Resource Boxing: Converting Realistic Cloud Task Utilization Patterns for Theoretical Scheduling

Bernhard Primas
School of Computing
University of Leeds, UK
scbjp@leeds.ac.uk

Peter Garraghan
School of Computing and Communications
Lancaster University, UK
p.garraghan@lancaster.ac.uk

Karim Djemame
School of Computing
University of Leeds, UK
k.djemame@leeds.ac.uk

Natasha Shakhlevich
School of Computing
University of Leeds, UK
N.Shakhlevich@leeds.ac.uk

ABSTRACT
Scheduling is a core component within distributed systems to determine optimal allocation of tasks within servers. This is challenging within modern Cloud computing systems – comprising millions of tasks executing in thousands of heterogeneous servers. Theoretical scheduling is capable of providing complete and sophisticated algorithms towards a single objective function. However, Cloud computing systems pursue multiple and oftentimes conflicting objectives towards provisioning high levels of performance, availability, reliability and energy-efficiency. As a result, theoretical scheduling for Cloud computing is performed by simplifying assumptions for applicability. This is especially true for task utilization patterns, which fluctuate in practice yet are modelled as piecewise constant in theoretical scheduling models. While there exists work for modelling dynamic Cloud task patterns for evaluating applied scheduling, such models are incompatible with the inputs needed for theoretical scheduling – which require such patterns to be represented as boxes. Presently there exist no methods capable of accurately converting real task patterns derived from empirical data into boxes. This results in a significant gap towards theoreticians understanding and proposing algorithms derived from realistic assumptions towards enhanced Cloud scheduling. This work proposes resource boxing – an approach for automated conversion of realistic task patterns in Cloud computing directly into box-inputs for theoretical scheduling. We propose four resource conversion algorithms capable of accurately representing real task utilization patterns in the form of scheduling boxes. Algorithms were evaluated using production Cloud trace data, demonstrating a difference between real utilization and scheduling boxes less than 5%. We also provide an application for how resource boxing can be exploited to directly translate research from the applied community into the theoretical community.

UCC '16, December 06-09, 2016, Shanghai, China

 ISBN 978-1-4503-4616-0/16/12…$15.00

DOI: http://dx.doi.org/10.1145/2996890.2996897

CCS Concepts
• Theory of computation~Scheduling algorithms • Theory of computation~Data structures design and analysis.

Keywords
Scheduling, task patterns, resource conversion

1. INTRODUCTION
Cloud computing has increasingly become an important component within internet infrastructure, capable of provisioning application service to millions of users globally. These systems composed by hundreds and thousands of inter-connected machines require highly effective scheduling algorithms in order to satisfy Service Level Agreements (SLA) imposed by users. Application scheduling is a critical component in Cloud computing, reflected by a large body of research proposing scheduling algorithms pursuing various objective functions ranging from performance [1][2], dependability [3-5], networking [6][7], and energy-efficiency [8-10].

These algorithms are created and validated through formal proof, simulation, and experimentation. More generally, algorithms can be categorized as stemming from applied and theoretical research communities. Applied algorithms are based on heuristics, realized through a relatively simple decision making (such as round-robin) and evaluated through simulation or experimentation.

Theoretical scheduling has intensely been studied by mathematicians for decades [11][12], providing important contributions to numerous fields such as manufacturing and services [13][14]. Theoretical scheduling studies optimal allocation of boxes (that are typically called jobs) to machines in adherence to an objective function. In terms of computing, boxes and machines are represented as tasks that execute within machines. While it has been demonstrated that important contributions have been made from theoretical scheduling, there exist challenges of their direct dissemination within Cloud computing systems. This is due to sacrificing realistic assumptions of system operation for more complete, yet simplistic models. An important strong assumption that theoretical scheduling models typically impose is that task resource demand can be represented by a piecewise constant function (i.e. boxes) [15][16]. Importantly, boxes are unable to capture the dynamicity of resource utilization patterns inherit within Cloud computing [17]. Failure to capture this behavior results in reduced applicability due to the disconnect between evaluating theoretical scheduling derived from real world operation in modern Cloud computing systems. Such a challenge represents

a significant gap between theoretical and applied scheduling, that are driven by completeness and heuristics, respectively.

As a result, there is a need to accurately map task execution patterns from real Cloud computing systems into the context of theoretical scheduling. In other words, there is a requirement to find an accurate representation of execution patterns that is composed by boxes. Such a technique would therefore allow a direct application of theoretical scheduling algorithms to real execution patterns. This is not currently possible as algorithm inputs for theoretical scheduling are not compatible with resource fluctuation intrinsic to task resource utilization patterns. A significant challenge towards bridging this gap are methods capable of automated conversion of realistic system behavior that are directly understood and applicable within a theoretical context. Such a method would allow for (1) evaluation of sophisticated algorithms with a strong mathematical basis driven by realistic Cloud operation, (2) enable the broader theoretical scheduling community direct access to understanding modern Cloud system behavior. The process of converting task patterns into boxes also poses a number of challenges, which in its current form requires significant manual effort bespoke to a specific studied system. Furthermore, which resource conversion method is the most effective in terms of accurately characterizing task execution with trade-offs of computation and data creation remains unclear.

This paper proposes a method for converting realistic task resource utilization patterns directly into boxes (termed resource boxing) making them directly exploitable for theoretical scheduling. The method is capable of automated conversion irrespective of underlying system architecture, resource type, and task dynamicity. Our contributions are listed as follows:

Identification of the knowledge gap between theoretical and applied scheduling in Cloud computing. This represents a serious endeavor within an unexplored research area towards introducing realistic assumptions from applied scheduling into theory. We provide a number of advantages and disadvantages in Cloud computing scheduling within each respective area, and challenges in addressing this identified gap.

Investigation and evaluation of four approaches for resource boxing. We detail four algorithms for resource boxing using event-based, periodic, and hybrid approaches. We evaluate their respective accuracy and trade-offs from resource boxing of a large-scale production Cloud datacenter. We apply our proposed method within experiments to study the relationship between resource utilization and power.

The paper is structured as follows: Section 2 introduces the background; Section 3 discusses related work; Section 4 details approaches for resource boxing, Section 5 evaluates the resource boxing algorithms; Section 6 details practical application of algorithms; Section 7 provides conclusions and future work.

2. BACKGROUND

Traditional theoretical scheduling focuses on assigning limited resources to tasks with the goal of minimizing a single objective function. In the context of theoretical scheduling, resources are typically referred to as machines and tasks are referred to as jobs. Within this paper we use the term task instead of job in order to avoid confusion with terminology within applied scheduling.

