
Proceedings Paper:
Primas, B, Djemame, K, Garraghan, P et al. (2016) Resource boxing: Converting realistic cloud task utilization patterns for theoretical scheduling. In: Proceedings - 9th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2016. 9th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2016, 06-09 Dec 2016, Shanghai, China. ACM, pp. 138-147. ISBN: 9781450346160.

https://doi.org/10.1145/2996890.2996897

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

Bernhard Primas, Peter Garraghan, Karim Djemame
School of Computing,
University of Leeds
{scbjp, p.m.garraghan, k.djemame} @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 numerous 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.**

Keywords— Scheduling, task patterns, resource conversion

I. 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 Agreement (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][4][5], networking [6][7], and energy-efficiency [8][9][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, and is realized through a relatively simple decision making (such as round-robin) evaluated through simulation or experimentation.

Theoretical scheduling has intensely been studied by mathematicians for decades [12][13], providing important contributions to numerous fields such as manufacturing and service [14][15]. 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) [16][17]. Importantly, boxes are unable to capture the dynamicity of resource utilization patterns inherit within Cloud computing [18]. 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 numerous approaches for resource boxing. We detail multiple 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 applied 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.

II. 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 in 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 [15] providing additional detail. We are given n tasks $T_1, \dots, T_n$ and m machines $M_1, \dots, M_m$. For every task T_j, for $1 \leq j \leq n$, there are a given number of problem specific parameters. Typical examples for task parameters include release time r_j denoting the time at which T_j becomes available, or processing p_j denoting the time that T_j needs to complete. The objective of a scheduler is then to decide for each task

(1) which machine the task should be assigned to, and

(2) when the task should start on the selected machine

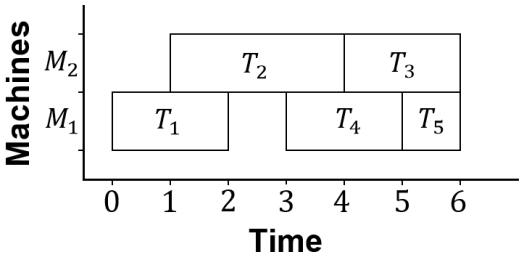


Figure 1. Example Gantt chart of optimal schedule

such that a number of problem specific constraints are satisfied and that a given objective function is minimized. Examples of constraints include that tasks which are assigned to the same machine must not overlap (i.e. simultaneous execution), and each task T_j can only be started after its release time r_j. 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 the tasks are provided in Table 1.

Table 1. Input parameters for an instance with 5 tasks.

	T_1	T_2	T_3	T_4	T_5
r_j	0	1	0	3	5
p_j	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 within the same machine overlap since no boxes overlap. Furthermore, it is easily verifiable that each task starts after its release time. This optimal solution consists of tasks T_1, T_4 and T_5 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 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 in not known in advance since tasks

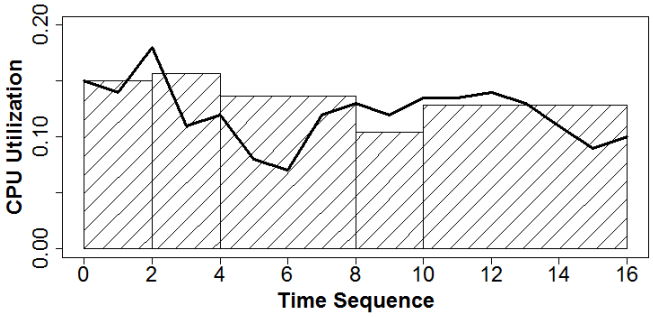


Figure 2. Representation of a task execution patterns by a series boxes.

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 can 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 [19] 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 in 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.

III. Related Work

Due to the mostly unexplored nature of this research area, there is limited efforts towards converting realistic assumptions of Cloud computing into a purely theoretical context. As a result, the body of work can be categorized within the context of studying and modeling 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. [20] classifies task execution qualitative boundaries for execution duration and uses *k*-means clustering to construct task classes. Using trace data from four Google compute cluster for 4 days. They construct eight different classes of tasks, separated by their respective duration and resource utilization.

