Journal of King Saud University – Computer and Information Sciences 34 (2022) 342–353

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

RTSLPS: Real time server load prediction system for the ever-changing cloud computing environment

Hajer Toumi [a,b,*], Zaki Brahmi [a,c], Mohhamed Mohsen Gammoudi [a,d]

[a] RIADI Lab, Manouba University, Tunisia
[b] Faculty of Sciences, ElManar, Tunisia
[c] Taiba University, Saudi Arabia
[d] ISAMM, Manouba University, Tunisia

ARTICLE INFO

Article history:
Received 31 August 2019
Revised 18 November 2019
Accepted 10 December 2019
Available online 20 December 2019

Keywords:
Cloud computing
Concept drift
Server load
Stream mining
Real time prediction
User behavior

ABSTRACT

Cloud Computing (CC) proposes a multi-tenant framework used by multiple concurrent users, each of which exhibits different and varied behavior. Such heterogeneity shapes a highly fluctuating load and creates new usage patterns overtime at the server level. Virtual Machines (VMs) interference also plays a big part in inducing changes at server load. Server load prediction is deemed crucial to ensure efficient resource usage. The execution of real-time interactive tasks constitutes an important part of CC. Thus, we propose, in this paper, a real-time server load prediction system based on incoming task classification and VM interference detection. The incoming task classification is used to capture the incoming workload trend and VM interference detection aims to capture the interference rate. Finally, load prediction considers server actual resources' usage, VM interference rate, and incoming workload trend. We propose an improved version of Hoeffding Adaptive Tree (HAT), augmented by ensemble drift detectors. Results show that our Real-Time Server Load Prediction System (RTSLPS) was able to deliver great flexibility dealing with changes and very good accuracy with quick evaluation time and a small memory footprint.

1. Introduction

According to Vic (2011) *CC refers to applications and services, which data centers around the world make available to users who have an Internet connection.* Thus CC can be seen as a form of client-server relationship Clarke (2010) where the user is responsible for driving the volume and behavior of task in term of requested resource and volume of task submission Ismael et al. (2013). The agility of Cloud framework made it the best choice for a wide variety of applications related to a heterogeneous mix of workloads. Interactive applications constitute a big part of CC with 3.6 billion consumer users in 2018 (Lobert, 2019). As the number of users continues to grow, CC providers must handle massive requests whilst ensuring, on one hand, the Quality of Services (QoS) and on the other hand, maximize their profits with lower investment. To ensure the on-demand availability of resources and the respect of SLA, proactive resources provisioning is deemed as a fundamental task. It seeks to predict future server load and provide the appropriate resources to a suitable workload accordingly. However, server load prediction in CC is a very delicate task as it requires an in-depth understanding of Cloud workload characteristics. Unlike greed computing, CC workload tends to be dynamic and highly variable (Di et al., 2014). The effect of user behavior on load variation is obvious as recently the debut of the actor Jennifer Aniston on Instagram caused a temporarily crash of the platform as fans flocked to follow her account leading to an overload on the provider side (Brennan, 2019). Such behavior cannot be anticipated from server historic, thus, considering the evolution of user behavior during the prediction is crucial to understand the future trend of servers load and to put in place an efficient load balancing strategy. In this regard, many studies were devoted to understand the impact of user behavior on CC workload variations (Ismael et al., 2014; Ismael et al., 2013; Shishira et al., 2017; Sehgal and Bhatt, 2018; Fehling et al., 2014; Gopal and Sunilkumar, 2016). In their paper, Ismael et al. (2014) propose a comprehensive analysis of CC workload characteristics. The analysis was conducted using the data from the second version of the Google Cloud trace-log (Google, 2011). The proposed study shows that CC workload *is always driven by the users*. In realistic conditions, a specific type of user detains a strong influence on

* Corresponding author.
E-mail addresses: hajertoumifst@gmail.com (H. Toumi), zakibrahmi@gmail.com (Z. Brahmi), gammoudimomo@gmail.com (M.M. Gammoudi).

https://doi.org/10.1016/j.jksuci.2019.12.004
1319-1578/

the overall CC workload. Sehgal and Bhatt (2018) propose a CC workload characterization from different points of view such as services customers and providers. The authors show how a workloads type may change in time according to its user behavior that can exhibit different resource usage than the expected. For instance, the authors present the example of a job who was characterized as big database creation and calculation; however, during the run-time, the key usage was found to be the disk space, not calculations. Thus the job needs to be reclassified. Fehling et al. (2014) propose a CC applications workload classification. According to users' behavior and routine, an application workload type can be static, periodic, once in a lifetime, unpredictable and continuously changing. The type defines the load variation of the application. For instance, a continuously changed workload means that the later experiences a long term change that can be planned or unplanned with continuous variation in resource usage. Shishira et al. (2017) classify CC applications according to its workload features into four classes: Web Workload WW, Online Social Network Workload OSNW, Video Service Workload VSW and, Mobile Device Workload MDW. For example, MDW refers to the workload created by the user during its interaction with mobile applications to access and share content and resources. According to the authors, user behavior is one of the most relevant aspects of these workloads and the main load producer.

Aside from users' behavior, by far the biggest factor affecting computational performance in the cloud is contention (Patrick, 2017). Public clouds are multi-tenant environments that allow the co-allocation of multiple applications. The fact that multiple VMs with various characteristics share the same hardware resources creates contention on shared resources (Zhu et al., 2012). The contention ratios directly affect applications response time and servers performance (Xu et al., 2014). If the contention is high then cloud server performance will deteriorate significantly (Patrick, 2017). In fact, an application running in the same VM on the same server may experience a wide disparity in its performance at a different time based on the work performed by other VMs on the same host (Zhu et al., 2012). Many works were devoted to study the impact of VMs interference (VMI) on server performance. Recent studies on Amazon EC2 have shown that, the disk I/O bandwidth of small instances can vary by 50%, and the network I/O bandwidth of medium instances can also vary by 66%, due to the contention on shared resources including CPU cores, cache space, and I/O bandwidth, as a result, a severe performance variation of VMs (Xu et al., 2014). In their paper Younggyum et al. (2007) show that the completion time of a task running with other tasks is greater by a *slowdownratio* than if it was running alone leading to unexpected behavior of server load. According to Javadi and Gandhi (2017), VMI is dynamic and unpredictable and can affect application performance by as much as 27 percent. Javadi and Gandhi (2017) showed that under interference, applications experience higher load than it would in the absence of interference. Xu et al. (2016) study the impact of VMI on application performance. The proposed results showed that the performance of benchmark applications running on two co-located VMs varies significantly from 0,7 to 29,1 percent in terms of resource usage. In particular, the performance variation on the cache and (network and disk) I/O resources was much severer than that on the CPU and memory resources. This is because the cache space and I/O bandwidth resources can hardly be isolated. Moreover, Xu et al. showed that increasing the number of co-located VMs increases the rate of VMI leading to further performance degradation. Zhu et al. (2012) study the interference from all type of resources considering the time-variant-resource usage. The authors introduce a dilation factor that accounts for the impact of resource contention from the co-located application. The dilation factor refers to the extra requirement of an application compared to if it was running alone. Caglar et al. (2014) showed that performance interference varies according to the running tasks types. This means that, for instance, hosting MEMORY intensive tasks on the same VM will cause a heavy contention on MEMORY and disk I/O. Moreover, an experimental study held by Pu et al. (2010) shows that co-locating CPU-Intensive workloads on a shared hardware platform incurs high CPU contention due to the demand for fast memory page exchanges in I/O channels.

The unique nature of CC workloads puts in place the following characteristics:

- The server load variations show high velocity as cloud applications' tasks tend to be short and very interactive (Hussain and Aleem, 2018).
- The users' access pattern tends to be significantly more diverse across different observational periods resulting in different execution length and resource usage patterns for the same task (Ismael et al., 2013).
- The VMs behavior tends to be dynamic and unpredictable due to the performance interference caused bu the contention on the shared resource (Javadi and Gandhi, 2017).