A typical theoretical scheduling problem can be formulated as follows, with [14] providing additional details. We are given n tasks $T_1, \dots, T_n$ and m machines $M_1, \dots, M_m$. For every task T_k, for $1 \leq k \leq n$, there are given a number of problem specific parameters. Typical examples for task parameters include release time r_k denoting the time at which T_k becomes available, or processing time p_k denoting the time that T_k needs to complete. The objective of a scheduler is then to decide for each task

1) to which machine the task should be assigned to, and
2) when the task should start on the selected machine

such that a number of problem specific constraints are satisfied and such that a given objective function is minimized. Examples of constraints include the requirements that tasks which are assigned to the same machine must not overlap (i.e. simultaneous execution), and that each task T_k can only be started at or after its release time r_k. A typical example for an objective is the makespan $C_{\max}$, which denotes the completion time of the task that finishes last. As an example, we consider an instance with five tasks and two machines, with input parameters for tasks as provided in Table 1.

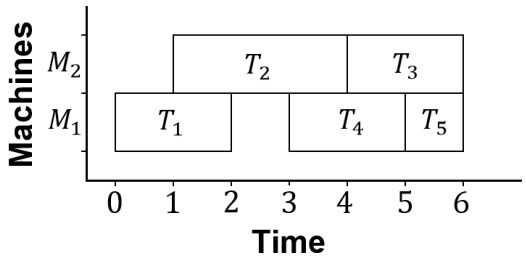


Figure 1. Example Gantt chart of an optimal schedule.

Table 1. Input parameters for an instance with five tasks.

	T_1	T_2	T_3	T_4	T_5
r_k	0	1	0	3	5
p_k	2	3	2	2	1

We consider the makespan as the objective function we want to minimize. Figure 1 illustrates a Gantt chart of an optimal solution for this problem (omitting a formal proof). No tasks overlap since no boxes overlap. Furthermore, it is easily verifiable that each task starts at or after its release time. This optimal solution consists of tasks T_1, T_4 and T_5 being assigned to machine $M_1$and tasks T_2 and T_3 being assigned to machine M_2. Note that interval $[0,1]$ is idle

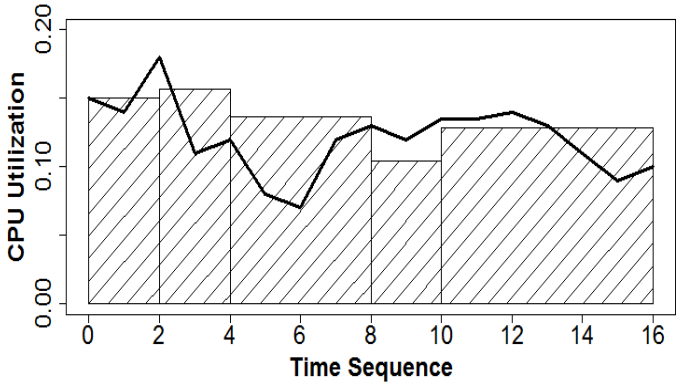


Figure 2. Representation of a task execution pattern by a series of boxes.

within machine M_2 whereas interval [2,3] is idle on machine M_1. The minimized value of the makespan is given by 6.

Applying traditionally theoretical scheduling within the context of Cloud computing systems is challenging for numerous reasons. First, there is only very limited information about the tasks available to the scheduler. For example, the release time is not known in advance since tasks are submitted to the system at random times. Moreover, it is not always clear how long the task execution will require, meaning that the processing time of a task may be unknown. Furthermore, Cloud computing tasks require computational resources such as CPU, memory or disk for execution. Since the resource demand varies over time, it is challenging to accurately represent this behavior using boxes as shown in Figure 2. However, theoretical scheduling algorithm typically require a detailed representation of tasks. This is highlighted within [18] stating that a large body of Cloud task assignment is based on *"a detailed representation of tasks to be executed, but a rather simplistic representation of the hosts"*, thus resulting in evaluation using simulation as opposed to measurements from a real system. As a result, we believe that there is an opportunity to convert Cloud workload patterns into boxes and still capturing realistic system behavior.

3. RELATED WORK

Due to the mostly unexplored nature of this research area, there is limited effort towards converting realistic assumptions of Cloud computing into a purely theoretical context. As a result, the body of work can be categorized into three components: modelling Cloud task patterns, applied scheduling, and theoretical scheduling.

There exist numerous works that attempt to study task patterns within Cloud computing that have been used in order to enhance scheduling. Mishra et al. [19] classifies task execution qualitative boundaries for execution duration and uses *k*-means clustering to construct task classes. Using trace data from 4 Google compute clusters for 4 days they construct eight different classes of tasks, separated by their respective duration and resource utilization.

Kuvulya et al. [20] present a statistical analysis of MapReduce jobs from the M45 supercomputer cluster to ascertain the statistical characteristics of resource utilization and job patterns. They provide details pertaining to number of tasks, completion rate, and average job distribution. They model job completion rate as a Lognormal distribution, and discovered 95% of jobs complete within under 20 minutes.

Solis Moreno et al. [17] propose a method of analyzing and modeling user and task patterns. Their approach is applied to a production Cloud datacenter of over 12,500 servers and 29 days operation. They are able to categorize and capture task resource utilization patterns through statistical analysis and probabilistic distribution functions, demonstrating numerous types of resource usage patterns in Cloud computing validated within CloudSim.

The statistical properties of derived Cloud task patterns have been directly used to construct assumptions and evaluate applied scheduling algorithms [16][21][22]. While these works can be leveraged by the research community to enhance scheduling, derived task utilization patterns are predominately based on coarse-grain statistics or probabilistic models that produce dynamic resource utilization, and thus cannot be readily translated into a box format as inputs for theoretical scheduling.

From the theoretical point of view, there have been numerous scheduling models proposed for distributed systems. Rahman et al. [23] propose a scheduling model for workflow applications. They represent the workflow application as a Directed Acyclic Graph (DAG) in which nodes correspond to tasks and edges to precedence dependencies between the tasks. The goal is to minimize the total estimated cost for running required services while assuring that the application finishes prior to a given deadline.

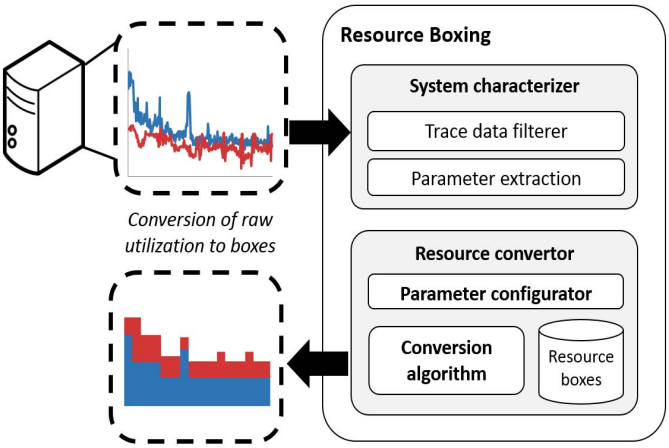


Figure 3. System model of resource boxing.