Kuvulya et al. [21] 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 using a Lognormal distribution, and that 95% of jobs complete within under 20 minutes.

Solis Moreno, et al. [18] propose a method of analyzing and modeling user and task patterns. Their approach was applied 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 pattern usage 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 [17][22][23]. 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. [24] propose a scheduling model for workflow applications. They represent the workflow application as Directed Acyclic Graph (DAG) in which nodes correspond to tasks and edges correspond 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.

Yin et al. [25] also considered 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 added 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. [16] consider the problem of concurrent workflow scheduling in high performance computing resources (HPC clouds). They presented 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 [9] 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 [17] detailing metaheuristic scheduling models and algorithms for Cloud computing studied from a theoretical point of view. Numerous novel approaches are presented, however it is also highlighted that both assumptions and objectives are often not objectively clear in the context of Cloud computing, differing between study.

IV. Resource Boxing

A. 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). 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.

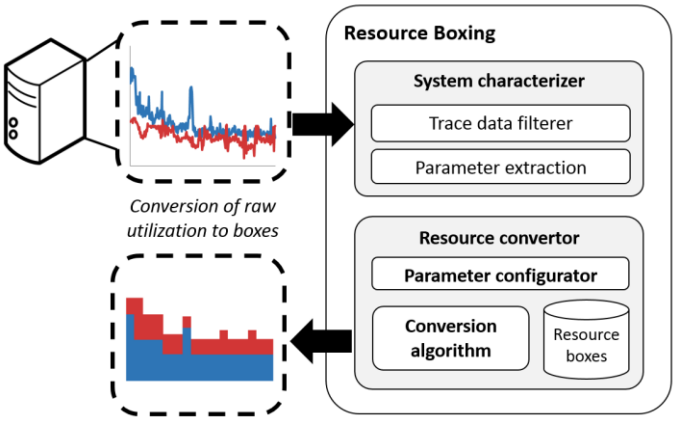


Figure 3. System model of resource boxing.

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 exploitation for analysis and exploitation.

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

B. 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 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 last time at which T_k runs on M_i. This includes 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_{kj} represents the CPU utilization at time t_{kj} for $1 \le j \le n$, 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. Resource boxing selects a set of update points $\mathcal{P}_k$ which is defined as

$$\mathcal{P}_k = \{p_{ki} |\, p_{ki} \text{ update point}, 1 \le i \le m\},$$

where an update point is the time the height of a box for task T may change, and m denotes the number of update points. Times t_{k1} and t_{kn} are not considered as update points for resource boxing. As an example, the update points in Figure 2 are given by $\mathcal{P} = \{2,4,8,10\}$.

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

C. 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_{ki} we distinguish the following three cases:

1) **Case $i = m$:** The update points coincides with the completion time of T_k and therefore no further box is created.
2) **Case $1 < i < m$**: The condition $i > 1$ implies that the update point p_{ki} is not the release time $r_k = p_{k1}$ of T_k. Therefore, the update point $p_{k(i-1)}$ exists and the time interval

   $$[p_{k(i-1)}, p_{ki}) = \{t_{kj} | p_{k(i-1)} \le t_{kj} < p_{ki}\}$$

   is non-empty. The height of the box in interval

   $$[p_{ki}, p_{k(i+1)}] = \{t_{kj} | p_{ki} \le t_{kj} < p_{k(i+1)}\}$$

   is then calculated by the weighted average CPU utilization over time interval $[p_{i-1}, p_i)$.

3) **Case $i = 1$**: 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, or a combination of both to achieve a more accurate estimator as found in [18]. However, the choice of the estimator will not have a significant impact on the overall accuracy since only the first box is affected.

D. Conversion Algorithms

In this subsection we propose a number of algorithms each of which uses different logic for selecting the set of update points for a 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 box updates performed. Therefore, the update set $\mathcal{P}_k = \mathcal{P}_k^0$ is empty:

$$\mathcal{P}_k^0 = \emptyset.$$

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 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 [20] 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 update set $\mathcal{P}_k = \mathcal{P}_k^E$ consists of all time points in (r_k, C_k) at which an event occurs. This can be represented as

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

Note that this definition also implies $r_k, C_k \notin \mathcal{P}_k^E$.

