BRAHMA: An Intelligent Framework for Automated Scaling of Streaming and Deadline-critical Workflows

Ankita Atrey, Hendrik Moens, Gregory Van Seghbroeck, Bruno Volckaert, Filip De Turck
INTEC-IBCN-iMinds, Ghent University, Gent, Belgium
{ankita.atrey,hendrik.moens,gregory.vanseghbroeck, bruno.volckaert,filip.deturck}@intec.ugent.be

***Abstract*—The prevalent use of multi-component, multi-tenant models for building novel Software-as-a-Service (SaaS) applications has resulted in wide-spread research on automatic scaling of the resultant complex application workflows. In this paper, we propose a holistic solution to *Automatic Workflow Scaling* under the combined presence of *Streaming* and *Deadline-critical* workflows, called *AWS-SD*. To solve the AWS-SD problem, we propose a framework *BRAHMA*, that learns workflow behavior to build a knowledge-base and leverages this info to perform intelligent automated scaling decisions. We propose and evaluate different resource provisioning algorithms through *CloudSim*. Our results on time-varying workloads show that the proposed algorithms are effective and produce good cost-quality trade-offs while preventing deadline violations. Empirically, the proposed hybrid algorithm – combining learning and monitoring, is able to restrict deadline violations to a small fraction (3–5%), while only suffering a marginal increase in average cost per component of 1–2% over our baseline naïve algorithm, which provides the least costly provisioning but suffers from a large number (35–45%) of deadline violations.**

***Index Terms*—Cloud simulation, Resource provisioning, Streaming and Deadline-Critical Workflows, Deadlines, SLA**

I. Introduction

Ubiquity and *pervasiveness* of the *Cloud* is no longer a new phenomenon. Cloud enabled services have become an integral part of the day-to-day life of almost every Internet user. On the one hand, cloud users enjoy *flexible* and *cost-effective* usage of various cloud services, however on the other hand, providing *quality of service* while maintaining cost-effectiveness, service level agreements (*SLAs*), scalability and *deadline* constraints, is the paramount concern of various service providers.

With the ever increasing use of multi-component, multi-tenant models [13], [32] for building SaaS applications and the *exponential* rise in popularity of cloud based services, automatically and correctly scaling these services at runtime is of paramount importance. Key challenges are: (1) Scaling the resulting application up or down depending on monitored user/tenant load in order to keep the SLA, no longer becomes an issue of scaling resources for a single service, but instead results in a complex problem of scaling all individual service endpoints in the workflow and (2) In a real-world setting, application workflows can possess a host of characteristics: ranging from execution flows being either *streaming* or *sequential* (details in Sec. III) to their deadlines being strict or fuzzy.

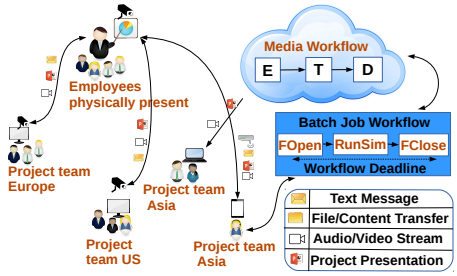


Fig. 1: Online collaborative meeting room use-case: the project lead is presenting to teams located across the globe, with attendees able to trigger ASAP workflows during the session.

There is no dearth of *real-world* scenarios where *streaming* and *deadline-critical* (*ASAP*) workflows occur concurrently. An obvious use case is an elastic based, multi-tenant online media cooperation / meeting room service (Fig. 1), consisting of both streaming and ASAP workflows, where user(s) are often required to trigger analytical workloads during the course of a meeting. Every stream consists of an encoder, a transcoder and a decoder, all of which have different multi-tenant SLA requirements in an attempt to provide a flawless service (no A/V interruptions, no stuttering, etc.). Each attendee can additionally trigger ASAP workflows during the meeting to e.g. run simulations, data analytical workloads etc. These ASAP workflow instantiations usually possess strict deadlines and can consist of one or more (service) components executing sequentially, thereby exhibiting runtime characteristics which differ from streaming workflows.

We propose *BRAHMA*, which uses machine learning to learn workflow behavior and curates a *knowledge base* (KB) to aid in making informed resource provisioning decisions. BRAHMA is comprised of classification and clustering modules, which analyse the resource request patterns of workflows to predict whether a new workflow will meet its deadline or not, and clusters workflows into groups possessing similar resource requirements. Our proposed algorithms use the information curated in the KB to take appropriate workflow scaling decisions. More specifically, the monitoring-based *proactive* algorithm proposed in our previous work [6] is used to scale the streaming workflows, while a *hybrid* method combining monitoring with the learning framework exposed by BRAHMA is used for the ASAP workflows.

In summary, this paper proposes a *holistic* solution to

978-3-901882-85-2

the automatic workflow scaling problem under the combined presence of streaming and ASAP workflows, called *AWS-SD* (Sec. III). Key contributions are as follows:

- A framework called BRAHMA (Sec. IV), which learns workflow behavior and stores this in a knowledge base.
- Algorithms (Sec. IV-C) that leverage BRAHMA to maintain SLAs and deadlines for streaming and ASAP workflows respectively, while keeping cost in line.
- Empirical analysis (Sec. VI) showing these algorithms to provide good *cost-quality* trade-offs while keeping *SLAs* in line and preventing *deadline* violations.