Yin et al. [24] also consider the problem of assigning application tasks to different processors such that the cost for the system is minimized and constraints limiting resource usage are satisfied. They add a penalty factor to the objective function to avoid solutions with too many tardy tasks. Due to the NP-completeness of the problem, a particle swarm optimization algorithm is presented to find near optimal solutions.

Jiang et al. [15] consider the problem of concurrent workflow scheduling in high performance computing resources (HPC clouds). They present a scheduling method that aims to minimize the total cost in terms of the computation cost, the communication cost as well as the earliest start time.

Li [26] studies the resource scheduling problem based on a service level agreement (SLA) which is known to be NP-hard. The SLA incorporates restrictions based on throughput, latency and cost. Using stochastic integer programming technique, an optimal solution is presented that satisfies all SLA constraints and minimizes the total cost.

Numerous additional works for metaheuristic scheduling techniques for Cloud computing also exist as surveyed in [16] detailing metaheuristics for Cloud scheduling problems from a theoretical point of view. Furthermore, while numerous novel approaches are presented, it is also highlighted that both assumptions and objectives are often not objectively clear in the context of Cloud computing, and frequently differ between each study.

4. RESOURCE BOXING

4.1 System Model

The objective of our approach is to convert realistic task resource utilization patterns in Cloud computing into boxes that can be directly understood and integrated into theoretical scheduling algorithms (defined as resource boxing). Using historical data is effective for yielding significant improvements on scheduler performance as demonstrated in [26]. Figure 3 depicts a high level system model of this approach that automates this entire process. The resource utilization patterns of servers are transmitted to the resource boxing process and consists of the following steps:

System characterizer: Raw trace data of server resource utilization patterns are studied to distinguish unique task execution patterns for each server within the Cloud computing system. The system filters and extracts key parameters of interest from the raw trace data in order to collect task ID, task utilization and respective timestamp of occurrence. This allows to massively reduce the computation and data size of the file for resource boxing.

Resource convertor: The filtered data for task resource patterns are then applied to an algorithm to conduct automated resource boxing. It is possible to select the type of algorithm for performing conversion, as well as configure its parameters dependent on administrator requirements. The output of this component is the equivalent resource utilization of the server within a box format, stored within a database and visualized for analysis and exploitation.

It is currently unclear what the optimal method is for resource boxing in terms of computation, number of update points, and accuracy within the context of task patterns within Cloud computing. Within the next section we propose four conversion algorithms to perform resource boxing.

4.2 General Notation

Resource boxing is the transformation of complex and dynamic utilization patterns of tasks into a series of boxes. Within this work we focus on CPU utilization, however the transformation is directly applicable to other resources such as memory, disk usage, and network.

We start by defining the notation that we use for the remainder of this paper based on a task T_k assigned to a machine M_i. The release time of T_k is denoted by r_k and corresponds to the time at which T_k is assigned to M_i. Note that this includes migration or rescheduling of T_k to M_i due to failures or eviction policies within the scheduler. Similarly, the completion time is denoted by C_k and corresponds to the time at which T_k leaves M_i. This can be due to completion of execution of T_k, migration of T_k to other machines, as well as task eviction or failure. The CPU utilization pattern of task T_k is given by a time sequence

$$\mathcal{T}_k = (t_{k1}, t_{k2}, \dots, t_{kN})$$

and by a sequence of CPU utilizations

$$\mathcal{U}_k = (u_{k1}, u_{k2}, \dots, u_{kN}).$$

Here $u_{k\ell}$ represents the CPU utilization at time $t_{k\ell}$ for $1 \leq \ell \leq N$, where N is the number of measurements available. Both sets $\mathcal{T}_k$ and $\mathcal{U}_k$ are part of the input within resource boxing. A set of update points

$$\mathcal{P}_k = \{p_{kj} |\, p_{kj} \text{ update point}, 1 \leq j \leq P\}$$

is selected where an update point is the time at which the height of a box for a task may change, and P denotes the number of update points. Times $r_k = t_{k1}$ and $C_k = t_{kN}$ are always considered to be update points in the context of resource boxing. As an example, the update points of the example from Figure 2 are given by {0,2,4,8,10,16}.

Having introduced the notation, we explain how to determine the box height at an update point and how to select the set of update points $\mathcal{P}_k$.

4.3 Box Height Determination

Consider a CPU utilization pattern with utilization sequence $\mathcal{U}_k$, time sequence $\mathcal{T}_k$ and a set of update points $\mathcal{P}_k$. For an update point p_{kj} we distinguish the following three cases:

1) **Case $j = P$:** The update point coincides with the completion time C_k and therefore no further box is created.
2) **Case $1 < j < P$**: The condition $j > 1$ implies that the update point p_{kj} is not the release time $r_k = p_{k1}$ of T_k. Therefore, the update point $p_{k,j-1}$ exists and the time interval
$$[p_{k,j-1}, p_{kj}) = \{t | p_{k,j-1} \leq t < p_{kj}\}$$
is non-empty. The height of the box in interval
$$[p_{kj}, p_{k,j+1}] = \{t | p_{kj} \leq t < p_{k,j+1}\}$$
is then calculated by the weighted average CPU utilization over time interval $[p_{j-1}, p_j)$. The weights for CPU utilization $u_{k\ell}$ for $1 \leq \ell < N$ are calculated according to the time differences $t_{k,\ell+1} - t_{k\ell}$.
3) **Case j = 1**: The update point coincides with the release time of T_k, thus taking the weighted average CPU utilization over the previous time interval is not possible. Instead, the height of the box for time interval $[p_{k1}, p_{k2}]$ is chosen using an estimator. A simple choice for an estimator is the CPU utilization u_{k1} at time t_{k1} as illustrated in Figure 2. It is also possible to use historical data, prediction techniques [27][28], or a combination of both to achieve a more accurate estimator as found in [17]. However, the choice of the estimator will not have a significant impact on the overall accuracy since only the first box is affected.

4.4 Conversion Algorithms

In this subsection we propose four algorithms each of which uses different logic for selecting the set of update points for T_k on machine M_i. It is worth noting that each of these approaches is capable of resource boxing multi-tenant servers (i.e. multiple tasks executing simultaneously). We consider four different resource conversion algorithms: *time zero*, *event-based*, *interval-based*, and *hybrid*.

Time Zero: Resource boxing is performed at the very beginning of task execution within the server and there are no further box updates performed. Therefore, the set of update points $\mathcal{P}_k = \mathcal{P}_k^0$ is given by

$$\mathcal{P}_k^0 = \{r_k, C_k\}.$$

As shown in Figure 4(a), the box height does not change irrespective of task utilization. This approach would appear to be effective in the event of very stable task patterns since updating would result in only minimal change.