The above-mentioned characteristics contribute to the depreciation of the existing techniques (Ismael et al., 2013) that establish their prediction only based on the history of server resource usage and/or does not consider sudden shifts in server load caused by changes in users' habits and the VMs interference. Thus, adaptive and evolving mechanisms that allow providers to obtain more accurate predictions are needed. Since users' requirement is continuously evolving (Sehgal and Bhatt, 2018) shaping new usage pattern over time and the rate VMI mainly depend on the type of the running VMs at the moment (Caglar et al., 2014), the prediction model has to incorporate and adapt its decision model to the recent information as it arrives. The idea is to create a prediction model able to evolve and adapt to changes in real-time. Compared to classical data mining techniques and deep learning techniques that are trained to excel at predefined circumstances (Shrestha and Mahmood, 2019; Moamar, 2016), stream mining techniques (SMT) are designed to create their decision model based on the most recent information received (Albert et al., 2015). SMT are capable of detecting and reacting to concept drift which allows the decision model to evolve with changing data stream (Albert and Ricard, 2007). In this paper, a Real-Time Server Load Prediction System *RTSLPS* is proposed. *RTSLPS* incorporates the most recent information about users' behavior and VMI to establish an accurate prediction. The proposed system is mainly based on stream mining algorithms to deliver a real-time prediction. Our contribution is presented as follows:

- Formalize server load according to user behavior and performance interference.
- Propose task classification of known and unknown tasks using (Uncertainty-handling and Concept-adapting Very Fast Decision Tree) UCVFDT proposed by Liang et al. (2010) to capture the incoming load trend.
- Propose server load prediction solution based on server actual load, VM interference rate, and Incoming workload trends.
- Propose an improved version of HAT augmented by ensemble drift detectors (HAT-nDetectors) to capture different kinds of data changes in server load.

Our paper is organized as follows: Section 2 presents the related works. Section 3 proposes an overview of Stream Mining. Section 4 proposes a server load formalization. In Section 5, we propose a detailed presentation of our proposed approach *RTSLP*. Section 6 presents experimentation and results and finally we close our paper with a conclusion in Section 7.

2. Related works

To deal with server load prediction, various works have been proposed. The used techniques can be classified into three main classes. The first class includes statistical techniques. Autoregression Integrated Moving Average (ARIMA) is one of the most widely used models for time series analysis due to its simplicity and speed (Zhang et al., 2012). This technique performs well in a stationary environment. To be more adapted to the particularities of the CC environment, many works tried to propose an improved version of ARIMA. Chen and Wang (2018) proposed a resource demand prediction method based on the Ensemble Empirical Mode Decomposition EEMD and ARIMA. EEMD decomposes the non-stationary host usage sequence into relatively stable intrinsic mode function (IMF) components and a residual component to improve prediction accuracy. Then ARIMA is used to predict the future value of each component. The final results are obtained by superposing the results of each component. The proposed method was able to tame the random variation of resource demands and to provide good accuracy. However, this solution held a high error accumulation and time cost due to the decomposition. Thus, in a more recent work, Chen and Wang (2019) tried to resolve these limits. Thus, they proposed a hybrid method for short-term host usage prediction that reduces time cost by selecting and reconstructing efficient IMF components. The experimentation showed that the proposed hybrid method was able to detain lower time cost and higher accuracy compared to the first proposed method. However, the results proved that this solution was unable to react efficiently to random and abrupt variations in resource usage. Zia Ullah et al. (2017) tried to propose a real-time resource usage prediction system based on ARIMA and Auto-regressive Neural Network (AR-NN). The idea is to take the real-time resource usage values and feeds them into several buffers. If the stocked data follow Gaussian distribution, the ARIMA is used else AR-NN is used. The results showed that ARIMA was enable to delivers good accuracy with real traces, however, AR-NN was able to deliver much better accuracy. The proposed system was supposed to produce real-time prediction by taking advantage of the quick execution time of ARIMA. However, the proposed approach fails to deliver a real-time prediction as the time complexity of AR-NN is very important (Chen and Wang, 2019). Jian et al. (2014) proposed a dynamic ensemble prediction algorithm using a combination of linear and non-linear prediction models. The proposed model includes a set of heterogeneous predictors where each set contains homogeneous predictors. The results show that the proposed ensemble model has better accuracy compared to a single model method. However, these results are valid in the context of a private CC environment which is more stable and contains less random variations compared to a public cloud environment (Ismael et al., 2014).

The second class includes deep learning techniques. Duggan et al. (2018) proposed a multiple-time-step-ahead based on Recurrent Neural Networks RNN. RNN showed better performance by delivering more accurate accuracy with non-stationary data compared to nonlinear and linear forecasting methods. However, RNN is characterized by its high time complexity (Rahmanian et al., 2018) as it needs a long time and an important amount of data to be efficiently trained. Song et al. (2017) proposed long short-term memory to predict the mean load over consecutive future time intervals and the actual load multi-step-ahead. The proposed method is based on RNN. However, the authors replaced the hidden layer of RNN with the Long Short-Term Memory LSTM model to learn long-term dependencies. The performed experimentation proved that the proposed method achieved high accuracy with Google data centers traces dataset. However, LSTM-RNN is hard to train as it requires a lot of data to be able to deliver good accuracy which imposes high training time. On a similar note, Huang et al. (2017) used LSTM and RNN with requests-to-vector to predict the performance and the future workload of web servers or data centers based on users requests sequence. They investigated the relationship between users requests sequence and web server performance. The experiment result showed that the proposed model has a good performance in predicting performance on the data set from nginx web server and mysql database server. Due to the nature of RNN, the proposed method still suffer from high time complexity. Rahmanian et al. (2018) proposed a Learning Automata LA-based ensemble prediction algorithm. Each component model has a weight that determines its accuracy. LA is used to determine the weight of each component of the ensemble. The final prediction is provided by combining the multiplication of prediction values of individual models to their weights. Experimentation showed that the proposed approach was able to outperform other techniques. However, the proposed solution suffers from a high execution time (Chen and Wang, 2019) and a high error rate since the final prediction is produced by the combination of base predictors results. Qiu et al. (2016) designed their model using a deep belief network and a regression layer. The deep belief network DBN is used to extract VM workload high features. The regression layer is used to predict future load. In their approach, the authors address two majors facts: i) The correlation among VMs and ii) The inaccuracy and incompleteness of workload data set. The proposed approach showed a better performance compared to the simple neural network, multi-layer neural network, and ARIMA. But it detains high time complexity as DBN is known to be very complex to train. Li et al. (2019) proposed a two-stage workload prediction model based on artificial neural network ANN, which is composed of one classification model and two prediction models. The results showed that the proposed model can achieve more accurate prediction compared with ARIMA, Linear Regression, SVR and Neural Network NN. However, ANN has always struggled with random and unpredictable variation as its prediction model is highly related to the past.

The third class includes Data Mining Techniques (DMT). Minarolli et al. (2017) proposed long-term resource usage prediction to detect when a host is overloaded or under-loaded using a supervised data mining approach based on Gaussian Process. In their proposed work, the authors tackle the problem of uncertainty of long-term predictions for overload detection. However, long-term prediction in a CC environment is very tricky as server load variations show high velocity as cloud applications' tasks tend to be short and very interactive (Hussain and Aleem, 2018). Zhong et al. (2018) propose a host load prediction model based on the Weighted Wavelet Support Vector Machine (WWSVM). The main idea is to differentiate samples that reflect more importance from the others. To find the optimal combination of the parameters to the WWSVM model, a parameter optimization algorithm based on particle swarm optimization(PSO) is proposed to reduce the training time. Gopal and Sunilkumar (2016) proposed a Bayesian Network BN model based on the analysis of user behavior and time frame. The model aims to determine short and long-term virtual resource requirements of CPU/memory-intensive applications and workload patterns at several data centers in the cloud during several time intervals. In their work, the authors addressed the dynamic and varied nature of the Cloud workload. The proposed approach considers the interaction among concurrent applications while producing the prediction. The results showed the proposed solution detains better accuracy compared to the regression technique and support vector technique. However, the authors ignore the impact of VM interference on application performance and on data centers load.

Table 1 presents a comparative study of the related works. We consider the following features for the comparison: UT depicts the used technique which can be a Data Mining Technique DMT, a

Table 1
Survey of load prediction related works.