II. Related Work

Cloud computing has spread to a wide-variety of domains like health care [11], Social Science/Mobile Cloud Computing [19] etc. A lot of research in this area revolves around virtualisation and resource scheduling. The targeted SaaS cloud platform is based on a multi-tenant model, where different user application's components can share the resources (VMs).

Literature has witnessed a plethora of works for *automatic workflow scaling* [7], [12], [27] with focus on maintaining quality of service parameters like SLAs [5], [6], [9], [21], [26], [33]–[35], deadlines [15], [25], [31] and many more. Research done by [7], [16], [30] discusses semi-automatic and automatic scaling for multimedia services. Soltanian et. al. [30] enlightened a subproblem related to media services scaling. Our previous work [6], focused solely on scaling and resource provisioning for streaming / non-ASAP workflows.

Despite wide-spread research in the area of workflow scaling, to the best of our knowledge none of the existing methods are capable of jointly scaling streaming and deadline-critical workflows. To this end, the work presented in this paper extends on [6] by adding support for deadline critical jobs, using an enabling framework and algorithms for provisioning cloud resources to ASAP jobs spawned from real-time streaming workflows (the latter having per-workflow component SLAs in order to guarantee seamless A/V streaming).

SLA based resource provisioning [9], [26], [33], [35] in cloud computing is another important line of research. [33] introduced an admission control method in order to avoid SLA violations. This work was further extended by L. Wu et al. [34], where different algorithms based on resource reservation and request rescheduling are proposed in order to improve the Customer lever satisfaction (CSL) and minimize the SLA violation. Other important research is dealing with workflow scheduling with strict deadlines [20], [4], [25], [22], [28], [23] [29]. One of the most popular works by Poola et al. [22] represented a robust resource scheduling algorithm. Their work targets robust and fault-tolerant scheduling algorithms with three multi-objective resource allocation policy.

III. Problem Description

This section presents a concise model of streaming and ASAP workflows. A SaaS application is defined as a workflow $W_j \in \mathcal{W}$ consisting of one or more components $C_{kj} \in W_j$ (see Fig. 2), where individual service components pass their

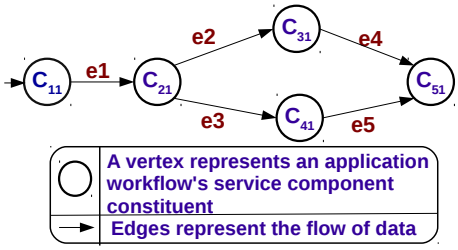


Fig. 2: An application workflow W_1 composed of multiple service components and inter-component data flows.

data along the workflow edges to the subsequent service component. These workflows can possess a wide-variety of characteristics in terms of execution flow, resource requirements etc.

In case of streaming workflows ($W_j \in \mathcal{W}_s$), components receive streaming data from the components which serve as input, while they themselves stream their output data to the next workflow components. Each streaming workflow service component possesses a separate SLA agreement which defines its minimal resource requirements (in terms of processing power, memory, storage etc.) to ensure proper execution. Note that, using this minimum resource requirement, we define the maximum number of components $\mathcal{N}^i_{max}$ that can simultaneously run on a VM $\mathcal{V}_i$ while ensuring SLAs are met.

In case of ASAP workflows ($W_j \in \mathcal{W}_{dc}$, analogous to the simulation-input-retrieval→run-simulation→simulation-output-storage operation discussed in Sec. I), the execution flow is sequential. Here, the data from one service component moves to the subsequent workflow component(s), once the former finishes processing and hence passes its full output to the latter. Similar to streaming workflows, each ASAP workflow possesses a deadline-constraint ($DC_{\mathcal{W}_j}$) which is used to identify a VM $\mathcal{V}_i$ that possesses the desired resources (in terms of processing power, memory, storage etc.) to ensure proper working of the workflow according to its specifications.

Given our use case of online collaborative A/V meetings, focus lies on the combination of streaming and ASAP workflows. Note that tenant requests for streaming and ASAP workflows follow time-varying distributions $\mathcal{D}_s(t)$ and $\mathcal{D}_{dc}(t)$ respectively. While streaming workflows do not benefit from assigning more resources to them than required, as one cannot *'speed up'* a meeting, ASAP workflows benefit from being allocated to more powerful resources. Owing to this difference in characteristics, jointly scaling service end-points of streaming and ASAP workflows is a challenging problem, defined as:

Problem. *Given a VM pool $\mathcal{V}$, a set of workflow requests ($\mathcal{W}$) consisting a combination of streaming ($\mathcal{W}_s$) and ASAP ($\mathcal{W}_{dc}$) workflow requests, following time varying distributions $\mathcal{D}_s(t)$ and $\mathcal{D}_{dc}(t)$ respectively, the maximum number of allowed requests ($\mathcal{N}^i_{max}$) and the processing power in MIPS (M_i) for each VM ($\forall V_i \in \mathcal{V}$), perform automatic resource provisioning to keep the SLAs ($SLA^{C_{kj}}_{status} = true$) and the deadline-constraints ($DEADLINE^{W_j}_{status} = true$) for all the workflow components $C_{kj} \mid \forall k, C_{kj} \in W_j, \forall j, W_j \in \mathcal{W}$, while retaining high cost-efficiency and quality of service for SaaS applications.*

IV. BRAHMA Framework

In this section, the BRAHMA framework and its associated resource-provisioning algorithms are described. The building blocks required for BRAHMA are:

