative and ready to start executing the applications, an SSH command is sent to each VM to start the execution of each workload. Stratus' manager is adapted to support the execution of client-server workloads (e.g., TailBench benchmark suite) and best-effort or batch workloads (e.g., stressor microbenchmarks or SPEC CPU benchmarks). In the case of

a client-server workload, an SSH command is sent to the client node to start running the clients, which send requests to the server (already running).

6) Perform actions in each quantum. Once the execution starts, the manager executes the *main loop* (see Figure 4) for the rest of the execution time. In each iteration, the manager is suspended for a given quantum length (established in the template). Then, data is collected from different sources (e.g., hardware performance counters, Linux file system, Intel library, or Libvirt) to monitor the main system resources (CPU usage, LLC occupancy, main memory, network, and disk bandwidth). Additionally, the manager is adapted to allow implementing and applying QoS policies. For instance, policies that manage resource sharing among VMs [7, 11, 12], predict interference among VMs [8, 9, 13, 15, 52] or schedule VMs [53].

7) Execution end. The main loop ends when the manager detects that all the VMs have finished running their applications, the moment at which it shuts down the running VMs.

All the data collected from the hardware performance counters and system resources are stored in CSV files, ready to be processed. Additionally, for characterization and debugging purposes, statistics and data are also collected inside the VMs. For instance, in Tailbench workloads, the clients report results such as latency per query, queries per interval, tail latency, etc.

4.2 Monitoring & Partitioning Main Shared Resources

Tenant applications compete for shared resources in cloud systems. This means that the performance of a given tenant application (or VM) will depend on the co-running applications. In other words, on the share of the resource that is able to use. Consequently, it is worth studying to which extent the performance of a given application is affected by varying the amount of share allocated to the application.

To perform this kind of experiment and help researchers in their decision-making, we need to define the tools to be implemented. In the last few years, server processors have been provided with advanced technologies that allow monitoring and partitioning of the major system resources.

Below, we explain how monitoring and partitioning of each shared resource is implemented in Stratus without relying on any external tool.

CPU Utilization. CPU utilization accounts for the percentage of time a CPU is active. It is a crucial metric in cloud environments since CPU utilization has been proven to be low (less than 20%) most of the time [54, 55], and thus, many resource provisioning strategies [12, 27, 56] seek to improve the CPU usage. To obtain the utilization of each CPU, we use the data collected from the file `/proc/stat`, which reports statistics about the kernel activity aggregated since the system first booted. To pin the VMs' VCPUs to logical cores of the physical machine, Stratus uses Libvirt's API [38].

Last Level Cache (LLC). Due to the high latency to access to main memory upon LLC misses, the LLC is one of the *critical* shared resources in current multi-core processors. Recently, some processor manufacturers like Intel have developed technologies that allow monitoring and partitioning of the LLC. In Intel processors, these technologies are known as Cache Monitoring Technology (CMT) and Cache Allocation Technology (CAT) [57]. Partitioning is performed using Classes of Service (CLOS), which can be defined either as groups of applications (PIDs) or as groups of logical cores to which a partition of the LLC is assigned. The LLC is partitioned in a *per-way* basis; that is, a cache way acts as the granularity unit of CLOS.

Memory Bandwidth. Memory bandwidth can considerably impact the performance or responsiveness of applications. For instance, in a server system with different VMs accessing the main memory, the inter-VM interference can significantly grow and make the most memory-sensitive VMs perform below an acceptable level, compromising the QoS. Recent Intel Xeon Scalable processors introduce Memory Bandwidth Allocation (MBA) [58], which allows to distribute memory bandwidth between the running applications. More precisely, it allows controlling the memory bandwidth between the L2 and the L3 (i.e., LLC) caches. Similarly to CAT, MBA works using CLOS. That is, MBA bandwidth limits apply only to CLOS, to which the user can assign tasks (PIDs) or cores. However, MBA **works on a *per-core* basis**. If the individual memory bandwidths of two applications running on the same core are limited with different values, the *maximum* limitation is the one that will apply to that core.

Disk Bandwidth. Many workloads operate on big data files or databases that cannot be completely loaded to main memory. Consequently, these workloads must constantly rely on the I/O system to access the disk and load/store the required data. Monitoring and partitioning this subsystem is, therefore of high interest. I/O access to the disks can be monitored using the virsh tool or Libvirt's API. Both mechanisms offer the same functionality and allow monitoring the number of read, write, and flush operations, the number of bytes read and written, as well as the total duration of the read, write, and flush operations.

Network Bandwidth. VMs running on the same physical machine share network resources whose bandwidth and latency play an important role in the QoS of tenant applications. Consequently, network resources should be monitored and partitioned to minimize inter-VM interference. The number of network packets or bytes that go through a network interface can be monitored with Libvirt's API.

