This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2350475, IEEE Transactions on Cloud Computing

Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS

Rodrigo N. Calheiros, Enayat Masoumi, Rajiv Ranjan, Rajkumar Buyya

Abstract—As companies shift from desktop applications to Cloud-based Software as a Service (SaaS) applications deployed on public Clouds, the competition for end-users by Cloud providers offering similar services grows. In order to survive in such a competitive market, Cloud-based companies must achieve good Quality of Service (QoS) for their users, or risk losing their customers to competitors. However, meeting the QoS with a cost-effective amount of resources is challenging because workloads experience variation over time. This problem can be solved with proactive dynamic provisioning of resources, which can estimate the future need of applications in terms of resources and allocate them in advance, releasing them once they are not required. In this paper, we present the realization of a Cloud workload prediction module for SaaS providers based on the Autoregressive Integrated Moving Average (ARIMA) model. We introduce the prediction based on the ARIMA model and evaluate its accuracy of future workload prediction using real traces of requests to web servers. We also evaluate the impact of the achieved accuracy in terms of efficiency in resource utilization and QoS. Simulation results show that our model is able to achieve an average accuracy of up to 91%, which leads to efficiency in resource utilization with minimal impact on the QoS.

Index Terms—Cloud Computing; Workload Prediction; ARIMA.

1 Introduction

Cloud computing [1] has evolved from a set of promising virtualization and data center technologies to a consolidated paradigm for delivery of computing as a service to end customers, which pay for such services according to its use, likewise utilities such as electricity, gas, and water. Adoption of the technology by enterprises is growing fast, and so is the number of Cloud-based companies offering Cloud-based solutions for end users.

The shift from desktop applications to public Cloud hosted Software as a Service (SaaS) business model has intensified the competition for Cloud providers. This is due to the presence of multiple providers in the current Cloud computing landscape that offer services under heterogeneous configurations. Selecting particular Cloud service configuration (e.g., VM type, VM cores, VM speed, cost, and location) translates to a certain level of Quality of Service (QoS) in terms of response time, acceptance rate, reliability, etc. In order to survive in such a competitive market, Cloud providers must deliver acceptable QoS to end-users of the hosted SaaS applications, or risk losing them.

However, one issue that arises from the transition to a SaaS model is the fact that the pattern of access to the application varies according to the time of the day, day of the week, and part of the year. It means that in some periods there are many users trying to use the service at the same time, whereas in others only a few users are concurrently accessing the servers. This makes static allocation of resources to the SaaS application ineffective, as during a period of low demand there will be excess of resources available, incurring unnecessary cost for the application provider, whereas during high utilization periods the available resources may be insufficient, leading to poor QoS and loss of costumers and revenue.

Clouds can circumvent the above problem by enabling *dynamic provisioning* of resources to applications based on workload behavior patterns such as request arrival rate and service time distributions. This means that extra resources can be allocated for peak periods and can be released during the low demand periods, increasing utilization of deployed resources and minimizing the investment in Cloud resources without loss of QoS to end users [2].

The challenge of dynamic provisioning is the determination of the correct amount of resources to be deployed in a given time in order to meet QoS expectations in the presence of variable workloads like what is observed by Cloud applications. This challenge has been tackled mainly via *reactive* approaches [3], [4], [5]—which increase or decrease resources when predefined thresholds are reached—or via *proactive* approaches [6], [7], [8]—which react to future load variations before their occurrence. The latter is typically achieved with techniques that can monitor, predict (e.g. estimating QoS parameters in advance), adapt according to these prediction models, and capture the relationship between application QoS targets, current Cloud resource allocation, and changes in workload patterns, to adjust resource allocation con-

- *R. N. Calheiros and R. Buyya are with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Australia*
- *R. Ranjan is with the Commonwealth Scientific and Industrial Research Organisation (CSIRO), Information and Communication Technologies (ICT) Centre, Acton, ACT, Australia*

figuration on-the-fly.

In previous work [9], we introduced an architecture for proactive dynamic provisioning via workload prediction—which determines how many requests per second are expected in the near future—combined with analytical models to determine the optimal number of resources in the presence of the predicted load. Although the proposed architecture recognized the need for workload prediction, it did not propose a concrete method for workload prediction. Thus, in this paper we present the design and evaluation of a realization of its workload prediction model using the Autoregressive Integrated Moving Average (ARIMA) model [10]. ARIMA is a method for non-stationary time series prediction that is composed of an autoregressive and a moving average model, and was successfully utilized for time series prediction in different domains such as finance. The **key contributions** of this paper are:

- We propose, design, and develop a workload prediction module using the ARIMA model. Our work applies feedback from latest observed loads to update the model on the run. The predicted load is used to dynamically provision VMs in an elastic Cloud environment for serving the predicted requests taking into consideration QoS parameters such as response time and rejection rate;
- We conduct an evaluation of the impact of the achieved accuracy in terms of efficiency in resource utilization and QoS of user requests.

Results show that our module achieves accuracy of up to 91%, which leads to efficiency in resource utilization with minimal impact in QoS for users.

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 introduces the application and system models that support our workload prediction architecture, which is detailed in Section 4. Section 5 contains experiments evaluating the accuracy of our proposed prediction architecture. Section 6 presents the simulation experiments evaluating the impact of the prediction in the efficiency of utilization of Cloud resources. Finally, Section 7 presents the conclusions and future work.

2 Related Work

The approaches for workload prediction in Clouds can be classified as reactive methods and proactive methods. Among reactive methods, Zhu and Agrawal [3] propose a method based on control theory to vertically scale resource configurations such as VM types, VM cores, VM speed, and VM memory. Vertical scaling is the process of increasing the resources available to each VM, rather than increasing the number of VMs (which is known as horizontal scaling). Their approach also addresses the budget constraints related to the workload execution. They apply the ARMAX model to predict CPU cycle and memory configurations required for hosting an application component. In contrast to this approach, we apply the ARIMA model to predict the future application workload behavior, which is fed into the queueing model for calculating the required VM configuration.

Bonvin *et al.* [4] propose a reactive method that scales servers based on the expected performance and profit generated by changes in the provisioning. This method is able to perform both horizontal and vertical scaling.

Similar to Bonvin *et al.*, Yang *et al.* [5] propose a reactive method for changing the resource configuration of cluster resources driven by the load incurred by the hosted application. It is based on user-defined threshold conditions and scaling rules that are automatically enacted over a virtualized cluster.