- **VM Allocation:** facilitates on demand creation of new VM instances based on a specific VM template from the pool of VMs $\mathcal{V}$. VM allocations under the *naïve* algorithm are performed in the beginning and remain fixed, whereas they are adapted based on the resource request patterns for the *advanced* and the *hybrid* algorithms.
- **Classification:** analyses the resource request patterns of ASAP workflows and learns a decision boundary capable of identifying whether the deadline of a workflow would be met or violated. Main benefit is the ability to predict the $DEADLINE_{status}$ of incoming ASAP workflows, facilitating better provisioning decisions.
- **Clustering:** facilitates fine-grained analysis of behavior exhibited by ASAP workflows. Here, resource request patterns are clustered, creating groups of ASAP workflows with similar resource requirements. With these clusters formed, it is easier to devise customized and informed resource provisioning strategies for each identified group. Any newly incoming ASAP workflow can be assigned to its most similar group, and hence utilize the already devised resource provisioning strategy.
- **Workflow Monitoring:** Keeps track of the progress for each component C_{kj} of a workflow W_j. Continuously probes the workflow components to monitor the time remaining for the component to finish execution. The monitoring capability plays a central role in the design of the more involved *hybrid* algorithm.

A. Learning Phase

BRAHMA (Fig. 3), operates in two phases. In the first, i.e. the *learning phase*, BRAHMA takes a large number of workflows (training data) generated using the *scenario generator* module. To facilitate robustness and generalizability of the learned models, the scenario generator creates a proper mix of ASAP workflows of varying number of components, component types etc. Each generated workflow, possesses resource requirements (in MI) for each of its constituent component, while also containing information about its deadline status (i.e.

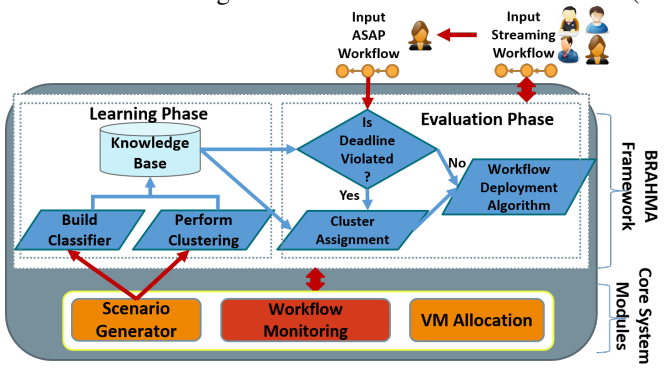


Fig. 3: Overview of the BRAHMA Framework.

Algorithm 1 Workflow Deployment Algorithm

Require: $\mathcal{V}$, $DC_{W_j} \mid \forall W_j \in \mathcal{W}$, $\mathcal{W} \sim \mathcal{D}(t)$, $provisionType$, $workflowType$, τ
Ensure: SLA_{status}, $DEADLINE_{status}$, η, $AvgCost$

1: **for** $t = 0$ to t_{max} **do**
2: $\quad \mathcal{W}^t \sim \mathcal{D}(t)$
3: $\quad$ **if** $workflowType = Streaming$ **then**
4: $\quad\quad \{SLA_{status}, AvgCost_s\} \leftarrow ProactiveDeploy(\mathcal{W}^t_s, \mathcal{V}, \tau = 0.6)$ ▷ Sec. 4 [6]
5: $\quad$ **else** //workflowType = *ASAP*
6: $\quad\quad numASAPDeploy \leftarrow |\mathcal{W}^t_{dc}|$
7: $\quad\quad$ **if** $numASAPDeploy \geq 0$ **then**
8: $\quad\quad\quad$ **if** $provisionType = Advanced$ **then**
9: $\quad\quad\quad\quad \{DEADLINE_{status}, \eta, AvgCost_{dc}\} \leftarrow KB(\mathcal{W}^t_{dc}, \mathcal{CM}, \mathcal{CC}, \mathcal{V}, Advnce)$
10: $\quad\quad\quad$ **else** //provisionType = *Hybrid*
11: $\quad\quad\quad\quad \{DEADLINE_{status}, \eta, AvgCost_{dc}\} \leftarrow KB(\mathcal{W}^t_{dc}, \mathcal{CM}, \mathcal{CC}, \mathcal{V}, Hybrid)$
12: $\quad AvgCost \leftarrow AvgCost_s + AvgCost_{dc}$

was the deadline broken or met). The first task of BRAHMA's learning phase is that of building a classifier. Here, we use the classification module described in the previous section to analyse the generated training data and learn a classifier model $\mathcal{CM}$, based on the resource request patterns, to predict whether the deadline of an ASAP workflow is going to be met.

Moving ahead, BRAHMA clusters similar workflows (based on the resource requirement pattern of its constituent components) from the training data to form semantically meaningful groups $\mathcal{CC}$. This allows to analyse workflow behavior at a finer level of granularity, facilitating appropriate resource provisioning decisions. Eventually both the classifier model $\mathcal{CM}$ and the created set of clusters along with their cluster centers $\mathcal{CC}$, are curated in the *Knowledge Base (KB)*. Note that the KB is curated by learning from historical/current resource request patterns and no information is required in advance.

