Performance Isolation of Data-Intensive Scale-out Applications in a Multi-tenant Cloud

Palden Lama
Department of Computer Science
University of Texas at San Antonio
San Antonio, TX, USA.
Email: palden.lama@utsa.edu

Shaoqi Wang, Xiaobo Zhou
Department of Computer Science
University of Colorado, Colorado Springs
Colorado Springs, Colorado, USA.
Email: swang,xzhou@uccs.edu

Dazhao Cheng
Department of Computer Science
University of North Carolina at Charlotte
Charlotte, North Carolina, USA.
Email: dazhao.cheng@uncc.edu

***Abstract*—Data-intensive applications often suffer from performance variability and degradation in the cloud due to intrinsically complex problem of performance interference that arises from multi-tenancy. Although application-level approach of straggler mitigation for scale-out data processing frameworks such as MapReduce and Spark, address the issue to some extent, they incur extra resource and often react after tasks have already slowed down. In this paper, we present *PerfCloud*, a novel system software that utilizes system level performance metrics for early detection of performance interference in a multi-tenant cloud, and provides non-invasive performance isolation through fine-grained resource control. Unlike existing works, PerfCloud does not require time-consuming workload profiling, or intrusive modification of the application framework and the operating system. We implemented PerfCloud on NSF Cloud's Chameleon testbed using KVM for virtualization, and OpenStack for cloud management. Experimental results with Hadoop MapReduce and Spark benchmarks show that PerfCloud effectively reduces their job completion time, decreases performance variability, and improves resource utilization efficiency while minimizing the performance degradation of other colocated VMs.**

***Keywords*-performance isolation; resource scheduling; cloud;**

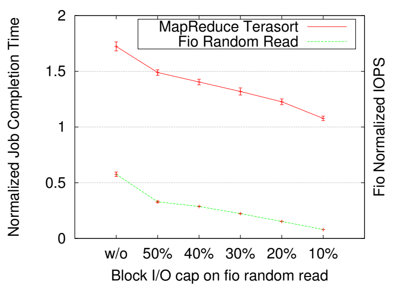
I. Introduction

Cloud applications often suffer from interferences from other applications and cannot fully leverage the on-demand resource elasticity in the cloud, leaving the abundant parallelism and potential scalability unexploited. In particular, scale-out data processing frameworks (e.g. MapReduce, Spark, etc.) hosted in the cloud are vulnerable to *stragglers* which occur due to a variety of reasons including performance interference [1], [2] and hardware heterogeneity [3], [4], [5]. Despite existing mitigation techniques, stragglers can be up to 6-8x slower than the median task in job on a production cluster, leading to high job completion time and inefficient resource utilization [6]. Furthermore, stragglers especially affect small jobs, i.e., jobs that consist of a few tasks, and have stringent requirement for low latencies.

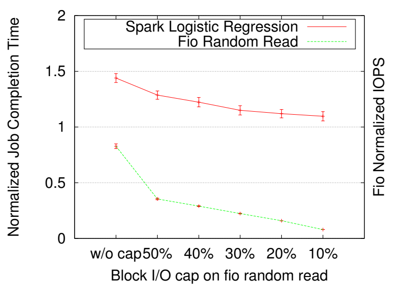
Existing efforts on resource management and scheduling for performance isolation in multi-tenant systems often require expensive, time-consuming workload profiling [7], [8], [2], [9], [10], [11], [12], [13], or intrusive instrumentation and modification of application framework [14], [15], [16], and guest operating system [17]. Hence, adoption of existing techniques for large scale applications in a real cloud platform faces a significant and practical barrier. At the application level, parallel data-processing frameworks attempt to mitigate these issues by marking slow running tasks as stragglers, relaunching multiple copies of them, and picking the earliest copy that finishes [5]. Such *wait-and-speculate* mechanisms are inefficient because a task is allowed to run for a significant amount of time before it can be identified as a straggler [18]. Furthermore, a speculative copy of a task has a direct impact on the resources available for other jobs, and the killed tasks lead to significant resource waste [19]. A recent study [20] found that in Facebook's Hadoop cluster, speculative tasks alone account for 25% of all tasks and 21% of resource usage. There are other replication based approaches [6] that avoid waiting and speculation altogether, and mitigate stragglers through full cloning of small jobs. However, such approach still incurs extra resources.

There are several challenges in enabling system support for performance isolation of data-intensive scale-out applications in a multi-tenant Cloud. First, it is difficult to shift the performance optimization efforts from application-level to the system software. This is due to the fact that cloud applications hosted on VMs appear as black boxes to the system software. Hence, system-level performance isolation mechanism needs to identify and address performance issues in a non-invasive manner with minimal interaction with the application. Second, quick detection and mitigation of performance interference is crucial for improving the performance of short jobs. Previous study [6] on Hadoop production cluster at Facebook shows that over 80% of the Hadoop jobs run for a short duration with fewer than ten tasks. Achieving low latencies for these small interactive jobs is of prime concern to datacenter operators.

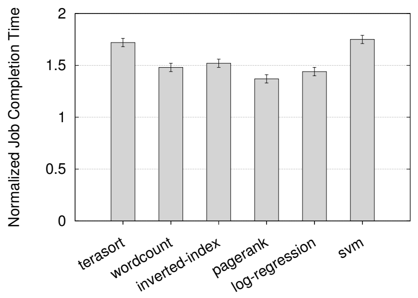
We present, *PerfCloud*, a novel system software that utilizes system-level performance metrics for early detection of performance interference experienced by data-intensive scale-out applications, and provides non-invasive performance isolation through fine-grained resource control. PerfCloud is designed to work at cloud scale, and is composed of lightweight and decentralized agents that monitor individual physical servers in a cloud datacenter. PerfCloud utilizes the disk I/O related performance metrics, `blkio.io_wait_time` and `blkio.io_serviced`, to detect I/O contention and the CPI (clock cycles per instruction) metric collected from



(a) MapReduce performance isolation with static I/O control.



(b) Spark performance isolation with static I/O control.



(c) Performance degradation in the absence of I/O control.

Fig. 1: Performance degradation due to colocated I/O intensive workload.

hardware performance counters to detect the contention of processor resources, such as the last level cache (LLC) and memory bandwidth. Hence, it is able to detect performance issues more quickly than application-level approaches such as speculative execution. It accurately identifies the antagonistic VMs based on online cross-correlation analysis between the resource usage pattern of colocated VMs.

To achieve performance isolation, PerfCloud applies a dynamic resource control algorithm that adjusts the CPU and disk I/O bandwidth of antagonistic VMs based on a TCP congestion control (CUBIC)-inspired technique [21]. We assume that a cloud administrator may set different priorities to various cloud instances/VMs possibly based on the cost of reserving the specific instance types. PerfCloud aims to achieve performance isolation of high-priority data-intensive workloads, while minimizing the performance degradation of low-priority antagonistic VMs. By mapping the problem of shared resource contention in a multi-tenant cloud to the problem of flow control in networks, PerfCloud achieves a stable control behavior guaranteed by the CUBIC congestion control technique, while avoiding time-consuming workload profiling and error-prone performance modeling.

