Estimating Effective Slowdown of Tasks in Energy-Efficient Cloud Systems

Altino M. Sampaio
CIICESI, Escola Superior de Tecnologia e Gestão de Felgueiras, Instituto Politécnico do Porto
Felgueiras, Portugal
Email: ams@estgf.ipp.pt

Jorge G. Barbosa
LIACC, Departamento de Engenharia Informática
Faculdade de Engenharia, Universidade do Porto
Porto, Portugal
Email: jbarbosa@fe.up.pt

***Abstract*—Consolidation consists in scheduling multiple virtual machines onto fewer servers in order to improve resource utilization and to reduce operational costs due to power consumption. However, virtualization technologies do not offer performance isolation, causing applications' slowdown. In this work, we propose a performance enforcing mechanism, composed of a slowdown estimator, and a interference- and power-aware scheduling algorithm. The slowdown estimator determines, based on noisy slowdown data samples obtained from state-of-the-art slowdown meters, if tasks will complete within their deadlines, invoking the scheduling algorithm if needed. When invoked, the scheduling algorithm builds performance and power aware virtual clusters to successfully execute the tasks. We conduct simulations injecting synthetic jobs which characteristics follow the last version of the Google Cloud tracelogs. The results indicate that our strategy can be efficiently integrated with state-of-the-art slowdown meters to fulfil contracted SLAs in real-world environments, while reducing operational costs in about 12%.**

***Keywords*-Kalman filter; virtualization; energy-efficiency; quality of service; performance interference.**

I. Introduction

Virtualization is an essential technology in cloud computing, because it renders flexible and scalable system services, creating a powerful computing environment. Cloud platforms employ virtualization to encapsulate workloads in multiple isolated execution environments through creating multiple virtual machines (VMs), and consolidate them on shared hardware platforms. Cloud computing is progressively being adopted in different scenarios, such as scientific computation and data analysis experiments [16]. Scientific applications and experiments are usually composed of a certain number of tasks, with various computation and data needs. By using cloud based technologies, scientists can have easy access to large distributed infrastructures, with the ability to scale up and down the computing resources according to the applications needs.

Despite the use of virtualization as a manner to guarantee manageability and availability properties, remains, though, big challenges such as reduce power consumption and guarantee service level agreements (SLAs), which is a key element to support and empower quality-of-service (QoS). However, guarantee the expected QoS for applications, and reduce the energy consumption at the same time, is not trivial due to QoS deviations. The cause of such deviations from expected performance relies in technological limitations and diverse scheduling policies. Regarding the former reason, virtualization does not guarantee performance isolation between VMs [10], [11], [12], meaning that applications performance can change due to the existence of other co-resident VMs sharing the underlying hardware resources. This phenomenon is known as performance interference. The later reason involves specific objectives such as reduction of operational costs, by scheduling multiple VMs onto fewer servers in order to minimize power consumption. However, optimizing energy efficiency through consolidation of VMs is challenging since it can degrade applications' performance. In the specific case of deadline-driven applications, users reserving a certain quantity of resources capacity cannot unfortunately expect that an application finishes in a specified amount of time (reserving a quantity of resources is not equal to achieving an application QoS).

In this paper, we propose to dynamically manage the execution of CPU-bounded tasks, deployed in a private cloud infrastructure. Additionally, co-located VMs suffer from performance interference due to contention in on-chip resources. Our strategy aims at improving energy efficiency, while maximizing the rate of completed jobs by their deadlines. This work represents an extension of [2], and the main contributions are: (i) a mechanism to determine the effective slowdown of tasks, based on noisy data samples provided by state-of-the-art slowdown meters; (ii) a scheduling algorithm that constructs interference- and power-aware virtual clusters. The aim is to build virtual clusters able to successfully accomplish applications QoS requirements (execute the tasks by their deadlines), with less energy consumption.

The remainder of this paper is organized as follows. Section II discusses related work. Section III details targeted scenario. In section IV, the mathematical formulation of the problem is established, as well the proposed mechanism to detect slowdowns and schedule tasks. Section V defines the evaluation metrics, characterizes the workloads utilized, and sets up the simulation scenario. Section VI discusses results. Section VII concludes this paper and introduces future research directions.

II. Related Work

There has been intensive work on quantifying and predicting effects of interference from on-chip resources (e.g. CPU, last-level-cache (LLC), and memory bandwidth), and optimization of energy-efficiency. Govindan et al. [17] propose Cuanta, a framework for predicting increase in task finish time among consolidated workloads due to shared LLC and memory bandwidth. It implies a profiling phase, by conducting a set of experiments to generate a full interference vector for an application, when executed with any set of $N-1$ applications co-located on the other cores. Authors underline the portability of the proposal, since it does not rely on cache-related hardware counters, and the needless of changing the software stack of the hosting platforms. Results demonstrate that Cuanta is able to predict application performance for a variety of co-location scenarios within 7% of the measured performance. Authors have also illustrated the usefulness of Cuanta for guiding VM consolidation decisions to yield better workload placement decisions for given performance and resource cost objectives, finding the the appropriate energy efficiency and performance trade-off. Our work differs from this approach in the sense that our slowdown estimator is complementary to Cuanta, removing any noise in its reads, and our proposed scheduling algorithm leverages virtualization mechanisms to mitigate interference effects, by assigning more resources as needed.