B. Evaluation Phase

In the *evaluation phase*, new streaming requests along with triggered ASAP requests are submitted to BRAHMA for inferring their execution behavior, resource requirements and deadline status. As a first step, BRAHMA probes the $\mathcal{CM}$ saved in the KB to predict the $DEADLINE_{status}$, i.e., whether the workflow under consideration would meet its deadline or not. If the deadline is going to be met, then there is no need to perform any specialized resource scaling, as the already assigned resources will be sufficient to meet the deadline-constraint of the workflow. However, if a violation is predicted, we query the KB's $\mathcal{CC}$ to assign this workflow to the cluster closest/most-similar to it in terms of exhibited resource requirements, guiding the resource provisioning algorithms.

C. Knowledge Base driven Resource Provisioning Algorithms

Algorithm 1 presents the pseudo-code of a generic algorithm for the deployment of streaming and ASAP workflows. As workflow requests follow a time-varying distribution $\mathcal{D}(t)$, we sample requests at different discrete time-instants, $t \in [0, t_{max}]$, denoted as $\mathcal{W}^t \sim \mathcal{D}(t)$ (line 2). If the workflow under consideration is a *streaming* workflow, we invoke the proactive algorithm proposed by us in [6] with $\tau = 0.6$ to scale its services up/down, while completely avoiding SLA violations and maintaining high cost-efficiency. On the other hand, the VM assignment of ASAP workflows is performed using the KB driven algorithms, namely – advanced and hybrid (lines 8–11), which are described next, in detail.

Alg. 2 presents the pseudo-code for the advanced and the hybrid algorithms. Both of them incorporate the use of the curated information from the knowledge base (KB) constructed by BRAHMA, and are thus highly similar in design with the only difference being the VM reservation and workflow migration procedure. As a first step, these algorithms invoke the classification model $\mathcal{CM}$ stored in the KB, to predict the $DEADLINE_{status}$ of each ASAP workflow (line 5). The workflows with the predicted $DEADLINE_{status} = true$ do not need any specialized scaling and thus, they are assigned to a "medium" VM (lines 6–8). For workflows whose deadlines are predicted to be violated, these algorithms query the $\mathcal{CC}$ stored in the KB and try to identify the cluster, which possesses workflows with the most similar requirement patterns (line 10). Next, with this derived information, the workflow components are assigned appropriate resources accordingly (lines 11–23).

A notable limitation of the KB driven algorithms are that, the resources are not pre-reserved, and hence, they are prone to suffer from various penalties incurred owing to new VM reservations and migration of workflow components from one VM to another. The *advanced* algorithm suffers from both of the previously stated penalties (line 15). On the other hand, the *hybrid* algorithm incorporates the use of *monitoring* (similar to the proactive algorithm in [6]) to continuously track the progress of an executing component. More specifically, for every clock tick Δt, a monitor event tracks the execution status of a currently running component C_{kj}, and as soon as the time left for its execution to complete, crosses the VM reservation and migration time $T^{V_l}_{reserve} + T^{V_l}_{migrate}$, a new VM reservation is triggered. This enables timely reservation of new VMs and migration of components, thereby mitigating the incurred penalties completely (lines 17–19). Thus, the cost-efficiency of hybrid is significantly better than the advanced algortithm.

Algorithm 2 KB driven Algorithm

Require: $\mathcal{V}$, $DC_{W_j} \mid \forall W_j \in \mathcal{W}$, $\mathcal{W}_{dc} \sim \mathcal{D}_{dc}(t)$, $\mathcal{CM}$, $\mathcal{CC}$, $provisionType$
Ensure: $DEADLINE_{status}$, η, $AvgCost_{dc}$

1: **procedure** KB($\mathcal{W}^t_{dc}, \mathcal{V}, \mathcal{CM}, \mathcal{CC}, provisionType$)
2: $AvgCost_{dc} \leftarrow 0$; $Penalty_{dc} \leftarrow 0$; $\eta \leftarrow 0$
3: **for** each $W_j \in \mathcal{W}^t_{dc}$ **do**
4: $MI_{W_j} \leftarrow 0$; $assignedMIPS_{W_j} \leftarrow 0$; $DEADLINE^{W_j}_{status} \leftarrow true$
5: $DEADLINE^{W_j}_{status} \leftarrow \mathcal{CM}_{predict}(\{C_{1j}, C_{2j}, \ldots, C_{kj}\} \in W_j)$
6: **if** $DEADLINE^{W_j}_{status} = true$ **then**
7: Deploy W_j on a pre-reserved "medium" VM V_i
8: $AvgCost_{dc} \leftarrow AvgCost_{dc} + (M_i + C_i + S_i) \times |W_j|$
9: **else**
10: Assign W_j to the closest cluster center $cc \in \mathcal{CC}$
11: **for** each $C_{kj} \in W_j$ **do**
12: $MI_{W_j} \leftarrow MI_{W_j} + MI_{C_{kj}}$
13: Deploy C_{kj} on VM V_l with $MIPS_l \geq MI_{C_{kj}}$
14: **if** $provisionType = Advnce$ **then**
15: $Penalty_{dc} \leftarrow Penalty_{dc} + P^{V_l}_{reserve} + P_{migrate}$
16: **else** //provisionType=Hybrid
17: Monitor the progress of C_{kj} for every Δt; $t_{cur} \leftarrow t_{cur} + \Delta t$
18: **if** $t_{C_{kj}} - t_{cur} = T^{V_l}_{reserve} + T^{V_l}_{migrate}$ **then**
19: Initiate reservation for VM V_l
20: $assignedMIPS_{W_j} \leftarrow assignedMIPS_{W_j} + MIPS_l$
21: $AvgCost_{dc} \leftarrow AvgCost_{dc} + (M_l + C_l + S_l)$
22: **if** $(MI_{W_j} / assignedMIPS_{W_j}) > DC_{W_j}$ **then**
23: $DEADLINE^{W_j}_{status} \leftarrow false$; $\eta \leftarrow \eta + 1$
24: $AvgCost_{dc} \leftarrow (AvgCost_{dc} + Penalty_{dc}) / |\mathcal{W}^t_{dc}|$; $\eta \leftarrow \eta / |\mathcal{W}^t_{dc}|$
25: **return** $DEADLINE_{status}, \eta, AvgCost_{dc}$