We implemented PerfCloud on NSF Cloud's Chameleon testbed using KVM for virtualization, and OpenStack for cloud management. Experiments results show that PerfCloud outperforms state-of-the-art techniques LATE [5] and Dolly [6] in achieving performance isolation and resource utilization efficiency of representative applications from Purdue MapReduce Benchmark Suite (PUMA) [22] and SparkBench [23]. The rest of this paper is organized as follows. Section II presents our motivational case study. Section III elaborates PerfCloud's key design and implementation details. Section IV gives the testbed setup and experimental results. Related work is presented in Section V. Conclusion is in Section VI.

II. Motivation

We present motivating examples to demonstrate the need and challenges of system level support for performance isolation in a multi-tenant cloud. We set up a Hadoop cluster of 6 VMs, where each VM is configured with 2-vcpu, 8GB RAM, and hosted on NSF Chameleon Cloud's Dell PowerEdge R630 bare-metal server. The Hadoop cluster is configured to run MapReduce and Spark workloads from the Purdue MapReduce Benchmark (PUMA) [22] and SparkBench [23] suite respectively. Here, terasort, wordcount, and inverted-index benchmarks belong to PUMA, while page-rank, logistic regression, and svm benchmarks belong to SparkBench.

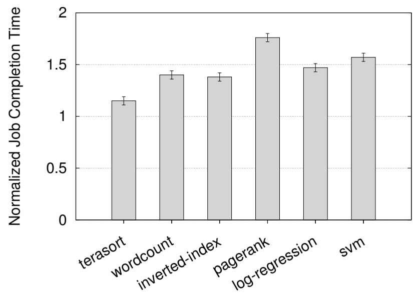
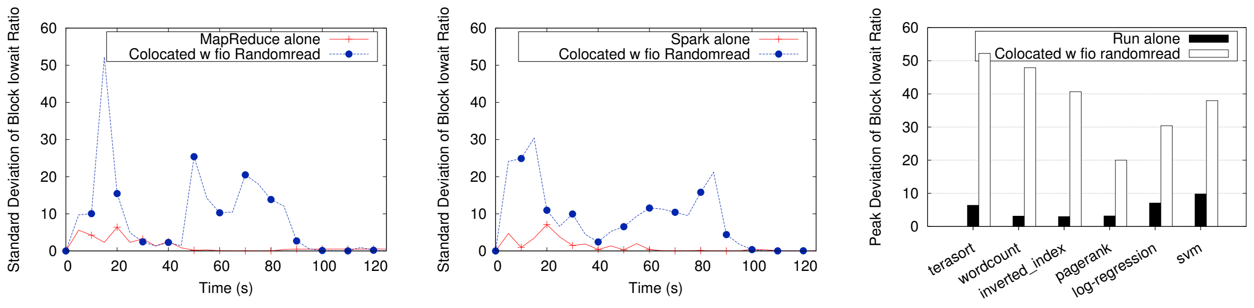


Fig. 2: Performance degradation due to colocated memory intensive workload.

A. Performance Interference

To analyze multi-tenant interference, we co-locate a VM that runs an I/O intensive random read benchmark using the Flexible I/O Tester (fio) tool. All the VMs were configured with caching option set to 'none', in order to avoid performance variability introduced by disk page caching on the host machine. The results presented are the average of five runs, where each VM's page cache is cleared before each run. We measure the job completion time of MapReduce, and Spark workloads normalized with respect to their respective job completion times when the antagonistic VM is not colocated with the Hadoop VMs. We also measure the I/O operations per sec (IOPS) of fio random read benchmark normalized with respect to its IOPS when it is running alone. As shown in Figures 1 (c), the interference caused by fio random read benchmark degrades the performance of MapReduce Terasort, and Spark Logistic Regression by 72% and 44% respectively. Next, we apply the linux block I/O subsystem's throttling policy to set various limits on the total (read and write) I/O throughput (bytes-per-sec) of the colocated VM. For example, an I/O cap of 50% limits the total I/O throughput of fio random



(a) MapReduce Terasort (b) Spark Logistic Regression (c) Various Benchmarks

Fig. 3: Standard deviation of block iowait ratio across Hadoop VMs with and without disk I/O contention.

read benchmark to 50% of its throughput obtained when it is running alone. MapReduce, and Spark performance improves as the I/O cap on the antagonistic VM decreases.

B. Challenges of Resource Throttling

Although resource throttling based performance isolation seems intuitive, it is challenging due to the following reasons. First, aggressive resource throttling comes at the cost of performance degradation of the colocated VMs. As shown in Figures 1 (a) and (b), the performance of fio random read benchmark decreases significantly as we lower the I/O cap on its VM. A naive strategy of applying I/O caps on all colocated VMs may lead to their unwarranted performance degradation. Second, it is difficult to determine how much I/O throttling is effective in achieving performance isolation, since the extent of performance interference varies across various colocated workloads as shown in Figure 1 (c). Furthermore, I/O throttling of antagonistic VMs beyond a certain level may not provide much performance gain in some workloads. For example, Figure 1 (b) shows that decreasing the I/O cap on the antagonistic VM beyond 20% shows very little improvement on the Spark application performance, while it degrades the performance of the colocated random read benchmark significantly. This is because disk I/O is no longer a bottleneck for the Spark workload in this case.

C. Contention of Shared Processor Resources

Performance interference may also arise from the contention of other prominent shared resources such as the last level cache (LLC) and memory bandwidth [24]. To measure the impact of contention in shared processor resources, we co-locate a VM running memory-intensive STREAM benchmark [25] on the physical server hosting Hadoop VMs. As shown in Figure 2, both MapReduce and Spark benchmarks suffer from significant performance degradation due to interference caused by the colocated memory-intensive workload.

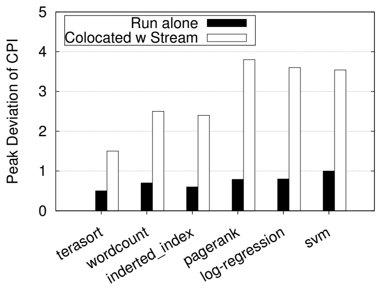


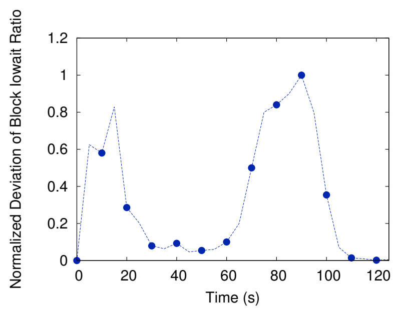
Fig. 4: Interference detection with CPI metric.