Kim et al. [18] propose a performance-maximizing VM scheduler algorithm that deals with performance impact due to contention in shared LLC. The aim is to identify VM arrangement that minimizes the performance interferences and the number of active servers, thus being possible to reduce energy consumption by turning off redundant servers. Authors propose a scheduling algorithm that co-locates a VM that has large cache demand with a VM that has small cache demand, in order to improve overall performance. Cache demand for a VM can be determined when it is running, without the need of prior profiling of workloads. The results have shown a decrease in average performance degradation ratio in about 16%. Even though, average performance degradation ratio remains above 14%. Again, our work differs from this approach in the sense that our proposal deals with performance mitigation disregarding the size of cache demand for co-hosted applications, whilst optimizes energy-efficiency. In this sense, our proposal can reduce average performance degradation ratio far below 14% achieved by these authors' proposal.

Zhu et al. developed pSciMapper [19] to consolidate scientific workloads with dissimilar requirements, focusing on the interference between the resource requirements, and reduction of total power consumption with lower execution slowdown. Authors start by investigating how the resource usage impacts the total power consumption, and then they analyse the correlation between performance and power consumption with consolidation of different types of workloads. Based on observations gained from experiments, the consolidation algorithm estimates the consumed power and execution time using the model trained offline, and apply online consolidation algorithm, to search for the optimal VMs to physical machines (PMs) consolidation mapping. Results demonstrated that pSciMapper is able to reduce power consumption by up to 56%, with less than 15% slowdown, with low scheduling overhead. Our work is complementary to pSciMapper, in the way it proposes to efficiently estimate slowdown in execution time of applications by using noisy slowdown samples retrieved from slowdown meters (as the one proposed by Zhu et al.). Besides that, our algorithm dynamically performs VMs to PMs adaptations in order to deal with performance interference and power inefficiencies.

Nathuji et al. [20] proposed Q-Clouds, which utilizes online feedback to build a multi-input multi-output model to capture performance interferences among consolidated CPU-bounded workloads. The framework reacts to performance degradation by adjusting processor allocation for each application based on the required SLA. Through the hypervisor, it sets the cap mechanisms on each VM to control the processing resources an application can use. It works by maintaining free resources to be lately used in the adjustment, without the need to determine the underlying sources of interference. Furthermore, the authors use Q-States to differentiate various levels of SLA so that they can dynamically provision underutilized resources, thereby improving system efficiency. Results show that performance interference is mitigated completely when resources are available, and the use of Q-states improves system utilization by up to 35%. Our proposal goes further and it additionally improves energy-efficiency, with acceptable impact in defined SLA.

Dwyer et al. [14] focused on modelling the degradation from sharing the multicore chip's resources. In this regarding, authors have developed an interesting machine learning model that estimates the slowdown of consolidated HPC applications with previously unseen workloads. The slowdown can be estimated inexpensively online, without requiring running applications in isolation or perturb their execution. By using machine learning approach, authors could discover complex relationships between a variety of factors and filter out the factors that are not important, allowing the methodology to be seamlessly ported between two machines with different hardware counters, cache and system bus architectures. Authors evaluated their model on an HPC cluster running scientific workloads, with an algorithm that improves performance, but uses more energy than a Best-Fit approach. Our work is complementary in the sense that it can be applied to decision making under uncertainty due to noisy slowdown samples. Moreover, our scheduling algorithm leverages dynamic update of Xen credit scheduler cap parameter to improve performance and energy efficiency.

III. System Overview

We consider a typical private cloud computing infrastructure, with homogeneous PMs having the same CPU C and memory M capacities, network bandwidth N, and equal access to a shared storage space S for storing VMs images. A job j submitted to the Cloud is a set of independent CPU-intensive workload tasks expressed in Mflop, $j = \{t_1, ..., t_n\}$. As studied by [10], [11], [12], co-hosted CPU-bounded tasks are subject of slowdown due interference caused by sharing of on-chip resources (e.g. CPU, LLC, and memory bandwidth). A VM encapsulates the task execution environment. Multiple distinct VMs can be mapped to a single PM, but a VM will run on top of one PM at a time. Each job has a completion deadline, which equals the deadline of its longest task. Jobs deadlines start counting as soon as cloud users submit them. A job is scheduled if a VM can be assigned to each one of its tasks. Unscheduled jobs are postponed for later scheduling. When a task completes, or reaches its deadline, the cloud manager collects it.