Interval-based: Updates are periodically performed at fixed time intervals as shown in Figure 4(c). The interval is determined by 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 update set $\mathcal{P}_k = \mathcal{P}_k^{I(L)}$ is given by

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

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

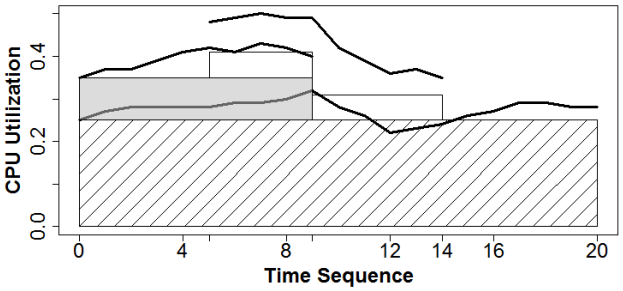
Hybrid approach: It is possible to combine both event-based and interval-based into a single algorithm. This allows to capture sudden changes to the task composition within a server whilst updating dynamic changes of long running tasks during extended periods of no event occurrence. Similar to interval-based,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

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

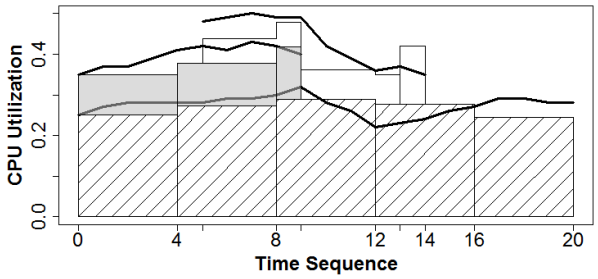
V. Algorithm Evaluation

A. *Metrics to Measure Similiarity*

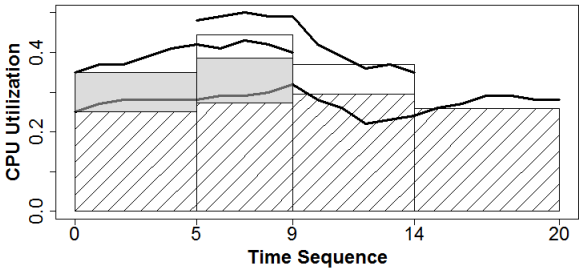
In order to analyze the accuracy of proposed algorithms for resource boxing, it is necessary to define a proper metric



(a)



(b)



(c)

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

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

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

We consider the *total resource utilization sequence*

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

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

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

where $B_j \in \mathfrak{B}$ is the aggregated height of boxes at time t_j.

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

$$\frac{1}{t_n - t_1}\sum_{j=1}^{n-1}(t_{j+1} - t_j)\left|U_j - B_j\right|.$$

Since there are n time points and only n-1 intervals, this metric does not take into account the value U_n.

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

$$\frac{1}{t_n - t_1}\sum_{j=1}^{n-1}(t_{j+1} - t_j)\frac{\left|U_j - B_j\right|}{U_j}.$$

For the same reason as before, the value U_n is not considered.

Ratio of Averages: Let $\overline{\mathfrak{U}}$ be the average total resource utilization and ler $\overline{\mathfrak{B}}$ 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

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

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.

B. Resource Boxing Evaluation with Production Data

In this section we apply resource boxing to the second version of Google Cloud tracelog [26][27] to analyze algorithm accuracy. The Google Cloud tracelog contains 12,580 servers, over 25 million tasks and 930 users for 29 days 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 parameters task ID (comprising JobId and taskindex), timestamp, CPU utilization and machine ID from the *task_resource_usage* table over the entire month. 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 the interval-based and hybrid approaches using a range of periodic update parameter {5, 10, 15, 25, 30} values for L.