Event-based: Resource boxing is performed based on a reactive approach to changes within the server environment. In other words, update points are calculated when a task is assigned or completed within the server as shown in Figure 4(b). Such an approach has been used for scheduling decisions in [19] under the assumption that significant alteration to utilization patterns occur due to event changes within the server. In this context, an event is defined as the timestamp when another task is assigned to, or completes within M_i. The set of update points $\mathcal{P}_k = \mathcal{P}_k^E$ consists of all time points in $[r_k, C_k]$ at which an event occurs. This can be represented as

$$\mathcal{P}_k^E = \{t | r_k \leq t \leq C_k, t \text{ is a task arrival/departure }\}.$$

Note that this definition also implies $r_k, C_k \in \mathcal{P}_k^E$ since there is the task arrival respectively the task departure of T_k at time r_k respectively at time C_k.

Interval-based: Updates are periodically performed at fixed time intervals as shown in Figure 4(c). The interval is determined by

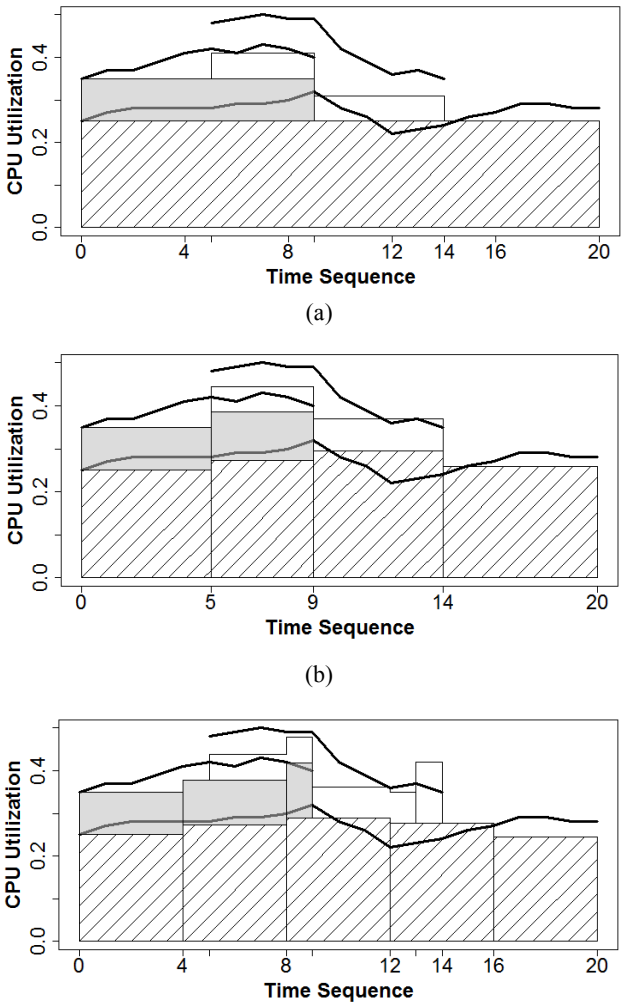


Figure 4. Visual depiction of resource boxing algorithms: (a) time zero, (b) event-based, (c) interval-based.

periodic updates at every L time units where L is a configurable input parameter determined by the system administrator. This allows for more coarse-grain or fine-grained formation dependent on parameter selection, and resource boxing irrespective of the server status and number of tasks. The set of update points $\mathcal{P}_k = \mathcal{P}_k^{I(L)}$ is given by

$$\mathcal{P}_k^{I(L)} = \{t | t = r_k + aL, a \geq 0, t \leq C_k\} \cup \{C_k\}.$$

Similar to above, this implies $r_k, C_k \in \mathcal{P}_k^{I(L)}$.

Hybrid approach: It is possible to combine both event-based and interval-based approaches into a single algorithm. This allows to capture sudden changes to the task composition within a server whilst updating dynamic changes of long running tasks during extended periods of no event occurrence. Similar to interval-based, parameter L is configurable. The set of update points $\mathcal{P}_k^{H(L)}$ is simply the union of the event-based and interval-based set of update points:

$$\mathcal{P}_k^{H(L)} = \mathcal{P}_k^{E} \cup \mathcal{P}_k^{I(L)}.$$

Since we have $r_k, C_k \in \mathcal{P}_k^{E}$ and $r_k, C_k \in \mathcal{P}_k^{I(L)}$, it follows that $r_k, C_k \in \mathcal{P}_k^{H(L)} = \mathcal{P}_k^{E} \cup \mathcal{P}_k^{I(L)}$.

5. ALGORITHM EVALUATION

5.1 Metrics to Measure Similiarity

In order to analyze the accuracy of proposed algorithms for resource boxing, it is necessary to define a proper metric that measures the similarity between Cloud resource patterns and the resource boxing representation. As resource boxing can be agnostically applied to various resource utilization patterns, it is unlikely that there is a single metric that is considered to be the overall optimal choice. For this reason, we propose several metrics for algorithm evaluation.

Consider machine M_i over a time horizon given by the *time sequence*

$$\mathcal{T} = (t_1, t_2, \dots, t_N).$$

Then, we consider the *total resource utilization sequence*

$$\mathfrak{U} = (U_1, U_2, \dots, U_N),$$

where $U_\ell \in \mathfrak{U}$ is the total resource utilization of M_i at time t_ℓ. In other words, U_ℓ is equal to the aggregated value of resource usage of each task on M_i at time t_ℓ. Similarly, we consider the total box height sequence

$$\mathfrak{B} = (B_1, B_2, \dots, B_N),$$

where $B_\ell \in \mathfrak{B}$ is the aggregated height of boxes at time t_ℓ.

Absolute Deviation: This metric calculates the error of resource boxing at every time and weights it according to the length of the subsequent interval:

$$\frac{1}{t_N - t_1} \sum_{\ell=1}^{N-1} (t_{\ell+1} - t_\ell) |U_\ell - B_\ell|.$$

Note that since there are n time points and only $N-1$ intervals, this metric does not take into account the error between U_N and B_N.

Relative Deviation: Similar to the absolute deviation, this metric calculates the relative error at every time and weights it accordingly:

$$\frac{1}{t_N - t_1} \sum_{\ell=1}^{N-1} (t_{\ell+1} - t_\ell) \frac{|U_\ell - B_\ell|}{U_\ell}.$$

As before, the error between U_N and B_N is not considered.

Ratio of Averages: Let $\overline{\mathfrak{U}}$ be the average total resource utilization and let $\overline{\mathfrak{B}}$ represent the average total box height (both weighted accordingly). Then the ratio of averages is defined by $\frac{\overline{\mathfrak{B}}}{\overline{\mathfrak{U}}}$. This metric is useful to check whether resource boxing is overestimating or underestimating the actual resource pattern on average.