IV. Problem Formulation

This section presents the mathematical basis of this multi-objective problem to schedule the tasks.

A. Power-Efficiency Objective

Based on [6], it works by estimating the power consumption P_i for a PM i, through the linear power model in (1).

$$P_i = p1 + p2 \times CPU_i \tag{1}$$

where CPU_i is the usage of CPU in PM i and varies within [0,1], $p1$ is the power consumption when PM is idle, and $p2$ is the power consumption due to CPU usage, which is proportional to the overall system load [3]. Equation (2) defines the power-efficiency E_{P_i}, and reflects how much useful work is produced under the power consumption [3].

$$E_{P_i} = \frac{CPU_i}{p1 + p2 \times CPU_i} \times (p1 + p2) \tag{2}$$

increases monotonically with CPU usage, reaching 1 when CPU usage is 1. For a PM i, if γ samples of CPU usage are detected below specified CPU threshold τ within the sliding window, then the scheduling algorithm is invoked to optimize the system overall power-efficiency $\overline{E_P}$, expressed by (3).

$$\overline{E_P} = \frac{\sum_{s=1}^{f} \frac{\sum_{i=1}^{u} E_{P_i}}{u}}{f}, \ \forall u \leq h \tag{3}$$

where u is the number of active PMs at sample s, f is the number of samples taken along the simulation, and h is the total number of PMs composing the infrastructure. By efficiently consolidating VMs, idle PMs can be switched to sleep mode, so as to reduce power consumption [5].

B. Performance Interference Mitigation Objective

This work extends our previous work [2] by considering the slowdown $\beta_i(t)$ experienced by task t in node i due to contention in shared on-chip resources, as expressed in (4).

$$\beta_i(t) = \frac{\frac{W(t,\mu)}{\widetilde{r_i}(t)} - \frac{W(t,\mu)}{r_i(t)}}{\frac{W(t,\mu)}{r_i(t)}} + \varepsilon \tag{4}$$

where $W(t, \mu)$ is the remaining workload (in Mflop) of task t at instant μ, and $r_i(t)$ and $\widetilde{r_i}(t)$ are the expected and effectively obtained CPU resources (in Mflops) assigned to the task, respectively. Thus, $\beta_i(t)$ can be negative (i.e. task is executing faster) or positive, and the same applies to prediction error ε (i.e. average error in slowdown samples plus random fluctuations). The scheduling algorithm is invoked if $\beta_i(t)$ prevents task t to finish by its deadline.

C. Performance Interference Detection Logic

This section describes the proposed mechanism to predict the execution time of tasks with performance deviations.

1) Kalman Filter (KF) Noise Removal: The simple Kalman filter [7] is a powerful estimator, that has the ability to smooth noisy data and to provide reliable estimates of signals affected by indirect, inaccurate and uncertain observations (e.g. Gaussian noise). It estimates a parameter with two steps, namely time update (predict) and measurement update (correct). In the time update step, the filter projects the current state estimate ahead in time, while the measurement update adjusts the projected estimate by an actual measurement at that time, so as to obtain an improved estimate. The mathematical formulation is presented along (5)–(9). In particular: the state prediction (i.e. predicts the new transitory slowdown ahead) is expressed by (5), and the covariance prediction (i.e. estimates how much error ahead) is represented by (6). These two equations convey the discrete Kalman filter time update.

$$\hat{\beta}_k = A\beta_{k-1} + Bu_k \tag{5}$$

$$\hat{P}_k = AP_{k-1}A^T + Q \tag{6}$$

Equation (7) defines the Kalman gain to moderate the prediction process, while (8) updates the new state (i.e. the new final slowdown) estimate with the last measurement. In turn, (9) updates the new final error estimate. These three equations together express the discrete Kalman filter measurement update:

$$K = \hat{P}_k H^T (H\hat{P}_k H^T + R)^{-1} \tag{7}$$

$$\beta_k = \hat{\beta}_k + K(z_k - H\hat{\beta}_k) \tag{8}$$

$$P_k = (I - KH)\hat{P}_k \tag{9}$$

2) Linear Regression-Based (LR) Completion Estimator: Regression analysis [8] is a common used statistical procedure to model relationships between variables. Its purpose is to predict dependent variable Y on the basis of explanatory variable X. Linear Regression Analysis (LRA) represents a simple form of regression analysis, and can be applied when the dependent variable is a linear function of explanatory variables. So, a straight line relationship fits the data satisfactory, in which $Y = aX + b$ is the estimated line. The best parameters a and b can be determined using algorithms such as Least Mean Squares (LMS). The a parameter represents the slope, and b is the intercept.