Zhang *et al.* [11] propose a reactive workload factoring architecture for hybrid Clouds that decomposes incoming workload in base workload and trespassing workload. The first one is derived from ARIMA-based prediction and handled by the local infrastructure, whereas the second is handled by a public Cloud.

The limitation of reactive platforms is that they react to changes in workload only after the change in utilization and throughput is observed in the system. Therefore, if the change is quicker than the reconfiguration time, end-users will observe poor QoS until the extra resources are available. Considering that changes in the workload typically follow patterns that are time-dependent, prediction techniques can avoid the above problem by triggering the reconfiguration before the expected increase of demand, so when the situation arises, the system is already prepared to handle it. Caron *et al.* [6] propose a method based on pattern matching for prediction of grid-like workloads in public Clouds. Gong *et al.* [12] propose a method for predicting resource demand of VMs based on predicted application workload. Islam *et al.* [8] apply Artificial Neural Networks (ANN) and linear regression for prediction of resources required for applications. Sladescu *et al.* [7] presents a system based on ANN to predict the workload to be experienced by an online auction in terms of intensity and location of the peaks.

Although techniques such as linear regression can generate predictions quicker than ARIMA, they also demand workloads that have simpler behavior than those that time series and ANN-based methods can accurately predict. Furthermore, studies [13], [14] show that web and data center workloads tend to present behavior that can be effectively captured by time series-based models. Thus, to increase the applicability of the proposed architecture, we adopt ARIMA-based prediction for our proposed architecture.

Tran *et al.* [14] applied the ARIMA model for prediction of server workloads. It targets long-time prediction (up to 168 hours), whereas we target short timespans to achieve timely reaction to workload changes. Our prediction, which is designed to be short-term and therefore quicker to be performed, is suitable for Clouds because Cloud platforms can quickly react to requests for more or less VMs. Our work also goes further ahead by applying feedback from latest observed loads to update the model on the run. Furthermore, in our work the predicted load is used for dynamically provisioning VMs for serving the predicted requests, and the impact of the prediction and provisioning is evaluated in regards to their effect on the QoS observed by end users.

Other domain-specific proactive approaches that are related to Clouds include the approach by Nae *et al.* for Massively Multiplayer Online Games [15]. Pacheco-Sanchez *et al.* [16] apply a Markovian model to predict server performance in Clouds. Roy *et al.* [13] apply the ARMA model for workload prediction in Clouds with the goal of minimizing cost, whereas the main objective of our approach is meeting QoS target of applications such as minimizing the request rejection rate, or maximizing resource utilization.

3 System and Application Models

The target system model of the architecture proposed in this paper consists of a public Cloud provider that offers to end users SaaS services backed by a PaaS layer (Figure 1) [9]. The PaaS in turn interacts with an IaaS provider that can be a third party provider. The target SaaS provider receives web requests, which are processed by the machines that are located at the IaaS layer.

For scaling up the infrastructure, the target provider deploys a number of Virtual Machines (VM) that process end user requests. To simplify the management of the infrastructure and to take advantage of profiling information, a single VM configuration, consisting of CPU power, amount of memory, and amount of storage is utilized by the SaaS provider. We also assume that the application has been profiled in the chosen VM configuration, so the provider has information about the VM's expected performance.

A single application instance executes on each VM, and since current Cloud providers do not support dynamic changes in the VM's specifications without downtime, increasing and decreasing the total number of VMs running the application is the most suitable option for utilization of elastic computing infrastructures[1], as it brings additional benefits such as higher fault tolerance and higher resilience to performance degradation caused by VM failures (as the crash of one of the VMs will not affect the others, enabling the application to continue serving customers using the VMs that are running).

The target application is web applications. Client requests consist of http requests that are processed by a web server running on the VMs. QoS targets of relevance to the system are response time T_s, defined as the maximum negotiated time in the SLA for serving a user's request and rejection rate $Rej(G_s)$, which is the proportion of incoming requests that cannot be served without violating T_s [9].

4 System Architecture

One of the key characteristics of Clouds is elasticity, which enables the infrastructure to be scaled or down to meet the demand of applications. However, instantiation of new VMs is not an immediate operation. Depending on Cloud providers' infrastructure architecture and their hypervisor policies, launching a new VM involves a non-negligible start-up time. The start-up time is long enough to be noticed by the clients and dramatically decreases the users' experience, which may result in abandonment of the application. Apart from potential financial losses due to decline in the number of users, the software provider may also be liable for not delivering the minimum required QoS.

Although standby VM instances may be helpful for tolerating sudden increases in number of requests, those standby VMs are more likely to be idle most of the times reducing the overall system utilization while increasing the operational cost. Furthermore, if the increase in the number of requests exceeds the load that standby VMs can handle, the problem of poor QoS arises again. Thus, a different approach must be sought for the Cloud provisioning problem.

One approach that has been explored is based on workload prediction: accurate predictions of the number of end-users' future service requests enable SLA's QoS targets to be met with reduced utilization of Cloud resources. As requests pattern vary depending on the application type, this paper focuses on request patterns that exhibit seasonal behavior, such as requests to Web or online gaming servers [15] [17]. To overcome the uncertainty in workload patterns in Cloud environments and minimize the estimation error in predicting future requests while maintaining optimal system utilization, in previous work [9] we proposed an adaptive provisioning mechanism in order to achieve the following QoS targets:

- Automation: The whole process of provisioning should be transparent to users;
- Adaptation: The provisioner should be aware of dynamic and uncertain changes in the workload and react to them accordingly;
- Performance assurance: In order to meet QoS targets, resource allocation in the system must be dynamic.

The key components of the proposed provisioning system, depicted in Figure 1 are [9]:

1) *Application Provisioner:* Receives accepted requests from the *Admission Control* module and forwards them to VMs that have enough capacity to process them. It also keeps track of the performance of the VMs. This information is passed to the *Load Predictor and Performance Modeler*. The Application Provisioner also receives from such module the

[1] Although CloudSigma claims to support dynamic changes in the hardware specifications of running VMs, such alterations require the VM to be stopped and restarted again, which imposes the same not-negligible setup time as starting new VMs in conventional IaaS providers.