Ratio of Variances: Let $\mathrm{Var}[\overline{\mathfrak{U}}]$ and $\mathrm{Var}[\overline{\mathfrak{B}}]$ be the variances of the average total resource utilization and of the total box heights. The ratio of Variances

$$\frac{\mathrm{Var}[\overline{\mathfrak{B}}]}{\mathrm{Var}[\overline{\mathfrak{U}}]}$$

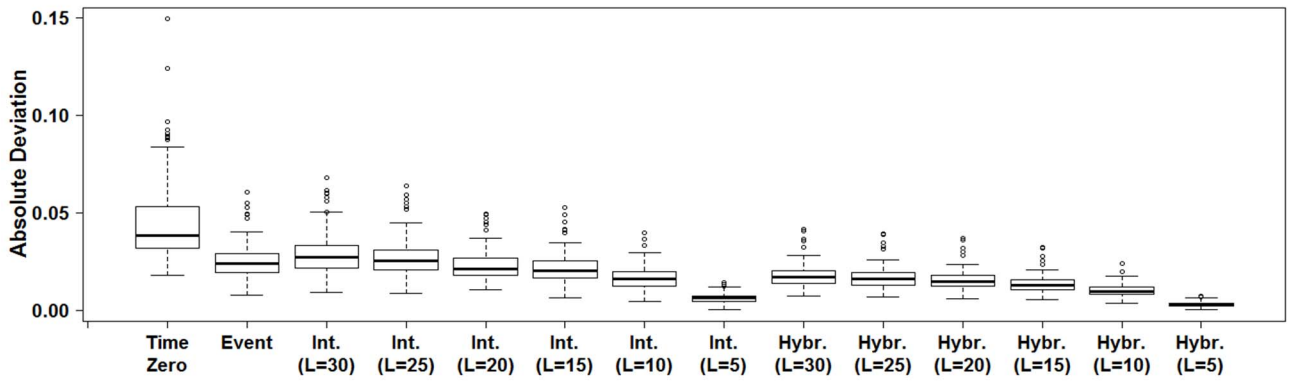


Figure 5. Absolute error deviation for all resource converson algorithms.

is a metric that can be used to verify that the variance of the box approach is close to the variance of the original data.

Ratio of Standard Deviations: The ratio of standard deviations is defined by

$$\frac{\sqrt{Var[\overline{\mathfrak{B}}]}}{\sqrt{Var[\overline{\mathfrak{U}}]}},$$

and is used to verify whether the standard deviation of the box approach is close to the standard deviation of the original data.

5.2 Resource Boxing Evaluation with Production Data

In this section we apply resource boxing to the second version of Google Cloud tracelog [29][30] to analyze algorithm accuracy. The Google Cloud tracelog contains 12,580 servers, over 25 million tasks and 930 users for 29 full days of operation. This data contains information about both CPU and memory utilization of tasks within machines normalized between 0 to 1. Each record refers to an average resource utilization value over a period of typically five minutes.

In our analysis we have randomly selected 150 machines of various architectures and execution from the data set and automatically extracted the following parameters: task ID (comprising JobId and taskindex), timestamp, CPU utilization and machine ID from the *task_resource_usage* table over the entire 29 days. We have applied resource boxing to the CPU utilization patterns of each machine. Conversion algorithms time zero and event-based have been applied, as well as the interval-based and hybrid approaches using a range of periodic update parameter values $L \in \{5,10,15,20,25,30\}$ which are measured in minutes.

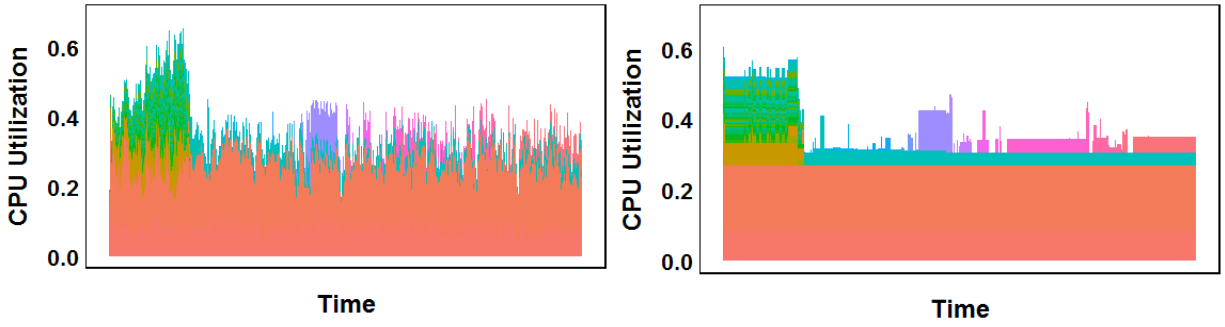


Figure 6. Day 3-6 utilization patterns of machineID 257408789 (a) real CPU utilization, (b) time-zero.

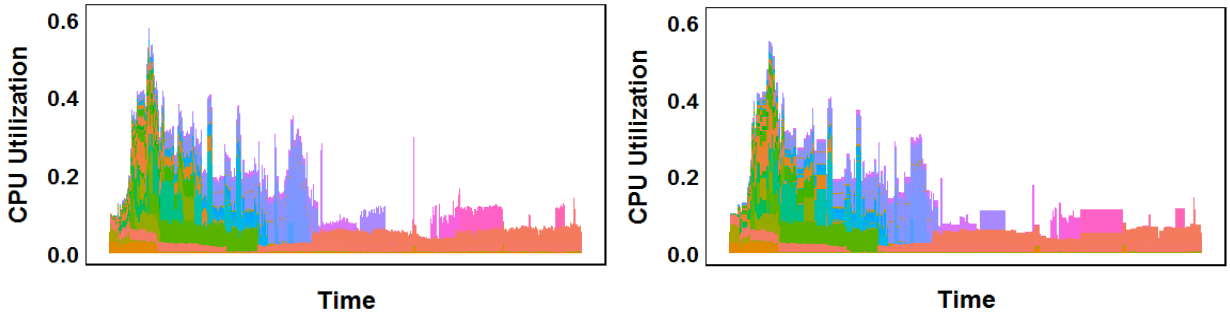


Figure 7. Day 3-6 utilization patterns of machineID 32915063 (a) real CPU utilization, (b) event-based.

Figure 6 and Figure 7 show a graphical comparison between real machine CPU utilization over three days comparing different resource boxing algorithms, with each color representing a unique task. It is observable that visually these patterns are similar and that the event based algorithm is capable of capturing the fluctuation of resource usage and scheduling/completion of tasks. Clearly, the plots visually demonstrate that the event-based approach is more accurate than the time zero algorithm.

Figure 5 provides details of the absolute deviation of resource boxing and empirical data from all 150 machines, respectively. It is observable that with the exception of time zero, all resource conversion algorithms result in an error rate less than 8% and a mean of less than 3%. The reason for the large deviation for time zero is due to the fact that the height of each box remains constant and cannot capture fluctuating resource patterns. The hybrid approach results in the lowest error rate of less than 4% and a mean of approximately 2.5%. However, as illustrated in Figure 8 and Figure 9, this also introduces the largest number of update points. This is an important consideration within the context of large-scale Cloud computing systems – heavy network use to transmit data and computation of resource boxing can potentially have a detrimental effect on the network performance of the system.