W	UT	PR		CP			WT	Perf	
		CPU	CPU + Others	History	User	Interf		A	T
Jian et al. (2014)	ST	*		*			D	++	+++
Chen and Wang (2019)	ST		*	*			D	++	+++
Rahmanian et al. (2018)	DL	*		*			D	+++	++
Zia Ullah et al. (2017)	ST	*		*			D	+++	++
Qiu et al. (2016)	DLT	*		*			D	++	+
Duggan et al. (2018)	DLT		*	*			D	+++	++
Song et al. (2017)	DLT	*		*			D	+++	+
Huang et al. (2017)	DLT	*		*	*		D	+++	+
Li et al. (2019)	DLT		*	*			D	+++	++
Minarolli et al. (2017)	DMT	*		*			DUC	++	++
Gopal and Sunilkumar (2016)	DMT		*	*	*		D	+++	+
Zhong et al. (2018)	DMT		*	*			D	+++	++
Proposed	SMT		*		*	*	DUC	+++	+++

Deep learning Technique DLT, a statistical Technique ST or Stream Mining Technique SMT, PR depicts the predicted resources which can be CPU or CPU and others resources (such as memory, BW and disk I/O), CP depicts the considered parameter for the prediction which can be server history, users' usage pattern and Performance interference, WT depicts the workload type which can be stationary S, dynamic D and dynamic with unpredictable changes DUC, Pref depicts the performance of the related technique and it is measured according to the accuracy A of the technique and the response time T (+++: very good, ++: good, +: Medium). The measures of time and accuracy were determined according to the experimentation held by the authors of the work and according to our literature review of the used techniques. For instance, the accuracy of the neural network depends on how deep the network is. This means that the accuracy will increase with more layers. Nerveless, adding more layers increases the time complexity (Shrestha and Mahmood, 2019). According to Table 1we can summarize the related works as follows:

- The majority of the proposed works establish their prediction models around the CPU usage ignoring the impact of other resources such as memory, BW and disk I/O on the load. Hosting memory-intensive tasks, a server may be overloaded by memory even though its CPU usage has not been overloaded (Fehling et al., 2014).
- The majority of works only consider history to produce their predictions. Yet, server load variations show high velocity as cloud applications' tasks tend to be short and very interactive (Di et al., 2014).
- The majority of the proposed works only consider the dynamic nature of Cloud Computing ignoring the fact that server load can record unforeseen changes that cannot be anticipated from the past.
- The majority of proposed works consider that server load variations are highly related to time. This makes a non-robust hypothesis as users can be seen as the main drive of server load variations (Huang et al., 2017).
- To the best of our knowledge, there is no server load prediction approach that takes into account VM interference along with user behavior to predict server load.

Although many proposed works were able to deliver very good accuracy, These works suffer from high training time as they are using DL or DM techniques. DMT and DLT require an important amount of data to train which imposes high training time (Chen and Wang, 2019). Moreover, DMT and DLT are trained to excel in predefined conditions, thus, they can't adapt quickly to unpredictable and new circumstances (Moamar, 2016; Shrestha and Mahmood, 2019). Unlike DMT and DLT, using SMT in our proposed approach helped us deliver high accuracy in short evaluation time as SMT does not require stored training set as the learning is established incrementally the newest data arrives. Moreover, our proposed approach was able to manage the highly dynamic and ever-changing nature of the CC environment considering unpredictable changes.

3. Data stream mining

Data Stream Mining is becoming the most efficient and fasted way to extract useful knowledge from what is happening now. It allows reacting in real-time when a change occurs or a new trend is detected while dealing with resources in an efficient and low-cost way (Albert et al., 2015). Unlike conventional data mining techniques that require prior knowledge of to-be analyzed data, data stream mining allows real-time processing of data when it arrives. A data stream S is seen as a sequence of data points $\vec{x_i}$ such that $S = \{\vec{x_1}, \vec{x_2}, \ldots, \vec{x_\infty}\}$. Each $\vec{x_i}$ arrives at specific time t_i and it is represented as features vector with d dimensions. Data distribution may change with time causing a concept drift which refers to the unforeseen changes in the stream caused by a change in the joint probability distribution such that $P_t(\vec{x_i}, Y_i) \neq P_{t+1}(\vec{x_i}, Y_i)$. The concept drift is characterized by its speed, and it can be abrupt or gradual, according to its speed.

Dealing with continuous changes, an efficient stream mining technique must be able to detect and react to all types of drift (abrupt, gradual) in the most effective way.

3.1. Concept drift detection methods

According to Albert and Ricard (2007), an efficient Data Stream solution for time-changing data have to effectively respond to the following questions: What to remember or forget? When to do the model upgrade? How to do the model upgrade. The first step is a very critical one as the quality of the input data will determine the quality of the output result.

To deal with concept drift, various methods were proposed, many of them use monitoring two distributions techniques, statistical process control techniques or Sequential Analysis techniques. Monitoring two distributions techniques use a reference window that summarizes the past information, and a sliding detection window containing the most recent data. They compare distributions aver the two windows with the null hypothesis that assumes that the two distributions are equal. Once the null hypothesis is rejected, a concept drift is announced at the start of the detection window. Nerveless, the big issue when using sliding windows is the size of the window. The small-sized window helps to react to changes quickly but loses its accuracy in periods of stability. A

large size helps to improve the accuracy of the learner during a stationary period but fails to quickly adapt to abrupt drift (rapidly changing concepts). ADWIN (ADaptive WINdowing) is a well-known windowing technique proposed by Albert and Ricard (2007). It is a change detector based on monitoring two distributions. It maintains a variable size of windows and adjusts it according to the observed changes. ADWIN is a parameter- and assumption-free, it automatically detects and adapts to the current change using only confidence bound parameter to indicate how confident we want to be in the algorithms output.

Statistical process control techniques consist of monitoring and controlling the evolution of the process. They are considered as trigger approaches. The drift detector reacts to concept changes and issues an alarm when the learner should be rebuilt or updated. One of the most popular techniques is called Drift Detection Method (DDM) (Joo et al., 2004). DDM is a change detector based on monitoring the online error-rate. It assumes that learning follows a sequence of trials. Many improvements of DDM were proposed such as EDDM RDDM and HDDM (Wiecki et al., 2013). STEPD (Kyosuke and Kaichiro, 2007) is a change detector based on monitoring the accuracy: the recent accuracy and the overall accuracy. The idea is that the accuracy of the online learner has to be equal to the overall accuracy from the beginning of the learning if not, that means that the concept is changing. STEPD proves a good performance facing gradual drift.

3.2. Data stream mining classification

In general, stream mining tasks are covering classification, clustering, outlier detection, etc. For the purposes of this paper, we will only discuss classification techniques. More specifically we will discuss the classification techniques based on decision trees since they are the most related to our research. Decision trees are among the most common and well-studied classifier models (Babu et al., 2017). In the context of the data stream, a well-known decision tree, Hoeffding Tree (HT) was proposed by Domingos and Hulten in Pedro and Geoff (2000). HT provides incremental and anytime learning that uses the first examples to choose the root of the decision tree. The following examples will be passed down to the corresponding leaves to choose the appropriate attributes and so on recursively. To choose the optimal splitting attribute, Hoeffding bound is used. With HT, a small sample can be enough to choose the optimal splitting attribute. HT works well with large data sets but it can't handle concept (Hulten et al., 2001). A Very Fast Decision Tree (VFDT) was proposed by in Pedro and Geoff (2000) as an extension of HT that can deal with concept drift. VFDT uses the information gain in the optimal splitting attribute choice and uses a sliding window for concept drift detection. It maintains statistics for each node and removes the outdated data by decreasing the statistic at nodes. VFDT is known to be Fast., In turn, an extension of VFDT was proposed by Hulten et al. (2001). proposed Concept-adapting Very Fast Decision Tree (CVFDT) that periodically evaluates the quality of the tree using a bunch of samples. The alternate trees are created after waiting a fixed number of incoming instances to confirm changes and the old tree is replaced by new more accurate ones after waiting a fixed number of incoming instances to confirm the accuracy of the new trees. CVFDT maintains the characteristic of VFDT but it is faster and accurate. An Uncertainty-handling and Concept-adapting Very Fast Decision Tree (UCVFDT) was proposed by Liang et al. (2010) as an extension of CVFDT that can handle data with uncertain attributes. UCVFDT is based on DTU (Biao et al., 2009) (Decision Tree for Uncertain Data). UCVFDT maintains the ability of CVFDT to cope with concept drift with high speed.

Hoeffding Window Tree (HWT) was proposed by Albert and Ricard (2007) and it is based on VFDT. HWT keeps a sliding window of the last data in the stream and it places the change detector ADWIN at every node of the tree. HWT maintains estimators of only relevant statistics at the nodes of the current sliding window and creates the alternates trees as soon as the changes are detected. The old trees are replaced as soon as there is evidence that the new one is more accurate. HWT Stores a bounded part of the window in main memory and the rest is stored in desk. Hoeffding Adaptive Trees (HAT) Evolution of HWT proposed by Albert and Ricard (2007). HAT does not need a fixed sliding window and it learns adaptively from arriving data stream. HWT places instances of estimators of frequency statistics at every node and keeps all relevant statistics from data in the nodes. Each node can decide which of the last instances is currently relevant for it. HAT keeps all its data in main memory and it uses less memory then HWT.

4. Real time server load prediction system RTSLPS formalization

In this section, we shall discuss the server load formulation. A server S has a capacity expressed in a form of resources R, hosts a set of VMs and detains a performance $Perf$ that has to respect a predefined QoS, such that $S =< R, VMs, Perf >$. Each $VM \in VMs$ belongs to a server S, uses a portion of server resource R_{VM} to operate, executes a set of tasks X_{VM} and suffers from interference I caused by sharing server resource with other VMs, such that $VM =< R_{VM}, X_{VM}, I >$. Each task x is executed in a VM, uses VM resource R_x to operate, is submitted by a user u and detains a Type, such that $x =< VM, R_x, u, Type >$. Each user $u \in U$ with U represents the set of users, submits a set of tasks X_u, demands a QoS to be respected and have a $profile$ that determine his behavior, such that $u =< X_u, profile, QoS >$.