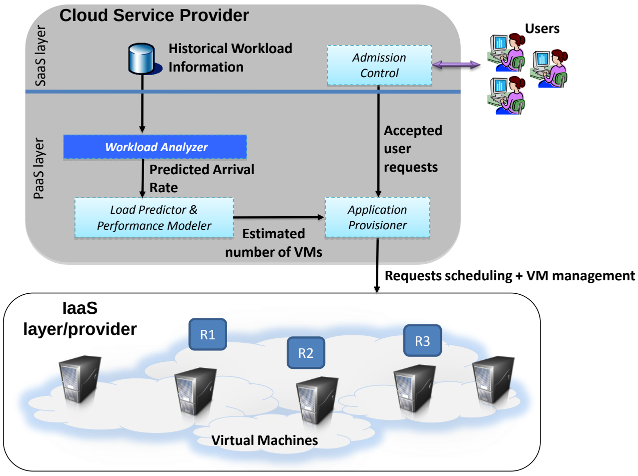


Fig. 1. Architecture for adaptive Cloud provisioning [9]. The key component of this paper, the Workload analyzer, is highlighted in the figure.

expected number of VMs required by the application. If the expected number of VMs differs from the number of provisioned VMs, the number is adjusted accordingly (by either provisioning new VMs or decommissioning unnecessary VMs).

2) *Load Predictor and Performance Modeler:* Decides the number of VMs to be allocated, based on the predicted demand by the *Workload Analyzer* module and on the observed performance of running VMs by the Application Provisioner. The performance is modeled via queueing networks, which, based on the predicted arrival rate of requests, return the minimum number of VMs that is able to meet the QoS metrics.
3) *Workload Analyzer:* Generates an estimation of future demand for the application. This information is then passed to the *Load Predictor and Performance Modeler* module.

To make the proposed architecture effective, a strong knowledge about the application workload behavior is required by the system so the performance model can be accurate. Therefore, the most suitable deployment model for the architecture is Software as a Service, where a queueing model can be built for each application offered to end users as a service. In the SaaS layer, the admission control module ensures that no application instance will get further requests if the capacity of the queue is exhausted. In this case, all the upcoming requests are rejected, because otherwise it is most likely that T_s would be violated. Accepted requests are forwarded to the Cloud provider's PaaS layer where the proposed system is implemented.

In our previous work [9], the system architecture was presented in a high-level view, without presenting a concrete implementation of each of its components. In this paper, we present a realization of the *Workload Analyzer* component of the architecture. The prediction method uses the Auto-Regressive Integrated Moving Average (ARIMA) model. The prediction gives the *Application Provisioner* enough time to react against any precipitous increase in workload by starting new VMs without compromising T_s while maintaining the overall system utilization above a given threshold.

4.1 Workload Analyzer

The Workload Analyzer realization we propose in this paper implements workload prediction using the general ARIMA time series process [10]. ARIMA has been chosen for the implementation of our module because the underlying workload fits well in the model: previous research observed that web workloads tend to present strong auto-correlation [17], [18].

At the start of the execution, in a preliminary step, the historical workload data is fed into the Workload Analyzer, where it fits the ARIMA model on them. When the system is operational, it delivers an estimation of the workload with one time-interval in advance. The length of the time interval can be adjusted to better fit the specific application. The only requirement for efficient system utilization is that the time interval should be long enough to allow extra VMs to be deployed. Therefore, time windows as short as 10 minutes could be suitable depending on the selected Cloud provider [19].

The request time series contains the number of observed requests at each time interval. It is implemented as a cyclic buffer so that at the next prediction cycle, the actual number of requests (obtained from the original dataset) is added to the time series used in prediction while discarding the oldest value. After constructing the request time series, the process of fitting the ARIMA model is initiated based on the Box-Jenkins method [10].

According to this method, the time series must be transformed into a stationary time series, that is, for each $(X_t, X_{t+\tau})$, τ being the time difference (lag) between two data points, the mean and variance of the process must be constant and independent of t. In addition, the auto-covariance between X_t and $X_{t+\tau}$ should be affected only by τ. This transformation is achieved by differencing the original time series. The number of times the original time series has to be differenced until it becomes stationary constitutes the d parameter of the $ARIMA(p,d,q)$ model.

The values of q and p are determined by analyzing the *autocorrelation* and *partial autocorrelation* plots of the historical data, respectively. In the context of this work, historical data means the observed number of requests per second received by the system in some past time interval.

The autocorrelation plot is used to determine how random a dataset is. In the case of random data, the autocorrelation values approach zero for all time-lagged values, otherwise, one or more autocorrelation values approach 1 or -1. In the autocorrelation plot, the horizontal axis represents the time lags. Values on the vertical axis are calculated using the autocorrelation coefficient R_h:

$$R_h = \frac{C_h}{C_0}, \tag{1}$$

where C_h is the auto-covariance function defined as:

$$C_h = \frac{1}{N}\sum_{t=1}^{N-\tau}(X_t - \bar{X})(X_{t+\tau} - \bar{X}), \tag{2}$$

where N is the number of samples and $\bar{X}$ is the average of samples $X_t, t = 1...N$. C_0 is the variance function:

$$C_0 = \frac{1}{N}\sum_{t=1}^{N}(X_t - \bar{X})^2 \tag{3}$$

The partial autocorrelation at τ is the autocorrelation between X_t and $X_{t-\tau}$ that is accounted only by lags above $\tau - 1$. If a stationary time series has an autoregression component of order p, its partial autocorrelation plot falls below the significant level at $\tau = p + 1$. The number of lags before the autocorrelation values drop below the significant level is the value of q for the moving average component of the ARIMA model.

Using the above method to determine the terms p, d, and q of the ARIMA model, the historical workload information is fit to the model to be used for prediction of future workload values.

4.2 System Design

The class diagram of the ARIMA-based workload prediction system is shown on Figure 2. The *ARIMAWorkloadAnalyzer* is the core component of the system and realizes the *WorkloadAnalyzer* component of Figure 1. By implementing the *IFeedbackable* interface, it is capable of taking feedbacks. In our system, current workload information, received from external components is modeled as a feedback signals and fed into the ARIMAWorkloadAnalyzer to make it aware of the most recent workload changes.

ARIMAWorkloadAnalyzer stores the given feedback signals into a cyclic buffer. The length of this buffer, which corresponds to the number of time intervals in past affecting the current prediction value, is configured at the start of the system based on characteristics of the application workload (number of received requests per second).

