Dottorato di Ricerca in
Computer Engineering and Science
International Doctorate School in
Information and Communication Technologies

XXVI CICLO

Sede Amministrativa di Modena
Università degli studi di MODENA e REGGIO EMILIA

TESI PER IL CONSEGUIMENTO DEL TITOLO DI DOTTORE DI RICERCA

Performance models for evaluation and management of large Internet systems and applications

Candidato:
Dott.ssa Stefania Tosi

Relatore:
Prof. Michele Colajanni

I never guess. It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts.

Jules-Henri Poincare'

Abstract

The advent of the cloud computing paradigm has led Internet service providers to set up large Internet systems distributed all over the world in order to efficiently make their business. Efficient management of large Internet systems requires several strategies that decide on request dispatching, load balance, admission control, and request redirection without direct intervention of human administrators. At the basis of most autonomic management decisions there is the need of performance models for supporting system management by taking real-time decisions on the basis of information related to the state of internal system components and resources. Performance models supporting large infrastructures must be able to operate at different time scales and should support prompt reconfigurations motivated by continuous dynamic changes in system, client, and business policies. Performance models should be scalable for increasing numbers of hardware and software components, they should be adaptive to heterogeneous data streams characteristics, and they should guarantee reliable results over changing system conditions and requirements.

This thesis presents a set of novel performance models proposed for online management of large amounts of variable and heterogeneous data streams coming from several system monitors and networks. In particular, this thesis extends the state-of-the-art in performance modeling in manifold directions: (i) it presents a novel approach for improving scalability when managing large amounts of data; (ii) it introduces new adaptive models for the on-line detection of anomalies in highly variable contexts, and for the identification of correlations and groups of related objects even when correlation is hidden by high variability; (iii) it uses predictive analytics for improving performance in the selection of cloud availability zones on the basis of user preferences.

Extensive evaluations of the proposed approaches in real systems demonstrate improvements in performance modeling with respect to the state-of-the-art solutions and the satisfaction of scalability, adaptivity, and reliability requirements that are mandatory for large systems management.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The advent of the cloud computing paradigm has massively changed the whole computing industry. It has led todays companies and Internet service providers to set up large Internet systems distributed all over the world in order to efficiently make their business and serve their customers expectations. Large Internet systems are characterized by a huge number of hardware resources (e.g., processors, memories, storage elements, virtual machines) and software components (e.g., applications, business processes) having the goal to make computing services readily available to the company and to users on demand, like any other utility service available in today's society.

Efficient management of these large systems requires several strategies that decide on request dispatching, load balance, overload and admission control, request redirection, and so on [10, 12, 16–19]. The large number of resources and components involved requires management strategies to be self-adaptive and autonomic because no amount of human administrators would be capable of cloning and migrating virtual machines in time, as well as re-distributing or re-mapping the underlying hardware. At the basis of most autonomic management decisions, there is the need of performance models for supporting system management by taking real-time decisions on the basis of information gathered from monitors and related to the state of internal system components and resources.

Monitors installed in large scale infrastructures produce thousands and thousands of heterogeneous data streams at the level of hardware resources, applications, events, and services [25, 26]. When dealing with large amounts of variable and heterogeneous data streams, performance models for management of cloud-based infrastructures should guarantee:

- **scalability** for increasing numbers of hardware and software components. Performance models must be efficient and have limited computational costs, since they have to cope with a large amount of data that must be collected,

analyzed, stored and transmitted at real-time so as to take timely corrective actions to meet SLAs.

- **flexibility** for heterogeneous data streams characteristics. Heterogeneity may be related to different sources, such as different types of resource they refer to, different statistical properties, and different time scales of monitoring.

- **adaptivity** of the models' settings and parameters over changing system conditions. Outcomes of performance modeling must be robust to changes in the system conditions and requirements, as well as to system errors.

This thesis presents a set of novel performance models proposed for online management of large amounts of variable and heterogeneous data streams coming from several system monitors and networks. All proposed solutions aim to fulfil the requirements of scalability, flexibility, and adaptivity that are mandatory for large systems management.

First, this thesis presents a novel approach for improving scalability when monitoring large amounts of data [5, 6]. By providing a data dimensionality reduction and a classification of information on the basis of the statistical properties of monitored data, this solution diminishes the computational complexity of managing big datasets without the loss of relevant information for management. Experiments carried on real data sets demonstrate that the proposed strategy finds useful applications in system management related to modern large Internet systems and improve the scalability of traditional approaches that work on all data sets.

Second, this thesis proposes new adaptive models for the on-line detection of anomalies in highly variable contexts [5–7, 140]. In particular, we propose a new adaptive algorithm for state change detection that is specifically tailored to be integrated with real-time management in Internet-based systems. The proposed algorithm combines for the first time an online version of the Wavelet-based model to achieve a continuously adaptive representation of the data flowing from the system monitors with an online adaptive version of the well known Cusum statistical test as a detector. Moreover, we address issues related to the identification of relevant behavioral changes in large Internet systems by introducing a novel approach that characterizes the statistical properties of the resource measures coming from system monitors, classifies them, and signals a change only when there is modification of the resource classification. By providing a data dimensionality reduction and a characterization of server statistical heterogeneity, we are able to detect changes in server operations without the need of statistically analyzing the entire set of resource measures.

Third, this thesis proposes a new correlation model that is able to capture both linear and non-linear dependences even among time series characterized by high variability [3]. When high variability affects the data streams, it is necessary to isolate perturbations and uncover all the patterns that compose the data set. By extracting trends in data that allow us to reveal the way in which a data stream may depend on the other one, the solution we propose extends the correlation detection and its applications also to domains that are characterized by highly variable data streams.

Fourth, this thesis extends the state-of-the-art in data clustering by presenting a novel algorithm for finding groups of related objects even when correlation is hidden by high variability [1, 2]. It is based on the extraction of the main patterns of the time series, the removal of perturbations, the selection of just the trend patterns, and the use of such trend patterns for discovering correlation-connected data also in highly variable domains. This approach allows to disclose dependences between data even when they are hidden by perturbations and makes the proposed solution effective for both system and network management, by finding groups of servers with related functionalities and network flows sharing similar statistical properties with high reliability levels.

Finally, this thesis uses predictive analytics for improving performance in the selection of cloud availability zones on the basis of user preferences [4]. We introduce a predictive approach to identify the cloud availability zone that maximizes satisfaction of an incoming request against a set of user requirements. The predictive models are built from historical usage data for each availability zone and are updated as the nature of the zones and requests change. This method successfully predicts the unpublished zone behavior from historical data and identifies the availability zone that maximizes user satisfaction against specific requirements. Moreover, the proposed method can be used to derive useful insights for availability zones, and can help cloud providers plan ahead their operations. We compare our selection method to non-predictive selection methods, which rely on a priori knowledge of unpublished zone attributes. The results show that the predictive approach performs better than the non-predictive ones in terms of user satisfaction, while maintaining high cloud performance.