Figure 6 and Figure 7 shows a graphical comparison between real machine CPU utilization over three days compared to resource boxing algorithms, with each color represented a unique task. It is observable that visually these patterns are similar – 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 results in an error rate less than 8% and a mean of 3%. The reason for the large deviation for

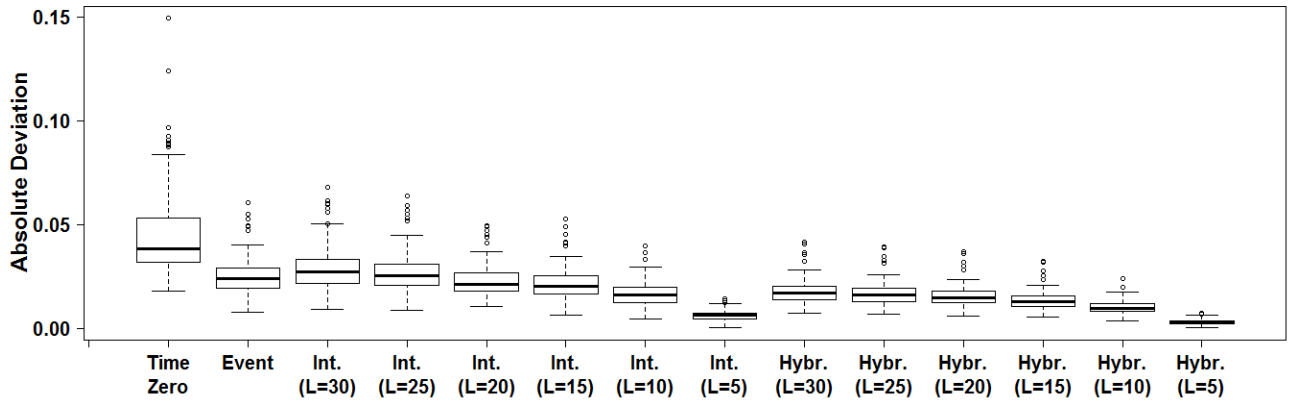


Figure 5. Absolute error deviation for all resource converson algorithms.

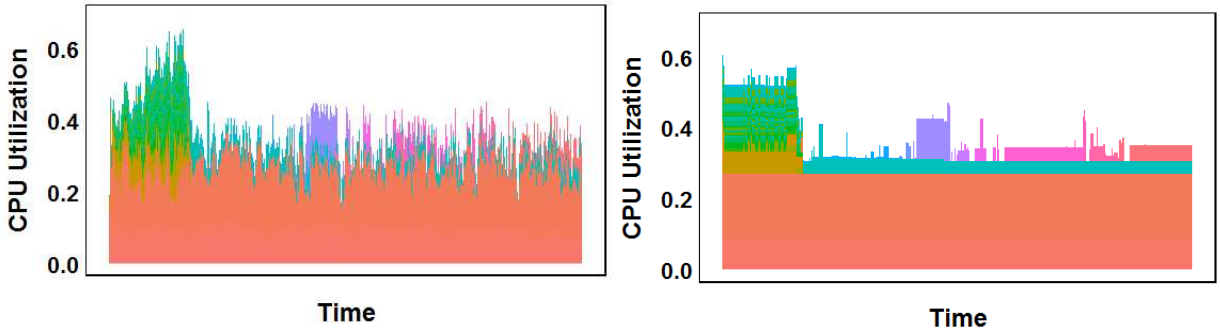


Figure 6. Day 3-6 utilization patterns of machineID 257408789 (a) real CPU utilization, (b) Time-zero

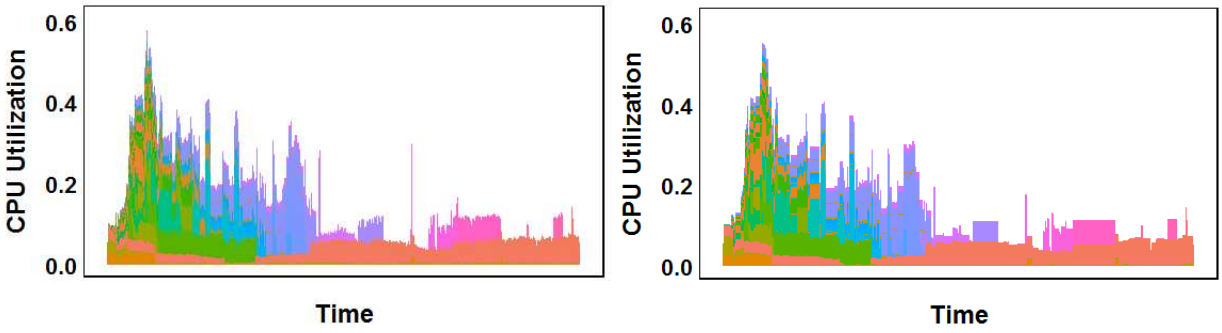


Figure 7. Day 3-6 utilization patterns of machineID 32915063 (a) real CPU utilization, (b) Event-based

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 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.