5 Case Studies

A modular approach to designing an experimental framework allows the addition of software layers (e.g., virtualization) and deployment in multiple machines with different architectures. This section presents two case studies illustrating how the proposed experimental framework can be used for performance evaluation studies.

Table 3: Shared resources utilization (average values across the entire execution) of the microbenchmark *stress_ng* when running in isolation.

Scenario	CPU (%)	Mem. (GB/s)	LLC (MB)	Disk (MB/s)	Net. (MB/s)
Low Stress	10.7	0.01	4.1	0.1	0.0
Medium Stress	44.7	3.6	13.4	0.1	0.0
High Stress	82.2	5.5	14.7	0.1	0.0

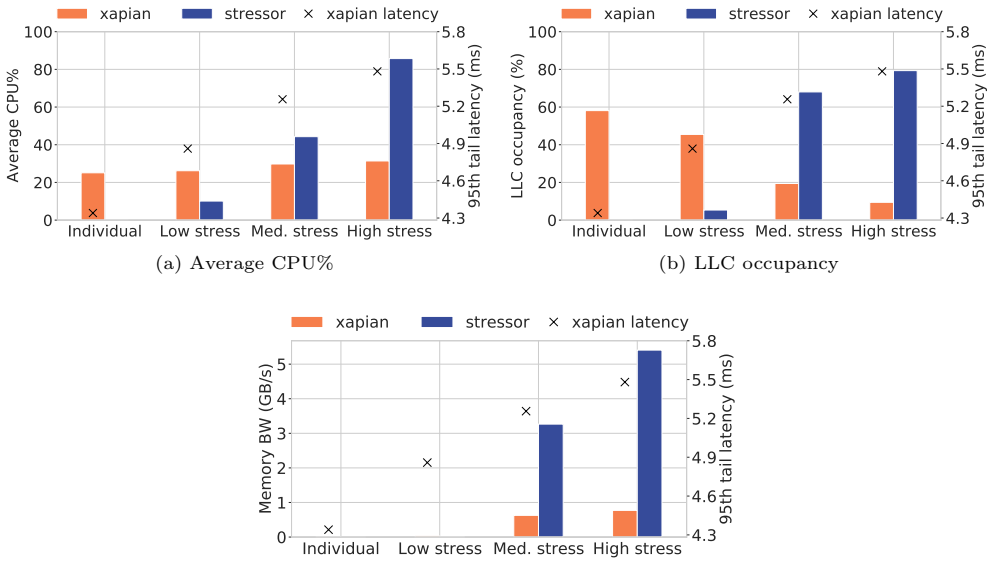
5.1 Co-locating Latency-Critical and Background Jobs

Current data centers seek to maximize resource utilization by co-locating multiple applications in the same server node. Among those applications, latency-critical are very common as they support interactive services (e.g., web search, video streaming, image recognition) used in many application domains. Due to the strict QoS requirements (quantified with the 95^{th} or 99^{th} percentile latency), a few latency-critical applications can be placed together on the same server. To maximize resource utilization, a common approach is to co-locate latency-critical jobs with background jobs with less strict QoS requirements [8, 11, 12].

However, when running multiple applications, interference appears at the shared resources [15, 54, 59], which can lead to violating a QoS agreement. A common approach to quantify how the interference at the shared resources affects the performance of applications is employing microbenchmarks [15, 60–62]. These synthetic benchmarks are specially designed to stress specific system resources (e.g., perform many accesses to the last level cache).

This case study presents some experimental results obtained with the `xapian` application from the TailBench suite [16] and the `stress-ng` [18] microbenchmark. Both `xapian` and `stress-ng` are executed each in a VM with 2 and 8 virtual CPUs, respectively, in the Stratus platform. Table 3 shows the three stressing scenarios tested and the utilization of the main shared system resources in each scenario of `stress-ng` when running alone. The stressing scenarios have an increasing level of CPU utilization (from Low to High) due to the pressure introduced in the last level cache (LLC) and main memory.

Figure 5 shows the resource consumption of `xapian` and `stress-ng` (average CPU utilization, LLC occupancy, and memory bandwidth) in each of the evaluated stressing scenarios. For comparison purposes, the results obtained when running `xapian` alone (*Individual*) have been included. Finally, the right Y-axis shows the 95^{th} percentile latency of `xapian` in each scenario. It can be observed that as the microbenchmark consumes more LLC space (Figure 5b) and memory bandwidth (Figure 5c), the average CPU utilization (Figure 5a) grows significantly. This causes the tail latency of `xapian` to grow as more interference is introduced. The LLC is the resource that affects most the performance of `xapian`. This application has very small memory bandwidth consumption (less than 100 MB/s) as its working set fits in the LLC. However, the bandwidth usage increases as `stress-ng` evicts `xapian`'s data. Notice that even in the low-stress scenario, the 95^{th} percentile latency of `xapian` grows by nearly 12%, which increases over 20% in the medium- and high-stress scenarios.