The proposed approaches extend the state-of-the-art in performance modeling in manifold directions: (i) they improve scalability by not requiring to model and analyze all data streams gathered from system monitors; (ii) they make no a-priori assumptions about the statistical characteristics of data; (iii) they are flexible enough to be applied to different data sets despite of time scales, ranges, or sources of data with the guarantee of high reliability of the results; (iv) they are direct to minimize the computational complexity of modeling and are therefore specifically tailored to work online; (v) they have special consideration for the characteristics of variability, dinamicity, heterogeneity, and high dimensionality typical of large

Internet systems monitoring.

The novel approaches proposed for performance modeling, their main quantitative and qualitative interpretations, and their improvements with respect to the state of the art are detailed in the following chapters.

Chapter 2 motivates the need of novel performance models for large system management.

Chapter 3 proposes novel techniques that guarantee high scalability by reducing the large amount of collected metrics to analyze to a subsets of aggregated information that are useful for management.

Chapter 4 presents new adaptive models for the on-line detection of relevant state changes and behavioral anomalies in highly variable contexts, in order to allow management systems to take real-time decisions for rapidly adapting to changing environments.

Chapter 5 proposes an innovative approach that is especially tailored to detect linear and non-linear correlations between highly variable data streams.

Chapter 6 presents novel algorithms for data clustering that guarantee high accuracy and robustness in finding groups of servers with related functionalities and network flows sharing similar statistical properties.

Chapter 7 presents an adaptive method that helps users to automatically select a cloud provider based on user-specific criteria and learns cloud provider properties by basing on information about previous provided services and by using this information to suggest the user an optimal decision making for cloud provider selection.

Chapter 8 concludes the thesis with some final remarks and future work.

Chapter 2

Motivation

Monitors installed in large scale systems produce data streams at the level of hardware resources, applications, events, and services [25, 26]. Any management decision needs performance modeling supports that are able to analyze these data streams in order to extract from raw performance measures the information that is useful for a given business or system objective.

Modeling data coming from system monitors is not an easy task, because of the three main properties that characterize data streams in large scale systems: volume, variety, and variability. These characteristics, together with velocity, define what is typically referred to as ”Big Data” [20], whose challenges are faced more and more by today’s organizations. These data characteristics lead to the need for a new set of capabilities for performance models to succeed in extracting insight from an immense amount of vary and heterogeneous data.

In the following, let us detail the characteristics of modern system data:

- **Volume**

  The massive volume of data monitored in modern systems is exploding. Twitter and Facebook generate about 10 terabytes of data every day, and some enterprises such as Google [21], Microsoft [22], and Yahoo [23] process terabytes of data every hour of every day of the year with large data centers of thousands of nodes [20]. According to [24], the digital universe will grow by a factor of 300 from 2005 to 2020, reaching 40 zettabytes and more than 5.200 gigabytes per person in 2020.

  As the number of resources that are monitored and the period of monitoring is growing [28, 139], the conversation about data volumes has changed from terabytes to petabytes with an inevitable shift to zettabytes [20]. This volumes of data cannot be analyzed in a timely fashion by traditional performance models, which assume that the number of data streams and the size of each of them allow the analysis of the entire set of data (e.g., [115–118]).

This assumption is unrealistic in modern Internet systems, where the computational cost associated to the analysis of each data set is impracticable. This scenario creates the need of *scalable* solutions to dominate the complexity of massive data analysis, through the identification of information inside raw data that is useful to gain a better understanding of system performance, business activity, and customers behavior in an online fashion. To achieve acceptable performance for large data analysis, we must abandon the idea of processing all raw data coming from system monitors and we must consider to focus on only a subset of data information that are useful for management.

In literature there are many reduction models oriented to limit the data that must be analyzed, such as the Principal Component Analysis (PCA) [110], the Correspondence Analysis [111], the Factor Analysis [112], the Non-Negative Matrix-Factorization [113], the Independent Component Analysis [114]. This thesis proposes an approach based on the PCA because it makes only weak assumptions on data characteristics and allows us to select the most important information on the basis of the so called ”energy” in data. Thanks to these properties, PCA has been extensively used in many computer related contexts, especially for workload characterization [81] and for network traffic analysis [121]. In the context of system management, the work in [14] integrates PCA into a two-stage log processing method that is oriented to discover the statistically dominant patterns in the data set and thereby to identify possible anomalies. Another paper [108] uses PCA to identify interactions among the components of large production systems. Even if the goals of these works are different, they support the possibility of using PCA in managing large data sets and finding relevant information.

- **Variety**

  With the explosion of the number of monitored resources and sensors, as well as of smart devices and social technologies, data monitored in a system has become complex, because it includes not only physical resource performance data, but also virtual ones, web log files, network packets, sensor data from active and passive systems, and so on. Such variety leads to data with different ranges and time scales, sampled at different time intervals, and referring to different resources or components. System management success must rely on the ability to draw insights from all kinds of data available despite of their variety.

  By basing on the static and a-priori choice of their setting and parameters, traditional performance models can hardly handle variety. As data variety grows, ad-hoc performance models suited to specific context as well

as models relying on many and hard-to-set parameters must leave room to *flexible* performance models that can adapt to different settings for monitoring and/or to different data characteristics. Since it is not feasible for human administrators to adjust models parameters to any different condition, it becomes mandatory for performance models to be autonomic, so to automatically adapt themselves to data characteristics with no human intervention.

Performance models proposed in this thesis solve most variety-related problems of existing algorithms. All proposed methods do not require any assumption about statistical properties of analyzed data, their performance does not depend on the type of data distribution, and they not rely upon a fixed number and setting of models parameters. These features make our methods flexible enough to model the performance of a system under changing and unpredictable conditions.

- **Variability**

  Resource measures obtained from load monitors in large Internet systems are extremely variable even at different time scales and tend to become obsolete rather quickly [42]. Monitored data sets are characterized by non-stationary and non-deterministic features, variable variance of the distribution, and likely affected by internal and external perturbations.

  Data sets reported in Figure 2.1 share the high variable and stochastic traits that are typical of the performance metrics coming from monitors in large Internet systems.

  These characteristics of resource measures are hard to model through the parametric techniques typically used by state of the art performance models and strongly affect their performance. Moreover, such characteristics are not static but change in time thus requiring performance models to *adapt* their modeling functions to varying conditions in order to accommodate variable contexts and changing requirements.