3) KFLR: The proposed mechanism to estimate if a executing task finishes before it's deadline is shown in Fig. 1 and leverages the KF and LR. KF is used to eliminate (Gaussian distributed) unpredictable disturbances in the slowdown estimates of running tasks. Slowdown estimates are provided by state-of-the-art tools [14], at constant sampling intervals. Then, the filtered slowdown samples are used as input of the LR estimator, so as to predict the tasks completion time, with more precision. The LR determines the remaining workload for the last ψ sampling intervals based on filtered ψ slowdown estimates and measured CPU used by the task. These ψ remaining workload estimates are then used to fill Y axis, and the ψ instants in which these estimates are observed is represented in the X axis. Then, based on the LRA equation, we calculate the value of X_t for $Y = 0$ (i.e. no remaining workload to process): $X_t = -b/a$. Basically, X_t represents the time in which task t is predicted to accomplish. If $X_t > t_{deadline}$ then an adaptation of the VMs to PMs mapping is triggered.

D. Scheduling Algorithm

The presented algorithm is based on [2] to deal with the slowdown of applications. We assume that each task t requires a minimum of resources, $min\ r(t)$, defined as the ratio of its workload $W(t, \mu)$ to the deadline $d(t_n)$, as well as a maximum amount of resources, $max\ r(t)$, to execute at maximum speed and which value is defined by the user. Tasks are to be scheduled in this priority, according to their reason of submission: (i) under performance; (ii) new tasks; (iii) power-inefficient PM. Reason (iii) is handled with migration, and (i) is solved with expansion, or migration in case of expansion failure due to scarcity of resources. Strategy followed by the scheduler is guided by (10)–(13).

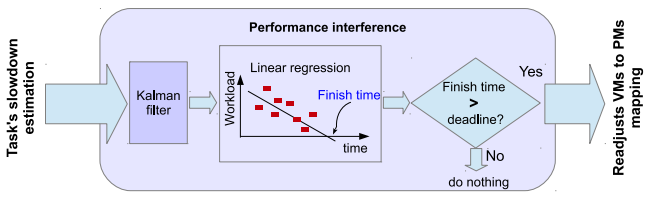


Figure 1: Performance interference detection block.

$$\beta = \begin{cases} \bar{\beta}_i \quad if\ task\ t\ is\ new \\ \beta_i(t) \quad if\ expanding\ task \\ \begin{cases} \beta_i(t) \quad if\ \beta_i(t) > \bar{\beta}_i \\ \bar{\beta}_i \quad otherwise \end{cases} if\ migrating\ task \end{cases},$$

$$where \quad \bar{\beta}_i = \frac{\sum_{k=1}^{n-1} \beta_i(t_k)}{n-1} \tag{10}$$

is the slowdown to use in (11)–(13), and is considered in assignment of physical resources to tasks. Its value depends on the reason why the algorithm is invoked. If we are scheduling a task for the very first time, then $\beta = \bar{\beta}_i$ observed in each candidate PM. The slowdown $\bar{\beta}_i$ is calculated based on slowdown experienced by the $n-1$ co-located tasks in PM i. For expansion, the slowdown considered is task's $\beta_i(t)$. The policy applied in case of migration is to choose the biggest of the two: (i) slowdown experienced by the task, or (ii) average slowdown in each candidate PM. Knowing β, we can now estimate the additional amount of resources (in terms of CPU) $\Delta r_i(t_n, \beta)$ to assign to task t_n to absorb the slowdown.

$$\Delta r_i(t_n, \beta) = \frac{W(t_n, \mu) \times \beta}{d(t_n) - \mu - m_{ei}} \tag{11}$$

where m_{ei} defines the migration time in case of migration. Selection of PMs to execute the tasks is ruled by (12)–(13).

$$r_i(t_n) + \Delta r_i(t_n, \beta) + \sum_{k=1}^{n-1} r_i(t_k) \leq C_i \tag{12}$$

imposes that a PM i is candidate to host a task t_n at time μ, if the amount of resources required by that task, increased by a supplementary amount of resources which depends on slowdown β, plus the overall resources required by all co-hosted tasks $\{t_1, ..., t_{n-1}\}$, do not exceed node's capacity C_i. For new, and expanding tasks, $m_{ei} = 0$.

$$\begin{cases} (r_i(t_n) + \Delta r_i(t_n, \beta)) \times (d(t_n) - \mu - m_{ei}) \geq W(t_n, \mu), \\ r_i(t_n) \leq max\ r(t_n) \end{cases} \tag{13}$$

defines that a task t_n can be scheduled, or migrated, to a PM i at time $\mu + m_{ei}$, if: (i) it provides the minimum resources $r_i(t_n) + \Delta r_i(t_n, \beta)$ required by the task, during

the $d(t_n) - \mu - m_{ei}$ time interval; (ii) and required $r_i(t_n)$ is less than $max\ r(t_n)$ to execute by its deadline.