As discussed previously, interval-based resource boxing enables the control of the number of update points created by the algorithm as shown in Figure 9. We observe that there exists a negative correlation between the number of update points and the error rate for resource boxing accuracy. This allows for system administrators to select an appropriate periodic interval dependent on accuracy required, or even dependent on the current utilization of the system (i.e. low level of cluster usage can allow for more fine grained data collection). Furthermore, this is an important consideration for online scheduling, requiring rapid decision making for effective task allocation. On the other hand, time zero algorithm only requires a single update point to calculate the box and results in an error rate up to 10-15%. This is important in order to capture and mitigate extreme or abnormal task behavior that may arise in Cloud computing systems (e.g. correlated failures).

While the hybrid algorithm achieves the highest accuracy, from the analysis it is observable that interval-based resource boxing is capable of achieving high levels of accuracy whilst keeping the number of update points at an acceptable level – with the advantage of the latter being controllable due to parameter configuration. The analysis of a production Cloud computing system demonstrates that it is possible to convert real task patterns into boxes applicable for theoretical scheduling. Specifically, we are capable of capturing deviation in task execution within multi-tenant environments and introduction of new tasks automatically. However, it is worth highlighting that although there exists dynamic change in resource utilization patterns over the month period, task patterns are observed to be relatively stable per individual task as analyzed in [1], and that the CPU utilization is a five minute aggregate, resulting in increased algorithm accuracy. As a result, it is necessary to study resource boxing at much higher fidelity of resource utilization patterns.

6. VALIDATION & APPLICATION

Within this section we detail a scenario to which resource boxing can be used to study and improve scheduling within Cloud computing from a theoretical context.

We used our method for resource boxing to investigate whether it is possible to translate empirical energy profiles into theoretical scheduling inputs. We conducted an experiment to collect the power profiles of a DELL D3400, Intel Core 2 Quad CPU @2.83GHz running Debian Ubuntu over a period of 450 seconds. We developed a program written in C++ to emulate multiple tasks simultaneously executing with their task patterns changing over time.

Each task was generated such that it performs the following life cycle:

1. The task is generated at time $t = 0$. The total life time $\mathcal{L}$ of the task is chosen randomly from interval [150,450]. Go to step 2.
2. There is a 30% probability that the task is put into sleep mode for s seconds, where s is stochastically selected from [0,250]. If the task was put to sleep, set $t = s$. If not, set $t = 0$. Go to step 3.
3. If $t \geq \mathcal{L}$, go to step 4. Otherwise, select a number δ randomly from [15,25]. The CPU utilization of the task in interval $[t, t + \delta]$ is then artificially forced to a constant, yet randomly chosen, level. Set $t = t + \delta$ and go to step 3.
4. The task is killed and ceases execution.

By using a Bash-script we recorded and extracted the CPU utilization of every task using the *top* command and its respective timestamp. It is possible for the task process to be recorded as 0 due to low scheduling priority (observable within production systems) [18]. Therefore, we collected the average record for each second and task which provides a more realistic CPU utilization.

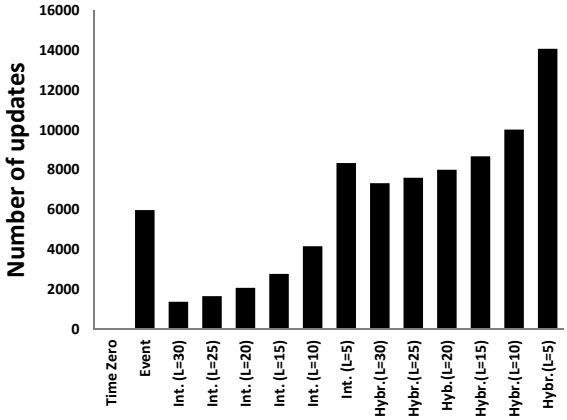


Figure 8. Number of update points per algorithm.

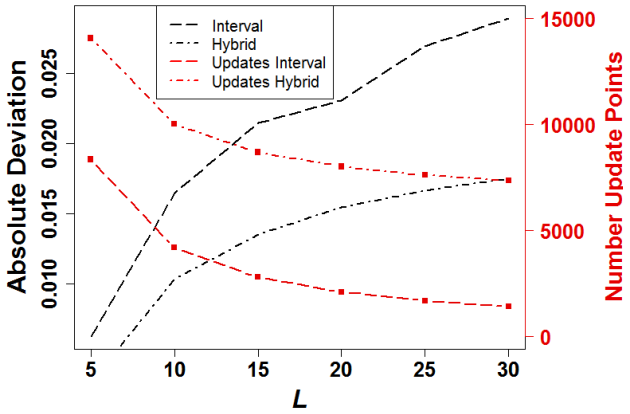


Figure 9. Parameter L sensitivity.

We collected the server power consumption using Voltech PM1000+ power meter that automatically measures and extracts the actual power consumption of the server throughout the experiment. We obtained exactly one record per second for the power consumption (measured in Watts).

We applied resource boxing algorithms to the CPU utilization pattern of tasks and analyzed the accuracy of our approach. Figure 10 gives a graphical comparison between the real CPU (top) and the box representation using the event –based algorithm (middle). The task behavior can be accurately represented every second with an error rate of 1.08%.

We exploit the produced resource boxing approximation to calculate server energy consumption and measure its similarity compared to actual energy recorded. Modelling the power as a linear function of the CPU utilization is widely recognized within the literature [31][32]. From data produced within the experiment we construct a linear function $P(U)$ with parameters a and b as follows:

$$P(U) = a + bU.$$

In order to determine suitable values for a and b we conducted a linear regression analysis based on the data we recorded previously with the Voltech PM1000+ power meter. This analysis yielded $a = 105.7$ and $b = 64.1$, therefore the power function reads now as follows:

$$P(U) = 105.7 + 64.1U.$$

This formula was applied to calculate the energy consumption based on both the recorded CPU utilization levels and the resource boxing approximation. The energy consumption is calculated by integrating power over time which translates to taking the weighted average mean of the recorded power consumption records in our discrete context. Our analysis findings show that the total energy consumed by the server is 57 kWs per experiment execution with less than a 1% error rate when applied to resource boxing. That shows the predicted energy comes very close to the actual consumed energy, indicating that using resource boxing can successfully capture realistic server power usage which can be exploited for evaluating theoretical power-aware scheduling models.