  Existing models and frameworks supporting management systems base on the assumption that resource measures are characterized by statistically homogeneous behaviors and/or stable conditions (e.g., [13, 81, 118, 130, 141]). This assumption is invalid in large Internet systems because the data sets coming from different servers and system resources are quite heterogeneous in statistical terms and extremely variable. For this reason, this thesis deals with the statistical heterogeneity of resource measures that must be considered as an intrinsic feature of modern systems [28, 81] leveraging virtualization technologies and providing multiple Internet-based services subject

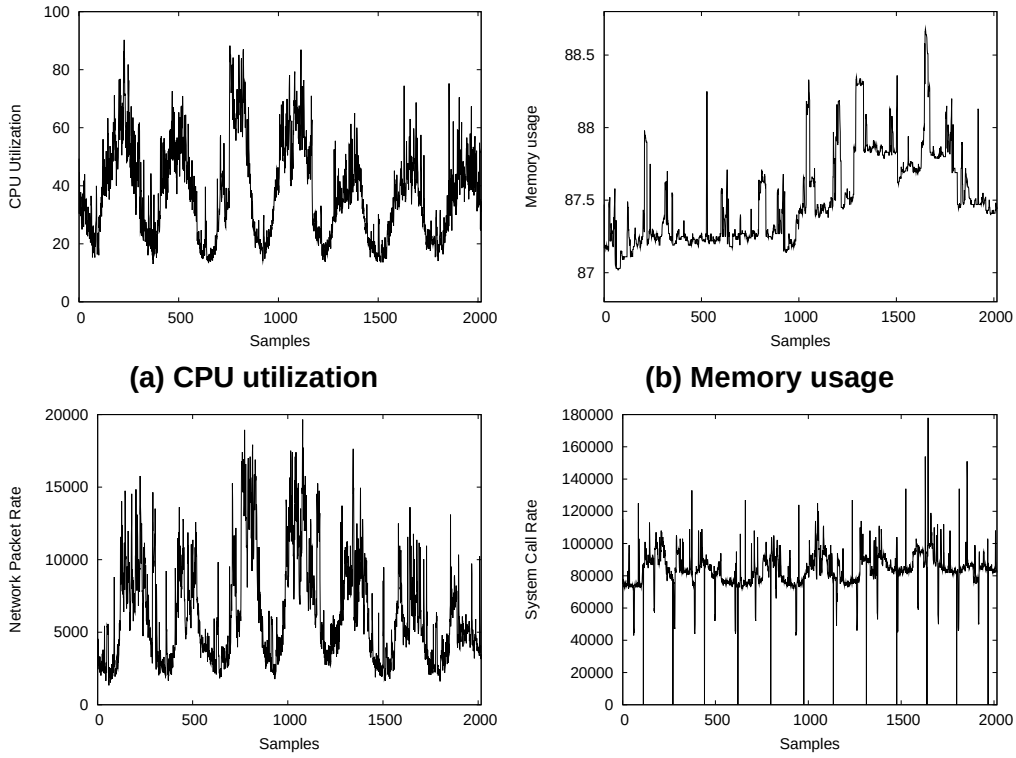


Figure 2.1: Examples of heterogeneous resource measures.

to varying demands. The methods proposed in this thesis do not filter heterogeneity, but they base on data representations that are able to remove all undesired perturbations while preserving the most relevant data information. This feature makes the proposed methods able to extract significant information for management form highly variable datasets without requiring any pre-analysis of data characteristics.

In the following sections, this thesis will introduce novel performance models to address the challenges of volume, variety, variability for different management purposes: for big data analytics, for anomaly detection, for correlation analysis, for data clustering, and for predictive analytics purposes.

Chapter 3

Big Data analytics

Even in a medium-sized Internet system, several thousands of resource data sets may reach the management system that should analyze, model and treat them at different temporal scales in order to guarantee scalability, reliability, and to avoid performance degradation and overloads. It would be possible to pass all resource data collected by the system monitors to the management system only in the case of short data sets consisting of few monitored samples. As this becomes impracticable when the data sets are collected from hundreds of components over long periods, it is necessary to identify all (and only) the relevant data carrying on useful information and to pass only this subset of relevant information to decision systems.

This chapter gives a contribution in the direction of simplifying the complexity of managing huge amounts of heterogeneous and highly variable data sets. The idea is to extract a concise representation of how the most important components and sets of components are acting through the investigation of a subset of relevant information that are obtained by *selecting* just the most important data sets. In this study, the importance is a measure of the impact that each resource has on the overall system behavior. We use the results of data set reduction as a mean to pass just relevant statistical information to a management system that is facilitated in the application of the most suitable decision strategy.

3.1 Proposed strategy

The proposed strategy is oriented to the selection and the statistical characterization of the resources of large Internet systems's servers. The measures related to system resources are typically huge in size because they refer to long monitoring periods, variable and heterogeneous. Hence, the first goal for management purposes is to discriminate between relevant and less relevant information.

We consider variability as the main statistical property that quantifies the degree of activity of a process and its importance in overall system behavior as in [109]. The index of variation of the data sets related to each monitored process is denoted by the *energy*. We use this measure as a means to distinguish between data sets carrying relevant and not-relevant information. In order to reduce the dimensionality of a data set analysis and to consider just the most relevant data sets in terms of energy, we refer to the *Principal Component Analysis* (PCA) model [110] that, unlike other models for dimension reduction [111–114], is able to express the intrinsic structure of a data set in terms of variance and requires no prior knowledge about statistical properties of data sets.

By considering only the relevant sources of information, we characterize the server statistical behavior emerging from the monitored data. Only relevant information is passed to the statistical analyzer that aims to classify data sets in one of the following classes:

- *deterministic*, presenting systematic trends and periodic patterns that are predictable and possible to model;
- *random*, manifesting isolated spikes and/or stochastic noises;
- *negligible*, giving a minor contribution to the overall system activity.

This classification allows a management system to apply the right model to the right data sets, as well as reducing the complexity of system management with null or minimum loss of information. The negligible category does not carry meaningful information for system management. Its contribution on system activity has a low impact and therefore a management framework can ignore data sets belonging to this class. When a data set is characterized by a prevalently deterministic behavior, a management system can adopt several models for prediction, anomaly detection, event identification. On the other hand, servers characterized by data sets with spiky and noisy behaviors complicate management. These data sets require some preliminary aggregation and filtering treatment before their utilization. In the most difficult cases they result useless or unfeasible for management even after the application of some data treatments.

Unlike existing solutions for server characterization that analyze each collected data set [115–118], the proposed solution classifies server behaviors by working on a subset of information. Our approach, that is based on PCA [110], aims (1) to extract from data sets the most relevant information, where the importance is measured as a function of the impact of each resource on the overall system behavior, and (2) to focus the classification analysis on this smaller subset of information. In this way, we are able to limit the computational complexity of the analysis, and to allow the integration of server characterization results with system management frameworks operating in large Internet systems.

The proposed strategy achieves the above mentioned goals through the steps outlined below.

1. **Initial analysis.** First, we evaluate the impact of each data set collected by system monitors on the overall system activity, so to distinguish *relevant* from *not-relevant* sources of information.