The pseudo-code for Performance- and Power-Aware Relaxed Time Task Execution (PP-RTTE) strategy is shown in Algorithm 1. At line 3, it eliminates tasks with negative slack time (i.e. difference between $d(t_n)$ and minimum execution time), and, for each reason (line 4), it sorts remaining ones in the ascending order according to slack time. Next, the algorithm picks up the first task in the list (priority reason, and least slack time), and proceeds according to the reason of scheduling. A task is scheduled in the PM that fulfils (12)–(13) and improves more (2). In the end, the algorithm applies the cap parameter from the Xen credit scheduler [15] to assign tasks the strict necessary amount of CPU to complete within their deadlines. During execution, cap parameter can be dynamically updated, in order to explore the available fraction of CPU in the physical server. For a task under performance, line 7 tries to assign additional resources. If current PM does not provide enough resources, the adaptation process reverts to a migration (line 8). If migration is infeasible due to time constraints, line 16 simulates migration away of co-located task with the higher slack time (line 11) so as to release resources. Line 17 verifies if resizing of the VM is now possible. If it is, the algorithm saves the new configuration (line 18), and continues to the next task in the list (line 5). Otherwise, it repeats the cycle at line 10. Job queue is updated as migrations occur. Jobs unscheduled due to scarcity of resources are postponed.

Algorithm 1 PP-RTTE Scheduling

```
1:  function PP-RTTE(pmList, jobList)
2:      map ← NULL
3:      jobList.taskList.removeNegativeSlack()
4:      jobList.taskList.sortTasksInReasonBySlack()
5:      for all task ∈ jobList.taskList do
6:          if task.reasonIsUnderPerformance() then
7:              if task.tryExpanding() = FALSE then
8:                  tmap.add(getPeffPM(pmList, task))
9:              end if
10:             while tmap.taskNotFound(task) do
11:                 tt ← task.getNextCohostedTask()
12:                 if tt == NULL then
13:                     tmap ← NULL
14:                     break
15:                 end if
16:                 tmap.add(getPeffPM(pmList, tt))
17:                 if task.tryExpanding() = TRUE then
18:                     map.add(tmap)
19:                     break
20:                 end if
21:             end while
22:         else
23:             map.add(getPeffPM(pmList, task))
24:         end if
25:         jobList.taskList.removeTasks(map)
26:     end for
27: end function
```

V. Characterization of the Simulation Scenario

This section presents the metrics used to assess algorithms performance, and describes workloads characteristics.

A. Performance Metrics

We apply three metrics, namely: (i) completion rate of users' jobs (14); (ii) the ratio of useful Mflop processed to the energy consumption (15); (iii) and working efficiency (16).

$$R_J = \frac{J_C}{J_S} \tag{14}$$

measures the completion rate of jobs, as the ratio of completed jobs J_C with respect to the number of submitted jobs J_S. A job is considered completed if all its tasks are finished by their deadlines.

$$E_M = \frac{\sum_{j=1}^{x} \theta_j \times \sum_{t=1}^{n} W_t}{\frac{\sum_{s=1}^{f} \frac{\sum_{i=1}^{u} P_i}{u}}{\cancel{f}} \times \cancel{f} \times 60}, \theta_j = \begin{cases} 1, \ if\ completed \\ 0, \ otherwise \end{cases} \tag{15}$$

is the amount of energy consumed, in Joule, to produce useful work. It is calculated by dividing the sum of all tasks' workloads, of successfully completed jobs, by the overall energy consumption, which is calculated by multiplying the average power consumption (i.e. for all active physical nodes u, at all sample times f), with the number of sample times f, times 60 (since samples are taken per minute).

$$E_W = \frac{\sum_{s=1}^{f} \frac{\sum_{i=1}^{u} E_{P_i}}{u}}{f} \times R_J, \ \forall u \leq h \tag{16}$$

quantifies the useful work done (i.e. R_J) by the consumed power. It is determined by multiplying (14) with (3), for all active physical nodes u, at all sample times f. Maximizing the working efficiency implies to diminish the power consumption and to increase the number of completed jobs.

B. Workloads Characteristics

We have created a set of 3614 synthetic jobs, and a total of 10357 tasks, which characteristics follow the last version of Google Cloud tracelogs [1]. Basically, 75% of jobs run one task only, and most of the jobs have less than 28. Jobs inter-arrival time is 4 seconds, each with a medium length of 3 minutes. The majority of jobs run in less than 15 minutes. Moreover, the jobs inter-arrival time, tasks length, tasks CPU capacity requirements, and number of tasks per job, are modelled using a Lognormal distribution. CPU usage per

task varies from near 0% to around 25%. Most of the tasks use less than 2.5% of the node's RAM. VMs are randomly assigned with RAM sizes of 256, 512 and 1024 MB. Each task deadline is rounded up to 10% more of its minimum necessary execution time.