In this paper, we believe that the load that will be created in the future $FLoad$ is deeply related to two main factors: i) User behavior and ii) VM interference. Users' behavior is defined by the type of submitted tasks during a period and VM-interference is defined by the contention of co-located VMs on the shared resources. Our proposed approach is composed of three main Modules: Incoming Workload Trend Detection IWTD, VM-Interference rate detection IRD, and Server Load Prediction SLP.

IWTD aims to extract the load trend of the incoming task based on task classification. The output of this model will be the rate of each task's type. Formally, IWTD problem (IWTDP) can be defined as follows:

$$IWTDP =< X, U, T > \tag{1}$$

- X is the stream of tasks such as $X^t = \{x_1^t, x_2^t, \ldots, x_n^t\}$ denotes the stream of tasks submitted at time t by a set of users U. Each task x_i is a tuples $< TaskID, t, u_j, RRes >$ such that $TaskID$ is the unique identifier of the task, u_j is the user submitting the task, t is the access time during which the task is submitted and $RRes$ is the requested resource which can be different from the actually used resources R_{x_i} during the task execution, as the user tends to overestimate the amount of needed resource.
- U: is the set of users submitting X. At a time t a user u_j is responsible of submitting a set of tasks X_j^t such that $X_j^t = \{x_1 j^t, x_2 j^t, \ldots, x_n j^t\}$ with n denotes the number of tasks submitted by u_j at t. Each user u_j have a $Profile_j$ such that $u_j^t =< X_j^t, Profile_j^t >$ with $Profile_j^t$ is the profile of u_j at t.
- T: is the time interval during which X is submitted by U

IRD aims to extract the rate of interference, formally, the IRD problem (IRDP) can be defined as follows:

$$IRDP =< M, RD, RS > \tag{2}$$

such that M is the number of co-located VM, RD_{vm} is the resource demand of a VM and RS_{vm} is the resource supply of a VM.

Finally, server load prediction model $LPred$ can be defined as follows:

$$LPred =< ALoad, IR, IWT, D > \quad (3)$$

such that:

- $ALoad : ALoad =< ru^1, ru^2, \ldots, ru^d >$, represents the actual load of d resources usage of the server. ru^i is the amount of the resource i usage.
- Interference Rate IR is based on the number of co-located VMs M and the difference between VM resource demand RD and VM resource supply RS in term of CPU and disk I/O (Xu et al., 2016).
- Incoming Workload Trend IWT includes the rate of incoming tasks' types in term of resource, such that: $IWT =< rtye^1, rtye^2, \ldots.., rtye^l >$ with $rtye^j$ is the rate of tye^j tasks.
- $D : (ALoad, IR, IWT) \mapsto FLoad$is the decision function that predicts the future workload $FLoad$ of the server.

Thus, future server load $FLoad$ can be computed according to Eq. 4.

$$FLoad = (ALoad - \alpha) + IW + IR \quad (4)$$

such that α is the released server resource related to living tasks (finished/ interrupted), IW is the incoming workload determined by the set of incoming tasks.

5. Real time server load prediction system description

Fig. 1 shows an overview of our proposed approach. As the stream of tasks arrives, the first step consists of classifying the incoming tasks into four main classes: CPU-Intensive ($CPU - I$), MEMORY-Intensive ($MEMI$), CPUMEMORY-Intensive ($CPUMEMI$) and Normal (N). Each class determines the task's $TYPE$. The rate of each type is calculated in order to detect the incoming workload trend, such that $IWT =< RCPUI, RMEMI, RCPUMEMI, RN >$ with $RCPUI$ is the rate of CPU-Intensive tasks, $MEMI$ is the rate of MEMORY-Intensive tasks, $RCPUMEMI$ is the rate of CPUMEM-Intensive tasks and RN is the rate of normal tasks. The Server Load Prediction (SLP) Module takes as input the IWT, the IR calculated by VM Interference Detection Module and the actual resources usage $ALoad$ of the server and produces a prediction of future server load $FLoad$ trend, such that $FLoad = \{OVERLOADED, CPUOVERLOADED, MEMOVERLOADED, UNDERUSED, NORMAL\}$. Producing a more refined prediction that determines which resource is overload will help tremendously in the migration step as it provides an insight on which VMs need to be migrated. In the following, we shall provide a more in-depth description of each module.

5.1. Incoming workload trend detection IWTD

This Model is composed of two main sub-models: Task classification and Trend detection.

5.1.1. Task classification

We believe that a task type $TYPE_{x_i}$ depends on the user behavior u_j during the submission time t. As a matter of fact, multi-tenancy in cloud computing means not only that IT infrastructure is shared by cloud users but also applications' tasks can be shared by various users with different requirements. Moreover, users' patterns tend to be significantly more diverse across different observational periods resulting in different execution length and resource usage patterns for the same task (Ismael et al., 2013). Thus, the resource usage of a specific task can be dynamically reapportioned per-user basis (Ismael et al., 2014). Therefore, the same task may belong to different types according to its user. As we mentioned in our paper (Hajer et al., 2017), the expectation of the task's type is given by its probability $P(x_{ij}^t)$ conditioned to the probability of $P(u_j^t)$ such as u_j^t is the profile of users at time t.

$$E(x_{ij}^t) = x_{ij}^t(P(x_{ij}^t)|P(u_j^t)) \quad (5)$$

A user profile u_j can be: CPU Intensive ($CPU - I$), MEMORY Intensive ($MEM - I$), CPU-MEMORY intensive ($CPUMEM - I$) or normal (N). This means that, for instance, a user with CPU intensive

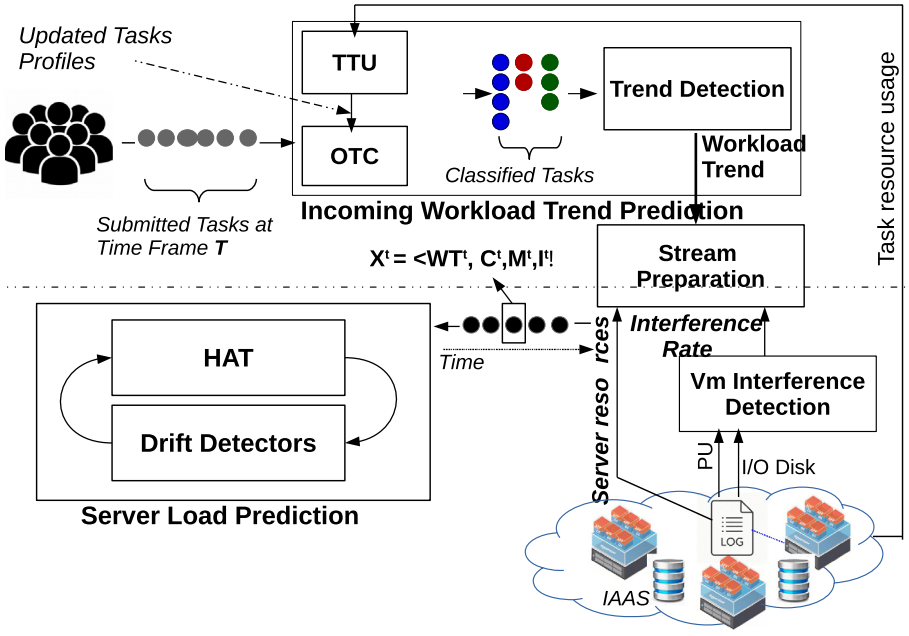


Fig. 1. Proposed Approach Overview.

profile means that the majority of its submitted tasks are CPU intensive. the profile of u_j is determined by his set of task X_{u_j} submitted during t.

The stream of submitted task X is composed of already known tasks, this means that x_i^t has been executed in the server S, and unknown tasks. It must be taken into consideration that over time the type of x_{ij}^t may change as task distribution may drift due to changes in user behavior. Moreover, the arrival of new users and new tasks must be expected.

Mainly task classification is based on u_j and t, this means that a user with $CPU-I$ profile during t is likely to submit $CPU-I$ task during the same time interval t, such that $\forall x_{ij}{}^t \in X_j^t$, the probability of $Type_{x_{ij}{}^t} = Type_y$ is $P(Type_{x_{ij}{}^t} = Type_y) = P(Type_y|u_j^t)$ such as:

$$P(Type_y|u_j^t) = \frac{P(u_j^t|Type_y)P(Type_y)}{P(u_j^t)} \tag{6}$$

An unknown task x_i means that one or more of its attributes is/are the uncertain attribute(s). Let's note ATT_{x_i} as the attributes set of x_i such that $ATT_{x_i} =< TaskID, t, u_j, RRes >$ and $ATT_{x_i}^a$ is uncertain attribute such that $ATT_{x_i}^a \in ATT\prime_{x_i}$ with $ATT\prime_{x_i}$ is the set of uncertain attributes. Given a belonging domain $Dom(ATT_{x_i}^a)$ such that $Dom(ATT_{x_i}^a) = \{v_1, v_2, \ldots, v_b\}, ATT_{x_i}^a$ detains a probability distribution over dom. $ATT_{x_i}^a$ can be a categorical attribute such as user attribute u_j or numerical attribute such as $RRes$.