2. **Server characterization.** Considering only the relevant sources of information, we then characterize the server statistical behavior emerging from the monitored data. We choose to classify the statistical behavior in three main classes. The idea is to focus the analysis only to those classes that can facilitate system management.

3. **Evaluation of system dynamics.** The evaluation of the system dynamics looks at the overall behavior of the system. A significant variation in the behavior of relevant sources of information denotes a significant change in the system that is useful to signal for better management.

This chapter only focuses on the first two steps of the strategy, while the last one will be treated in Section 4.2, when considering the anomaly detection problem.

It is important to observe that a similar strategy can be applied to evaluate the system dynamics of resources belonging to groups of servers up to the whole system. By looking at the overall energy of the system, we can dynamically discriminate whether the system has a prevalent deterministic behavior and therefore it is manageable, or if the system is mainly driven by a random behavior that makes the system hard to treat and to model. A significant variation in the amounts of deterministic and random energies of the system denotes a significant change in system activity that is useful to signal for adapting management decision to changing environments.

A final remark is in order. Since the proposed strategy does not make any a-priori assumption on the statistical properties of the data sets, as it is required by other approaches [81, 119], it is flexible to characterize any type of resource and metrics, such as CPU utilization, memory and swap usage, disk usage, network packet rate. In the present version, it operates on homogeneous resource metrics belonging to distributed servers belonging to the same data center. Extensions to heterogeneous resources are possible, but out of the scope of this chapter.

3.2 Initial analysis

In large Internet systems, the data sets collected from the monitored resources form a multivariate structure characterized by multiple dimensions, each one con-

sisting of several samples. The proposed approach reduces the computational complexity of the analysis when the number of samples is huge with respect to the number of monitored resources (that is, dimensions). In these large structures, a reliable management decision requires a preliminary identification of the most relevant information. A common approach is to find a new coordinate space consisting of a lower dimensionality that is representative of the original space [120]. When a structure can be approximated through a smaller number of dimensions in a way that minimizes the error and the loss of information, we can refer to the smaller number of dimensions as the *structure intrinsic dimensionality*.

In literature, there are many reduction models to extract the intrinsic dimensionality of system resource measures, such as the Principal Component Analysis (PCA) [110], the Correspondence Analysis [111], the Factor Analysis [112], the Non-Negative Matrix-Factorization [113], the Independent Component Analysis [114]. Among these algorithms, we choose the PCA model because it is able to express the intrinsic structure of a data set in terms of variance without requiring any prior knowledge about the statistical characteristics of data sets.

At time t, the sets of sampled measures originate a matrix X_t corresponding to the highly dimensional multivariate structure. It is a n x p matrix, where n is the number of considered samples (for example, $n = 2016$ if we consider a five minutes sampling in a one week period), and p is the number of considered system resource measures. Our approach is more efficient than existing solutions when we consider long monitoring periods where $n > p$. As resource measures are statistically heterogeneous, it is useful to normalize them as data sets characterized by zero mean and unit variance, as suggested in [121]. The PCA is a coordinate transformation method that maps X_t on a new set of axes [120] called components. Calculating the components is equivalent to solve the symmetric eigenvalue problem for the matrix $\Psi_t = X_t^T X_t$ that is, a measure of the covariance of the data sets deriving from samples. In practice, each component v_i is the i-th eigenvector computed from the spectral decomposition of Ψ_t [121]:

$$\Psi_t v_i \quad = \quad \lambda_i v_i \qquad i = 1, \ldots, p \tag{3.1}$$

where λ_i is the eigenvalue corresponding to the eigenvector v_i and represents the magnitude of the variation along each component v_i. Previous work uses the term *energy* obtained as a sum of quadratic terms to quantify how well the eigenvectors describe the original data set [70, 108]. On the other hand, we compute the energy by considering the output of the PCA and evaluate the energy σ_i associated to the component v_i through the percentage of variation related to its eigenvalue λ_i, that is, $\sigma_i = \frac{\lambda_i}{\sum_{j=1}^{p} \lambda_j} * 100$. As Ψ_t is symmetric positive definite, its eigenvectors are orthogonal and the corresponding eigenvalues are non-negative real numbers. By convention, the eigenvectors are unit norm and the eigenvalues are arranged from large to small, so that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_p$.

In this new dimensional space, a *dimension* u_i ($i = 1, \ldots, p$) is defined as a vector of size n obtained by the contribution of the data set matrix X_t and of the component v_i. As suggested in [121], this vector is normalized to unit length by dividing it by $\sqrt{\lambda_i}$. Hence, for each principal component v_i we have:

$$u_i = \frac{X_t v_i}{\sqrt{\lambda_i}} \quad i = 1, \ldots, p \tag{3.2}$$

Through the above equation all server behaviors weighted by v_i produce one dimension of the transformed data. Hence, the vector u_i can capture the temporal variation common to all server measures along the component v_i. Since the components are ordered with respect to their contribution to the overall energy, u_1 captures the strongest temporal trend that is common to all server measures, u_2 captures the second strongest trend, and so on.

The energy characterizing each component is used to reduce the data dimensionality: we exclude the least *energetic* components of the structure, that is, we ignore the less variable components. There are several methods [122] to choose how many components it is useful to retain and to exclude. They include graphical approaches, such as the *scree plot* [123], the quantitative tests based on PCA singular values [124], the percentage of variability expressed by the principal components [125]. In this chapter, we pursue this last criterion by determining in advance the amount of residual variability that we want to tolerate and by excluding residual components. A typical threshold [52] establishes that the cumulated percent on variation expressed by the retained dimensions should be higher than the 90-percentile. Our method does not depend on the choice of this threshold, hence in this chapter we consider not-relevant the dimensions contributing to the overall variance of the data set for less than 10%.

When a smaller set of r dimensions are relevant, we can have interesting implications: X_t can be mapped on an r-dimensional subspace of $\mathbb{R}^p$, and $r \ll p$ is the intrinsic dimension of X_t. This result allows the method to distinguish between the *relevant* dimensions $U_t = \{u_1, \ldots, u_r\}$ that are contributions shared by all resource measures on the r principal components, and the *not-relevant* dimensions $\overline{U_t} = \{u_{r+1}, \ldots, u_p\}$ corresponding to the least important components in terms of system energy.

We validate the proposed approach by taking into account long data traces collected from an Internet system consisting of distributed servers that support different types of applications, including Web sites, databases, access controls, CMS, mail server, management software. In this large system, monitors collect resource measures every five minutes referring to CPU utilization, primary and secondary memory-related metrics, network activities. For the sake of simplifying a complex procedure, we limit our presentation to a week of data sets referring to the CPU utilization of 50 servers hosting Web sites providing static, dynamic and

interactive content. These data sets originate a matrix X_t with $n = 2016$ rows and $p = 50$ columns.