C. Simulation Setup

1) System Parameters: We have simulated a cloud infrastructure composed of 50 homogeneous PMs. The CPU capacity of PMs was considered to be 800 Mflops. The consuming power of fully loaded physical nodes was considered to be 250 W. The $p1$ and $p2$ parameters in (1) were set up to 70% and 30% of full power consumption, respectively. VMs migration overhead ($\{8, 10, 12\}$ seconds) depend on their assigned RAM. For power-efficiency objective [6], the size of the sliding-window was set to 5 minutes, each sample taken at 1 minute intervals, and with $\tau = 0.5$ and threshold $\gamma = 3$. Fluctuations in slowdown samples provided by upstream slowdown meter followed a normal distribution. For KF, we assumed a constant value for process noise covariance $Q = 5 \times 10^{-4}$ and measurement noise covariance $R = 0.2^2$. The line for LR is determined based on $\psi = 2$ samples. Less number of points leads to slowdown estimator to react faster. The simulated amplitude of interference can vary from almost 0% to about 50%, and differs for each VM depending on the number of co-hosted VMs, and according to a randomly generated value that follows a normal distribution.

2) Comparative Slowdown Detection Mechanisms: We have implemented three other common mechanisms for comparison purposes. For all mechanisms, analysis of slowdown samples is performed over a sliding window containing 5 samples per minute, taken at constant intervals. The threshold was set up to 10% more of tasks minimum necessary execution time. For the first one, Simple ThresholD (STD), VM adaptation is triggered if 3 slowdown data samples exceed the specified threshold. The second mechanism, Average ThresholD (ATD), determines first the average slowdown from samples within the window, and triggers a condition if the average value exceeds the specified threshold. The third approach, Extended ThresholD (ETD), calculates the average slowdown from samples within 3 consecutive windows (each window differing from the prior one in one sample), and invokes the scheduling algorithm if the average result exceeds the mentioned threshold.

3) Comparative Scheduling Algorithms: We have implemented two other algorithms for comparison purposes, namely: (i) Optimistic Best-Fit (OBFIT) [4]; and (ii) Relaxed Time Task Execution (RTTE) [2]. Due to constraints of space, algorithms will not be explained here. They do not consider tasks' slowdown.

VI. Results and Analysis

This section discusses the performance results for the various simulations. First, we compare the algorithms' performance in the absence of slowdown, and then we applying performance interference among co-hosted tasks. Last section combines proposed KFLR with scheduling algorithm.

A. Performance of the Scheduling Algorithms

In Table I is shown the results for algorithms' performance, in the absence of slowdown, but considering power-efficiency optimization. Results show that PP-RTTE and RTTE algorithms behave equal. PP-RTTE achieves the best completion rate of jobs, together with OBFIT, but improves more the energy efficiency and working efficiency. This means that PP-RTTE strategy can produce the same work as OBFIT, but for less energy. The reason is that PP-RTTE achieves higher levels of consolidation (VMs / PM) due to the use of cap parameter from the Xen credit scheduler, running more VMs in each PM, thus consuming less energy.

In table II is registered the results for same scheduling algorithms, but applying performance interference. PP-RTTE algorithm is combined with the KFLR slowdown estimator to deal with possible slowdown situations. The upstream slowdown meter provides slowdown data samples with an average slowdown error of 10%, and standard deviation of 20%. As observed, the two algorithms that do not consider the slowdown due to performance interference degrade substantially. In particular, RTTE degrades more due to high levels of consolidation. In turn, PP-RTTE strategy is able to maintain similar levels of completion rate of jobs, outperforming OBFIT and RTTE algorithms in more than 158%. Moreover, the number of active physical servers increased compared to Table I, which is understandable if we recall that PP-RTTE assigns additional CPU resources to VMs in order to maintain the required levels of performance. Furthermore, the number of running VMs per time unit increased as well, since tasks execute for longer and new tasks continue being submitted. More tasks means more opportunities to improve power-efficiency (higher level of

Table I: Performance of scheduling algorithms, without slowdown (with power optimization).

Algorithm	$R_J(\%)$	E_M	$E_W(\%)$	VMs/PM
OBFIT	100.00	0.0524	73.57	14.51
RTTE	100.00	0.0531	74.82	15.70
PP-RTTE	100.00	0.0531	74.82	15.70

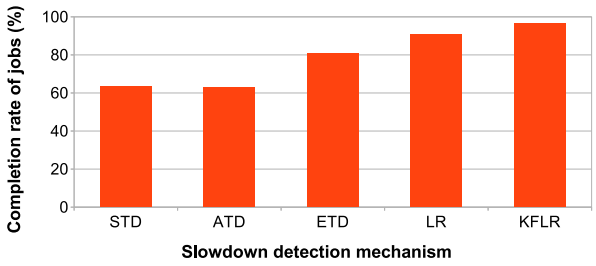
Table II: Performance of scheduling algorithms, with interference among co-hosted tasks (noisy slowdown samples with average error of 10%, and standard deviation of 20%).