ARIMAWorkload accomplishes the workload prediction through the *Forecaster* class. This class has a connection to a statistical backend (the R `forecast` package [20], which for simplicity is not presented on the diagram). It accepts a time series from the ARIMAWorkloadAnalizer and prepares it for submission to the statistic engine, where ARIMA model is fitted on them. For a given time series, the statistical back-end replies with a predicted value, along with its corresponding 80% and 95% confidence levels. The Forecaster class then parses and encapsulates this reply into an instance of *ForecastEntity* class and passes it back to the ARIMAWorkloadAnalyzer. The accuracy of the ARIMA-based workload prediction is evaluated in the next section.

The steps of the prediction procedure and its components are shown on Figure 3.

4.3 Modeling and Forecast Complexity

There are many different methods that can be applied for ARIMA fitting. We adopted the fitting process from R, which implements the Hyndman-Khandakar algorithm [20]. This method is broadly composed of three sequential stages [21]: (i) determination of the number of differencing steps of the model (parameter d); (ii) actual differentiation of the time series d times; and (iii) selection of the best fit model.

The method used by R for the first stage applies successive Kwiatkowski-Phillips-Schmidt-Shin (KPSS) tests [22] to determine d. This method has complexity O(n^2), where n is the number of points in the workload used for prediction. The second stage applies d differentiations over the data, what makes this stage O(n). The third stage evaluates a fixed number of variations of the ARIMA model and selects the one that better fits the input data [20]. The fitting process can be accomplished with complexity O(n^2) [23]. For k repetitions, the complexity of this stage is O(kn^2). Given that k is finite and does not grow with the size of the input, the complexity of this stage can also be estimated as O(n^2). Because each of these steps is executed sequentially, the complexity of the fitting method can be established as O(n^2). Although the complexity is determined by the number of observations, in our application this value is constant because of the use of a cyclic buffer. Also, the number is reduced because of the use of lags to operate only with data from relevant time periods for the period being predicted.

The prediction procedure is straightforward once the ARIMA has been determined and values from previous observations are available. Because the propose method predicts one time interval ahead, it has complexity O(p), where p is the order of the autoregressive component of the ARIMA model.

5 Performance Evaluation

The system was evaluated with real traces of requests to the web servers from the Wikimedia Foundation[2]. These traces contain the number of http requests received for each of the project's resources (static pages, images, etc) aggregated in 1-hour intervals and are publicly available for download[3]. It also contains the project name associated with each resource being requested and the language of each accessed resource. We consider only requests to English Wikipedia resources in these

[2] http://www.wikimedia.org

[3] http://dumps.wikimedia.org/other/pagecounts-raw

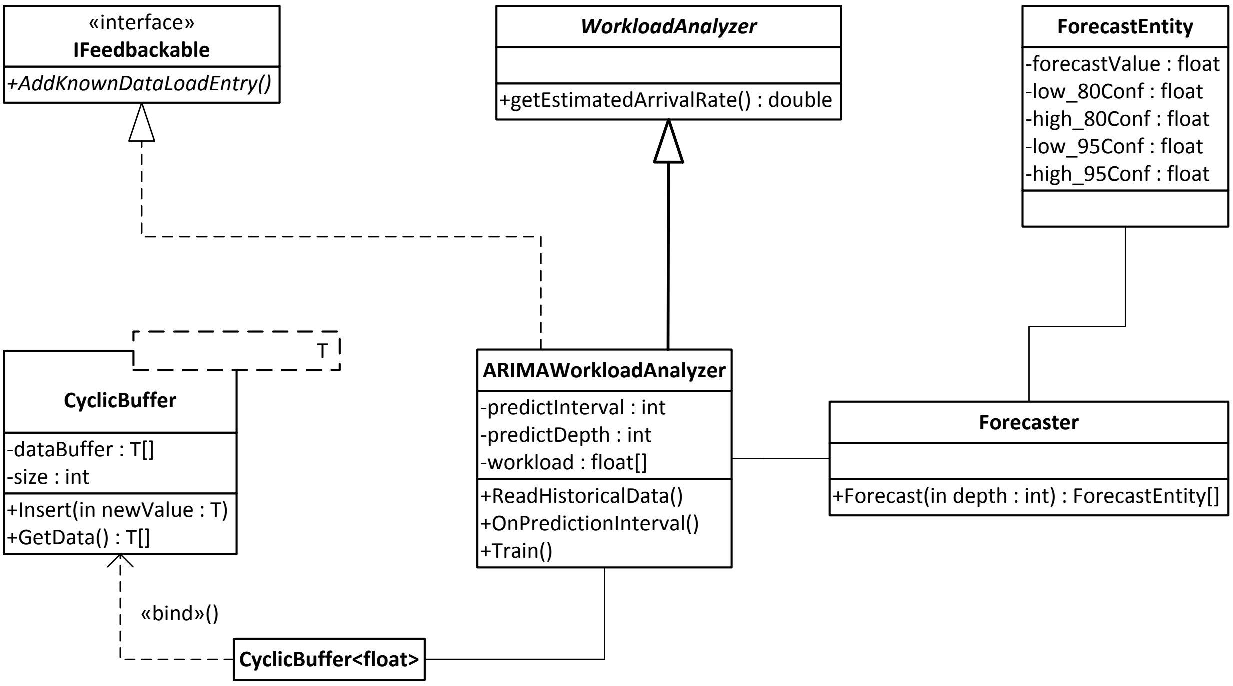


Fig. 2. Class diagram for ARIMA-based workload prediction.

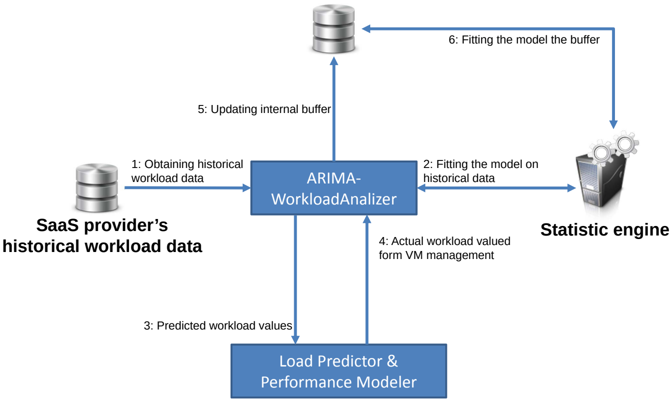

6: Fitting the model the buffer
5: Updating internal buffer
1: Obtaining historical workload data
ARIMA-WorkloadAnalizer
2: Fitting the model on historical data
SaaS provider's historical workload data
Statistic engine
4: Actual workload valued form VM management
3: Predicted workload values
Load Predictor & Performance Modeler

Fig. 3. Workload prediction steps and its involving components.

experiments. An analysis of patterns of web requests to Wikipedia servers was presented by Urdaneta *et al.* [17].