Task Classification Module is composed of two sub-modules: Online Task Classification (OTC) and Task Type Update (TTU). The principle is as follows: Once a task arrives, the OTC sub-module affects the task into a type using UCVFDT. UCVFDT is an extension of CVFDT proposed by Hulten et al. (2001) that aims to classify uncertain data points. The reason behind our choice of UCVFDT is that it is an extension of CVFDT. In our recent work (Hajer et al., 2017), we held a comparative study of some stream mining algorithms including CVFDT for task classification in CC. The results show that CVFDT detains a better performance in terms of accuracy and completion time compared to the other tested algorithms. Once the task has been executed, the TTU sub-module aims to actualize the task's type using its execution information from the server log. The task type is updated in real-time with the actual information about the task execution considering used CPU and used MEMORY. This helps to keep our model updated in real-time. Algorithm 1 shows the pseudo-code of the Task Classification Model.

Algorithm 1 OTC Algorithm
Input: $X^t = \{x_{11}^t, x_{12}^t, \ldots, x_{nm}^t\}$
OUTPUT: *TaskType*
For Each incoming task $x\prime^t$
  if $X^t.Contains(x\prime^t)$ **then**
    if $x^t.getUser \in X^t.getUsers()$ **then**
      $TaskType \leftarrow$ ClassifyKnownTask($x\prime^t$)
    else
      $TaskType \leftarrow$ ClassifyUnKnownCATGTask($x\prime^t$)
    end if
  else
    $TaskType \leftarrow$ ClassifyUnKnownNUMTask($x\prime^t$)
    $X^t.add(x^t\prime)$
  end if
  $x^t.SetType(TaskType)$

As Algorithm 1 shows, OTC is based on three main functions: ClassifyKnownTask(x), ClassifyUnKnownCATGTask(x) and ClassifyUnKnownNUMTask(x).

ClassifyKnownTask(x) aims to classify the known tasks.

ClassifyUnKnownCATGTask(x) and ClassifyUnKnownNUMTask(x) aim to classify unknown tasks.

5.1.2. Incoming workload trend detection

Trend Detection sub-module takes as an input the classified task and calculates the rate of each type $rtye^j$. With N is the number of arriving task at t, such that:

$$rtye^j = \frac{rtye^j tasks}{N} \tag{7}$$

5.2. VM-interference rate detection IRD

The majority of existing works define the workload created at Cloud servers as the amount of resource used by servers responding to users' requests; however, the fact that multiple VMs with various characteristics share the same server creates contention on the shared resources and cause additional and random load variations. Performance interference affects the completion time of tasks leading to unexpected behavior of server load (Caglar et al., 2014). According to Xu et al. (2016), performance interference can be seen as the mismatch between the resource supply provided by the hosting server and the resource demand of co-located VMs. Thus, to capture the rate of VM interference of a VM, (Xu et al., 2014) proposes a supply-demand ratio model presented as follows:

$$f_d^i = c_0 + c_1.\frac{S_{cpu}^i}{D_{cpu}^i} + c_2.\frac{S_{io}^i}{D_{io}^i} \tag{8}$$

Such that $c0, c1$ and $c2$ are the coefficients determined by Xu et al. (2016). For each VM instance i, D_{cpu}^i and D_{io}^i denote the aggregated CPU utilization and disk I/O utilization of its co-located VMs, respectively. S_{cpu}^i and S_{io}^i denote the available physical CPU resource (cores) and I/O resource on its hosting server, respectively.

For $f_d^i \in (0, 1]$, a small f_d^i rate means that the interference is high. The overall performance degradation factor of VM interference f_d is formulated as the average interference value in all the VMs and it is given by:

$$f_d = \frac{\sum if_d^i}{m}; \forall i \in [1, n] \tag{9}$$

Where m is number of co-allocated VMs.

5.3. Data stream preparation

Indeed, the server loads dataset exhibits the properties of evolving data stream as they follow those three properties: unbounded data series, high velocity, changes in data distribution over time as it records a combination of stationary periods, gradual change and abrupt change. This module aims to prepare our data stream to be fed to the SLP module and it contains two steps: Merge data. The output result data of task classification, VM interference rate data and server resource usage data merge into a single stream of tuples. This stream contains information about Time t, the submission rate $SubR$, CPU intensive tasks rate $CPU-I$, MEMORY intensive tasks rate $MEM-I$, CPU-MEMORY intensive tasks rate $CPUMEM-I$, normal tasks rate $Normal$, interference rate $Interf$, actual CPU usage $CPU-U$ and actual MEMORY usage $MEM-U$. Each data of the stream is a tuples $< t, SubR, CPU-I^t, MEM-I^t, CPUMEM-I^t, Normal^t, Interf^t, CPU-U^t, MEM-U^t >$ Quantize data. Quantization Interval Selection helps in maintaining

Table 2
Quantization intervals of selected attributes.

Attribute	Quantization Interval	Range
SubR	Low, Medium, High	$[0-300), [300-800), [800-\infty)$
CPU − *I* (%)	Low, Medium, High	$[0-20), [20-45), [45-100)$
MEM − *I* (%)	Low, Medium, High	$[0-20), [20-45), [45-100)$
CPUMEM − *I* (%)	Low, Medium, High	$[0-20), [20-45), [45-100)$
Normal (%)	Low, Medium, High	$[0-20), [20-60), [60-100)$
Inter (%)	Low, Medium, High	$[0-20), [20-50), [50-100)$
CPU − *U* (%)	Low, Medium, High	$[0-20), [20-45), [45-100)$
MEM − *U* (%)	Low, Medium, High	$[0-20), [20-45), [45-100)$

important data patterns. Thus, our data are quantified into numbered ranges as follows: 0*forLow*, 1*forMedium* and 2*forHigh*. Table 2 presents the Quantization intervals considered in our analysis.

5.4. Server load prediction SLP using HAT-ndetectors

The main component of our system is SLP based on a real-time classification using HAT proposed by Albert and Ricard (2007) augmented by a set of drift detectors $n-Detectors$. Since our stream typically can record over time any combination of long stationary periods, abrupt changes and gradual changes, a single detection method only works for one type of drift, it can only handle concept drift with particular speed and it has limited generalization capacity (Du et al., 2014). Thus, it's more beneficial to combine the strength of different detectors to achieve a more accurate prediction.

HAT-nDetector is based on two main phases: Online learning and Concept drift detection. As our stream of data *DS* comes continuously, it is incrementally provided to the online learner (classifier). The old data are used to train the classifier, and the newest data are used for testing. Concept Drift Detection uses the test results provided by the online learner and then it returns relevant information concerning if there is a change or not to the classifier. According to the concept drift detection phase feedback, the classifier will be updated.

5.4.1. HAT-nDetectors

HAT is known for its ability to learn from fluctuating data streams without needing a fixed-sized parameter/window. Since HAT uses a sliding window, choosing the right size of the window is very difficult because it depends on the rate of changes of the dataset distribution. Ideally, a large parameter is needed to ensure a high accuracy during stationary periods and a small one to react quickly to the drift. Nerveless, in our case, the size of the window will be adjusted according to the rate of distribution change observed by the set of detectors placed at every node of HAT.

Algorithm 2 presents the pseudo-code of HAT-nDetectors. Once a new instance x arrives, it will be tested by the function Test(x) and sent to nDetector (SubSection 5.4.2). When a change or warning occurs at a node, the nDetectors will raise a hand.

The $up-to-now-dataset$ contains the training data set needed for the training step. The $Rebuilding-dataset$ will be used when concept drift occurs to learn the new concept. The size of $Rebuilding-dataset$ depends on the rate of change. As long no concept drift is detected $Rebuilding-dataset$ is empty. Once a Warning is detected, the set will be incremented by the set of data exhibiting the new trend. If the drift is confirmed at a node, $up-to-now-dataset$ will be cleared and refilled by $Rebuilding-dataset$ and new sub-tree $Alternate-sub-tree$ is created. The accuracy of the original sub-tree $Existing-sub-tree$ and the alternate sub-tree is compared. If the alternate sub-tree shows better results than the original sub-tree, then the original sub-tree will be dropped and replaced by the alternate sub-tree.