V. Experimentation Setup

A. Media Workflows

The media workflow illustrated in Fig. 4, represents an instance of a *streaming* workflow with three components namely encoder, transcoder and decoder. These streaming workflows can trigger multiple ASAP workflows that correspond to deadline-critical jobs like e.g. running a simulation, decision support, data analysis. Each service component is executed on a VM V_i chosen from the pool of available VMs $\mathcal{V}$. Service components corresponding to streaming workflows, possess an SLA which can be either met or broken. Moreover, each ASAP workflow possesses a deadline-constraint, which, similar to streaming workflows, can be either met or broken, and if broken causes delays in e.g. simulation/data analysis tasks. Note that even though much more elaborate workflows exist, these particular workflows have been chosen to showcase the strength of BRAHMA and the presented algorithms.

B. Evaluation Scenario

As shown in Fig. 5, 200 user requests for streaming workflows are generated following a normal distribution, with the time 12 noon set as mean and 3.5 hours as standard deviation. At every time instant, each generated streaming worfklow further possesses a 5% chance of triggering an ASAP workflow. This graph portrays that the number of requests for both streaming and ASAP workflows will vary in between the start of the day up to the end of the day.

ASAP workflows possess varying characteristics, namely – number of components, type of resource requirements etc. To this end, our scenario generator module, generates a variety of three, four and five component workflows, where the resource requirements (in terms of million-instructions (MI)) for each component follow the templates as shown in Table I. For example, a map-reduce job could follow the <low→high/very-high→low→high/very-high→low> template, etc. The deadline-constraint, in terms of MI requirements, for an ASAP workflow W_j possessing k components is calculated as k times the expected component resource requirement. For each user/tenant request a new instance of the workflow W_j is created and the constituent service components $C_{kj}, \forall k \mid C_{kj} \in W_j$ are provisioned on different VMs V_i, available from the VM pool $\mathcal{V}$ (the choice of which VM and how this VM pool grows / shrinks is driven differently depending on the choice of algorithm). To deploy VMs in the resource pools, eight types of VM images were defined, as detailed in Table II. The costs for the VM templates used, were

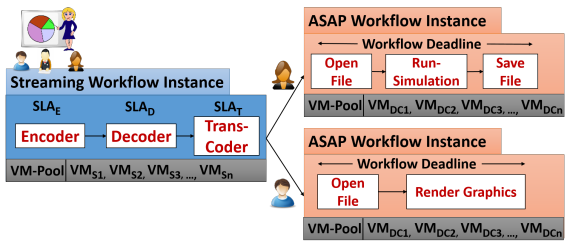


Fig. 4: Streaming workflows spawning ASAP workflows.

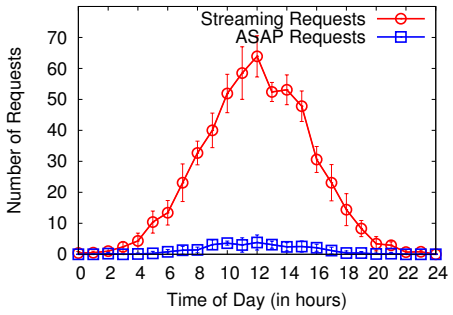


Fig. 5: Nnumber of worfklow requests based on time of day.

TABLE I: **Resource Requirement Templates**

Template Name	Very-Low	Low	Medium	High	Very-High
Resource requirement range (MI)	100 – 300	250 – 550	500 – 800	750 – 1050	1000 – 1200

parameterized based on the Amazon EC2 image $c3.8x$large [2], with a monthly price of 1.680 to provide 32 vCPUs (17476 MIPS [1]), 60 GB of RAM and 2*320 GB of storage. This cost was divided equally between secondary-storage, main-memory and CPU, and the converted unit prices (per MB/hour and MI/hour [3]) were used to calculate the costs for the VM templates used in this paper. For the simulations, the values of the time required to reserve new VMs ($T^{V_i}_{reserve}$) and the time required to migrate one component from an existing VM instance to another ($T_{migrate}$) were defined as uniform distributions between [40s,55s] and [0.5s,2s] respectively.