We initially evaluate the energy σ_i related to each dimension u_i in order to identify how many dimensions bring energy information to the data set. Figure 3.1 shows the cumulative distribution of the energy provided by each dimension [126] by reporting for each dimension u_i the proportion of the energy related to the previous dimensions that is, $\sum_{j \leq i} \sigma_j$.

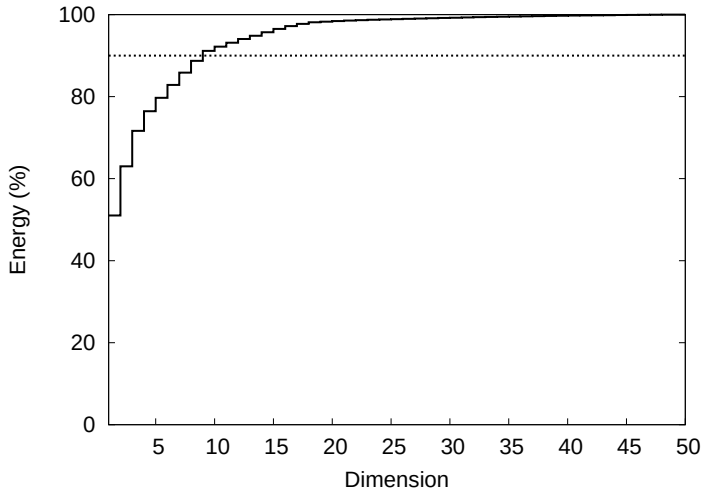


Figure 3.1: Cumulative distribution of energy.

Thanks to the rapid growth of the distribution curve in Figure 3.1, the 90-percentile of the energy of the overall system is captured just by the first nine dimensions that contribute to most server variability. In this example, our approach evaluates the first 9 dimensions as *relevant*, while the other 41 dimensions are marked as *not-relevant* from the point of view of the energy. This is an important result of the proposed methodology applied to a real Internet system because it confirms that, in terms of energy, the resource measures all together form a structure with an intrinsic dimensionality of $r = 9$, that is much lower than the original number (50) of the considered set of data.

As a confirmation of this result, we evaluate the degree of correlation between the original resource traces sampled from the servers and the reconstructed data sets as a function of the number k of dimensions. We use the following equation for reconstruction:

$$X'_t \approx \sum_{i=1}^{k} \sqrt{\lambda_i} u_i v_i^T, \quad k = 1, \ldots, p \tag{3.3}$$

Figure 3.2 reports an example of the high correlation existing between original and reconstructed CPU utilizations of three servers as a function of the considered dimensions (k spans from 1 to 20). This figure evidences that the first 9 dimensions achieve a perfect positive linear correlation between original and reconstructed data sets for servers 7 and 11, and a very high correlation for server 3. These results confirm that a PCA-based methodology can be applied to identify the most relevant information for management decisions in large Internet systems.

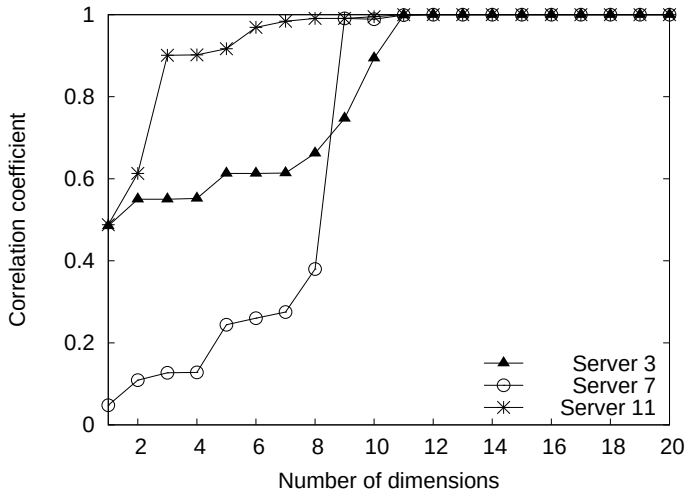


Figure 3.2: Correlation coefficient between original and reconstructed data sets as a function of the number of considered dimensions.

3.3 Server characterization

The first application of the proposed methodology is to provide a statistical characterization of the server behaviors as *relevant* and *deterministic*, *relevant* and *random*, or *negligible*. For each data set corresponding to a server, we propose an algorithm activated periodically and based on the following steps. We consider the activation at a generic time t. The contributions in terms of energy allow the algorithm to distinguish between relevant dimensions U_t and not-relevant $\overline{U_t}$ dimensions.

Then, we distinguish the relevant dimensions between *correlated* U_t^C and *low-correlated* U_t^L by evaluating the autocorrelation function (ACF) of each relevant dimension U_t as in [127] and in many other contexts (e.g., [11, 119]). A quick decrease of the ACF means that the observed dimension values exhibit low (or null) autocorrelation. This is the case of dimensions capturing the random perturbations and/or spikes varying in time and intensity in the system behavior. On

the other hand, a slow decay of the autocorrelation function indicates that the dimension shows a dependency among its values. The dimension typically exhibits trends, periodicity and seasonal fluctuations due to the diurnal activity, as well as the difference between weekday and weekend activity of the system [127].

We use the classification of the dimensions to distinguish three behavioral values for each data set referring to a server resource. The idea is to add all the contributions for each type of dimensions and then to evaluate the maximum of them. This method is applied to each server resource k as following.

- $D_t^{Ser}[k] = \sum_j v_j[k], \forall j \mid u_j \in U_t^C$ denotes the contributions provided by the relevant correlated dimensions;
- $R_t^{Ser}[k] = \sum_j v_j[k], \forall j \mid u_j \in U_t^L$ denotes the contributions provided by the relevant low-correlated dimensions;
- $N_t^{Ser}[k] = \sum_j v_j[k], \forall j \mid u_j \in \overline{U_t}$ accumulates the contributions of the not-relevant dimensions.

For each server resource k, we evaluate the highest term that is, $M_t^{Ser}[k] = max\{D_t^{Ser}[k], R_t^{Ser}[k], N_t^{Ser}[k]\}$. In such a way, we can characterize each server resource on the basis of its main behavioral value:

$$Server\ k = \begin{cases} \textit{Deterministic} & \text{if } M_t^{Ser}[k] = D_t^{Ser}[k] \\ \textit{Random} & \text{if } M_t^{Ser}[k] = R_t^{Ser}[k] \\ \textit{Negligible} & \text{if } M_t^{Ser}[k] = N_t^{Ser}[k] \end{cases} \tag{3.4}$$

It is important to compare the proposed strategy against those obtained through other approaches that analyze each monitored data set, such as [115–118]. Existing models require computationally expensive analyses to evaluate correlation [127], probability distributions [128], and mean loads [129]. The complexity becomes excessive especially when each data set consists of several entries, and it is even increased by the necessity of applying some pre-filtering technique when the measures are highly variable, as it is typical in the considered contexts.