Algorithm	$R_J(\%)$	E_M	$E_W(\%)$	VMs/PM
OBFIT	37.43	0.0027	24.58	13.60
RTTE	12.25	0.0012	9.21	15.94
PP-RTTE	96.49	0.0368	75.27	16.13

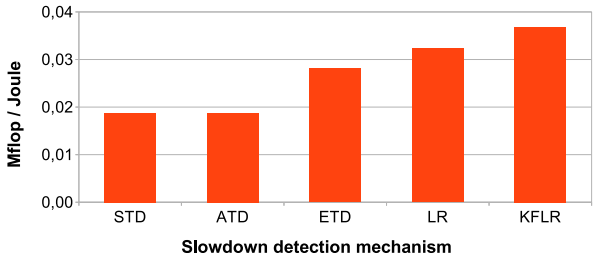
consolidation), which explains why working efficiency E_W remains almost unchanged. Despite the higher level of consolidation, which induces more interference, it is interesting to observe that PP-RTTE can even though complete more jobs. This explain why PP-RTTE is capable of doing more work for less energy, as evidenced by E_M and E_W.

B. Performance of the Slowdown Estimators

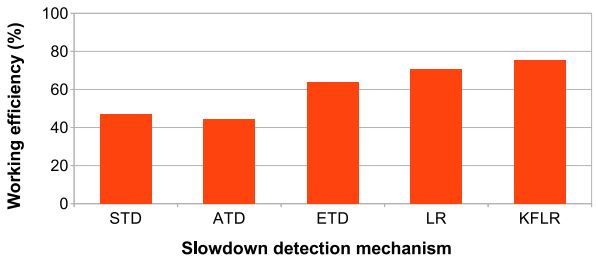
In Fig. 2 is depicted a comparison of performance achieved by tested slowdown estimators. Simulations were carried out by combining different slowdown detector alternatives with PP-RTTE algorithm, which performs better as seen in section VI-A. Upstream slowdown meter provides slowdown samples with an average slowdown error of 10%, and standard deviation of 20%. In Fig. 2(a) we see that KFLR completes more than 96% of jobs, which represents about 7% more relatively to LR, and about 52% more than so common applied STD method. Both STD and ATD methods seem to respond quickly to threshold violations, since they trigger almost 50% more of VMs migrations, leading to inferior rate of completed jobs due to migration overhead. Moreover, they consume more energy than all the other alternatives as well. In turn, ETD seem to react too late, triggering VMs migrations when tasks are no longer capable of complete by their deadlines. As expected, KFLR achieves superior performance than all the other estimator techniques. Comparing this technique with the specific case of using only LR, the better results for KFLR comes from the removal of noise from slowdown samples before its use from LR. Moreover, KFLR reduces the average slowdown estimated error in 25% compared to LR. Fig. 2(b) and Fig. 2(c) quantify the useful work done by the consumed energy and power, respectively. Again, we observe that KFLR is the estimator that performs better, delivering more work, for less energy consumption. Regarding other estimators, this property suggests that KFLR provides higher confidence levels, taking better decisions despite the noisy data, thus completing more jobs and consuming equal or less energy.



(a) Completion rate of users jobs.



(b) Energy efficiency.



(c) Working efficiency.

Figure 2: Comparison among slowdown detection mechanisms, applied to PP-RTTE scheduling algorithm (noisy slowdown samples with average error of 10%, and standard deviation of 20%).

C. Combining PP-RTTE scheduler and KFLR estimator

The slowdown meter proposed by Dwyer et al. [14] is referred as having an Average Slowdown Error (ASE) of 10%, which is the value we have used so far. Other researchers propose alternatives, with different ASEs [11]. We now evaluate the performance of PP-RTTE combined with KFLR, while varying AES in samples provided by the upstream slowdown meter. Table III shows the evolution for all considered metrics, average estimated system slowdown $\bar{\beta}$, and average Estimated Error (EE), this last determined by the difference between real and estimated slowdown. As we could expect, the higher the uncertainty in slowdown data samples provided by upstream slowdown meters, the more difficult to take the right decision (EE reaches higher values as uncertainty increases). This explains why the completion rate of users jobs decreases, with consequent negative impact in energy efficiency and working efficiency, as ASEs augments. However, an important remark from results is that if we consider that upstream slowdown meters [14] can estimate slowdown of applications with an ASEs of 10%, and that in real-world environments the SLA contract terms allow a mean performance degradation of 1%–5% [9], then our mechanism can be applied effectively.

VII. Conclusions

In this paper, we have proposed a performance enforcing mechanism, composed of a slowdown estimator, and a scheduling algorithm. The slowdown estimator collects noisy slowdown samples provided by general state-of-the-

Table III: Performance of PP-RTTE + KFLR, while varying the average slowdown error (ASE) in data samples.