We demonstrated for interval-based resource boxing that it is possible to control the number of updates created by the algorithm as shown in Figure 9. We observe that there exists a negative correlation between the number of updates 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 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 to calculate the box, and resulting 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).

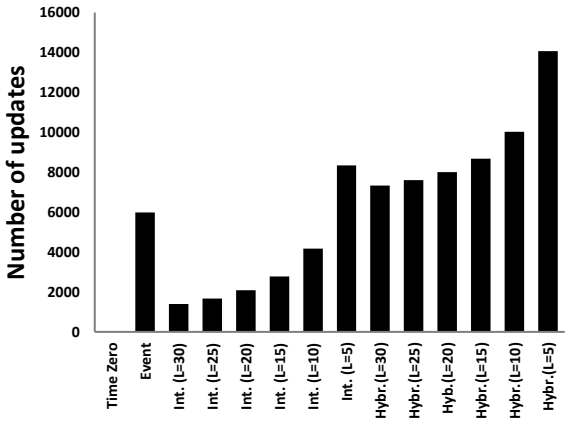


Figure 8. Resource boxing update per algorithm

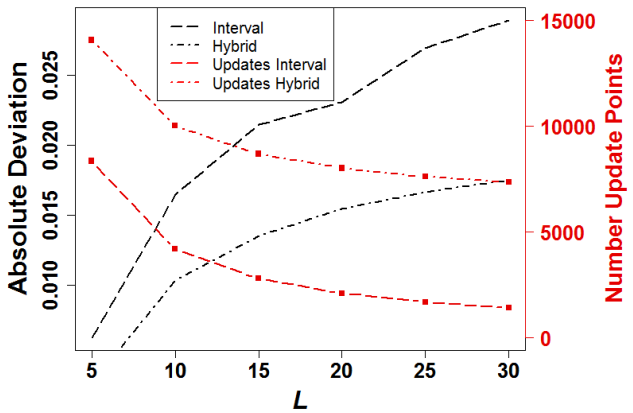


Figure 9. Parameter *L* sensitivity

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 minimizing the number of updates – 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.

VI. 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 ℓ randomly from $[15,25]$. The CPU utilization of the task in interval $[t, t + \ell]$ is then artificially forced to a constant, yet randomly chosen, level. Set $t = t + \ell$ and go to step 3.
4. The task is killed and ceases execution.

By using a Bash-script we have 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)[19]. Therefore, we collect the average record for each second and task which provides a more realistic CPU utilization.

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 apply resource boxing algorithms to the CPU utilization pattern of tasks and analyzed the accuracy of our approach. Figure 10 gives graphical comparison for the event-based algorithm to the recorded CPU utilization (top). 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