For validation purposes, we consider again the data set referring to the CPU utilization of the 50 servers introduced in Section 3.2. We compare the classification results of servers achieved by the proposed strategy and other approaches that analyze each data set. The results of these models are reported in the second column of Table 4.5. The results obtained through the proposed approach are shown in the three *Proposed strategy* columns. If we compare the classes with bold indexes to the server characterization arising from the baseline characterization reported in the second column, we see a perfect correspondence of results.

The proposed strategy is able to identify the main statistical behavior of 50 servers through a classification and a statistical analysis of only the most relevant 9 dimensions of the data set. On this small set of dimensions, we do not need to apply any time consuming filtering technique since PCA itself allows us to isolate perturbations from data. Thanks to the dimensionality reduction, we can integrate the server characterization with any decision support for system management in large distributed systems.

Let us detail the steps of this algorithm applied to $r = 9$ relevant dimensions and 41 not-relevant dimensions. We apply the ACF to the $r = 9$ relevant dimensions. For server characterization purposes, we need to be selective in the choice of the correlated dimensions and we set a high cut-off value of 0.8. The ACF gives that 5 dimensions are *correlated* and 4 are *low-correlated* (Figure 3.3). The goal is now to determine which dimensions actually bring information to which CPU traces of the data set. By evaluating the D_t^{Ser}, R_t^{Ser} and N_t^{Ser} values, we compute the extent to which correlated U_t^C, low-correlated U_t^L and not-relevant $\overline{U_t}$ dimensions are present in the data set.

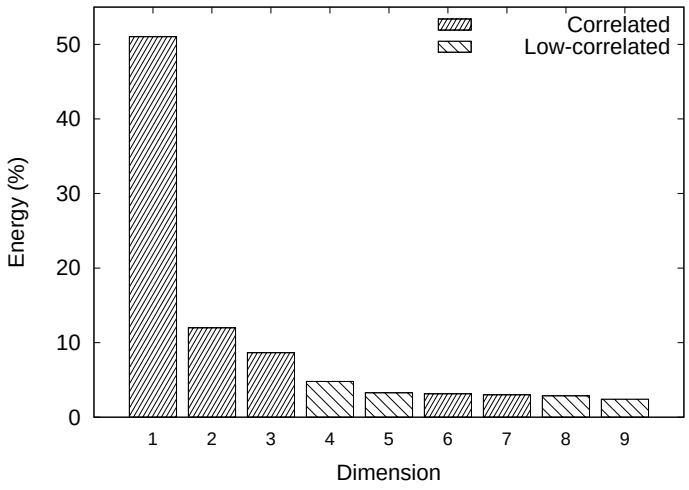


Figure 3.3: Percentage of energy and classification of the relevant dimensions.

The proposed strategy classifies the characteristics of the server k through the three behavioral values $D_t^{Ser}[k]$, $R_t^{Ser}[k]$ and $N_t^{Ser}[k]$ without the need of applying multiple statistical tests to all data sets. In the results reported in the *Proposed strategy* columns of Table 4.5, each column highlights in bold text the $M_t^{Ser}[k]$ value for each server. This value denotes the behavioral characterization that is assigned by the proposed strategy. For example, the baseline analysis of the server 3 computes a mean load utilization of filtered data of 6.39% that classifies it as

a negligible server. Our strategy reaches the same characterization through the evaluation of only three behavioral values: a predominant $N_t^{Ser}[3]$ value of 0.1866 characterizes the server as under-loaded, thus avoiding the analysis and treatment of the entire data set.

	Analysis of each data set	**Proposed strategy**				Analysis of each data set	**Proposed strategy**		
k	Behavior	D_t^{Ser}	R_t^{Ser}	N_t^{Ser}	k	Behavior	D_t^{Ser}	R_t^{Ser}	N_t^{Ser}
1	Random	0.2211	**0.2379**	0.0181	26	Negligible	0.0569	0.0402	**0.1912**
2	Negligible	0.0006	0.0009	**0.1297**	27	Negligible	0.0071	0.0186	**0.2028**
3	Negligible	0.0443	0.0334	**0.1866**	28	Negligible	0.0220	0.0268	**0.2112**
4	Random	0.0391	**0.3012**	0.1185	29	Random	0.0432	**0.1261**	0.1127
5	Negligible	0.0186	0.0116	**0.1955**	30	Random	0.0207	**0.2498**	0.0137
6	Negligible	0.0162	0.0128	**0.1938**	31	Deterministic	**0.3955**	0.0846	0.0927
7	Random	0.0615	**0.2379**	0.0181	32	Deterministic	**0.4183**	0.2324	0.0277
8	Negligible	0.0492	0.0370	**0.2030**	33	Deterministic	**0.3037**	0.2695	0.0293
9	Random	0.1160	**0.2206**	0.1236	34	Random	0.2114	**0.2388**	0.0166
10	Negligible	0.0005	0.0005	**0.1309**	35	Negligible	0.0478	0.0410	**0.1858**
11	Deterministic	**0.3539**	0.1612	0.0540	36	Negligible	0.0387	0.0353	**0.2114**
12	Negligible	0.0099	0.0049	**0.1960**	37	Random	0.0275	**0.3535**	0.0616
13	Negligible	0.0139	0.0122	**0.2052**	38	Negligible	0.0236	0.0842	**0.1214**
14	Deterministic	**0.3158**	0.2091	0.0150	39	Random	0.0611	**0.2059**	0.0287
15	Deterministic	**0.2302**	0.0792	0.0860	40	Deterministic	**0.3810**	0.2160	0.0421
16	Deterministic	**0.3057**	0.2396	0.0176	41	Deterministic	**0.2462**	0.0784	0.0815
17	Deterministic	**0.3368**	0.1016	0.0402	42	Deterministic	**0.3902**	0.0892	0.0960
18	Random	0.0357	**0.3350**	0.0712	43	Negligible	0.0003	0.0008	**0.1094**
19	Random	0.0685	**0.2085**	0.0071	44	Negligible	0.0039	0.0100	**0.1755**
20	Deterministic	**0.2755**	0.1612	0.0251	45	Deterministic	**0.3879**	0.1695	0.0458
21	Random	0.1170	**0.2858**	0.0239	46	Deterministic	**0.3002**	0.0716	0.0866
22	Random	0.0837	**0.3412**	0.0213	47	Negligible	0.0090	0.0198	**0.1659**
23	Deterministic	**0.3955**	0.0846	0.0927	48	Negligible	0.0221	0.0237	**0.2081**
24	Deterministic	**0.3163**	0.3154	0.0374	49	Negligible	0.0006	0.0003	**0.1388**
25	Negligible	0.0002	0.0006	**0.1037**	50	Deterministic	**0.2032**	0.0898	0.0844

Table 3.1: Characterization of servers behavior.