III. PerfCloud Design and Implementation

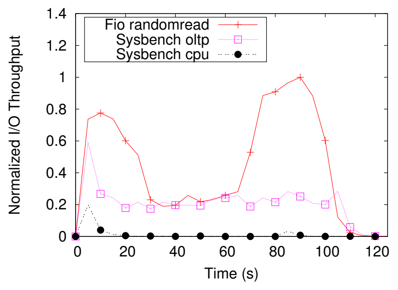
A. Non-invasive Detection of Performance Interference

1) I/O Contention: We now describe how PerfCloud detects I/O contention experienced by data-intensive scale-out applications by utilizing system-level performance metrics at run time. The key insight behind the technique is that since scale-out applications such as MapReduce and Spark are designed to distribute the work evenly across multiple worker nodes, the deviation in I/O behavior among the worker nodes can indicate I/O contention related issues. Our study shows that the standard deviation of the ratio of blkio.io_wait_time and blkio.io_serviced across the various VMs running a data-intensive application on a physical server can serve as an early indicator of the underlying performance issues. blkio.io_wait_time reports the total time I/O operations by a cgroup spent waiting for service in the scheduler queues, and blkio.io_serviced reports the number of I/O operations performed by a cgroup. Each cgroup is mapped to a VM. These metrics are easily obtainable from the Block I/O (blkio) subsystem, a linux kernel module, that controls and monitors access to I/O on block devices by tasks in cgroups.

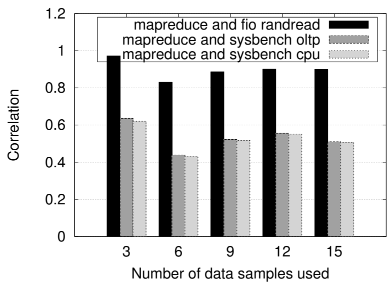
As a case study, we measure the standard deviation of block iowait ratio across the Hadoop VMs at fixed time intervals when a MapReduce Terasort job with 10 map and 10 reduce tasks was executed. We collect this metric when the MapReduce job is running alone, and when the Hadoop VMs are colocated with another VM running an I/O intensive fio benchmark. All the VMs are hosted on a single physical server. As shown in Figure 3 (a), the disk I/O contention caused by fio random read benchmark significantly increases the standard deviation of $blkio.io\_wait\_time/blkio.io\_serviced$ across Hadoop VMs. On average, the peak standard deviation of block iowait ratio increases by a factor of 8.2 due to disk I/O contention. Correspondingly, MapReduce job performance



(a) MapReduce Terasort



(b) Colocated workloads



(c) Correlation between deviation of block iowait ratio and colocated VM's I/O throughput.

Fig. 5: Antagonist identification based on the correlation of Block I/O wait and I/O throughput between colocated VMs.

degrades by a factor of 1.7. As shown in Figure 3 (b), we observe similar results when the VMs are configured to run the Spark Logistic Regression benchmark, while being colocated with a VM running fio random read benchmark. Futhermore, Figure 3 (c) shows that the standard deviation of block iowait ratio shows a similar pattern for various MapReduce and Spark benchmarks, when they are colocated with fio random read benchmark. We also observe that this metric does not exceed a threshold of 10 when the MapReduce and Spark workloads are running alone. An important aspect of the proposed approach is that it is able to identify performance interference within a few seconds. This is in sharp contrast to application-level speculative execution approaches, which need the tasks to run for a significant amount of time for straggler detection.

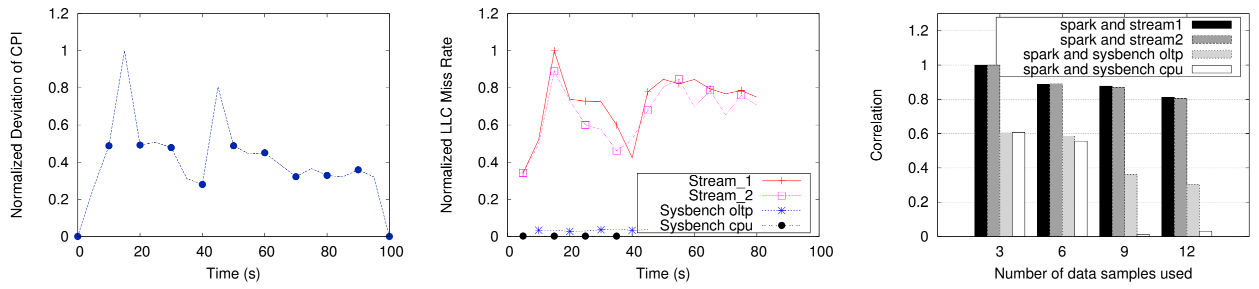
2) Contention of Shared Processor Resources: In order to detect contention of shared processor resources, PerfCloud utilizes a hardware performance counter based metric, CPI (clock cycles per instructions). Our study shows that the standard deviation of CPI measured across the various VMs running a data-intensive application on a physical server increases significantly, when a memory-intensive VM is colocated. As shown in Figure 4, the peak deviation in CPI does not exceed a threshold of 1 when the MapReduce and Spark workloads are running alone. However, with a colocated memory-intensive STREAM benchmark, the peak deviation is much higher than 1 for various MapReduce and Spark benchmarks. From Figures 2 and 4, we observe that the deviation in CPI is correlated with the amount of performance degradation caused by the STREAM benchmark. Furthermore, compared to MapReduce, the Spark benchmarks experience a larger impact of contention in shared processor resources in terms of performance degradation. Spark jobs are more sensitive to LLC miss rates and memory bandwidth contention as it frequently reuses intermediate results residing in memory.

B. Identifying Antagonists

PerfCloud performs online cross-correlation analysis to quickly determine which colocated VMs are the likely cause of performance interference with data-intensive scale-out applications hosted in a multi-tenant system. In particular, it identifies the antagonists by looking for correlations between the system-level performance metrics of victim VMs and the colocated VMs. A good correlation means that the suspect is highly likely to be a real antagonist. PerfCloud uses the Pearson Correlation Coefficient for this purpose.

To identify antagonists responsible for I/O contention, PerfCloud calculates the correlation between a time series of standard deviation of block iowait ratio among the data-intensive application VMs, and a time series of I/O throughput of a colocated VM respectively. As a case study, we colocate VMs running the MapReduce Terasort benchmark, with VMs running fio random read, sysbench oltp, and sysbench cpu benchmarks on the same physical server. The sysbench oltp benchmark is run for 120 seconds with eight threads to perform read-only test on a MySQL database table of size 10000000. The sysbench cpu benchmark is run with four threads each to calculate prime numbers with a maximum value of 12000000. Figure 5 (a) shows the standard deviation of block iowait ratio among the Hadoop VMs, normalized by the peak deviation. Figure 5 (b) shows the total I/O throughput (bytes-per-sec) measured from other colocated VMs, normalized by the peak throughput observed across the VMs over the duration of the experiment. Figure 5 (c) compares the correlation results obtained with various sizes of dataset collected from our experiment. There is a strong correlation between the I/O metrics of Hadoop VMs, and VM running the fio benchmark. PerfCloud is able to identify an antagonistic VM with dataset size as small as three, which can be obtained quickly within three measurement intervals.