Algorithm 2 HAT-ndetectors algorithm

```
Input: DS = {x_1, x_2, ..., x_n}
  up-to-now-dataset ← Training DataSet
  Rebuilding-dataset ← NULL
  Concept-Drift ← FALSE
  Warning ← FALSE
  result ← NO-CONCEPT-DRIFT
  Output: SERVERSTATE
For Each: x in DS
  Test ← Test(x)
  result ← nDetectors(Test)
  if result = WARNING then
    Warning ← TRUE
  end if
  if result = CONCEPT-DRIFT then
    Concept-Drift ← TRUE
  end if
  while Concept-Drift = FALSE do
    up-to-now-dataset.Add(x)
    if Warning = TRUE then
      Rebuilding-dataset.Add(x)
    else
      Rebuilding-dataset.clear()
    end if
end while
Alternate-sub-tree.Build(Rebuilding-dataset)
up-to-now-dataset.Clear()
up-to-now-dataset.AddAll(Rebuilding-dataset)
if Alternate-sub-tree.calculateAccuracy()> Existing-sub-tree.
  calculateAccuracy() Existing-sub-tree.Replace(Alternate-su
  b-tree) then
end if
```

5.4.2. Concept drift detection with n-detectors

The main idea of n-Detectors is that an ensemble of various detectors will help to locate more concept drifts (Du et al., 2014). As a matter of fact, there are so many types of data streams, but usually, a single detection method only works well for one type of them (Maciel et al., 2015). Our concept drift model includes multiple detectors, each of which has a different detection model. Every detector uses a variable as a change indicator to signal concept drifts or warning. The change indicator is compared with a predefined threshold β to determine the current state of the data stream. A fusion rule is required to consolidate detection results from different detectors, in our case, the majority voting and weighted voting are used.

Usually, a set of drift detectors find a concept drift at different time-steps, thus we use *maxWait*. Once a detector signals a drift or a warning, *maxWait* is the number of instances to wait to confirm or deny the drift. Thus, we have a set of detectors *DT* such that $DT = dt_1, dt_2, \ldots, dt_n$. Each detector $dt_i \in DT$ has a detection model DM_i and a weight w_i such that $dt_i = DM_i, w_i >$. Initially, the majority voting is used to signal a concept drift or a warning. This means that with respect to a specific amount of time *maxWait*, we have $DT_{drift}^{maxWait}$ the set of detectors that signaled a concept drift within *maxWait*. dt_k^t is the k-th detector that signal a concept drift in time t with $t \leqslant maxWait$, such that $dt_k^t \in DT_{drift}^{maxWait}$ and $DT_{drift}^{maxWait} \subseteq DT$. The feedback of each detector is evaluated to update the weight w_i of each detector.

In each time, our detection model chooses the most effective detector in order to increase its weight. The evaluation considers the accuracy of the detection and the signal time. This means that,

In each time, the first one that signals a correct concept drift will be rewarded with a point. the effectiveness of a detector $dt - i$ is computed according to the Eq. 10 such that $EFF_{dt_i} \in [0..1]$by dividing the weight w_i by the total number of instances IN. The more $EFF_{dt_i}$is close to 1 higher the effectiveness of dt_i.

$$EFF_{dt_i} = \frac{w_i}{IN} \tag{10}$$

Once the model is efficiently updated, the prediction will start to rely, beside the majority voting, on the weighted voting. This means that during *maxWait*, once a detector with high weight w_i signals a drift such that $w_i \geqslant \beta$, the model will directly confirm the drift. However, at each time a detector provides a false detection, its weight will be penalized by taking away a point.

Our model helps to cover different types of drift in a highly dynamic environment such as the cloud environment. Unlike the ensemble-based method that requires higher response time compared to a single method, the response time of n-detectors method decreases over time as, with each specific type of stream, it allows the most effective detector to take the lead.

6. Experimentation and results

Our Experimentation is divided into two main parts: i) The first part focuses on evaluating the performance of the prediction algorithm *HAT – ndetectors* compared to other stream mining techniques. This section aims to prove the ability of *HAT – ndetectors* to deliver high accuracy facing different types of concept drift in a short time. ii) The second part focuses on evaluating RTSLPS performance.

6.1. HAT-ndetectors performance evaluation

6.1.1. Evaluation metrics

In this part, we are focusing on evaluating the performances of HAT-nDetectors. We used the hyper-plane generator to simulate changes into our data-set. We evaluate HAT-nDetectors performance against its original version HAT and two well-known stream mining algorithms: Adaptive Random Forest (ARF) (Gomes et al., 2017) and OzaBagADWIN (Albert et al., 2009). ARF is an adaptation of the original Random Forest algorithm proposed by Breiman (2001) which uses bagging. OzaBagADWIN is an adaptation of the Bagging algorithm proposed in Nikunj and Stuart (2001) that uses ADWIN as a drift detector.

A prequential evaluation technique is used and a prequential error is computed based on an accumulated sum S of a loss function L between the prediction y_t and observed values $\hat{y}_t$ (Albert et al., 2015) as shown in Eq. 11.

$$S = \sum_{t=1}^{n} L(y_t, \hat{y}_t) \tag{11}$$

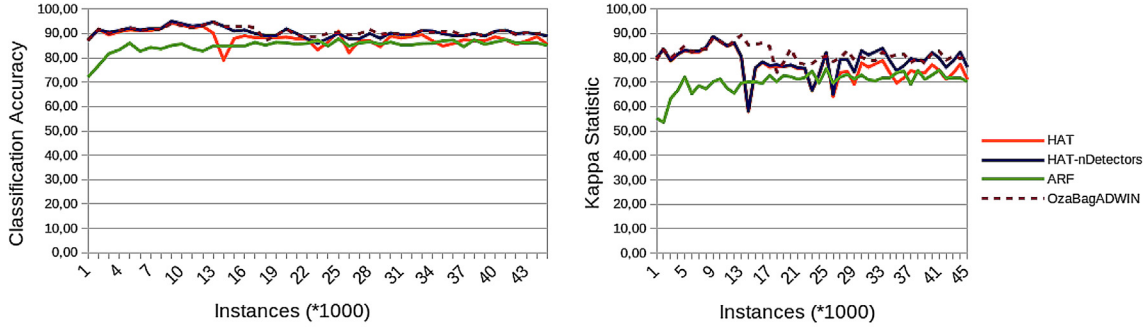
To evaluate the performance of our proposed algorithm, classification accuracy, Kappa Statistic and evaluation time are used. Classification accuracy and Kappa statics provide an indication of the algorithms ability to classify individual samples correctly. Evaluation Time is important to evaluate the ability of the algorithm to perform real-time classification. The tests were performed by MOA (Massive Online Analysis) (Albert et al., 2010). All metrics were plotted every 1000 instances. Fig. 2a and b show valleys in classification accuracy and kappa statistic as Hat-nDetectors encounters changes. These valleys are related to the period during which the algorithm recovers from changes and adapts its model. As Fig. 2a and b show, Hat-nDetectors was able to maintain a better accuracy and a better Kappa comparing to HAT and ARF with a mean classification accuracy of 90% and mean kappa statistic of 80%. However, HAT-nDetectors and OzaBagADWIN showed nearly similar behaviors.

6.1.2. Results and discussion

Although, OzaBagADWIN has similar behavior to HAT-ndetectors in term of accuracy and kappa, Fig. 3 shows that HAT-nDetectors was able to maintain the lower evaluation time with a mean evaluation time of 1,98 CPU seconds compared to 5,04 CPU seconds for OzaBagADWIN. Fig. 3 shows that HAT-nDetectors detains a slightly higher time than HAT. However, experimentation showed that beyond 54 000 instances, the evaluation time of HAT-nDetectors becomes aligned with HAT evaluation time. It boils down to the fact that at first, our proposed algorithm relies on the whole ensemble of detectors to detect changes, then, it starts to rely on the most effective detector. Fig. 3 shows the number of changes detected by HAT-nDetectors compared to HAT as the real amount of changes is represented by the discontinuous red curve. HAT-nDetectors was able to detect correctly more changes than HAT showing a better capacity in detecting concept drifts.