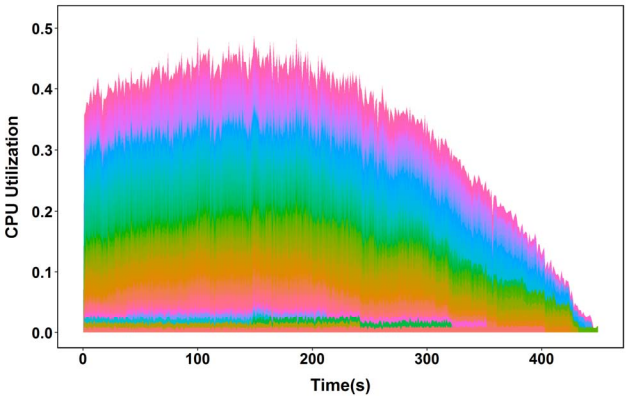
7. CONCLUSIONS

In this paper we have presented resource boxing - an approach for translating realistic resource utilization patterns of Cloud computing tasks into inputs that are useable by theoretical scheduling algorithms. We have identified a knowledge gap which exists between theoretical and applied scheduling, and have an automated technique for schedule box creation derived from real Cloud utilization in order for these closer collaboration between these communities. Our conclusions are summarized as follows:

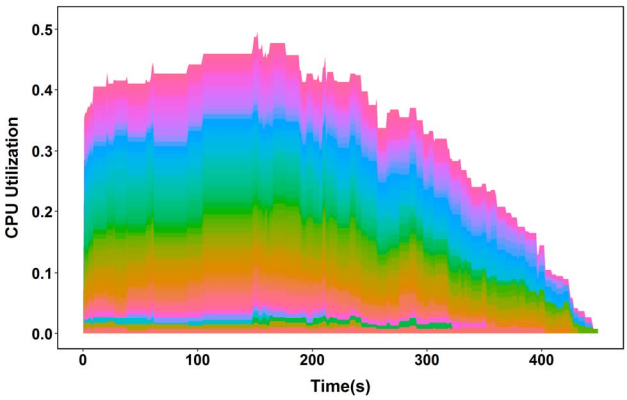
Real task patterns can be directly converted for theoretical scheduling inputs. We demonstrate from analysis of production trace data and experiments that it is possible to create box equivalents of diverse task execution within multi-tenant Cloud servers with less than 3% error rate in absolute deviation.

Interval-based resource boxing is highly effective. In terms of accuracy, number of update points, and computation time, it appears that interval-based resource conversion is the most effective approach within our case studies. However, this particular algorithm requires a system administrator to manually specify the time period between update points which may not be always possible or desired. For such cases we recommend using event-based resource boxing since we demonstrated it to have an acceptable accuracy and since it requires no input parameters.

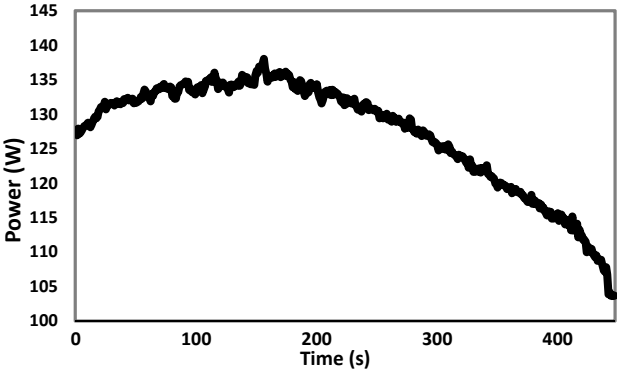
Future work includes introducing more extreme workflow patterns to evaluate the accuracy of resource boxing, as well as evaluate numerous theoretical algorithms by using the derived boxes from this analysis. We plan to directly apply this approach in order to perform online decision making using theoretical scheduling algorithms within real systems. Finally, we intend to generate a workload generator that creates boxes based on realistic task behavior which enables the validation of theoretical algorithms within real world systems.



(a)



(b)



(c)

Figure 10. Server resource patterns for (a) real CPU, (b) event-based CPU, (c) power consumption.

8. ACKNOWLEDGMENT

This work is partly supported by the European Commission under H2020-ICT-20152 contract 687584 - Transparent heterogeneous hardware Architecture deployment for eNergy Gain in Operation (TANGO) project.

9. REFERENCES

[1] Van, H. N., Tran, F. D. and Menaud, J.-M. 2009. SLA-Aware Virtual Resource Management for Cloud Infrastructures. In *Proceedings of the 2009 Ninth IEEE International Conference on Computer and Information Technology - Volume 02* (Xiamen, China, Aug 08 – 11, 2009). CIT '09. IEEE Computer Society, Washington, DC, USA, 357-362. DOI=http://dx.doi.org/10.1109/CIT.2009.109.

[2] Tsakalozos, K., Roussopoulos, M. and Delis A. 2011. VM placement in non-homogeneous IaaS-Clouds. In *Proceedings of the International Conference on Service-Oriented Computing* (Paphos, Cyprus, Dec 05 – 08, 2011). ICSOC '11. Springer-Verlag Berlin, Heidelberg, 172-187. DOI=http://dx.doi.org/10.1007/978-3-642-25535-9_12

[3] Machida, F., Kawato, M. and Maeno, Y. 2010. Redundant virtual machine placement for fault-tolerant consolidated server clusters. In *Proceedings of the IEEE Network Operations and Management Symposium-NOMS* (Osaka, Japan, Apr 19 – 23, 2010). NOMS '10. IEEE Computer Society, Washington, DC, USA 32-39. DOI=http://dx.doi.org/10.1109/NOMS.2010.5488431

[4] Zhang, Y., Zheng, Z. and Lyu, M. R. 2011. BFTCloud: A Byzantine Fault Tolerance Framework for Voluntary-Resource Cloud Computing. In *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing* (Washington, DC, USA, Jul 04 – 09, 2011). CLOUD '11. IEEE Computer Society, Washington, DC, USA, 444-451. DOI=http://dx.doi.org/10.1109/CLOUD.2011.16.

[5] Javadi, B., Abawajy, J. and Buyya, R. 2012. Failure-aware resource provisioning for hybrid Cloud infrastructure. *J. Parallel Distrib. Comput.* 72, 10 (October 2012), 1318-1331. DOI=http://dx.doi.org/10.1016/j.jpdc.2012.06.012

[6] Meng, X., Pappas, V. and Zhang, L. 2010. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *Proceedings of the 29th Conference on Information Communications* (San Diego, CA, USA, Mar 15 – 19, 2010). INFOCOM '10. IEEE Press, Piscataway, NJ, USA, 1154-1162.