To identify antagonists responsible for contention of processor resources, PerfCloud calculates the correlation between a time series of standard deviation of CPI among the data-intensive application VMs, and a time series of LLC miss rates of a colocated VM respectively. VMs showing high LLC miss rates are more likely to put pressure on the shared last level cache and memory bandwidth [24]. As a case study, we colocate VMs running the Spark Logistic Regression benchmark, with two VMs each running a memory-intensive STREAM benchmark, and remaining two VMs running sysbench oltp, and sysbench cpu benchmarks. Each STREAM benchmark is run with only eight threads and an array size of 2 billion elements to create a situation where a group of antagonists



(a) Spark Logistic Regression

(b) Colocated workloads

(c) Correlation between deviation of CPI and colocated VM's LLC miss rate.

Fig. 6: Antagonist identification based on the correlation of CPI and LLC miss rates between colocated VMs.

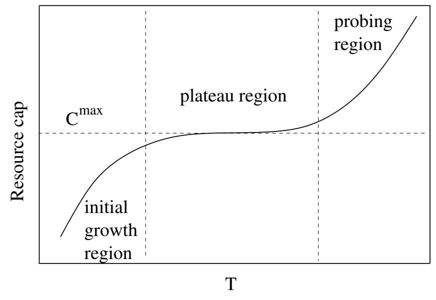


Fig. 7: Cubic function for dynamic resource capping.

together cause significant performance interference, but which individually do not have much effect. In our testbed, a STREAM benchmark with 16 threads is sufficient to cause significant performance interference. Figures 6 (a) and (b) show the normalized standard deviation of CPI among the Hadoop VMs, and the normalized LLC miss rates of colocated VMs respectively. The LLC miss rates are not counted when the VMs are not running any workload. In such cases, PerfCloud treats missing values in time series vectors as 0 (rather than omitting them, as is typically done when computing the Pearson correlation). As a result, PerfCloud is able to avoid over-emphasizing similarities computed over little data. As shown in Figure 6 (c), the standard deviation of CPI among Hadoop VMs, and the LLC miss rates of the two VMs running the STREAM benchmarks are highly correlated, with a correlation coefficient value greater than 0.8.

C. Mitigating Performance Interference

PerfCloud utilizes the disk I/O throttling policy of Block I/O subsystem and CPU hard-capping [26] to reduce the I/O throughput and CPU usage of low-priority antagonistic VMs for achieving performance isolation of high-priority data-intensive applications. While disk I/O throttling and CPU hard-capping are effective in mitigating the contention of disk I/O and shared processor resources respectively, an important question is how much resource capping is required to achieve performance isolation, while avoiding unwarranted performance degradation of antagonistic VMs. A naive approach may apply ad-hoc resource capping on antagonists, whenever resource contention is detected. However, such ad-hoc policies may lead to oscillatory and unstable system behavior [27]. To address this challenge, PerfCloud uses a control mechanism shown to exhibit the stable behavior of CUBIC congestion control [21]. Here, we map the problem of shared resource contention among colocated VMs to the problem of flow control in networks. In principle, resource contention experienced by high-priority data-intensive applications is analogous to TCP network congestion, and allocating disk I/O and CPU bandwidth to colocated VMs while avoiding significant resource contention is similar to congestion control. PerfCloud initiates its dynamic resource control algorithm according to Equation 1, whenever I/O or processor resource contention is detected, and one or more antagonistic VMs are identified on a physical server. The rationale for using CUBIC congestion control-inspired technique is that it provides good control stability without requiring time-consuming workload profiling and performance modeling, which is often prone to inaccuracy.

$$c_i(t+1) = \begin{cases} (1-\beta)\cdot c_i(t), & \text{if } I(t) > \mathscr{I} \\ \gamma\cdot(T_i - \sqrt[3]{\frac{C_i^{max}\cdot\beta}{\gamma}})^3 + C_i^{max}, & \text{otherwise} \end{cases} \qquad (1)$$

Here $c_i(t)$ denotes the resource (CPU or I/O) cap applied on antagonistic VM i at sampling interval t, I(t) is the standard deviation of block iowait ratio or CPI among the VMs belonging to a high-priority application, $\mathscr{I}$ is the threshold parameter for I(t), T_i is the elapsed time (number of intervals) since the last resource cap-decrease event on VM i, C_i^{max} is the resource cap at the time of the last cap-decrease event, and $\gamma \in [0,1]$ is a scaling constant. The resource cap, $c_i(t)$, for each antagonistic VM, at $t = 1$, is initialized to be equal to the VM's observed CPU usage or I/O throughput. Whenever I(t) $> \mathscr{I}$ implying disk I/O or processor resource contention, PerfCloud decreases the CPU or I/O cap on the antagonistic VMs multiplicatively by a factor of β. When the performance interference subsides and I(t) $< \mathscr{I}$, PerfCloud raises the resource caps according to the cubic function given by Equation 1.

After a reduction in resource cap of an antagonistic VM, the cubic increase function operates in three regions as shown in Figure 7. (1) *Initial growth region*: the resource cap increases

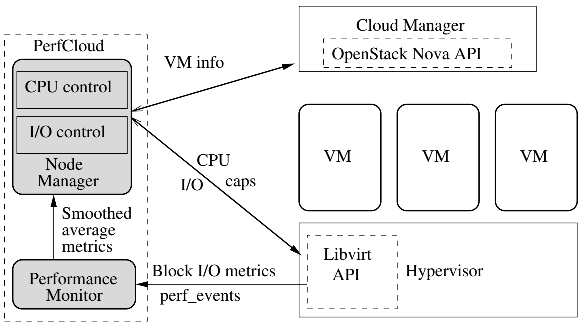


Fig. 8: PerfCloud Architecture.

steeply, as long as I(t) $< \mathscr{I}$, and the cap is smaller than C_i^{max}. This allows the antagonistic VMs to quickly get a fair allocation of CPU or I/O bandwidth as long as the performance of high-priority application is not affected. Here, fairness is measured among low-priority VMs only, since high-priority VMs always get higher preference. *The motivation here is to penalize low-priority VMs only when they are misbehaving*. (2) *Plateau region*: As the resource cap gets closer to C_i^{max}, it slows down its growth and increases very conservatively. As a result, a high-priority application that has a temporary drop in resource demand followed by a demand surge will not suffer from resource contention. The length of the plateau region is determined by the scaling factor γ. (3) *Probing region*: If no resource contention is detected after spending time in the plateau region, the resource cap will increase aggressively to probe for more CPU or I/O bandwidth.