In order to observe weekly patterns, we use four weeks of the traces, dated from midnight, 01 January 2011 to 5 pm, 04 February 2011. The first three weeks are used for training purposes. The requests corresponding to such a period are transformed to a time series process (i.e. the values p, d and q of the ARIMA model are defined). At runtime, the model is constantly updated: whenever new requests arrive, they are incorporated to the time series and older data is removed from the time series in the same amount. The fitting process is then repeated, what may lead to changes in the values of p, d, and q.

The fourth week is used for evaluation purposes. Based on the training dataset, the demand for each hour of the fourth week is predicted. The output of the prediction procedure is a number, accompanied by two confidence ranges, covering the 80% and 95% bands, for each hour of the fourth week. Figure 4 presents the predicted and actual values (i.e., the value observed in the traces) and corresponding confidence ranges, for the fourth week of the workload.

The accuracy of the prediction is evaluated using various error metrics. The results are presented in Table 1. The *Predicted* column contains the accuracy according to different metrics. The *Low 80%* and *High 80%* contain the limits for the 80% confidence interval for the prediction. The table also reports the same for the 95% confidence interval. The output of the confidence intervals can be used when one is willing to sacrifice SLA in favor of utilization (by choosing the lower 80% or 95%) or decreasing utilization in order to provide better response times (by choosing the higher 80% and 95%).

Figures 4(b) and (c) show that, although utilization of the high edges of the confidence intervals minimize the occurrences of underestimations, it also decreases the prediction accuracy down to an average of 78% in the high 95% case. If high to very high system utilization, which minimizes the operational cost, is the main priority the number of VMs should be selected based on either low 80% or 95% values. As a side effect, this decision increases the underestimation cases, which leads to an average of 85% prediction accuracy in the low 95% case. Results from the column *Predicted* of Table 1 show the accuracy for different metrics when the optimal number of VMs for increasing the performance and user experience, while decreasing underutilization cases (Figure 4(a)), is selected. In this case, the average

TABLE 1
Prediction accuracy by various metrics. Predicted: use of the actual output of the prediction. Low and High 80%: use of one of the edges of the 80% confidence interval as the predicted value. Low and High 95%: use of one of the edges of the 95% confidence interval as the predicted value.

Accuracy metric	Predicted	Low 80%	High 80%	Low 95%	High 95%
Root mean square deviation (RMSD)	1146.26	1570.16	1959.95	2136.40	2582.36
Normalized root mean square deviation (NRMSD)	0.15	0.20	0.25	0.27	0.33
Mean absolute deviation (MAD)	876.98	1151.56	1461.03	1680.13	2038.47
Mean absolute percentage error (MAPE)	0.09	0.10	0.16	0.15	0.22

prediction accuracy increases to 91%.

These results show general trends expected to be achieved by our approach when it is applied for workloads that contain auto-regressive and moving average components on its composition, regardless of the particular Cloud deployment model (e.g., public Cloud or private Cloud). As suggested in previous work [13], [14], this is expected in requests for web servers and Cloud data centers, and such a behavior in the Wikipedia traces has been firstly identified by Urdaneta *et al.* [17]. For this type of workloads, similar results to those presented in this section can be expected. The particularities of the workload is one factor to be considered when selecting a prediction technique, as different techniques can present different performance depending on the characteristic of each workload. For example, Sladescu *et al.* [7] successfully utilized artificial neural networks to predict workloads bursts in auction websites.

6 Impact of Prediction Accuracy on Application's QoS

Although the proposed method can generate predictions all the times, and the achieved accuracy indicate that our proposed method achieves good accuracy, it does not say too much about how the obtained accuracy impacts the QoS of applications and data center utilization, which are the metrics of interest for Cloud providers willing to apply the method. Thus, in this section we present experiments aiming at evaluating how these important metrics are impacted by the accuracy of the prediction mechanism.

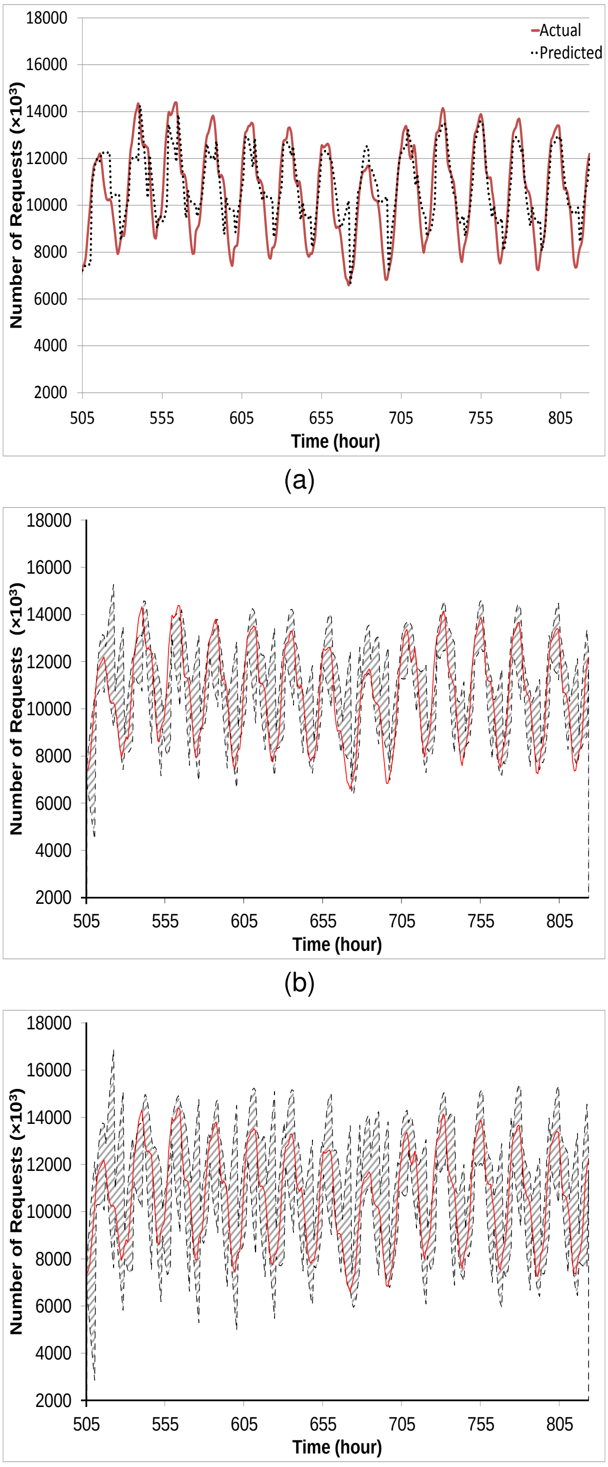
The experiment was performed via simulation using the CloudSim [24] toolkit. CloudSim is a toolkit that contains a discrete event simulator and classes that enable users to model Cloud environments, from providers and their resources (physical machines, virtual machines, and networking) to customers and requests. During the simulation execution, user requests for resources and application execution trigger provisioning and scheduling decisions in the data center that affects the execution time of the simulated application, and thus it enables the evaluation of effects of policies for scheduling and provisioning in the performance of applications, ultimately resulting in the observed response time of requests at the user side.