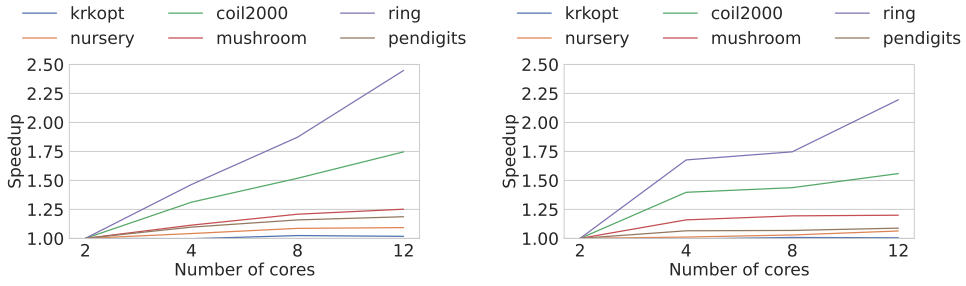


(a) Average CPU%

(b) LLC occupancy

(c) Memory bandwidth

Fig. 5: CPU and memory consumption of `xapian` and the stressor microbenchmark (`stress-ng`). Performance of `xapian` (tail latency) is also plotted.



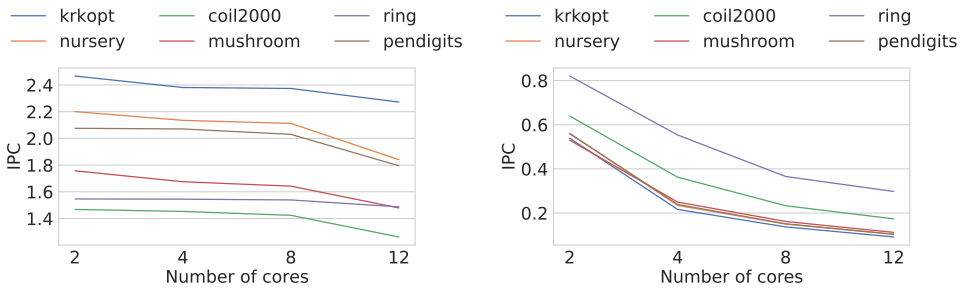
(a) Intel

(b) Arm

Fig. 6: Speedup obtained with Decision Tree classifier ML method and a set of datasets (one per line) by increasing the number of cores with respect to 2 cores.

5.2 Intel vs. Arm Performance Comparison

Traditional x86 architectures dominate the server CPU market share, mostly Intel processors. However, alternative processor architectures such as Arm are also recently driving into the server market. For this reason, recent studies [63–65] have focused on comparing the performance of Intel and Arm-based CPUs. To this end, using the



(a) Intel (b) Arm

Fig. 7: Average IPC per core obtained with the Decision Tree classifier ML method and a set of datasets (one per line) with an increasing number of cores.

same interface to perform the experiments in each machine results in a more fair comparison, easing the researcher's work.

For illustrative purposes, this case study presents some experimental results on how the application performance is improved with an increasing number of cores when executed in two server processors: an Intel processor with a Xeon Silver 4116 CPU with 12 Hyper-Threading (2 threads per core) cores, and an Arm Cavium ThunderX2 processor (CPU CN9975) with 28 cores that support up to 4 threads per core. To ease the comparison, both processors are configured to run at the same frequency, 2.10 GHz. In each processor, we have evaluated a set of datasets from the Penn Machine Learning Benchmark (PMLB) suite [66] with the Decision Tree classifier machine learning (ML) method [67].

Figure 6 shows the results obtained when running each dataset in single-threaded mode (i.e., one thread running in each core) with an increasing number of cores, from 2 to 12 (maximum number of cores in the Intel processor) both in the Intel and Arm processors. Regardless of the underlying machine, datasets present similar trends. We found that the difference in speedup among datasets is directly related to the amount of floating-point (FP) operations that are required to execute. That is, the higher the number of FP operations, the more the dataset benefits from running with additional cores. For instance, in the Arm machine, those applications (*krkopt*, *nursery* and *pendigits*) that barely change their speedup (i.e., always close to one), the fraction of FP operations executed is less than 5%. In contrast, those datasets that show higher speedup, *ring* and *coil2000*, show a significant fraction of FP operations, around 13% and 7.5%, respectively.