[7] Biran, O., Corradi, A., Fanelli, M., Foschini, L., Nus, A., Raz, D. and Silvera, E. 2012. A Stable Network-Aware VM Placement for Cloud Systems. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (Ottawa, Canada, May 13 – 16, 2014). CCGRID '12. IEEE Computer Society, Washington, DC, USA, 498-506. DOI=http://dx.doi.org/10.1109/CCGrid.2012.119

[8] Sampaio, A. M., Barbosa, J.G. and Prodan, R. 2015. PIASA: a power and interference aware resource management strategy for heterogeneous workloads in Cloud data centers. *Simulation Modelling and Practise Theory*. 57 (July 2015), 142-160. DOI= http://dx.doi.org/10.1016/j.simpat.2015.07.002

[9] Graubner, P., Schmidt, M. and Freisleben, B. 2011. Energy-Efficient Management of Virtual Machines in Eucalyptus. In *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing* (Washighton, DC, USA, Jul 04 – 09, 2011). CLOUD '11. IEEE Computer Society, Washington, DC, USA, 243-250. DOI=http://dx.doi.org/10.1109/CLOUD.2011.26

[10] Srikantaiah, S., Kansal, A. and Zhao, F. 2008. Energy aware consolidation for cloud computing. In *Proceedings of the 2008 Conference on Power Aware Computing and Sstems* (San Diego, CA, USA, Dec 08 – 10, 2008). HotPower'08. USENIX Association, Berkeley, CA, USA, 10-10

[11] Johnson, S. M. 1954. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1 (March 1954), 61-68. DOI=http://dx.doi.org/ 10.1002/nav.3800010110

[12] Bellman, R. 1956. Mathematical aspects of scheduling theory. *Journal of the Society for Industrial and Applied Mathematics*. 4, 3 (September 1956), 168-205.

[13] Pinedo, M. and Chao, X. 1999. Operations Scheduling with Applications in Manufacturing and Services. McGraw-Hill Companies.

[14] Brucker, P. 2001. Scheduling Algorithms. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[15] Jiang, H.-J., Huang, K.-C., Chang, H.-Y., Gu, D.-S. and Shih., P.-J. 2011. Scheduling concurrent workflows in HPC cloud through exploiting schedule gaps. In *Proceedings of the 11th International Conference on Algorithms and Architectures for Parallel Processing* (Melbourne, Australia, Oct 24 – 26, 2011). ICA3PP '11. Springer-Verlag, Berlin, Heidelberg, 282-293.

[16] Tsai, C.-W. and Rodrigues, J. J. P. C. 2014. Metaheuristic scheduling for Cloud: a survey. *Syst. J.* 8, 1 (March 2014), 279-291. DOI=http://dx.doi.org/10.1109/JSYST.2013.2256731

[17] Moreno, I. S., Garraghan, P., Townend, P. and Xu, J. 2014. Analysis, modeling and simulation of workload patterns in a large-scale utility Cloud. *Trans. on Cloud Comp.* 2, 2 (June 2014), 208-221. DOI= http://dx.doi.org/ 10.1109/TCC.2014.2314661

[18] Wang, L. and Gelenbe, E. 2015. Adaptive dispatching of tasks in the Cloud. *IEEE Trans. on Cloud Comp.* PP, no. 99, 1–1.

[19] Mishra, A. K., Hellerstein, J. L., Cirne W. and Das, C. R. 2010. Towards characterizing cloud backend workloads: insights from Google compute clusters. *ACM SIGMETRICS Perform. Eval. Rev.* 37, 4 (March 2010), 34-41. DOI=http://dx.doi.org/10.1145/1773394.1773400

[20] Kavulya, S., Tan, J., Gandhi, R. and Narasimhan P. 2010. An Analysis of Traces from a Production MapReduce Cluster. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing* (Melbourne, Australia, May 17 – 20, 2010). CCGRID '10. IEEE Computer Society, Washington, DC, USA, 94-103. DOI=http://dx.doi.org/10.1109/CCGRID.2010.112

[21] Moreno, I. S., Yang, R., Xu, J., and Wo, T. 2013. Improved energy-efficiency in cloud datacenters with interference-aware virtual machine placement. In *Proceedings of the IEEE International Symposium on Autonomous Decentralized Systems* (Mexico City, Mexico, Mar 06 – 08, 2013). ISADS '13, pp. 1-8, 2013. DOI=http://dx.doi.org/ 10.1109/ISADS.2013.6513411

[22] Piraghaj, S. F., Calheiros, R. N., Chan, J., Dastjerdi, A. V. and Buyya, R. 2016. Virtual Machine Customization and Task Mapping Architecture for Efficient Allocation of Cloud Data Center Resources. *The Computer Journal.* 59, 2 (November 2015), 208–224. DOI=http://dx.doi.org/10.1093/comjnl/bxv106.

[23] Rahman, M., Li, X. and Palit, H. 2011. Hybrid Heuristic for Scheduling Data Analytics Workflow Applications in Hybrid Cloud Environment. In *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum* (Anchorage, AK, USA, May 16 – 20, 2011). IPDPSW '11. IEEE Computer Society, Washington, DC, USA, 966-974. DOI=http://dx.doi.org/10.1109/IPDPS.2011.243

[24] Yin, P.-Y., Yu, S.-S., Wang, P.-P. and Wang, Y.-T. 2006. A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems. *Comput. Stand. Interfaces* 28, 4 (April 2006), 441-450. DOI=http://dx.doi.org/10.1016/j.csi.2005.03.005

[25] Li, Q. 2011. An optimal algorithm for resource scheduling in Cloud computing. *Adv. Multim., Softw. Engin. Comp.* 2 (2012), 293-299. DOI=http://dx.doi.org/ 10.1007/978-3-642-25986-9_46

[26] Beloglazov, A. and Buyya, R. 2012. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers. *Concurr. Comput. : Pract. Exper.* 24, 13 (September 2012), 1397-1420. DOI=http://dx.doi.org/10.1002/cpe.1867.

[27] Djemame, K. and Haji, M. H. 2007. Grid application performance prediction: a case study in BROADEN. In *Proceedings of the First international conference on Verification and Evaluation of Computer and Communication Systems* (Algiers, Algeria, May 05 – 06, 2007). VECoS '07. British Computer Society, Swinton, UK, 158-169.

[28] Wang, K. and Franklin, M. 1997. Highly accurate data value prediction using hybrid predictors. In *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture* (Research Triangle Pk, North Carolina, USA, Dec 01 – 03, 1997). MICRO '30. IEEE Computer Society, Washington, DC, USA, 281-290.

[29] Reiss, C., Wilkes, J. and Hellerstein, J. 2014. Google cluster-usage traces: format + schema. *Google Inc.* https://drive.google.com/file/d/0B5g07T_gRDg9Z0lsSTEtTWtpOW8/view

[30] ClusterData2011_2 traces. [Online] Available: https://github.com/google/cluster-data/blob/master/ClusterData2011_2.md

[31] Beloglazov, A. and Buyya, R. 2010. Energy Efficient Resource Management in Virtualized Cloud Data Centers. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing* (Melbourne, Australia, May 17 – 20, 2010). CCGRID '10. IEEE Computer Society, Washington, DC, USA, 826-831. DOI=http://dx.doi.org/10.1109/CCGRID.2010.46.

[32] Qiang, H., Gao, F., Wang, R. and Qi, Z. 2011. Power consumption of virtual machine live migration in Clouds. In *Proceedings of the International Conference on Communications and Mobile Computing* (Qingdao, China, Apr 18 – 20, 2011). CMC '11. IEEE Computer Society, Washington, DC, USA, 122-125. DOI=http://dx.doi.org/10.1109/CMC.2011.62