The simulated environment is composed of a data center implementing the architecture described in Section 4. The data center contains 1000 hosts, each of which having 8 cores and 16GB of RAM. At the start of the simulation, the data center hosts 50 VMs for processing incoming requests.

The prediction method using the ARIMA model—described and evaluated in the previous section—is implemented in the Workload Analyzer component. The expected number of upcoming requests for the next hour is predicted based on the number of previously observed requests, and when the actual value is available, it replaces the predicted value, and is used in the next round of prediction.

The Load Predictor and Performance Modeler operates as in our previous work [9]. It consists of a network of n M/M/1/k queues, where n is the number of VMs in the system. The module operates over a single type of virtual machine, whose average response time of the application is assumed to be known (for example, based on historic information). The queue size k is inferred from the expected response time of requests on the VM and the maximum response time that is the QoS attribute agreed between provider and customers. For these experiments, VMs have the following configuration: 1 CPU core with 1 ECU, 2GB of RAM and 10GB of storage. If a different VM type was chosen, the execution time of requests would need to be estimated, resulting in a new value for k.

The input received from the Workload Analyzer is used by the Load Predictor and Performance Modeler as the arrival rate for the queue systems. It then searches for the optimal value of n that is compliant with QoS expectation at the maximum possible utilization rate. The search is conducted in a way that, at each iteration, it narrows down the candidate answers, eliminating the reevaluation of values which already have been concluded not to be appropriate. Once the optimal value of n is



(a)

(b)

(c)

Fig. 4. Results of the ARIMA-based prediction for one week period of the workload. (a) Predicted and actual (obtained from the trace) values. (b) 80% confidence interval and actual values. (c) 95% confidence interval and actual values.

found, it is forwarded to the Application Provisioner as the required number of VM for the next time interval. This value is compared with the current number of VMs so a decision can be made about creation of VMs (if the current number of VM is smaller than n), decommission of VMs (if the current number is bigger than n), or no action (if the current number of VMs is equal to n). Notice that, if decommission of VMs is necessary, it is not triggered immediately. Instead, the exceeding machines stop receiving new requests and are destroyed only when all the requests they are processing are completed.

The Application Provisioner component of the adaptive Cloud provisioning architecture issues VM creation and decommission commands. This action is based on the calculated number of VMs—by the Load Predictor and Performance Modeler module—necessary to serve accepted requests. In this experiment, there was no restriction on the maximum number of VMs, apart from those imposed by the physical constraints of the simulated date center infrastructure.

Finally, the Admission Control ensures the QoS of already accepted requests by rejecting any upcoming requests whenever all VM queues are full. The queues are designed in a way that they only accept a number of requests whose product of average execution time and size is equal or smaller than the maximum response time as defined by the QoS.

The simulation was performed for the fourth week of the Wikipedia workload traces mentioned in Section 5. The input requests were generated as follows. The base value for each hour of simulation was obtained from the corresponding hour in the workload traces. This value was divided by 60 to define the base number of average requests per minute for that hour. Within each hour, requests were submitted at each minute, following a Poisson distribution using the base number of requests/min for the hour as the average. Each request is modeled to require 50 ms with a positive variation of 10%, uniformly distributed. The maximum response time defined by the QoS is 150 ms, rejection rate below 20%, and data center utilization above 80%. At the end of each hour, the value for the next hour was read from the trace and the process was repeated, with the updated requests/min rate used as the average for the Poisson-distributed arrivals. For CloudSim simulation purposes, submissions were grouped in batches of 1000, and as many batches as defined by the traces were submitted on each 1 minute interval. The total number of batches submitted during the one-week simulation was 3,139,260.

As the simulation runs, requests are submitted to the system, and either are rejected (if queues are full) or processed. If processed, execution times are collected and averaged for each VM. This value is also used by the Performance Modeler to estimate the number of required VMs. The simulation runs for the same time interval as the previous experiment (from midnight, 01 January 2011 to 5 pm, 04 February 2011). The Performance Modeler is executed 15 minutes before the next hour. At the end of the simulation, we collected the following metrics: execution time of accepted requests; number of rejected requests; total number of QoS violations; total utilization of the infrastructure; minimum and maximum number of VMs running at any moment; and total number of hour of VMs required for the simulation.

The above process was executed five times. At each execution, the Workload Analyzer was configured to return a different value for the prediction, similarly to the experiment in the previous section: (i) the exact prediction value; (ii) the value of the lower 80% confidence interval; (iii) the value of the lower 95% confidence interval; (iv) the value of the higher 80% confidence interval; and (v) the value of the 95% confidence interval.

6.1 Results and Discussion

Table 2 presents the results of the simulation. It contains the value obtained for each output metric. It can be observed that the assumption discussed in the previous section about using the edges of the confidence intervals or the predicted value holds: the more the prediction overestimates the arrival rate of requests, smaller the response time and rejection rate, at the cost of a smaller utilization of the data center.