In this chapter we do not report all possible exploitations of the proposed strategy. Just to give another example, we observe that considering the amount of servers characterized by negligible activities offers a valuable information for management optimizations. In our case study, we have that 20 over 50 servers are prevalently negligible, that is, mainly underloaded during two weeks of observation. In a similar context, a resource management system that logically pools all server resources would increase the overall system utilization and diminish operational costs by turning off some unnecessary servers.

Chapter 4

Anomaly detection

At the basis of most real-time strategies for large Internet systems management there is the need to continuously evaluate system state behavior and to detect when an anomaly is occurring. Modern Internet systems open new and interesting scenarios in the field of the research on the online detection models.

In this chapter, we propose flexible and adaptive detection models that we demonstrate they are suitable to analyze continuous streams of data coming from Internet-based systems characterized by high variability and non stationarity of the monitored resource measures that result in not-acceptable false alarm rates. Our models solve the limits of the traditional solutions while retaining their computational efficiency. The solutions we present address two distinct detection problems: state change detection (Section 4.1) and behavioral change detection (Section 4.2).

4.1 State change detection

Most real-time management decisions related to modern Internet-based systems are activated after a notification that a relevant state change has occurred in some system resource(s). Anomaly detection, quality and access control, request redirection, process and virtual machine migration, hardware re-mapping, diagnosis and fault detection, to name a few, are examples of processes that are activated after the detection of a significant and non-transient system state change. For this reason, the most important system resources should be continuously monitored and data be passed to some statistical models that decide almost immediately whether a relevant state change has occurred or not.

Internet-based systems pose novel challenges in the field of state change detection. We are moving from the more traditional area of offline contexts, where the models are applied to well defined sets of data with (almost) no temporal

constraints, to online contexts, where models receive continuous streams of data and have to answer almost immediately. We have to consider novel types of data streams. Indeed, because of internal system operations, such as context switching, virtualization, and I/O operations, and of the unpredictable variability of the user request rates, monitored resource measures are *non-stationary* and typically both the mean and the variance vary over time. Moreover, these measures exhibit what we can call *high variability*, that corresponds to a standard deviation comparable to (and even larger than) the mean and that can be quantified by a coefficient of variation larger than one. Finally, non-stationary behaviors occur at a faster time scale with significant variations even over short time intervals.

In these contexts, the most common models used to support the detection of relevant state changes, such as Particle Filtering [132], Kalman filter [133] and Sequential Monte Carlo Method [134], are not effective because they require the knowledge of some statistical characteristics of the time series [135, 136]. Other popular state change detectors, such as the threshold-based detectors, the Shewhart chart, the Exponential Weighted Moving Average (EWMA) chart and the traditional Cumulative Sum (Cusum) chart, are often inadequate because when data streams are characterized by non-stationary behaviors and high variability, they cause either a large number of false detections or absence of detections depending on the chosen parameters of the model [45, 137].

In this section, we propose a new adaptive model that combines two key ingredients:

- a *detection rule* based on a new adaptive implementation of the Cusum model [138] that is able to keep the detector performance (in terms of both false detection rate and absence of detection rate) close to optimal;
- a *data representation* that uses an online Wavelet-based filter that is able to rectify the monitored data by eliminating random non-Gaussian errors while retaining the main features of the original data stream.

We demonstrate that the proposed adaptive state change detection model is able to provide the best results for several statistical characteristics of data coming from real and emulated contexts, and that it is able to satisfy the temporal constraints that are typical of real-time resource management systems.

4.1.1 Problem definition

Online detection problem

We are interested in an online version of the state change detection problem where the time series is represented by a continuous stream of measurements of the resource usage of Internet-based servers.

In time series characterized by non-stationary conditions, a *state* is defined as a subset of the continuous data stream where data are characterized by the same statistical properties that we generically denote by ϑ. It can refer to one attribute (e.g., mean, variance, distribution) or a combination of them. We define *relevant state change* as a shift larger than $\Delta > 0$ in the ϑ metric that the model should be able to timely detect. The actual value of Δ is actually application dependent.

We consider a continuous stream of temporally ordered samples $\{y\} = \{y_1, \ldots, y_i, \ldots\}$, and we focus on operations and parameters that the model has to apply at a generic sample y_i. Before this sample, we can assume that the model has identified one or more state(s), each characterized by an online computation of the statistical characteristics $\widehat{\vartheta}$ of interest for the detection model. Let us denote by y_s and by $\widehat{\vartheta}_s$ the first sample of the present state of the data subset including y_i and its estimated statistical characteristics, respectively. (We implicitly assume that there was no relevant state change between y_s and y_i.) At the sample y_i, the online detection model has to evaluate whether a relevant state change occurs or not. To do this, a statistical representation $f_y(y_s, \ldots, y_i)$ of the samples between y_s and y_i is dynamically evaluated to provide an estimation of the statistical properties of the data stream and a *detection rule* must verify whether the deviation among the estimated statistical properties of the current state $\widehat{\vartheta}_s$ and the statistical representation $f_y(y_s, \ldots, y_i)$ of the samples between y_s and y_i overcomes a certain threshold. This is typically done on the basis of the likelihood ratio [151], that estimates the deviations among the statistical representation of the data samples and the current state.

We distinguish two classes of detection models, where the differences reside mainly in the way to consider these deviations, the way to choose the statistical threshold, and the way to represent the samples between y_s and y_i.

In particular, the first class uses as a threshold a function f^1_Δ of the minimum value Δ of the shift that the model should capture and compares $f^1_\Delta(\Delta)$ with a punctual evaluation of the deviation between the estimation of the current state $\widehat{\vartheta}_s$ and the function $f_y(y_s, \ldots, y_i)$. The models belonging to this class (e.g., the Threshold-based models [149] and the Exponential Weighted Moving Average (EWMA) [45]) can detect the presence of a relevant state change on the basis of the following equation:

$$\text{detection rule} = \begin{cases} \text{state change}, & \text{if} \quad |\widehat{\vartheta}_s - f_y(y_s, \ldots, y_i)| \geq f^1_\Delta(\Delta) \\ \text{no state change}, & \text{otherwise.} \end{cases} \tag{4.1}$$

The second class of state change detection algorithms are based on a *gain function* g (e.g., the Cumulative Sum [138]). The gain function g can be considered as an accumulator of the detection model: during a relatively stable state it should be close to zero, and it should depart from zero when a relevant change occurs in the

time series. It sums up the consecutive partial increments (decrements) among the online estimation of the current state $\widehat{\vartheta}_s$ and the sample y_i every time that the deviation between the current state estimation $\widehat{\vartheta}_s$ and the online data representation $f_y(y_s, \ldots, y_i)$ reaches a reference value of the minimum shift Δ computed by the function f^2_Δ. In such a case:

$$g_i = g_{i-1} + f_g(|\widehat{\vartheta}_s - y_i|), \quad \text{if} \quad |\widehat{\vartheta}_s - f_y(y_s, \ldots, y_i)| \geq f^2_\Delta(\Delta) \tag{4.2}$$

$$g_i = g_{i-1}, \quad \text{if} \quad |\widehat{\vartheta}_s - f_y(y_s, \ldots, y_i)| < f^2_\Delta(\Delta) \tag{4.3}$$

where g is initialized to $g_0 = 0$ and it is reset to 0 every time that a state change is detected.