Overall, applications achieve higher scalability when running on the Intel machine, even though applications behave similarly in both machines. To compare the performance, Figure 7 shows the average instructions per cycle (IPC) obtained for each application in each core configuration. Important performance differences can be observed between the IPCs obtained, especially among those applications that show very low speedup with an increasing number of cores. For instance, when running with two cores, *krkopt* achieves an average IPC of 2.45 on the Intel machine, while

this value drops to 0.5 in the Arm machine. Notice that the Arm machine has fewer issue ports than the Intel machine (6 compared to 8), which can hinder performance significantly. In addition, the L2 cache is four times smaller (256 KB compared to 1 MB). Unfortunately, unlike Intel, Arm microarchitecture does not provide performance events to measure statistics about the ports' utilization or cache misses, which makes the performance analysis difficult.

6 Discussion and Future Work

Stratus allows experimental research in cloud systems considering all three components: the client, the server, and the storage. This platform is aimed at studying resource interference that takes place among applications running on the same server (thus, only one server node is deployed). If there were to be multiple independent servers, an instance of the resource and application manager should run on each server (see Figure 1).

As for future work, we plan to extend Stratus to support research in complex platforms consisting of multiple *cooperating* servers and several storage nodes. Current clouds replicate many system components to deal with availability, reliability, and performance (i.e., meeting users' QoS) [68]. Availability refers to increasing the capability of shared resources to support higher user requests. Reliability refers to supporting fault tolerance. That is, the system can continue to support the same services, although some hardware components (e.g., a server or a storage node) fail. Variability appears as a function of the server's load that replies to the user's request as well as the load of the storage node (disk). Also, some contention can appear in the interconnection network as servers compete for storage nodes. To address these concerns, cloud systems replicate applications in distinct servers and data in separate storage disks.

We plan to upgrade the current infrastructure by adding more servers and storage nodes, as well as a *master* node to schedule requests as depicted in Figure 8. Notice that the original Stratus version needs to be slightly modified at the server nodes must communicate with the master node (instead of directly with the client's requests generator) and with multiple disk nodes. The master machine receives client requests and schedules them to the servers according to a given policy. This hardware-software infrastructure will open the research scope in many directions, for instance, scheduling policies at the master node or latency reduction techniques [69–72].

7 Conclusions

Research on resource management spans a wide range of applications, such as high-performance computing, machine learning, and mobile computing. Research and tests should be performed in controlled experimental platforms. Nevertheless, proposed frameworks often fail regarding the types of nodes included, virtualization support, and management of critical shared resources.

This paper has presented Stratus, a modular experimental platform for research on resource management that overcomes the mentioned limitations. Stratus includes diverse types of computing nodes, a comprehensive virtualization stack, and the ability

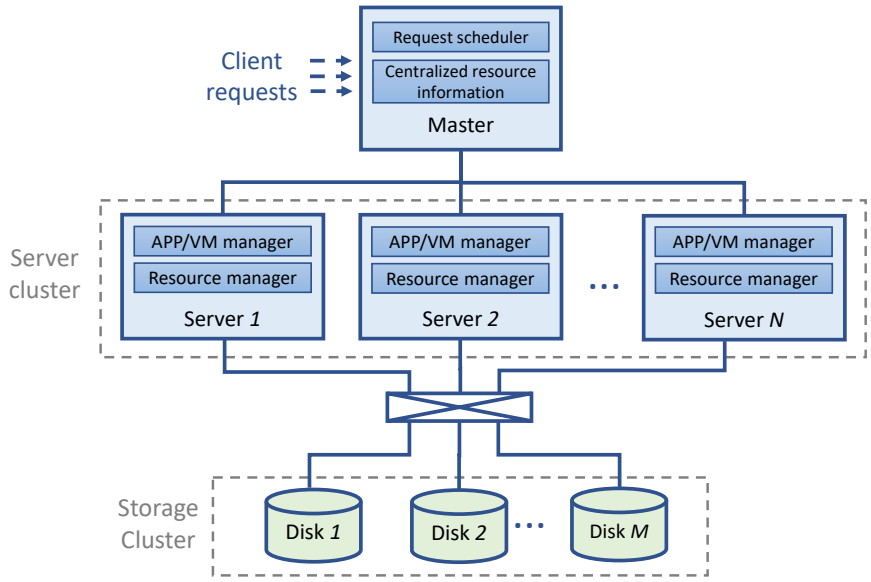


Fig. 8: Design of a possible future expansion of Stratus considering more nodes.

to partition major shared resources among the tenant's applications. Stratus implements a software manager that integrates three main functionalities: VM management, monitoring of performance counters and system resource utilization, and partitioning of main shared resources. This work presented case studies illustrating how Stratus can be used for research in different computing domains and platforms.