We can observe that all prediction values are able to meet the QoS established for the simulation, with the worst case scenario being 5% of rejected requests, below the set target of 20%. The increase in average execution time is explained by an increase in the time each requests stays in the queue.

We also can notice that the rate of rejected requests does not improve when the value of High 95% of confidence interval is used. Improvements in this metric saturated at the High 80% value. Moreover, the Low 95% is the only value where rejection rate is above 10%. Therefore, the 95% confidence interval values give extreme results, with a marginal gain when the higher band is used, and high rejection rate in the case of the lower band. Thus they are likely to be unsuitable for most practical applications. Thus, using an 80% confidence interval value for the prediction is enough for practical purposes. More accurate prediction does not result in any benefit in the operation of the Cloud infrastructure.

The consequences of underestimating the number of incoming requests are twofold. First, since it causes less VMs that required to be deployed, the response time of the requests in the execution queues increase. This increase, in turn, causes the queues to be full most of the time, affecting the upcoming requests' rejection rate.

In order to measure the magnitude of execution time violations, we calculated the amount of requests that were executed within the required time, along with the proportion of them missed the deadline by 5%, 10%, 15%, 20%, and more than 20%. As depicted in Figure 5, for all cases, more than 80% of deadlines were misses by a margin below 5%, and more than 99% of requests were delayed by a margin up to 10%. All the missed deadlines occur by a margin that was equal or less than 15% of the required execution time. This demonstrates that our method is effective in meeting QoS expectations in terms of execution time and rejection rate.

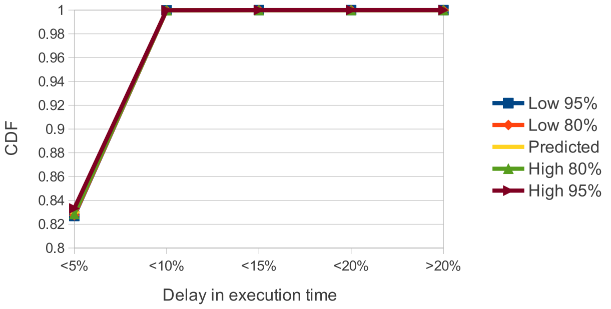


Fig. 5. Cumulative distribution of delayed requests in function of delay size. It can be noticed that 80% of requests were delayed by less than 5%, and 99% of requests were delayed by less than 10%.

Regarding to data center utilization, all the prediction values were able to achieve the target of keeping utilization above 80%. As expected, the minimum and maximum number of VMs is determined by the confidence level of prediction in use, the higher the degree of overestimation of incoming requests, the higher the minimum and maximum allocated VMs and consequently the higher the number of VM hours required to process the workload. The savings in total VM hours between the provisioned with the prediction and static allocations based on the maximum VMs varies from 16.64% (High 80%) to 34.08% (Low 95%), demonstrating the advantages of elasticity provided by Clouds against provisioning for peak demands. Furthermore, achieving the same number of VM hours with a static allocation (i.e., the same number of VMs running at all times, irrespective of the load) would allow in the best case allocation of 172 machines (for High 95%), a value that is 21% below the maximum number of VMs allocated in the same case, and therefore still unable to meet the QoS demands that our approach achieves.

About execution time of the prediction process, it executed in an average of 1.1 s, with standard deviation of 34.29 ms on a Intel CORE i7-2600 CPU[4] with 8 GB of RAM. This execution time is much smaller than the typical deployment time of a virtual machine (which is in the order of minutes) and thus it does not compromise the whole operation of the system. Thus, for workloads such as the one studied in the section, our proposed method is able to deliver accurate prediction within a small amount of time, enabling the data center to timely react to changes in the incoming workload without impacting applications' QoS.

[4] Quad core, 3.4 GHz and 8 MB of cache.

TABLE 2
Results of the simulation showing how different prediction values affect the QoS of requests and utilization of the data center.

Output metric	Low 95%	Low 80%	Predicted	High 80%	High 95%
Average service time (ms)	110.51	99.39	85.48	73.19	65.42
Standard deviation of service time	27.83	31.68	33.05	28.49	23.27
% of rejected requests	13	8	4	1	1
% of QoS violations	5	3	2	1	0
Data center utilization (%)	98.74	96.67	91.00	86.44	83.31
Minimum number of VMs	37	70	89	101	95
Maximum number of VMs	193	184	197	198	218
VM Hours	40582.87	43985.14	48605.68	52678.68	55017.67

7 Conclusions and Future Work

Together with the increasing shift from desktop-based applications to SaaS-based applications hosted on Clouds, there are growing concerns about the QoS of such Cloud applications. Because of the raising competitiveness in the SaaS market, application providers cannot afford to lose their customers to the competitors as a result of insufficient QoS.

One of the key factors affecting QoS is the dynamicity in the workload, which leads to variable resource demands. If at any given moment the workload exceeds resources' capacity, QoS on that particular interval will be poor, affecting customers' experience with the application.

In order to circumvent the above problem, we proposed a proactive approach for dynamic provisioning of resources for SaaS applications based on predictions using the ARIMA model. The approach realizes the *Workload Analyzer* component of the architecture presented in our previous work [9] and feeds the rest of the components with accurate predictions that enable the rest of the system to scale the resources without waste of resources.

We introduced the prediction based on the ARIMA model and evaluated its accuracy of future workload prediction using real traces of requests to web servers from the Wikimedia Foundation. We also evaluated the impact of the achieved accuracy in terms of efficiency in resource utilization and QoS. Simulation results showed that our model is able to achieve an accuracy of up to 91%, which leads to efficiency in resource utilization with minimal impact in response time for users.

In future, we plan to integrate to the architecture a reactive module that can act as a second line of defense against poor QoS by compensating errors in the prediction with ad hoc decision on dynamic provisioning. We also plan to explore more robust techniques for workload prediction, able to predict peak in resource utilization that cannot be fit in the ARIMA model. With these techniques available, we plan to investigate methods for automatic selection of the best approach for workload modeling and load prediction given user-defined accuracy and computational requirement trade-offs. We will also apply the methods proposed in this paper in a prototype private Cloud system.

Acknowledgments

This work was partially supported through The University of Melbourne Early Career Research (ECR) grant and Australian Research Council (ARC) Future Fellowship.