C. Evaluation Metrics

We consider the following metrics:

- **Deadline status:** of a workflow W_j, running on a VM V_i, is defined as a binary variable which assumes the value of $false$ if the deadline is violated, or $true$ otherwise. The fraction of the workflows whose deadlines are violated is denoted by η. Mathematically,

$$\eta = \frac{1}{w}\Big(\sum_{j=1}^{w} I\Big) \tag{1}$$

  I is the indicator function: $I = 1$ if $DEADLINE^{W_j}_{status} = false$; and 0 otherwise.
- **VM Cost:** is defined as the sum of all costs related to resource usage. Thus, for a simulation with w workflow requests, each one with c service components, and M_k, S_k, C_k, representing, memory, storage and CPU costs respectively for a component C_k, we mathematically define the average VM cost as follows:

$$\frac{1}{w}\Big(\sum_{j=1}^{w}\Big(\sum_{k=1}^{c}(M_k + S_k + C_k)\Big)\Big) \tag{2}$$

- **Penalty:** is defined as the extra cost incurred (over-and-above the normal resource utilization costs) on components while waiting for (1) a new VM reservation $P_{reserve}$ and (2) migration of components from one VM to another $P_{migrate}$. We mathematically state the average Penalty as:

$$\frac{1}{w}\Big(\sum_{j=1}^{w}\Big(\frac{1}{c}\sum_{k=1}^{c}(P_{reserve_k} + P_{migrate_k})\Big)\Big) \tag{3}$$

TABLE II: **Parameterized VM Templates**

Template	**CPU**	**RAM**	**Storage**	**Hourly Cost ($)**
$Template_{01}$	150 MIPS	4 GB	128 GB	$0.154
$Template_{02}$	300 MIPS	8 GB	256 GB	$0.308
$Template_{03}$	450 MIPS	12 GB	384 GB	$0.462
$Template_{04}$	600 MIPS	16 GB	512 GB	$0.616
$Template_{05}$	750 MIPS	20 GB	640 GB	$0.77
$Template_{06}$	900 MIPS	24 GB	768 GB	$0.924
$Template_{07}$	1050 MIPS	28 GB	896 GB	$1.078
$Template_{08}$	1200 MIPS	32 GB	1024 GB	$1.232

TABLE III: **Classifier Parameters**

Classifier	**Parameters**
Decision Tree	Pruning Confidence Factor=0.25, Min. #leaf-instances=2
Random Forest	#Trees=50, #Random-features=3
Functional Tree	Min. #instances for node-splitting =15, #boosting-iterations=15

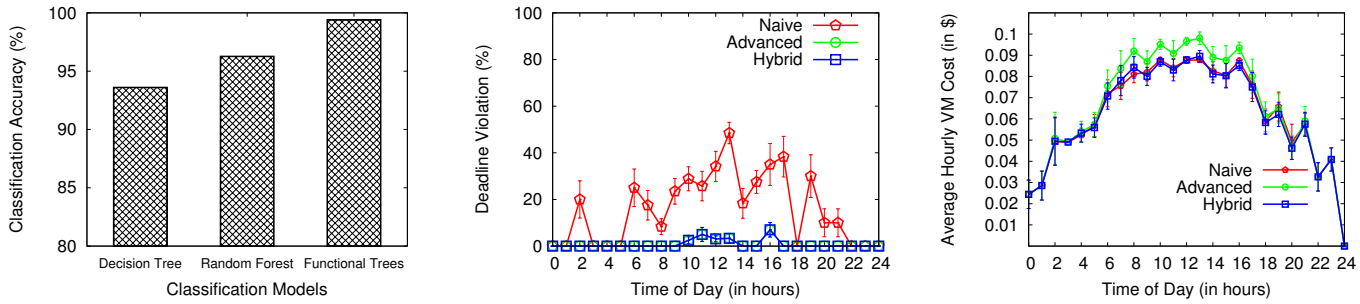
VI. Simulation Results

All simulations were executed using the extended CloudSim simulator [10], on an Intel(R) Core i5 4-core machine with a 1.7 GHz CPU and 8 GB RAM running Linux Ubuntu 15.04. We use the publicly available implementations of the classification and clustering models from the WEKA [17] data mining software. Results are averaged over 10 simulation runs.

We first analyse the performance of various classification modules used under the BRAHMA framework. As mentioned in Section V, we use the scenario generator module to generate 6000 ASAP workflow requests (for training the learning phase of BRAHMA) possessing varying (3, 4 and 5) lengths/number of service components. Using the deadline-constraint estimation discussed in the previous section, each ASAP workflow is then assigned a class label, i.e., whether the deadline of this workflow was violated or met. If the total MI requirements of an ASAP workflow is greater than the estimated deadline-constraint (in terms of MI), then the deadline is marked to be violated, while met otherwise. We use the *decision tree* (J48 algorithm) [24], *random forest* [8] and *functional tree* [14] methods for classification. A grid-search was performed to choose the optimal set of internal classifier parameters, summarized in Table III. The reader is referred to [17], for an in-depth understanding of these parameters.

Fig. 6a portrays the classification accuracy using 10-fold cross validation. It is clear that, functional tree classification possesses the highest accuracy ($\approx 99\%$), while the decision tree possesses the least ($\approx 94\%$). Nevertheless, using any of the three classifiers stated previously, BRAHMA is able to predict with a reasonably high accuracy, whether an ASAP workflow will violate its deadline or not.

With the capability to perform high-confidence deadline status predictions, next, we cluster the generated ASAP workflows for fine-grained near-optimal resource provisioning. We use the *k-means* algorithm [18] to cluster the ASAP workflows into groups with similar resource requirement patterns. The *silhouette coefficient* metric is used to correctly identify the optimal number of clusters (ρ) from the data. The higher the silhouette value, the better the produced clustering. To this end, we choose ρ as 9, 11 and 18 for the length 3, 4 and 5 workflows respectively. The clustering module is not only *statistically sound* but also *qualitatively effective*, as it



(a) Classifier Performance (b) ASAP Workflow Deadline Violation (%) (c) Average Hourly VM Cost

Fig. 6: (a) Performance of various classifiers under BRAHMA. (b) A comparison of the variation in the ASAP workflow deadline violation percentage and (c) the average total cost (combining costs for streaming and ASAP workflows) versus the time of day for the naïve, advanced and hybrid algorithm.

generates a wide-variety of semantically meaningful clusters, a representative set of which is described in Table IV.