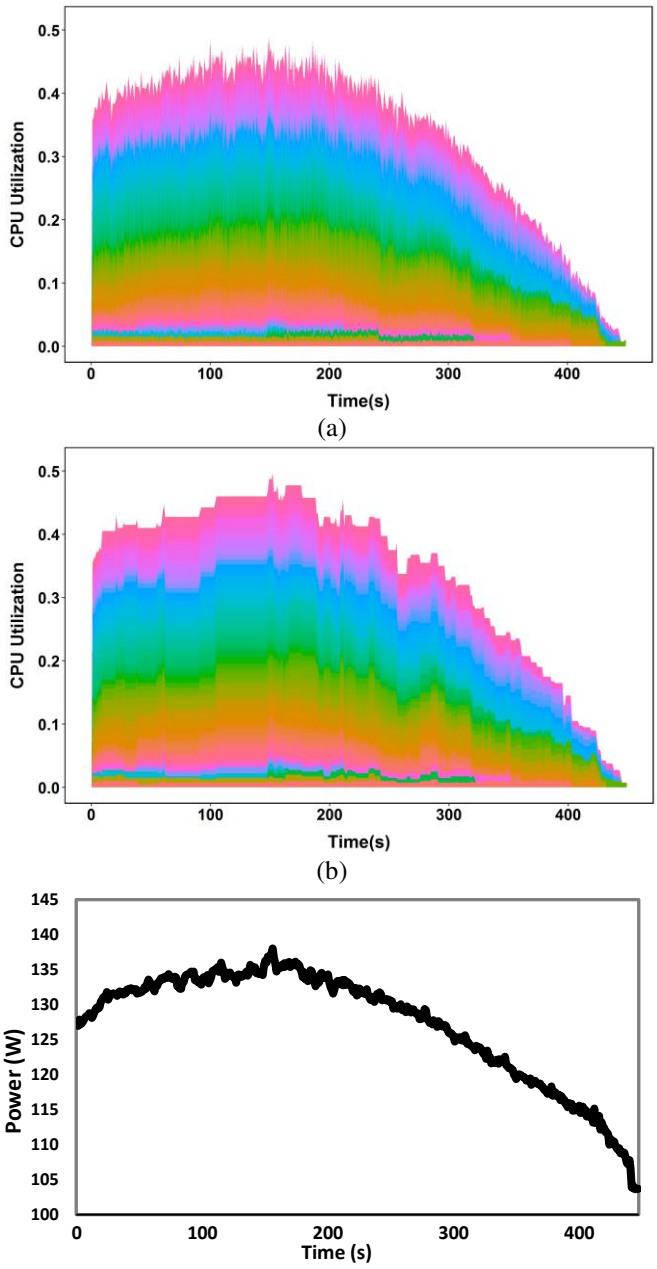


Figure 10. Server resource patterns for
(a) real CPU, (b) event-based CPU, (c) power consumption.

similarity compared to actual energy recorded. Modelling the power as a linear function of the CPU utilization is widely recognized within the literature [29][30]. 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 have conducted a linear regression analysis based on the data we recorded previously with the Voltech PM1000+ power meter. This analysis yielded $a = 64.1$ and $b = 105.7$, therefore the power function reads now as follows:

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

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. This 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.

VII. Conclusion

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 equivalent 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 updates, and computation time, it appears that interval-based resource conversion is the most effective within our case studies. However, this particular algorithm requires a system administrator to manually specify the time period between updates 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 even more extreme workflow patterns to evaluate its accuracy, 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 that can provide theoretical inputs for evaluating based on realistic task behavior.

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.

References

[1] H. N. Van, F. D. Tran, J.M Menaud, SLA-aware Virtual Resource Management for Cloud Infrastrtuctures, IEEE International Conference on Computer and Inforamtion Technology, 2009, vol. 1, pp. 357-362.

[2] K. Tsakalozos, M. Roussopoulos, A. Delis, VM placement in non-homogeneous IaaS-Clouds. In International Conference on Service-Oriented Computing, 2011, pp. 172-187.

[3] F. Machida, M. Kawato, Y. Maeno, Redundant virtual machine placement for fault-tolerant consolidated server clusters. IEEE Network Operations and Management Symposium-NOMS, 2010, pp. 32-39.

[4] Y. Zhang, Z. Zheng, Z., M.R. Lyu, BFTCloud: A byzantine fault tolerance framework for voluntary-resource cloud computing, IEEE International Conference on Cloud Computing (CLOUD), 2011, pp. 444-451.

[5] B. Javadi, J. Abawajy, R. Buyya, Failure-aware resource provisioning for hybrid Cloud infrastructure, Journal of parallel and distributed computing, 2012, 72(10), pp. 1318-1331.

[6] X. Meng, V. Pappas, L. Zhang, Improving the scalability of data center networks with traffic-aware virtual machine placement. IEEE INFOCOM, 2010, pp. 1-9.

[7] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, E. Silvera, A stable network-aware vm placement for cloud systems. IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid), 2012, pp. 498-506.

[8] A.M. Sampaio, J.G. Barbosa, R. Prodan, PIASA: a Power and Interference Aware Resource Management Strategy for Heterogeneous Workloads in Cloud Data Centers, Simulation Modelling and Practise Theory, 2015.