As for future work, we plan to extend Stratus to support research in systems consisting of many servers and storage nodes. Finally, some research topics that the future version will support have been drawn.

Acknowledgments. Not applicable.

Funding. This work has been supported by the Spanish Ministerio de Universidades under the grant FPU18/01948 and by the Spanish Ministerio de Ciencia e Innovación and European ERDF under grants PID2021-123627OB-C51 and TED2021-130233B-C32, and by Generalitat Valenciana under Grant AICO/2021/266.

Authors contributions. All authors contributed equally to the work's conception and design. Platform design and software development was performed by LP, SP, and JP. Data collection and representation were performed by LP. JS was in charge of the project administration, supervision, and data validation. JS, MEG, and SP were responsible for the funding acquisition and administration. All the authors contributed in the writing and reviewing of the manuscript.

Data availability. Tailbench applications are publicly available at https://github.com/supreethkurpad/Tailbench. Stress_ng microbenchmark can be downloaded using

common package manager tools if Linux distributions (e.g., Ubuntu's APT). Datasets from PMLB are publicly available at https://github.com/EpistasisLab/pmlb, as well as the code for the Decision Tree classifier ML method https://github.com/rhiever/sklearn-benchmarks.

Declarations

Conflict of interest. The authors declare that they have no competing interests

Ethical approval. Not applicable.

Consent to participate. Not applicable.

Consent for publication. Not applicable.

References

[1] Gupta, A., Milojicic, D.: Evaluation of hpc applications on cloud. In: 2011 Sixth Open Cirrus Summit, pp. 22–26 (2011). https://doi.org/10.1109/OCS.2011.10

[2] Netto, M.A.S., Calheiros, R.N., Rodrigues, E.R., Cunha, R.L.F., Buyya, R.: Hpc cloud for scientific and business applications: Taxonomy, vision, and research challenges. ACM Comput. Surv. **51**(1) (2018) https://doi.org/10.1145/3150224

[3] Hormozi, E., Hormozi, H., Akbari, M.K., Javan, M.S.: Using of machine learning into cloud environment (a survey): Managing and scheduling of resources in cloud systems. In: 2012 Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, pp. 363–368 (2012). https://doi.org/10.1109/3PGCIC.2012.69

[4] Sahoo, J., Mohapatra, S., Lath, R.: Virtualization: A survey on concepts, taxonomy and associated security issues. In: 2010 Second International Conference on Computer and Network Technology, pp. 222–226 (2010). https://doi.org/10.1109/ICCNT.2010.49

[5] Serrano, D., Bouchenak, S., Kouki, Y., de Oliveira Jr., F.A., Ledoux, T., Lejeune, J., Sopena, J., Arantes, L., Sens, P.: Sla guarantees for cloud services. Future Generation Computer Systems **54**, 233–246 (2016) https://doi.org/10.1016/j.future.2015.03.018

[6] Suresh, A., Gandhi, A.: Servermore: Opportunistic execution of serverless functions in the cloud. In: Proceedings of the ACM Symposium on Cloud Computing. SoCC '21, pp. 570–584 (2021). https://doi.org/10.1145/3472883.3486979

[7] Chen, Q., Xue, S., Zhao, S., Chen, S., Wu, Y., Xu, Y., Song, Z., Ma, T., Yang, Y., Guo, M.: Alita: Comprehensive performance isolation through bias resource management for public clouds. In: Proceedings of SC20: International Conference

for High Performance Computing, Networking, Storage and Analysis, pp. 1–13 (2020). https://doi.org/10.1109/SC41405.2020.00036

[8] Patel, T., Tiwari, D.: Clite: Efficient and qos-aware co-location of multiple latency-critical jobs for warehouse scale computers. In: Proceedings of 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 193–206 (2020). https://doi.org/10.1109/HPCA47549.2020.00025

[9] Chen, S., Jin, A., Delimitrou, C., Martínez, J.F.: Retail: Opting for learning simplicity to enable qos-aware power management in the cloud. In: 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA), pp. 155–168 (2022). https://doi.org/10.1109/HPCA53966.2022.00020

[10] Nishtala, R., Petrucci, V., Carpenter, P., Sjalander, M.: Twig: Multi-agent task management for colocated latency-critical cloud services. In: Proceedings of 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 167–179 (2020). https://doi.org/10.1109/HPCA47549.2020.00023

[11] Chen, S., Delimitrou, C., Martínez, J.F.: PARTIES: QoS-Aware Resource Partitioning for Multiple Interactive Services. In: Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 107–120 (2019). https://doi.org/10.1145/3297858.3304005