ASE	$R_J(\%)$	E_M	$E_W(\%)$	$\bar{\beta}$	$EE(\%)$
5%	98.70	0.0397	77.58	16.39	4.36
10%	96.49	0.0368	75.27	19.50	4.86
15%	93.53	0.0339	71.67	23.09	4.50
20%	84.45	0.0252	61.91	26.30	5.24
25%	70.98	0.0192	55.26	29.70	13.21

art upstream slowdown meters, and invokes the scheduling algorithm to handle tasks predicted to complete beyond their deadlines. Considering that upstream slowdown meters [14] can estimate slowdown of applications with an average error of 10%, and that in real-world environments the SLA contract terms allow a mean performance degradation of 1%–5% [9], our PP-RTTE + KFLR proposal can be applied effectively to fulfil expected QoS of applications and reduce energy costs in about 12%. As future work, we intend to extend this proposal to schedule workflows. Moreover, we intend to consider physical servers reliability, which has become an important concern as data centers grow in size and in complexity. Construction of fault-tolerant virtual clusters is crucial to fulfil applications QoS requirements.

REFERENCES

[1] Z. Liu, S. Cho, "Characterizing Machines and Workloads on a Google Cluster," Proc. International Conference on Parallel Processing Workshops (ICPPW), Sept. 2012, pp. 397–403.

[2] A. M. Sampaio, J. G. Barbosa, "Dynamic Power- and Failure-Aware Cloud Resources Allocation for Sets of Independent Tasks," Proc. IEEE International Conference on Cloud Engineering (IC2E), March 2013, pp. 1–10.

[3] J. Xu, and J. Fortes, "A Multi-objective Approach to Virtual Machine Management in Datacenters," Proc. 8th ACM International Conference on Autonomic Computing (ICAC), Jun. 2011, pp. 225–234.

[4] S. Fu, "Failure-aware resource management for high-availability computing clusters with distributed virtual machines," Journal of Parallel and Distributed Computing, vol. 70, April 2010, pp. 384–393.

[5] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Q. Dang, and K. Pentikousis, "Energy-Efficient Cloud Computing," The Computer Journal, vol. 53, no. 7, Aug. 2009, pp. 1045–1051.

[6] A. M. Sampaio, J. G. Barbosa, "Optimizing Energy-Efficiency in High-Available Scientific Cloud Environments," Proc. IEEE International Conference on Cloud and Green Computing (CGC), October 2013, pp. 76–83.

[7] G. Welch, G. Bishop, "An introduction to the Kalman filter," University of North Carolina, North Carolina, 1995.

[8] G. A. Seber, A. J. Lee, "Linear regression analysis", John Wiley & Sons, vol. 936, 2012.

[9] A. Beloglazov, J. Abawajy, R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," Future Generation Computer Systems, vol. 28, no. 5, 2012, pp. 755–768.

[10] Q. Huang, P. P. Lee, "An experimental study of cascading performance interference in a virtualized environment," ACM SIGMETRICS Performance Evaluation Review, vol. 40, no. 4, 2013, pp. 43–52.

[11] Y.Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, C. Pu, "An analysis of performance interference effects in virtual environments," Proc. IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS), April 2007, pp. 200–209.

[12] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, C. Pu, Y. Cao, "Who is your neighbor: Net i/o performance interference in virtualized clouds," IEEE Transactions on Services Computing, vol. 6, no. 3, July-Sept. 2013, pp. 314–329.

[13] R. Nathuji, A. Kansal, A. Ghaffarkhah, "Q-clouds: managing performance interference effects for qos-aware clouds," Proc. 5th European conference on Computer systems, April 2010, pp. 237–250.

[14] T. Dwyer, A. Fedorova, S. Blagodurov, M. Roth, F. Gaud, Jian Pei, "A practical method for estimating performance degradation on multicore processors, and its application to HPC workloads," Proc. International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Nov 2012, pp. 1–112.

[15] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, "Xen and the art of virtualization," Proc. 19th ACM Symposium on Operating Systems Principles (SOSP), 2003.

[16] C. Vecchiola, S. Pandey, R. Buyya, "High-performance cloud computing: A view of scientific applications," Proc. 10th International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN), 2009, pp. 4–16.

[17] S. Govindan, J. Liu, A. Kansal, A. Sivasubramaniam, "Cuanta: quantifying effects of shared on-chip resource interference for consolidated virtual machines," Proc. 2nd ACM Symposium on Cloud Computing, 2011, pp. 22.

[18] S. Kim, H. Eom, H. Y. Yeom, "Virtual machine scheduling for multicores considering effects of shared on-chip last level cache interference," Proc. International Green Computing Conference (IGCC), 2012, pp. 1–6.

[19] Q. Zhu, J. Zhu, A. Jiedan, G. Agrawal, "Power-aware consolidation of scientific workflows in virtualized environments," Proc. ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, 2010, pp. 1–12.

[20] R. Nathuji, A. Kansal, A. Ghaffarkhah, "Q-clouds: managing performance interference effects for qos-aware clouds," Proc. 5th European conference on Computer systems, 2010, pp. 237–250.