The design parameters, β and γ, used in Equation 1 are tuned to achieve good performance isolation in a timely manner, while avoiding unwarranted performance degradation of antagonists. We empirically set β to 0.8, γ to 0.005. Based on our observation in section III-A, we set the threshold parameter $\mathscr{I}$ to 10 for detecting I/O contention, and 1 for detecting processor resource contention. The threshold is determined by the peak standard deviation of block iowait ratio, and CPI that are observed when there is no resource contention.

D. Putting It All Together

Figure 8 shows the architecture of PerfCloud. It is composed of lightweight and decentralized agents that run on individual physical servers in a cloud datacenter. Each agent, called the *node manager* is responsible for the performance isolation of high priority data-intensive applications hosted on a physical server. Without loss of generality, this paper implements the proposed ideas on a prototype cloud system, using KVM for virtualization, and running the *node manager* as a daemon process on the host machine.

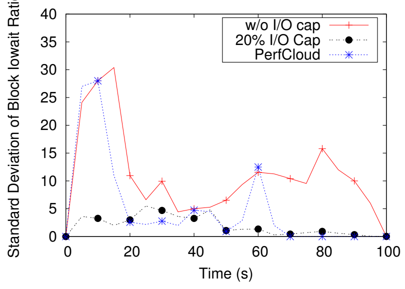
1) Performance Monitor: The *performance monitor* periodically measures the blkio.io_wait_time, blkio.io_serviced, and CPI metrics for each VM belonging to a high-priority data-intensive application hosted on the physical server. It also measures the I/O throughput in terms of blkio.io_service_bytes, LLC miss rate, and CPU usage for each low-priority VM colocated on the same server. The Libvirt API is used to

Algorithm 1 Dynamic Resource Control Algorithm.

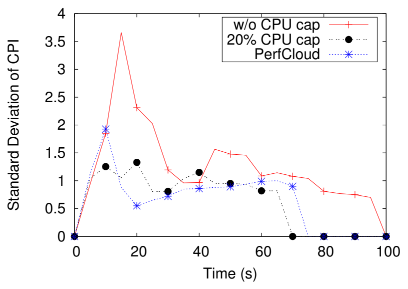
```
1: Variables: $H$: list of VMs belonging to high-priority application; $L$: list of low-priority VMs; $A$: list of antagonistic VMs; $\theta$: list of I/O throughput (MB/s) (for I/O control) and CPU usage scaled to [0,1] (for CPU control) of antagonistic VMs;
2:
3: while $true$ do
4:     $H, L = getvminfo()$
5:     calculate I(t): the standard deviation of CPI (for CPU control) and iowait (for I/O control).
6:     if $t\%\tau == 0$ then
7:         $A, \theta = identifyAntagonists()$
8:     end if
9:     if $I(t) > \mathscr{I}$ then
10:        for $\theta_i$ in $\theta$ do
11:            if $t == 1$ then
12:                resource cap $c_i(t) = \theta_i$
13:            end if
14:            $c_i(t+1) = (1-\beta) \cdot c_i(t)$
15:            apply resource cap $c_i(t+1)$ on VM $a_i$
16:        end for
17:    else
18:        for $a_i$ in $A$ do
19:            $c_i(t+1) = \gamma \cdot (T_i - \sqrt[3]{\frac{C_i^{max} \cdot \beta}{\gamma}})^3 + C_i^{max}$
20:            apply resource cap $c_i(t+1)$ on VM $a_i$
21:        end for
22:    end if
23:    $t = t + 1$
24: end while
```

collect the Block I/O metrics from the hypervisor. Since these metrics provide cumulative values from the time the VMs were booted, we calculate the delta values between consecutive measurement intervals. CPI and LLC miss rates are measured on a per cgroup basis by using the perf_event [28] tool. The *performance monitor* applies an exponentially weighted moving average (EWMA) technique to smooth out short-term variations in the data collected over 5 second intervals.

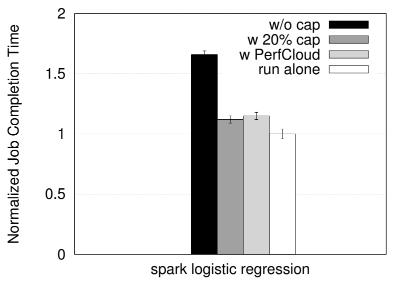
2) Node Manager: As shown in Algorithm 1, the *node manager* periodically contacts the *cloud manager* to fetch relevant information about the VMs hosted on the physical server, including VM priority (high/low), and a list of VMs that belong to the same high-priority application. As a result, it is aware of possible changes in VM placement caused by arrival of new VMs, VM migration, etc. We use an open-source cloud management software, OpenStack, to build a cloud environment in our testbed. The *node manager* uses OpenStack Nova API to interact with the cloud manager. Next, it calculates the standard deviations of the block iowait ratio and CPI among the high-priority application VMs based on the data collected by the *performance monitor*. It periodically identifies the antagonistic VMs. As described in Section III-B, the low-priority VMs that have a correlation of 0.8 or more are



(a) Block Iowait.

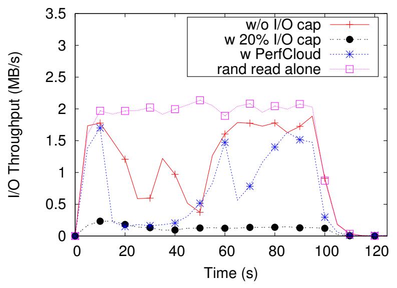


(b) CPI.

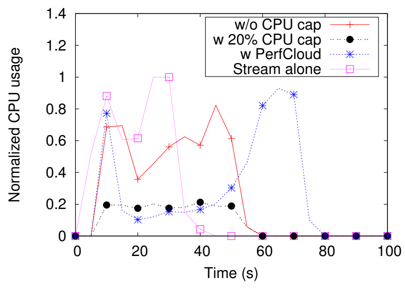


(c) Performance Isolation.

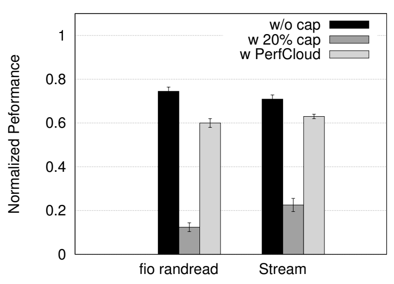
Fig. 9: Performance isolation of spark workload colocated with I/O intensive and memory intensive workloads.



(a) Fio random read.



(b) Stream benchmark.



(c) Impact on Antagonists.

Fig. 10: PerfCloud's dynamic resource control on antagonistic VMs.

considered to be the antagonists. The *node manager* runs both CPU control and I/O control modules to calculate the resource caps, $c_i(t)$, to be set on the antagonistic VMs according to Equation 1. It uses the Libvirt API to apply the CPU caps through *vcpu_quota*, and the I/O caps through block I/O subsystem's throttling policy.