[12] Javadi, S.A., Suresh, A., Wajahat, M., Gandhi, A.: Scavenger: A black-box batch workload resource manager for improving utilization in cloud environments. In: Proceedings of the ACM Symposium on Cloud Computing (SoCC), pp. 272–285 (2019). https://doi.org/10.1145/3357223.3362734

[13] Shekhar, S., Abdel-Aziz, H., Bhattacharjee, A., Gokhale, A., Koutsoukos, X.: Performance interference-aware vertical elasticity for cloud-hosted latency-sensitive applications. In: Proceedings of the IEEE 11th International Conference on Cloud Computing (CLOUD), pp. 82–89 (2018). https://doi.org/10.1109/CLOUD.2018.00018

[14] Pons, L., Feliu, J., Puche, J., Huang, C., Petit, S., Pons, J., Gómez, M.E., Sahuquillo, J.: Effect of hyper-threading in latency-critical multithreaded cloud applications and utilization analysis of the major system resources. Future Generation Computer Systems **131**, 194–208 (2022) https://doi.org/10.1016/j.future.2022.01.025

[15] Pons, L., Feliu, J., Sahuquillo, J., Gómez, M.E., Petit, S., Pons, J., Huang, C.: Cloud white: Detecting and estimating qos degradation of latency-critical workloads in the public cloud. Future Generation Computer Systems **138**, 13–25 (2023) https://doi.org/10.1016/j.future.2022.08.012

[16] Kasture, H., Sanchez, D.: Tailbench: a benchmark suite and evaluation methodology for latency-critical applications. In: 2016 IEEE International Symposium on Workload Characterization (IISWC), pp. 1–10 (2016)

[17] Ferdman, M., Adileh, A., Kocberber, O., Volos, S., Alisafaee, M., Jevdjic, D., Kaynak, C., Popescu, A.D., Ailamaki, A., Falsafi, B.: Clearing the clouds: A study of emerging scale-out workloads on modern hardware. Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (2012)

[18] Canonical Ltd: Ubuntu manpage: stress-ng. Available at https://manpages.ubuntu.com/manpages/artful/man1/stress-ng.1.html. Accessed: 2022-11-20 (2020)

[19] ESnet, NLANR, DAST: iPerf tool for network bandwidth measurements. Available at https://iperf.fr/. Accessed: 2022-11-20 (2020)

[20] Why would a cloud computing company use the SPEC CPU2017 benchmark suite? Available at https://www.spec.org/cpu2017/publications/DO-case-study.html. Accessed: 2019-08-02 (2017)

[21] Pons, L., Petit, S., Pons, J., Gómez, M.E., Huang, C., Sahuquillo, J.: Stratus: A Hardware/Software Infrastructure for Controlled Cloud Research. In: Proceedings of PDP, pp. 299–306 (2023)

[22] Badia, S., Carpen-Amarie, A., Lèbre, A., Nussbaum, L.: Enabling large-scale testing of iaas cloud platforms on the grid'5000 testbed. In: Proceedings of the 2013 International Workshop on Testing the Cloud, pp. 7–12 (2013)

[23] Duplyakin, D., Ricci, R., Maricq, A., Wong, G., Duerig, J., Eide, E., Stoller, L., Hibler, M., Johnson, D., Webb, K., *et al.*: The design and operation of cloudlab. In: USENIX Annual Technical Conference, pp. 1–14 (2019)

[24] Keahey, K., Anderson, J., Zhen, Z., Riteau, P., Ruth, P., Stanzione, D., Cevik, M., Colleran, J., Gunawi, H.S., Hammock, C., *et al.*: Lessons learned from the chameleon testbed. In: Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference, pp. 219–233 (2020)

[25] Sfakianakis, Y., Marazakis, M., Bilas, A.: Skynet: Performance-driven resource management for dynamic workloads. In: 2021 IEEE 14th International Conference on Cloud Computing (CLOUD), pp. 527–539 (2021). https://doi.org/10.1109/CLOUD53861.2021.00069

[26] Bienia, C., Kumar, S., Singh, J.P., Li, K.: The parsec benchmark suite: Characterization and architectural implications. In: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, pp. 72–81 (2008)

[27] Cai, B., Li, K., Zhao, L., Zhang, R.: Less provisioning: A hybrid resource scaling engine for long-running services with tail latency guarantees. IEEE Transactions on Cloud Computing **10**(3), 1941–1957 (2022) https://doi.org/10.1109/TCC.2020.3016345

[28] Ma, L., Liu, Z., Xiong, J., Jiang, D.: Qwin: Core allocation for enforcing differentiated tail latency slos at shared storage backend. In: 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS), pp. 1098–1109 (2022). https://doi.org/10.1109/ICDCS54860.2022.00109