[9] Q. Li, An optimal algorithm for resource scheduling in cloud computing. In Advances in Multimedia, Software Engineering and Computing, 2011, Vol. 2, pp. 293-299.

[10] P. Graubner, M. Schmidt, B. Freisleben, Energy-efficient management of virtual machines in eucalyptus, IEEE International Conference on Cloud Computing (CLOUD), 2011, pp. 243-250.

[11] S. Srikantaiah, A. Kansal, A., F. Zhao, Energy aware consolidation for cloud computing. In Proceedings of the 2008 conference on Power aware computing and systems, 2008, Vol. 10, pp. 1-5.

[12] S.M. Johnson, Optimal two-and three-stage production schedules with setup times included, Naval research logistics quarterly, 1954, 1(1), pp. 61-68.

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

[14] J.W. Herrmann, Operations Scheduling with Applications in Manufacturing and Services, Journal of Scheduling, 5(1), 2002, 95-96.

[15] P. Brucker, Scheduling Algorithms. Springer 1998.

[16] H.J. Jiang, K.C. Huang, H.Y. Chang, D.S. Gu, J.P. Shih, Scheduling concurrent workflows in HPC cloud through exploiting schedule gaps. In International Conference on Algorithms and Architectures for Parallel Processing, 2011, pp. 282-293.

[17] C.W. Tsai, J.J. Rodrigues, Metaheuristic scheduling for cloud: A survey. IEEE Systems Journal, 2014, 8(1), 279-291.

[18] I.S. Moreno, P. Garraghan, P. Townend, J. Xu, Analysis, modeling and simulation of workload patterns in a large-scale utility cloud. IEEE Transactions on Cloud Computing, 2014, 2(2), 208-221.

[19] L. Wang, E. Gelenbe, Adaptive dispatching of tasks in the cloud, IEEE Transactions on Cloud Computing 2015.

[20] A. K. Mishra, J.L. Hellerstein, W. Cirne, C.R. Das, Towards Characterizating Cloud Backend Workloads: Insights from Google Computer Clusters, SIMETRICS Performance Evaluation Review, 2010, vol. 37, pp. 34-41.

[21] S. Kavulya, J. Tan, R. Gandhi, P. Narasimhan, An Analysis of Traces from a Production MapReduce Cluster, IEEE/ACM International Conference on Cluster, Cloud, and Grid Computing, 2010, pp. 94-103.

[22] I. S. Moreno, R. Yang, J. Xu, T. Wo, Improved energy-efficiency in cloud datacenters with interference-aware virtual machine placement, IEEE International Symposium on Autonomous Decentralized Systems (ISADS), 2013, pp. 1-8.

[23] S. F. Piraghaj, et al. "Virtual Machine Customization and Task Mapping Architecture for Efficient Allocation of Cloud Data Center Resources", The Computer Journal,, vol. 59, no.2, 2016, pp. 208 – 224.

[24] M. Rahman, X. Li, H. Palit, Hybrid heuristic for scheduling data analytics workflow applications in hybrid cloud environment. IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011, pp. 966-974.

[25] P.Y. Yin, S.S. Yu, P.P. Wang, Y.T. Wang, A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems. Computer Standards & Interfaces, 2006, 28(4), pp. 441-450.

[26] C. Reiss, J. Wilkes, and J. Hellerstein, "Google Cluster-Usage Traces: Format + Schema," Google Inc. [https://drive.google.com/file/d/0B5g07T_gRDg9Z0lsSTEtTWtpOW8/view]

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

[28] C. Wilke, S. Götz, S. Richly, JouleUnit: a generic framework for software energy profiling and testing, ACM workshop on Green in/by software engineering, 2013, pp. 9-14 ACM.

[29] A. Beloglazov, R. Buyya, "Energy Efficient Resource Management in Virtualized Cloud Data Centers," IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010, pp. 826-831.

[30] H. Qiang, et al., "Power Consumption of Virtual Machine Live Migration in Clouds," International Conference on Communications and Mobile Computing (CMC), 2011, pp. 122-125.