IV. Evaluation

A. Testbed Setup

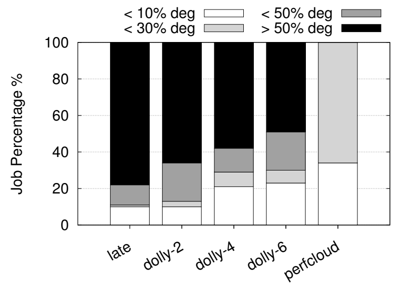
We have implemented and evaluated PerfCloud on the NSF Cloud's Chameleon testbed, using Dell PowerEdge R630 bare-metal servers equipped with 2.3 GHz 48 core Intel Xeon processor and 125 GB memory. We built a cloud environment by using KVM for server virtualization, and OpenStack for cloud management. Experiments were conducted both at a small scale with a 12-node virtual Hadoop cluster running on a single bare-metal server, and at a large scale with a 152-node virtual Hadoop cluster running on 15 bare-metal servers. Each node was configured with 2 VCPU and 8 GB memory. Two nodes were configured as the `NameNode` and `JobTracker/Spark Master`, respectively. The rest of the nodes ran as slave nodes for HDFS storage and MapReduce/Spark task execution. We set the HDFS block size to its default value 64 MB.

For performance evaluation, we used the workloads derived from the Purdue MapReduce Benchmark Suite (PUMA) [22] and SparkBench [23] suite as representative data-intensive scale-out applications. In the PUMA suite, we used various MapReduce benchmarks with real-world test inputs from Wikipedia as well as data generated by the *TeraGen* tool. In the SparkBench suite, we used various Spark benchmarks with data generated from SparkBench's synthetic data generator.

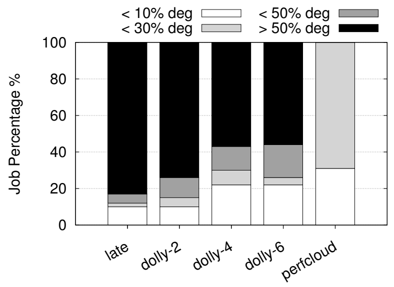
B. Impact of Dynamic Resource Control

First, we evaluate the effectiveness of PerfCloud's dynamic resource control technique in achieving performance isolation of a Spark Logistic Regression benchmark with at most 40 tasks per stage running on a 12-node virtual Hadoop cluster. To create performance interference, we co-locate two VMs running the I/O intensive fio random read benchmark and memory-intensive STREAM benchmark respectively. In addition, physical server also hosts VMs running sysbench oltp and sysbench cpu benchmarks. Figures 9 (a) and (b) show that compared to default system without resource capping, PerfCloud significantly reduces the standard deviation of block iowait ratio and CPI across Hadoop VMs. This result demonstrates that PerfCloud is able to detect and reduce the resource contention experienced by the Hadoop VMs. The relationship between resource contention, block iowait ratio and CPI has been established in Section III-A.

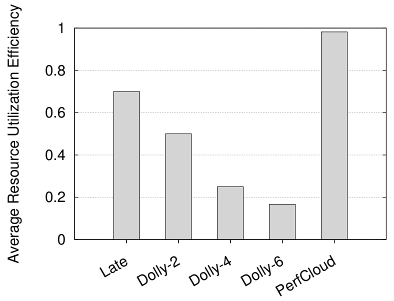
Figure 9 (c) compares the performance isolation effectiveness of PerfCloud with the default system, and a static resource capping policy that applies 20% I/O cap on the VM running fio random read benchmark, and 20% CPU cap on the VM running STREAM benchmark. PerfCloud and the static resource capping policy outperform the default system by 31% and 33%



(a) Breakdown of MapReduce job performance.

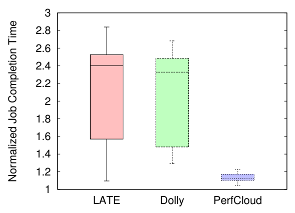


(b) Breakdown of Spark job performance.

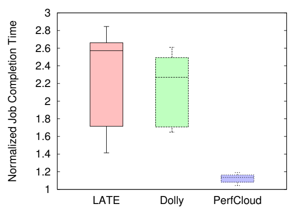


(c) Average resource utilization efficiency.

Fig. 11: Large-scale evaluation of LATE, Dolly, and PerfCloud.



(a) MapReduce Terasort



(b) Spark Log Regression

Fig. 12: Performance Variability

respectively in terms of the normalized job completion time of the Spark application. This is because both schemes quickly throttle the antagonistic VMs and mitigate the performance interference issue. However, as shown in Figure 10 (c), PerfCloud results in significantly less performance degradation of antagonistic VMs than that caused by the static capping policy. The normalized throughput of fio random read benchmark and STREAM triad kernel are 4.8X and 2.8X higher respectively in case of PerfCloud. This is due to PerfCloud's dynamic resource control algorithm, which throttles the colocated VMs only when there is resource contention, and allows them to quickly regain CPU and I/O bandwidth when there is no contention. As shown in Figures 10 (a) and (b), PerfCloud throttles fio random read and STREAM benchmarks during the time 15 to 40 seconds, which correspond to the initial growth and plateau region of CUBIC control. After time 40 seconds, resource caps are allowed to increase rapidly (probing region). In case of fio random read, another I/O throttling occurs at time 65 seconds due to a sudden increase in the standard deviation of block iowait ratio as shown in Figure 9(a). The STREAM benchmark finishes at different times under different schemes.

C. Large Scale Evaluation

We now present the results of large-scale experiments conducted on a 152-node virtual Hadoop cluster, which is distributed over 15 physical servers. We use two workload mixes of 100 MapReduce and 100 Spark benchmarks derived from the PUMA and SparkBench suite respectively. Similar to the related work [6], 80% of the MapReduce jobs have less than 10 map/reduce tasks, and 20% of the jobs have 10 to 50 tasks. In the Spark workload mix, 80% of the applications have at most 10 tasks per stage, and 20% of the applications have at most 50 tasks per stage. These workload mixes are generated by using different sizes of input data for the MapReduce and Spark benchmarks. On each job execution, we randomly distribute antagonistic VMs running the I/O intensive fio, and memory-intensive STREAM benchmarks.

For performance comparison, we use the state-of-the-art techniques - LATE [5] scheduler and Dolly [6]. While LATE applies speculative task execution for straggler mitigation, Dolly relies on full cloning of jobs, avoiding waiting and speculation altogether. Dolly launches multiple clones of a job and uses the result of the first clone that finishes. This paper uses Dolly's job-level cloning feature instead of its finer-grained task-level cloning alternative since the latter requires significant modification of the MapReduce and Spark frameworks. Since the effectiveness and resource efficiency of Dolly depends on the number of clones, we evaluate Dolly with various number of clones (Dolly-2, Dolly-4, Dolly-6).