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, Jun. 2009.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[3] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC'10)*. Chicago, USA: ACM, Jun. 2010.

[4] N. Bonvin, T. G. Papaioannou, and K. Aberer, "Autonomic SLA-driven provisioning for cloud applications," in *Proceedings of the 11th International Symposium on Cluster, Cloud and Grid Computing (CCGrid'11)*. Newport Beach, USA: IEEE Computer Society, May 2011, pp. 434–443.

[5] J. Yang, T. Yu, L. R. Jian, J. Qiu, and Y. Li, "An extreme automation framework for scaling cloud applications," *IBM Journal of Research and Development*, vol. 55, no. 6, pp. 8:1–8:12, Nov. 2011.

[6] E. Caron, F. Desprez, and A. Muresan, "Forecasting for grid and cloud computing on-demand resources based on pattern matching," in *Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom'10)*. Indianapolis, USA: IEEE Computer Society, Dec. 2010, pp. 456–463.

[7] M. Sladescu, A. Fekete, K. Lee, and A. Liu, "Event aware workload prediction: A study using auction events," in *Proceedings of the 13th International Conference on Web Information Systems Engineering (WISE'12)*, ser. Lecture Notes in Computer Science, X. S. Wang, I. F. Cruz, A. Delis, and G. Huang, Eds. Berlin, Germany: Springer, 2012, vol. 7651, pp. 368–381.

[8] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 155–162, Jan. 2012.

[9] R. N. Calheiros, R. Ranjan, and R. Buyya, "Virtual machine provisioning based on analytical performance and QoS in cloud computing environments," in *Proceedings of the 40th International Conference on Parallel Processing (ICPP'11)*. Taipei, Taiwan: IEEE Computer Society, Sept. 2011, pp. 295–304.

[10] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*, 4th ed. Hobokem, USA: Wiley, 2008.

[11] H. Zhang, G. Jiang, K. Kenji Yoshihira, H. Chen, and A. Saxena, "Intelligent workload factoring for a hybrid cloud computing model," in *Proceedings of the 2009 IEEE Congress on Services (SERVICES'09)*. Los Angeles, USA: IEEE Computer Society, Jul. 2009.

[12] Z. Gong, X. Gu, and J. Wilkes, "PRESS: PRedictive Elastic ReSource Scaling for cloud systems," in *Proceedings of the 6th International Conference on Network and Service Management (CNSM'10)*. Niagara Falls, Canada: IEEE, Oct. 2010, pp. 9–16.

[13] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *Proceedings of the 4th International Conference on Cloud Computing (CLOUD'11)*. Washington DC, USA: IEEE Computer Society, Jul. 2011, pp. 500–507.

[14] V. G. Tran, V. Debusschere, and S. Bacha, "Hourly server workload forecasting up to 168 hours ahead using seasonal ARIMA model," in *Proceedings of the 13th International Conference on Industrial Technology (ICIT'12)*. Athens, Greece: IEEE, Mar. 2012, pp. 1127–1131.

[15] V. Nae, A. Iosup, and R. Prodan, "Dynamic resource provisioning in massively multiplayer online games," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 3, pp. 380–395, Mar. 2011.

[16] S. Pacheco-Sanchez, G. Casale, B. Scotney, S. McClean, G. Parr, and S. Dawson, "Markovian workload characterization for QoS prediction in the cloud," in *Proceedings of the 4th International Conference on Cloud Computing (CLOUD'11)*. Washington DC, USA: IEEE Computer Society, Jul. 2011, pp. 147–154.

[17] G. Urdaneta, G. Pierre, and M. van Steen, "Wikipedia workload analysis for decentralized hosting," *Computer Networks*, vol. 53, no. 11, pp. 1830–1845, Jul. 2009.

[18] M. Arlitt and T. Jin, "A workload characterization study of the 1998 World Cup Web site," *IEEE Network*, vol. 14, no. 3, pp. 30–37, May 2000.

[19] M. Mao, J. Li, and M. Humphrey, "Cloud auto-scaling with deadline and budget constraints," in *Proceedings of the 11th International Conference on Grid Computing (GRID'10)*. Brussels, Belgium: IEEE Computer Society, Oct. 2010, pp. 41–48.

[20] R. J. Hyndman and Y. Khandakar, "Automatic time series forecasting: The forecast package for R," *Journal of Statistical Software*, vol. 27, no. 3, pp. 1–22, Jul. 2008.

[21] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. OTexts, 2012. [Online]. Available: https://www.otexts.org/fpp

[22] D. Kwiatkowski, P. C. Phillips, P. Schmidt, and Y. Shin, "Testing the null hypothesis of stationarity against the alternative of a unit root," *Journal of Econometrics*, vol. 54, no. 1-3, pp. 159–178, Oct. 1992.

[23] F. Sowell, "Maximum likelihood estimation of stationary univariate fractionally integrated time series models," *Journal of Econometrics*, vol. 53, no. 1–3, pp. 165–188, Jul. 1992.

[24] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

Rodrigo N. Calheiros Rodrigo N. Calheiros is a Post-Doctoral Research Fellow in the Cloud Computing and Distributed Systems Laboratory (CLOUDS Lab) in the Department of Computing Information Systems, University of Melbourne, Australia. His research interests include Cloud and Grid computing and simulation and emulation of distributed systems.

Enayat Masoumi Enayat Masoumi was a Research Assistant with the Cloud Computing and Distributed Systems Lab at the University of Melbourne. His research interests include Cloud computing, multi-agent systems, and distributed systems.

Rajiv Ranjan Dr. Rajiv Ranjan is a Senior Research Scientist (equivalent to Associate Professor in North American University System) and Julius Fellow in the CSIRO Computational Informatics, Canberra, where he is working on projects related to Cloud and big data computing. He has been conducting leading research in the area of Cloud and big data computing developing techniques for: (i) Quality of Service-based management and processing of multimedia and big data analytics applications across multiple Cloud data centers (e.g., CSIRO Cloud, Amazon and GoGrid); and (ii) automated decision support for migrating applications to data centers.

Rajkumar Buyya Dr. Rajkumar Buyya is Professor of Computer Science and Software Engineering and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in Cloud Computing. He has authored 400 publications and four text books. He is one of the highly cited authors in computer science and software engineering worldwide.