To effectively evaluate the relevance of the proposed KB driven resource provisioning algorithms, we devised a baseline *naïve* algorithm.The naïve algorithm reserves all the resources with different properties (storage, CPU, memory etc.) at the beginning of the application session. Every incoming ASAP workflow is assigned to a pre-reserved "medium" (say Template_{04} in Table II) sized VM. An intuitive approach is to identify a VM possessing MIPS equal to the expected MI requirement of a workflow component as the medium-sized VM. The reason being that in expectation, this VM would be able to meet the deadline of half of the ASAP workflows.

The naïve algorithm possesses the least cost, owing to pre-assignment of resources and the lack of need for new VM reservations and workflow migrations, if any. Moreover, no new VM reservations happen even if $\forall V_i \in$ pre-reserved $\mathcal{V}$, the $MIPS_i$ is not sufficient to fulfill the requirements MI_{W_j} of a workflow W_j, in which case the deadlines get violated.

Fig. 6b presents a comparison of the naïve, advanced and hybrid algorithms in terms of the percentage of ASAP workflows whose deadline gets violated. Since the naïve algorithm does not perform intelligent resource provisioning, it suffers from a large number of deadline violations, ranging from as low as 15% to 45% in the worst-case. On the other hand, the advanced and the hybrid algorithms, which leverage the classification and clustering modules of BRAHMA, are able to better model the workflow request patterns and behavior, thereby performing informed resource provisioning and scaling. Thus, the percentage of workflows that suffer deadline violations is kept as low as 3–5%.

Lastly, we perform a comparison of the variation in average hourly VM costs for the proposed algorithms with the time of day. Note that, this analysis includes the costs for both streaming and ASAP workflows as well as the penalties incurred, if any. We use the *pro-active* algorithm, proposed by us in [6], for scaling streaming workflows. We set $\tau = 0.6$, thus, SLAs are always met [6], and hence no penalties are incurred due to SLA violations. The proposed naïve, advanced, and hybrid algorithms act in unison with the pro-active algorithm to jointly scale the combination of streaming and ASAP workflows. Fig. 6c shows that the naïve algorithm possesses the least cost. On the other hand, the *advanced* algorithm possesses the highest cost, owing to penalties incurred due to workflows waiting for new VM reservations and component migrations. To this end, the *hybrid* algorithm incorporates the *monitoring* capability as used in the *pro-active* algorithm, to mitigate the above discussed penalties. It is evident that, the *hybrid* algorithm closely mirrors the cost of the naïve algorithm, and thus, is as *cost-effective* as naïve.

To summarize, the *proactive* algorithm with $\tau = 0.6$ and the *hybrid* algorithm, which intelligently congregates the *monitoring* capability of the former with the fine-grained decision making capability exposed by BRAHMA, serve as the best possible trade-off for minimizing the costs while also keeping the SLAs and the deadline-constraints of the workflows in line.

TABLE IV: **Sample mapping of identified resource requirement patterns / clusters to workflow types**

Resource Requirement Patterns	Job Type
Low→High/Very-High→Low	File open, Run-Simulation, File close
Low→High/Very-High→Low→High/Very-High→Low	A basic Map-Reduce job
High/Very-High→Low→Low→High/Very-High	Indexing & Querying job
High→Medium→Low	A Post-processing job

VII. Conclusion

In this paper, we addressed the problem of *Automatic Workflow Scaling* under the combined presence of *Streaming* and *Deadline-critical* workflows, called *AWS-SD*. We identified the need for methods that are capable of jointly scaling these workflows. Consequently, we devised a holistic solution to the AWS-SD problem; by coming up with a framework *BRAHMA* that curates a knowledge base (KB) of learned workflow behavior(s), and knowledge base driven resource provisioning algorithms – *advanced* and *hybrid*, that leverage BRAHMA for effective scaling of such workflows. Our empirical studies show that our algorithms are *effective* and provide good cost-quality tradeoffs while preventing deadline violations. In the future, we will implement a BRAHMA prototype running on OpenStack and evaluate its runtime behavior while scaling an elastic A/V collaborative cloud-based service.

Acknowledgment

This research is partly funded by the IWT SBO DeCoMAdS project.

REFERENCES

[1] "Amazon EC2 MIPS," 2013, http://www.cmips.net/category/cup-results/.

[2] "Amazon EC2 Images," 2015, http://aws.amazon.com/pt/ec2/instance-types/.

[3] "Benchmarking the new Amazon c4 Instances," 2015, http://www.cmips.net/tag/intel-xeon-e5-2666-v3-2-90ghz/, Last accessed on May 31, 2015.

[4] S. Abrishami and M. Naghibzadeh, "Deadline-constrained Workflow Scheduling in Software-as-a-Service Cloud," *Scientia Iranica*, vol. 19, no. 3, 2012.

[5] A. F. Antonescu and T. Braun, "SLA-Driven Simulation of Multi-Tenant Scalable Cloud-Distributed Enterprise Information System," in *ARMS-CC*, 2014.