Figures 11 (a) and (b) show the breakdown of MapReduce and Spark job performance in case of LATE, Dolly, and PerfCloud respectively. We observe that PerfCloud outperforms LATE and Dolly by limiting the performance degradation of 34% of the MapReduce jobs to be less than 10%, 31% of the Spark jobs to be less than 10%, and that of 100% of all jobs to be less than 30%. Dolly outperforms the LATE scheduler, since its proactive job cloning approach does not have to wait and observe a task before acting. With the increase in number of clones, Dolly achieves better performance for more jobs. However, as shown in Figure 11 (c), Dolly's average resource utilization efficiency decreases significantly as more clones are used for straggler mitigation. We measure resource utilization efficiency as the ratio of sum of successful task execution times and the sum of all task execution times (including the tasks that are killed). Using a large number of clones implies that Dolly has to kill a large number of jobs, thus decreasing its resource utilization efficiency. Compared to LATE and Dolly, PerfCloud achieves consistently better performance for all jobs without sacrificing resource utilization efficiency. This is because PerfCloud addresses the root cause of performance interference by throttling the antagonistic VMs on all physical servers, and does not require any extra resource usage.

Finally, we evaluate PerfCloud's ability to reduce the performance variability of MapReduce and Spark workloads in a multi-tenant environment. We run a MapReduce Terasort job with 50 tasks, and a Spark Logistic Regression benchmark with 50 tasks per stage, while randomly colocating antagonistic VMs (fio and STREAM benchmarks) on the 15 physical servers on each job execution. We repeat the experiment 30 times. As shown in Figures 12 (a) and (b), the median and the spread of the normalized job completion time is much smaller in case of PerfCloud, as compared to the cases with LATE and Dolly. This is because unlike PerfCloud, LATE and Dolly's ability to mitigate performance interference depends on the distribution of antagonistic VMs among the physical servers. For instance, if antagonistic VMs run on most servers, most duplicate copies of tasks or jobs will also face interference.

D. Discussion

1) Overhead Analysis. PerfCloud incurs minimal overhead in achieving performance isolation. The use of Linux Block I/O subsystem for I/O monitoring does not introduce any additional overhead, since this kernel module is enabled by default. The overhead of measuring hardware performance counters is minimal as the perf_event tool is used in counting mode on a per-cgroup basis. Our study conducted on Chameleon Cloud's Dell PowerEdge R630 bare-metal server does not show any visible impact of perf_event tool on application performance. Furthermore, applying resource caps on a VM takes less than 30 ms. This overhead increases linearly with the number of antagonists, and is limited since each *node manager* acts on a single physical server only.

2) Future Work Due to its decentralized design, PerfCloud does not take into account the hardware heterogeneity of physical servers. As a result, VMs running on slower machines may still cause some tasks to straggle. In such cases, application-level approaches such as speculative execution can complement PerfCloud in collectively improving the performance of data-intensive applications. Furthermore, if multiple high-priority applications are colocated on the same server, the *node manager* can notify the *cloud manager* to address the issue through complementary solutions such as VM migration. We will evaluate PerfCloud on a heterogeneous server cluster along with VM migration in future. Furthermore, we will study the impact of other optimizations such as shared-memory communication among Hadoop VMs, and NUMA architecture-aware VM mapping on the effectiveness of PerfCloud.

V. Related Work

Performance isolation of co-hosted applications has become increasingly important as data centers tend to consolidate more and more workloads on fewer machines for improving server utilization, and reducing data center costs [10], [29], [30], [31], [32], [33], [13], [34], [35]. Delimitrou et al. [10], [29] developed a collaborative filtering technique to classify an unknown, incoming workload with respect to how much interference it will cause to co-scheduled applications and how much interference it can tolerate. Lo et al. [33] applied online monitoring and offline profiling information for latency-critical jobs to identify shared resource saturation, and provided performance isolation through a hybrid of hardware and software mechanisms. Chang et al. [2] proposed a statistical machine learning based I/O interference prediction model, and an interference-aware scheduler for data-intensive applications in virtualized environments. However, these techniques suffer from the overheads associated with data collection from previously scheduled applications, offline model training and profiling of incoming workloads. Furthermore, they rely on the assumption that interference-free workload placement is always possible. However, it is increasingly difficult to find interference-free placement for all the VMs belonging to scale-out applications, as the degree of parallelism grows.

There are application level approaches [3], [6], [36], [20], [18], [5] that aim to achieve performance isolation of data-parallel applications through straggler mitigation. Zaharia et al. [5] proposed a scheduling algorithm that prioritizes stragglers based on their expected time to finish, relaunches multiple copies of them, and picks the earliest copy that finishes. Ananthanarayanan et al. [36] proposed an algorithm that carefully uses speculation to mitigate the impact of stragglers in approximation jobs in data analytics. Ren et al. [20] presented a speculation-aware job scheduler that integrates the tradeoffs associated with speculation into job scheduling decisions. However, existing *wait-and-speculate* mechanisms are inefficient due to lack of timely straggler detection.

In work [6], the authors avoided waiting and speculation altogether by proposing full cloning of small jobs for straggler mitigation. However, such approach still incurs extra resources. Yadwadkar et al. [18] proposed a system that predicts stragglers using a statistical modeling technique based on cluster resource usage and uses these predictions to inform scheduling decisions. However, such approach suffers from large overheads involved with capturing training data per node in a cluster, and training straggler prediction models across workloads. Unlike exiting approaches, PerfCloud is significantly more agile, does not incur extra resource, and avoids the overheads of data collection and performance modeling.

Mace et al. [16] presented Retro, a resource management framework that monitors per-tenant resource usage within a shared distributed system, and exposes this information to a centralized resource management policy for performance isolation. Chen et al. [14] optimized MapReduce performance by dynamically provisioning map tasks to match the distinct machine capacity in a heterogeneous cluster. However, unlike PerfCloud, these approaches requires intrusive instrumentation, and modification of existing data processing frameworks.

VI. Conclusions

Performance isolation of data-intensive scale-out applications in a multi-tenant cloud has been a challenging problem. Application-level approaches of straggler mitigation alone are ineffective due to the lack of timely straggler detection and inefficient due to the use of extra resources for task duplication. In this paper, we designed and implemented *PerfCloud*, a novel system software that utilizes system-level performance metrics for early detection of performance interference in a multi-tenant cloud, and provides non-invasive performance isolation through fine-grained resource control. PerfCloud does not require time-consuming workload profiling, or intrusive modification of the application framework and the operating system. Experimental evaluation shows that PerfCloud outperforms the state-of-the-art techniques, LATE and Dolly, in achieving performance isolation and resource utilization efficiency of representative MapReduce and Spark benchmarks.

VII. Acknowledgments

This research was supported in part by U.S. NSF research grants CNS-1422119 and CNS-1649502. Results presented in this paper were obtained using the Chameleon testbed supported by NSF.