6.1.3. Conclusion

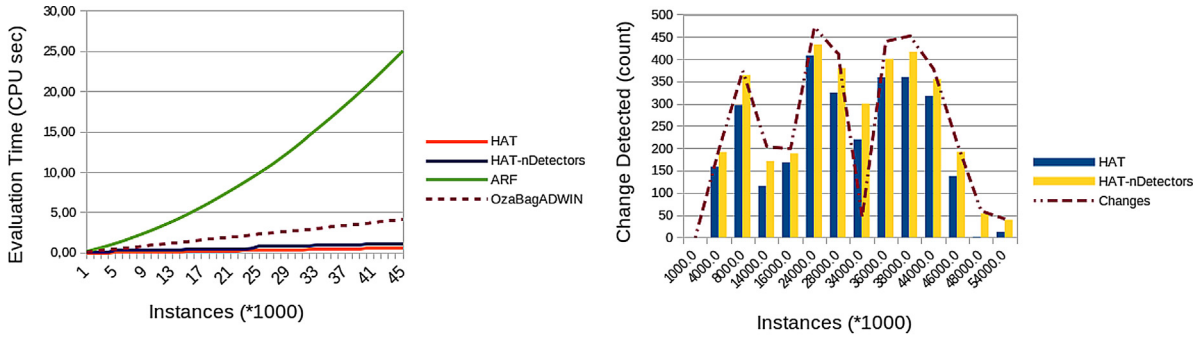
Compared to HAT, ARF and OzaBagADWIN, HAT-nDetectors was able to deliver high accuracy with lower evaluation time. HAT-nDetectors performance is very promising as its small evaluation time, high accuracy, and high Kappa statistic support the premise that HAT-nDetectors can be used to establish an accurate and a



(a) Classification accuracy of HAT-ndetectors vs HAT, AdaptiveRandomForest, OzaBagADWIN

(b) Kappa Statistic of HAT-ndetectors vs HAT, AdaptiveRandomForest, OzaBagADWIN

Fig. 2. (a) Classification accuracy of HAT-ndetectors vs HAT, AdaptiveRandomForest, OzaBagADWIN. (b) Kappa Statistic of HAT-ndetectors vs HAT, AdaptiveRandomForest, OzaBagADWIN.



(a) Evaluation Time of HAT-ndetectors vs HAT, AdaptiveRandomForest, OzaBagADWIN

(b) Change Detection of HAT-nDetectors vs HAT

Fig. 3. (a) Evaluation Time of HAT-ndetectors vs HAT, AdaptiveRandomForest, OzaBagADWIN. (b) Change Detection of HAT-nDetectors vs HAT.

real-time prediction in a highly dynamic and ever-changing environment such as Cloud environment.

6.2. Real time server load prediction system (RTSLPS) performance evaluation

6.2.1. Simulation setup and evaluation metrics

Our simulation is built based on real workload traces analysis of Amazon EC2 and Google CE data centers proposed in Di et al. (2014) and Gopal and Sunilkumar (2016). The data has been collected during the period from 21st July 2014 to 27 Th September 2014. The gathered traces are related to data collected from 12500 servers and contain approximately 650000 jobs. It includes information about CPU, Memory, disk I/O and network I/O. To have a more accurate simulation of the CC environment, we propose an extension of CloudSim Framework that consists of a combination of CloudSimPlus (Campos, 2016) and CloudSimEX (Grozev, 2014). The proposed extension aims to provide a simulation of Cloud workload considering: i) dynamic resource usage, ii) VM interference. Beside cloudSimplus and CloudSimEx merging, we added into the framework VM interference detection model based on Eqs. 8 and 9 and also the Geo-localization at session-level.

In our simulation, The workload is represented in terms of user sessions. Each session has an arriving frequency based on Time zone (day of week and time interval), geographical location and submission rate. Each task that belongs to the session is submitted by a user and it requires CPU, RAM and disk IO. Initially, our workload contained 50% of normal tasks, 10% Memory-Intensive tasks, 5% of CPU-Memory-Intensive tasks and 35% of CPU-Intensive tasks distributed across multiple time frames. To simulate changes, we change resource distribution at the task level and the rate of submission across different time frames to simulate changes at users' behavior. To simulate drift at the server level, our simulation considers the dynamic migration strategy and CPU core failure. This means that migration can be triggered at any time and that over time some of CPU cores may fail which will decrease the server capacity increasing its risque of becoming overloaded.

The confusion matrix is used for RTSLPS Evaluation as it gives an interesting overview of how well a model is doing. As shown in Table 3 the accuracy, the recall, F1 score and AUC-ROC curve of each class is computed to study the performance of our proposed System.

6.2.2. Results and discussion

Results show that RTSLPS exhibits a good performance with a mean accuracy of 96,5%. As the Table 3 shows, our system was able to maintain high accuracy, high recall and a high F1 score across all classes. This means that it is doing very well dealing with the dynamic and changeable nature of server load in cloud computing. Moreover, the results show that RTSLPS detains a good AUC-ROC across all classes, which means that the system shows great ability in disguising between classes.

Finally, RTSLS was able to maintain a mean completion time of 0,85 *CPUsec* over 100 000 instances.

6.2.3. Conclusion

RTSLPS was able to handle different kinds of changes while delivering good accuracy in short completion time. The overall performance of the proposed system was very promising as all metrics show that the proposed system was able to handle different kinds of changes related to changes at users' level or at the server level.

7. Conclusion

In this paper, we presented a real-time server load prediction system RTSLPS based on Hoefding Adaptive Tree augmented by ensemble drift detectors HAT-nDetectors to detect and react to changes of server load distribution over time. As a matter of fact, the dynamic and unpredictable nature of Cloud workload requires flexible techniques that can handle load fluctuations and sudden

Table 3
Confusion matrix from real time server load prediction system classification.

	Normal	Overloaded	CPU-Overloaded	MEM-Overloaded	UnderUsed
Normal	48641	192	152	72	50
Overloaded	646	11568	513	282	29
CPU-Overloaded	232	137	17244	217	21
MEM-Overloaded	90	58	86	9420	2
UnderUsed	391	9	5	9	5898
Accuracy (%)	97,2	96,4	95,8	94,2	98,3
Recall	95,7	88,7	96,6	97,5	93,4
F1 Score	96,4	92.4	96.2	95.8	95.8
AUC-ROC	87,35	85.5	83	87.5	88.4

spikes in real-time. In this work, we considered users' behavior and VM interference as main factors that contribute to shaping the unpredictable nature of server load. Incoming trend detection that aims to classify the incoming tasks is proposed in order to monitor users' behavior. In this paper, unknown task classification is considered. The experimentation showed that our approach was able to maintain an accurate prediction facing concept drift with a short response time.

This work is a step toward building a real-time load balancing framework that predicts server load and makes migration decisions accordingly. As future work, we are working on the following improvements: i) Improving uncertain task classification module, ii) Proposing real-time load balancing strategy based on RTSLPS that considers dynamic VMs migration in order to maximize server resource usage and minimize VMs interference rate.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Albert, B., Ricard, G., 2007. Learning from time-changing data with adaptive windowing. In SDM, SIAM:443-448.

Albert, B., Geoff, H., Bernhard, P., Ki, Richard, Ricard, G., 2009. New ensemble methods for evolving data streams. In: 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

Albert, B., Geoff, H., Richard, K., Bernhard, P., 2010. MOA: massive online analysis. J. Mach. Learn. Res. 11, 1601–1604.

Albert, B., Gianmarco, D.F.M., Jesse, R., Geoff, H., Bernhard, P., 2015. Efficient Online Evaluation of Big Data Stream Classifiers. KDD. 59-68.

Babu, D., Ramadevi, Y., Ramana, K., 2017. PGNBC: Pearson Gaussian Naïve Bayes classifier for data stream classification with recurring concept drift. Intell. Data Anal. 21 (5), 1173–1191.

Biao, Q., Yuni, X., Fang, L., 2009. DTU: A Decision Tree for Uncertain Data. In Advances in Knowledge Discovery and Data Mining.(PAKDD09), pp. 4–15.

Breiman, L., 2001. Mach. Learn. 45, 5. https://doi.org/10.1023/A:1010933404324.

Brennan, M., 2019. Jennifer Anistons Instagram debut sends platform crashing. Retrieved 16 October 2019, fromhttps://www.theguardian.com/film/2019/oct/15/jennifer-anistons-instagram-debut-sends-platform-crashing.

Caglar, F., Shashank, S., Aniruddha, S.G., 2014. Towards a performance interference-aware virtual machine placement strategy for supporting soft real-time applications in the cloud. REACTION.

Campos, M., 2016. CloudSim Plus. [online] CloudSim Plus. Available at: http://cloudsimplus.org/ (accessed 31 Aug. 2016).

Chen, J., Wang, Y., 2018. A resource demand prediction method based on EEMD in cloud computing. Proc. Comput. Sci. 131, 116–123.

Chen, J., Wang, Y., 2019. A hybrid method for short-term host utilization prediction in cloud computing. J. Electrical Comput. Eng. 2019, 1–14.

Clarke, R., 2010. User Requirements for Cloud Computing Architecture. CCGrid 2010 – 10th IEEE/ACM International Conference on Cluster, Cloud, and Grid Computing, pp. 625–630.https://doi.org/10.1109/CCGRID.2010.20.