[29] Zhang, Y., Chen, J., Jiang, X., Liu, Q., Steiner, I.M., Herdrich, A.J., Shu, K., Das, R., Cui, L., Jiang, L.: Libra: Clearing the cloud through dynamic memory bandwidth management. In: 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), pp. 815–826 (2021). https://doi.org/10.1109/HPCA51647.2021.00073

[30] Google Cloud Compute Engine - CPU platforms [online]. Available at https://cloud.google.com/compute/docs/cpu-platforms. Accessed: 2022-11-14 (2022)

[31] Amazon's EC2 [online]. Available at https://aws.amazon.com/ec2/instance-types/?nc1=h_ls. Accessed: 2022-11-14 (2022)

[32] Huawei Elastic Cloud Server (ECS) [online]. Available at https://www.huaweicloud.com/intl/en-us/product/ecs.html. Accessed: 2022-11-14 (2022)

[33] Sefraoui, O., Aissaoui, M., Eleuldj, M.: Openstack: Toward an open-source solution for cloud computing. International Journal of Computer Applications **55**(3), 38–42 (2012)

[34] Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A.: kvm: the linux virtual machine monitor. In: Proceedings of the Linux Symposium, vol. 1, pp. 225–230 (2007). Dttawa, Dntorio, Canada

[35] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. ACM SIGOPS operating systems review **37**(5), 164–177 (2003)

[36] Amazon Web Services [online]. Available at https://aws.amazon.com/ec2/faqs/?nc1=h_ls. Accessed: 2022-11-28 (2022)

[37] Google Compute Engine FAQ [online]. Available at https://cloud.google.com/compute/docs/faq. Accessed: 2022-11-28 (2022)

[38] ibvirt: The virtualization API [online]. Available at https://libvirt.org. Accessed: 2022-11-28 (2022)

[39] QEMU [online]. Available at https://www.qemu.org. Accessed: 2022-11-28

(2022)

[40] Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., Casado, M.: The design and implementation of open vSwitch. In: 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), pp. 117–130. USENIX Association, Oakland, CA (2015). https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

[41] Russell, R.: virtio: towards a de-facto standard for virtual i/o devices. ACM SIGOPS Operating Systems Review **42**(5), 95–103 (2008)

[42] DPDK [online]. Available at https://www.dpdk.org/. Accessed: 2022-11-28 (2022)

[43] Weil, S., Brandt, S., Miller, E., Long, D., Maltzahn, C.: Ceph: A scalable, high-performance distributed file system. In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI), pp. 307–320 (2006)

[44] Padoin, E.L., Pilla, L.L., Castro, M., Boito, F.Z., Alexandre Navaux, P.O., Méhaut, J.-F.: Performance/energy trade-off in scientific computing: the case of ARM big. LITTLE and Intel Sandy Bridge. IET Computers & Digital Techniques **9**(1), 27–35 (2015)

[45] Criado, J., Garcia-Gasulla, M., Kumbhar, P., Awile, O., Magkanaris, I., Mantovani, F.: Coreneuron: performance and energy efficiency evaluation on intel and arm cpus. In: 2020 IEEE International Conference on Cluster Computing (CLUSTER), pp. 540–548 (2020). IEEE

[46] Mitra, G., Johnston, B., Rendell, A.P., McCreath, E., Zhou, J.: Use of SIMD Vector Operations to Accelerate Application Code Performance on Low-Powered ARM and Intel Platforms. In: Proceedings of IPDPSW, pp. 1107–1116 (2013)

[47] Flynn, P., Yi, X., Yan, Y.: Exploring source-to-source compiler transformation of openmp simd constructs for intel avx and arm sve vector architectures. In: Proceedings of PMAM, pp. 11–20 (2022)

[48] T. Gleixner, I.M.: Performance counters for linux (2009)

[49] Intel Corporation: Intel RDT Library. Available athttps://github.com/intel/intel-cmt-cat (2021)

[50] Michael Bayer et al.: Mako Templates. Available at http://www.makotemplates.org/ (2019)

[51] Mills, D.L.: Internet time synchronization: the network time protocol. IEEE Transactions on communications **39**(10), 1482–1493 (1991)

[52] Jia, R., Yang, Y., Grundy, J., Keung, J., Hao, L.: A systematic review of scheduling approaches on multi-tenancy cloud platforms. Information and Software Technology **132**, 106478 (2021) https://doi.org/10.1016/j.infsof.2020.106478

[53] Wang, Z., Xu, C., Agrawal, K., Li, J.: Adaptive scheduling of multiprogrammed dynamic-multithreading applications. Journal of Parallel and Distributed Computing **162**, 76–88 (2022) https://doi.org/10.1016/j.jpdc.2022.01.009