References

[1] X. Bu, J. Rao, and C.-Z. Xu, "Interference and locality-aware task scheduling for mapreduce applications in virtual clusters," in *Proc. ACM Int'l Symp. on High-Performance Parallel and Distributed Computing (HPDC)*, 2013.

[2] R. C. Chiang and H. H. Huang, "TRACON: interference-aware scheduling for data-intensive applications in virtualized environments," in *Proc. Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011.

[3] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. N. Vijaykumar, "Tarazu: optimizing mapreduce on heterogeneous clusters," in *Proc. Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012.

[4] R. Gandhi, D. Xie, and Y. C. Hu, "PIKACHU: How to rebalance load in optimizing mapreduce on heterogeneous clusters," in *Proc. USENIX Conf. on Annual Technical Conf. (ATC)*, 2013.

[5] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Proc. USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, 2008.

[6] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones," in *Proc. USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, 2013.

[7] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou, "Apollo: Scalable and coordinated scheduling for cloud-scale computing," in *Proc. of the USENIX Conf. on Operating Systems Design and Implementation (NSDI)*, 2014.

[8] W. Chen, J. Rao, and X. Zhou, "Preemptive, low latency datacenter scheduling via lightweight virtualization," in *Proc. USENIX Conf. on Annual Technical Conf. (ATC)*, 2017.

[9] P. Delgado, F. Dinu, A.-M. Kermarrec, and W. Zwaenepoel, "Hawk: Hybrid datacenter scheduling," in *Proc. USENIX Conf. on Annual Technical Conf. (ATC)*, 2015.

[10] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," in *Proc. Int'l Conf. on Architecture Support for Programming Language and Operating System (ASPLOS)*, 2013.

[11] I. Gog, M. Schwarzkopf, A. Gleave, R. N. Watson, and S. Hand, "Firmament: Fast, centralized cluster scheduling at scale," in *Proc. USENIX Symp. on Operating System Design and Implementation (OSDI)*, 2016.

[12] A. K. Maji, S. Mitra, B. Zhou, S. Bagchi, and A. Verma, "Mitigating interference in cloud services by middleware reconfiguration," in *Proc. ACM/IFIP/USENIX Int'l Conf. on Middleware (Middleware)*, 2014.

[13] D. Novaković, N. Vasić, S. Novaković, D. Kostić, and R. Bianchini, "Deepdive: Transparently identifying and managing performance interference in virtualized environments," in *Proc. of the USENIX Conf. on Annual Technical Conf. (ATC)*, 2013.

[14] W. Chen, J. Rao, and X. Zhou, "Addressing performance heterogeneity in mapreduce clusters with elastic tasks," in *Proc. IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS)*, 2017.

[15] A. Harlap, H. Cui, W. Dai, J. Wei, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing, "Addressing the straggler problem for iterative convergent parallel ml." in *Proc. ACM Symp. on Cloud Computing (SoCC)*, 2016.

[16] J. Mace, P. Bodik, R. Fonseca, and M. Musuvathi, "Retro: Targeted resource management in multi-tenant distributed systems." in *Proc. USENIX Conf. on Networked Systems Design and Implementation (NSDI)*, 2015.

[17] X. Wang, S. Chen, J. Setter, and J. F. Martínez, "Swap: Effective fine-grain management of shared last-level caches with minimum hardware support," in *Proc. Int'l Symp. on High-Performance Computer Architecture (HPCA)*, 2017.

[18] N. J. Yadwadkar, G. Ananthanarayanan, and R. Katz, "Wrangler: Predictable and faster jobs using fewer resources," in *Proc. ACM Symp. on Cloud Computing (SoCC)*, 2014.

[19] A. Ros, L. Y. Chen, and W. Binder, "Understanding the dark side of big data clusters: An analysis beyond failures," in *Proc. of IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN)*, 2015.

[20] X. Ren, G. Ananthanarayanan, A. Wierman, and M. Yu, "Hopper: Decentralized speculation-aware cluster scheduling at scale," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, 2015.

[21] S. Ha, I. Rhee, and L. Xu, "Cubic: A new tcp-friendly high-speed tcp variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, 2008.

[22] "PUMA: Purdue mapreduce benchmark suite," http://web.ics.purdue.edu/ fahmad/benchmarks.htm.

[23] M. Li, J. Tan, Y. Wang, L. Zhang, and V. Salapura, "Sparkbench: A comprehensive benchmarking suite for in memory data analytic platform spark," in *Proc. ACM Int'l Conf. on Computing Frontiers (CF)*, 2015.

[24] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," in *Proc. Int'l Conf. on Architecture Support for Programming Language and Operating System (ASPLOS)*, 2010.

[25] J. D. McCalpin, "Stream benchmark," *Link: www. cs. virginia. edu/stream/ref. html# what*, 1995.

[26] P. Turner, B. B. Rao, and N. Rao, "Cpu bandwidth control for cfs," in *Linux Symp.*, vol. 10, 2010, pp. 245–254.

[27] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing–degrees, models, and applications," *ACM Computing Surveys*, vol. 40, no. 7, pp. 1–28, 2008.

[28] S. Eranian, "perfmon2: the hardware-based performance monitoring interface for linux," http://perfmon2.sourceforge.net/.

[29] C. Delimitrou and C. Kozyrakis, "Quality-of-service-aware scheduling in heterogeneous data centers with paragon," *IEEE Micro*, vol. 34, no. 3, pp. 17–30, 2014.

[30] P. Lama and X. Zhou, "NINEPIN: Non-invasive and energy efficient performance isolation in virtualized servers," in *Proc. of the IEEE/IFIP DSN*, 2012, pp. 1–12.

[31] P. Lama, Y. Guo, C. Jiang, and X. Zhou, "Autonomic performance and power control for co-located web applications in virtualized datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1289–1302, 2016.

[32] J. Leverich and C. Kozyrakis, "Reconciling high server utilization and sub-millisecond quality-of-service," in *Proc. European Conf. on Computer Systems (EuroSys)*, 2014.

[33] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: improving resource efficiency at scale," in *Proc. Int'l Symp. on Computer Architecture (ISCA)*, 2015.

[34] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes, "Cpi2: Cpu performance isolation for shared compute clusters," in *Proc. of the ACM European Conf. on Computer Systems (EuroSys)*, 2013.

[35] H. Zhu and M. Erez, "Dirigent: Enforcing qos for latency-critical tasks on shared multicore systems," in *Proc. Int'l Conf. on Architecture Support for Programming Language and Operating System (ASPLOS)*, 2016.

[36] G. Ananthanarayanan, M. C. Hung, X. Ren, I. Stoica, A. Wierman, and M. Yu, "Grass: Trimming stragglers in approximation analytics," in *Proc. USENIX Conf. on Networked Systems Design and Implementation (NSDI)*, 2014.