Di, S., Kondo, D., Cappello, F., 2014. Characterizing and modeling cloud applications/jobs on a Google data center. J. Supercomputing 69 (1), 139–160.

Du, L., Song, Q., Zhu, L., Zhu, X., 2014. A selective detector ensemble for concept drift detection. Comput. J. 58 (3), 457–471.

Duggan, M., Shaw, R., Duggan, J., Howley, E., Barrett, E., 2018. A multitime-steps-ahead prediction approach for scheduling live migration in cloud data centers. Software: Practice and Experience 49 (4), 617–639.

Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P., 2014. Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications, first ed., vol. 1, Springer, Ed. India: Springer-Verlag Wien.

Gomes, H., Bifet, A., Read, J., Barddal, J., Enembreck, F., Pfharinger, B., et al., 2017. Adaptive random forests for evolving data stream classification. Mach. Learn. 106 (9–10), 1469–1495.

Google. Google Cluster Data V2. Available:http://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1.

Gopal, K.S., Sunilkumar, S.M., 2016. Virtual resource prediction in cloud environment: a Bayesian approach. J. Netw. Comput. Appl. 65, 144–154.

Grozev, N., 2014. CloudSim and CloudSimEx [Part 1]. [online] Nikgrozev.com. Available at:https://nikgrozev.com/2014/06/08/cloudsim-and-cloudsimex-part-1/ (accessed 26 Jul. 2017).

Hajer, T., Zaki, B., Zoulayka, B., Mohammed, M.G., 2017. Server load prediction using stream mining. International Conference on Information Networking (ICOIN), Da Nang, pp. 653–661.

Huang, Z., Peng, J., Lian, H., Guo, J., Qiu, W., 2017. Deep recurrent model for server load and performance prediction in data center. Complexity 2017, 1–10.

Hulten, G., Spencer, L., Domingos, P., 2001. Mining Time-changing Data Stream. In: Proc. Of the 7th International Conference on Knowledge Discovery and Data Mining. San Francisco, USA: [s.n.].97-106.

Hussain, A., Aleem, M., 2018. GoCJ: Google cloud jobs dataset for distributed and cloud computing infrastructures. Data 3 (4), 38.

Ismael, S.M., Peter, G., Paul, T., Jie, X., 2013. An approach for characterizing workloads in Google cloud to derive realistic resource utilization models. In: IEEE 7th International Symposium on Service Oriented System Engineering, Redwood City, pp. 49–60.

Ismael, S.M., Peter, G., Paul, T., Jie, X., 2014. Analysis, modeling and simulation of workload patterns in a large-scale utility cloud. J. IEEE Trans. Cloud Comput. 2 (2), 208–221.

Javadi, S.A., Gandhi, A., 2017. DIAL: Reducing Tail Latencies for Cloud Applications via Dynamic Interference-aware Load Balancing. IEEE International Conference on Autonomic Computing (ICAC), Columbus, OH, pp. 135–144.

Jian, C., Jiwen, F., Minglu Li Chen, J., 2014. CPU load prediction for cloud environment based on a dynamic ensemble model. Softw. Pract. Exp. 44 (7), 793–804.

Joo, G., Pedro, M., Gladys, C., Pedro, R., 2004. Learning with drift detection. In: SBIA Brazilian Symposium on Artificial Intelligence, pp. 286–295.

Kyosuke, N., Kaichiro, Y., 2007. Detecting concept drift using statistical testing. In: Proc. of the 10th Intern. Conf. on Discovery Science, ser. DS 07, V. Corruble, M. Takeda, and E. Suzuki, Eds., vol. 4755. Springer, pp. 264–269.

Liang, C., Zhang, Y., Song, Q., 2010. Decision Tree for Dynamic and Uncertain Data Streams. In: Proceedings of 2nd Asian Conference on Machine Learning, 13:209–224.

Li, L., Wang, Y., Jin, L., Zhang, X., Qin, H., 2019. Two-stage adaptive classification cloud workload prediction based on neural networks. Int. J. Grid High Performance Comput. 11 (2).

Lobert, C., 2019. 7 important cloud computing statistics and why they matter. Retrieved 23 mars 2019, fromhttps://www.vcsolutions.com/7-important-cloud-computing-statistics-and-why-they-matter/.

Maciel, B.I.F., Silas, G.T.C.S., Roberto, S.M.B., 2015. A Lightweight Concept Drift Detection Ensemble. IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI).

Minarolli, D., Mazrekaj, A., Freisleben, B., 2017. Tackling uncertainty in long-term predictions for host overload and underload detection in cloud computing. J. Cloud Comput. 6 (1).

Moamar, S.M., 2016. Handling Concept Drift. In Learning from Data Streams in Dynamic Environments. In: SpringerBriefs in Applied Sciences and Technology. Springer International Publishing, pp. 33–59.

Nikunj, O., Stuart, R., 2001. Online bagging and boosting. In: Artificial Intelligence and Statistics 2001. Morgan Kaufmann, pp. 105–112.

Patrick, P., 2017. Benchmarking cloud servers: A Cloud Computing Insider's Guide CloudSigma. [online] CloudSigma. Available at:https://www.cloudsigma.com/benchmarking-cloud-servers-a-cloud-computing-insiders-guide/ (accessed 4 Nov. 2018).

Pedro, D., Geoff, H., 2000. Mining high-speed data streams. In: Proc. 6th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, pp. 71–80.

Pu, X., Liu, L., Mei, Y., Sivathanu, S., Koh, Y., Pu, C., 2010. Understanding Performance Interference of I/O Workload in Virtualized Cloud Environments. 2010 IEEE 3rd International Conference on Cloud Computing, Miami, FL, pp. 51–58.

Qiu, F., Zhang, F.B., Guo, J., 2016. A deep learning approach for VM workload prediction in the cloud. 17th IEEE/ACIS International Conference on Software Engineering. Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Shanghai, pp. 319–324.

Rahmanian, A., Ghobaei-Arani, M., Tofighy, S., 2018. A learning automata-based ensemble resource usage prediction algorithm for cloud computing environment. Future Gener. Comput. Syst. 79, 54–71.

Sehgal, N.K., Bhatt, P.C.P., 2018. Cloud workload characterization. In: Cloud Computing. Springer, Cham.

Shishira, S.R., Kandasamy, A., Chandrasekaran, K., 2017. Workload Characterization: Survey of Current Approaches and Research Challenges. ICCCT-2017.

Shrestha, A., Mahmood, A., 2019. Review of Deep Learning Algorithms and Architectures. IEEE Access, vol. 7, pp. 53040–53065.

Song, B., Yu, Y., Zhou, Y., Wang, Z., Du, S., 2017. Host load prediction with long short-term memory in cloud computing. J. Supercomputing 74 (12), 6554–6568.

Vic (J.R.) W., 2011. Introduction to Cloud Computing and Security, Cloud Computer Security Techniques and Tactics, pp. 1–27.

Wiecki, T., Sofer, I., Frank, M., 2013. HDDM: hierarchical bayesian estimation of the drift-diffusion model in python. Front. Neuroinf., 7

Xu, F., Liu, F., Liu, L., Jin, H., Li, B., Li, B., 2014. iAware: making live migration of virtual machines interference-aware in the cloud. IEEE Trans. Comput. 63 (12), 3012–3025.

Xu, F., Liu, F., Jin, H., Vasilakos, A.V., 2014. Managing performance overhead of virtual machines in cloud computing: a survey, state of the art, and future directions. Proc. IEEE 102 (1), 11–31.

Xu, F., Liu, F., Jin, H., 2016. Heterogeneity and interference-aware virtual machine provisioning for predictable performance in the cloud. IEEE Trans. Comput. 65 (8), 2470–2483.

Younggyum, K., Rob, K., Paul, B., Mic, B., Zhihua, W., Calton, P., 2007. An Analysis of Performance Interference Effects in Virtual Environments. IEEE International Symposium on Performance Analysis of Systems and Software, San Jose, CA, pp. 200–209.

Zhang, Q., Zhani, M.F., Zhang, S., Zhu, Q., Boutaba, R., Hellerstein, J.L., 2012. Dynamic energy-aware capacity provisioning for cloud computing environments. In: Proceedings of the 9th International Conference on Autonomic Computing. ACM, pp. 145–154.

Zhong, W., Zhuang, Y., Sun, J., Gu, J., 2018. A load prediction model for cloud computing using PSO-based weighted wavelet support vector machine. Appl. Intell. 48.

Zhu, Q., Zhu, Tung, T., 2012. A performance interference model for managing consolidated workloads in QoS-aware clouds, in Proc. 5th Int. Conf. Cloud Comput. pp. 170–179.

Zia Ullah, Q., Hassan, S., Khan, G.M., 2017. Adaptive resource utilization prediction system for infrastructure as a service cloud. Computational intelligence and neuroscience 1–13.