[6] A. Atrey, H. Moens, G. V. Seghbroeck, B. Volckaert, and F. D. Turck, "Design and Evaluation of Automatic Workflow Scaling Algorithms for Multi-tenant SaaS," in *Proceedings of the 6th International Conference on Cloud Computing and Services Science*, 2016.

[7] D. Bellenger, J. Bertram, A. Budina, A. Koschel, B. Pfander, and C. Serowy, "Scaling in Cloud Environments," in *WSEAS*, 2011.

[8] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[9] R. Buyya, S. K. Garg, and R. N. Calheiros, "SLA-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions," in *CSC*, 2011.

[10] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Softw. Pract. Exper.*, vol. 41, no. 1, 2011.

[11] P. C. Church, Y. Wang, Q. Xie, and M. Pedram, "Task Scheduling with Dynamic Voltage and Frequency Scaling for Energy Minimization in the Mobile Cloud Computing Environment," *IEEE Transactions on Services Computing*, vol. 8, no. 2, 2014.

[12] J. Espadasa, A. Molinab, G. Jimneza, M. Molinab, R. Ramreza, and D. Conchaa, "A Tenant-based Resource Allocation Model for Scaling Software-as-a-Service Applications over Cloud Computing Infrastructures," *FGCS*, vol. 29, no. 1, 2013.

[13] J. Fiaidhi, I. Bojanova, J. Zhang, and L.-J. Zhang, "Enforcing Multitenancy for Cloud Computing Environments," *IT Professional*, vol. 14, no. 1, 2012.

[14] J. Gama, "Functional Trees," *Machine Learning*, vol. 55, no. 3, pp. 219–250, 2004.

[15] T. A. L. Genez, L. F. Bittencourt, and E. R. M. Madeira, "Workflow scheduling for SaaS / PaaS cloud providers considering two SLA levels," in *NOMS*, 2012, pp. 906–912.

[16] R. H. Glitho, "Cloud-based Multimedia Conferencing: Business model, Research agenda, State-of-the-art," in *CEC*, 2011.

[17] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, 2009.

[18] J. Hartigan and M. Wong, "Algorithm AS 136: A K-means clustering algorithm," *Applied Statistics*, pp. 100–108, 1979.

[19] X. Lin and A. M. Goscinski, "A Survey of Cloud-Based Service Computing Solutions for Mammalian Genomics," *IEEE Transactions on Services Computing*, vol. 7, no. 4, 2014.

[20] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost and Deadline-constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds," 2012.

[21] H. Morshedlou and M. Meybodi, "Decreasing Impact of SLA Violations: A Proactive Resource Allocation Approach for Cloud Computing Environments," *IEEE TCC*, vol. 2, no. 2, 2014.

[22] D. Poola, S. K. Garg, R. Buyya, Y. Yang, and K. Ramamohanarao, "Robust Scheduling of Scientific Workflows with Deadline and Budget Constraints in Clouds," in *IEEE-AINA*, 2014.

[23] D. Poola, K. Ramamohanarao, and R. Buyya, "Enhancing Reliability of Workflow Execution Using Task Replication and Spot Instances," *TAAS*, vol. 10, no. 4, 2016.

[24] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[25] M. A. Rodriguez and R. Buyya, "Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, 2012.

[26] D. Serrano, S. Bouchenak, Y. Kouki, F. A. de Oliveira Jr., T. Ledoux, J. Lejeune, J. Sopena, L. Arantes, and P. Sens, "SLA guarantees for cloud services," *Future Generation Computer Systems*, vol. 54, no. *, 2016.

[27] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "CloudScale: Elastic Resource Scaling for Multi-tenant Cloud Systems," in *SoCC*, 2011.

[28] J. Shi, J. Luo, F. Dong, J. Zhang, and J. Zhang, "An Enhanced Workflow Scheduling Strategy for Deadline Guarantee on Hybrid Grid/Cloud Infrastructure," *Journal of Applied Science and Engineering*, vol. 18, no. 1, 2015.

[29] ——, "Elastic Resource Provisioning for Scientific Workflow Scheduling in Cloud under Budget and Deadline Constraints," *Cluster Computing*, vol. 19, no. 1, 2016.

[30] A. Soltanian, M. A. Salahuddin, H. Elbiaze, and R. Glitho, "A Resource Allocation Mechanism for Video Mixing as a Cloud Computing Service in Multimedia Conferencing Applications," in *CNSM*, 2015.

[31] G. L. Stavrinides and H. D. Karatza, "A Cost-Effective and QoS-Aware Approach to Scheduling Real-Time Workflow Applications in PaaS and SaaS Clouds," in *FiCloud*, 2015.

[32] W.Tsai and P. Zhong, "Multi-Tenancy and Sub-Tenancy Architecture in Software-as-a-Service (SaaS)," in *SOSE*, 2014.

[33] L. Wu, S. K. Garg, and R. Buyya, "SLA-based admission control for a Software-as-a-Service provider in Cloud computing environments," *JCSS*, vol. 78, no. 5, 2012.

[34] L. Wu, S. K. Garg, S. Versteeg, and R. Buyya, "SLA-Based Resource Provisioning for Hosted Software-as-a-Service Applications in Cloud Computing Environments," *IEEE TSC*, vol. 7, no. 3, 2014.

[35] L. Wu, S. K. Garg, and R. Buyya, "SLA-based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments," in *CCGrid*, 2011.