[54] Lu, C., Ye, K., Xu, G., Xu, C.-Z., Bai, T.: Imbalance in the cloud: An analysis on alibaba cluster trace. In: 2017 IEEE International Conference on Big Data, pp. 2884–2892 (2017). https://doi.org/10.1109/BigData.2017.8258257

[55] Liu, Q., Yu, Z.: The elasticity and plasticity in semi-containerized co-locating cloud workload: A view from alibaba trace. In: Proceedings of the ACM Symposium on Cloud Computing (SoCC), pp. 347–360 (2018). https://doi.org/10.1145/3267809.3267830

[56] Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., Bianchini, R.: Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In: Proceedings of the 26th Symposium on Operating Systems Principles (SOSP), pp. 153–167 (2017). https://doi.org/10.1145/3132747.3132772

[57] Intel: Improving real-time performance by utilizing cache allocation technology. Intel Corporation, April (2015)

[58] Andrew H., Abbasi, Khawar M., Marcel C.: Introduction to Memory Bandwidth Allocation. Available at https://software.intel.com/en-us/articles/introduction-to-memory-bandwidth-allocation (2019)

[59] Lo, D., Cheng, L., Govindaraju, R., Ranganathan, P., Kozyrakis, C.: Heracles: Improving resource efficiency at scale. In: Proceedings of ISCA, pp. 450–462 (2015)

[60] Yang, H., Breslow, A., Mars, J., Tang, L.: Bubble-Flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers. In: Proceedings of ISCA, pp. 607–618. Association for Computing Machinery, New York, NY, USA (2013)

[61] Moradi, H., Wang, W., Zhu, D.: DiHi: Distributed and Hierarchical Performance Modeling of Multi-VM Cloud Running Applications. In: Proceedings of HPCC/SmartCity/DSS, pp. 1–10 (2020)

[62] Masouros, D., Xydis, S., Soudris, D.: Rusty: Runtime interference-aware predictive monitoring for modern multi-tenant systems. IEEE Transactions on Parallel and Distributed Systems **32**(1), 184–198 (2020)

[63] Jackson, A., Turner, A., Weiland, M., Johnson, N., Perks, O., Parsons, M.: Evaluating the Arm Ecosystem for High Performance Computing. In: Proceedings of PASC, pp. 1–11. Association for Computing Machinery, ??? (2019)

[64] Chen, S., Galon, S., Delimitrou, C., Manne, S., Martínez, J.F.: Workload characterization of interactive cloud services on big and small server platforms. In: Proceedings of IISWC, pp. 125–134 (2017)

[65] Hammond, S.D., Hughes, C., Levenhagen, M.J., Vaughan, C.T., Younge, A.J., Schwaller, B., Aguilar, M.J., Pedretti, K.T., Laros, J.H.: Evaluating the Marvell ThunderX2 Server Processor for HPC Workloads. In: Proceedings of HPCS, pp. 416–423 (2019)

[66] Olson, R.S., La Cava, W., Orzechowski, P., Urbanowicz, R.J., Moore, J.H.: PMLB: a large benchmark suite for machine learning evaluation and comparison. BioData mining **10**(1), 1–13 (2017)

[67] Olson, R.S., La Cava, W., Mustahsan, Z., Varik, A., Moore, J.H.: Data-driven Advice for Applying Machine Learning to Bioinformatics Problems. arXiv e-print. https://arxiv.org/abs/1708.05070 (2017)

[68] Slimani, S., Hamrouni, T., Ben Charrada, F.: Service-oriented replication strategies for improving quality-of-service in cloud computing: a survey. Cluster Computing **24**, 361–392 (2021)

[69] Barroso, L.A., Hölzle, U., Ranganathan, P.: The Datacenter as a Computer: Designing Warehouse-scale Machines. Springer, ??? (2019)

[70] Kang, Y., Zheng, Z., Lyu, M.R.: A latency-aware co-deployment mechanism for cloud-based services. In: 2012 IEEE Fifth International Conference on Cloud Computing, pp. 630–637 (2012). https://doi.org/10.1109/CLOUD.2012.90

[71] Zhang, Y., Hua, W., Zhou, Z., Suh, G.E., Delimitrou, C.: Sinan: ML-Based and QoS-Aware Resource Management for Cloud Microservices. In: Proceedings of ASPLOS. Association for Computing Machinery, New York, NY, USA (2021)

[72] Zhang, I., Raybuck, A., Patel, P., Olynyk, K., Nelson, J., Leija, O.S.N., Martinez, A., Liu, J., Simpson, A.K., Jayakar, S., *et al.*: The demikernel datapath os architecture for microsecond-scale datacenter systems. In: Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, pp. 195–211 (2021)