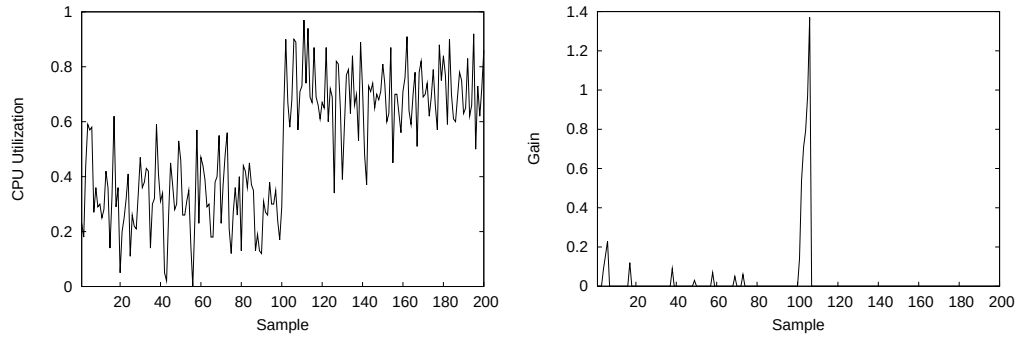
g is characterized by a stable value when the condition $|\widehat{\vartheta}_s - f_y(y_s, \ldots, y_i)| \geq f^2_\Delta(\Delta)$ is not satisfied. The function $f_g(|\widehat{\vartheta}_s - y_i|)$ computes the actual value of increment (decrement) among the online estimation of the current state $\widehat{\vartheta}_s$ and the actual sample y_i that must be summed to g_{i-1}. The choice of the function f_g differs from a detection model to another [151] and depends on the statistical characteristics representative of a time series. When the intensity of consecutive detected shifts is over a given threshold H, the model has to signal a relevant state change. Therefore, the detection rule of this class of models can be written as follows:

$$\text{detection rule} = \begin{cases} \text{state change}, & \text{if } g_i \geq H \\ \text{no state change}, & \text{otherwise.} \end{cases} \tag{4.4}$$

The choice of the characteristic threshold H depends on the specific gain function g and on the performance required by the state change detection algorithm.

We give an example of gain function computed on the time series of Figure 4.1(a). It refers to the CPU utilization of a server sampled every five seconds and that is characterized by a relevant load change at sample 100. Figure 4.1(b) reports the results of the gain function g applied to that time series. As expected, g is characterized by a significant increment at the instant of the relevant change in the system load.

For both classes of detectors, when a relevant state change is detected, the detection model has to provide an estimation of the statistical characteristic $\widehat{\vartheta}$ of the novel state. For simplicity reasons, without loss of generality, in this chapter we consider the state change detection problem applied to the case of relevant changes in the mean of the time series. Hence, we consider $\widehat{\vartheta} = \mu$. In this way, after a state change the estimation of novel state should be able to capture the central tendency of the time series during that state.



(a) Time series (b) Gain function g

Figure 4.1: Detecting relevant state changes through a gain function

Online detection problem in non-stationary, highly variable and spike contexts

The performance of the state change detection algorithms depends heavily on the statistical properties of the time series [45]. The context of the Internet-based servers where data streams are continually collected from monitored system resources opens interesting new challenges. These servers are subject to variable resource demands, heterogeneous processes, different levels of virtualizations. Consequently, monitored system resources are characterized by high levels of utilization, unexpected and unpredictable events and several sources of perturbations which result into spikes. The highly variable, non-stationary and unpredictable usage of resource measures and the presence of spikes and other perturbations with variable intensities complicate the identification of the system state and state changes. Detecting relevant state changes is further complicated by the temporal constraints imposed by real-time management strategies that receive the outputs of the online detection algorithms. This novel perspective prevents the application of state of the art algorithms working offline (e.g., [138]). We will show that it causes poor performance even of the most popular algorithms that can be applied to online continuous streams.

We give some examples of the major issues that affect the online detection algorithms when applied to Internet based servers. The architecture that we use for our evaluations is a typical distributed multi-tier Web system that is based on the implementation presented in [152]. The application servers are deployed through the Tomcat servlet container, and are connected to MySQL database servers. In our experiments, we exercise the system through realistic traces. We emulate more variable scenarios by means of the TPC-W workload generator [153]. Our workload scenarios are described by distinct step variations of the external load obtained by changing the number of emulated browsers. The algorithm that we

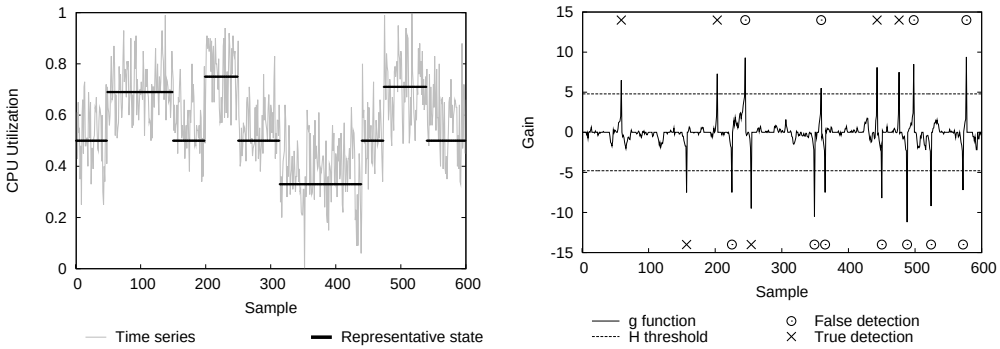
use in the following evaluation is the Cumulative Sum (Cusum) that is a widely used sequential analysis technique for detecting changing points in time series.

In the following examples, we use the CPU utilization of the database server as the representative system resource measure, because of the strong correlation of its utilization level with the external load (larger than 0.8). We set as representative states value $\{\mu\}$ the mean of the CPU utilization samples during the period in which the external load was constant and select the samples of change $\{s\}$ as the sample in which the number of emulated browser changes. Figure 4.2(a), Figure 4.3(a) and Figure 4.4(a) show the CPU utilization of three database servers that are subject to different external loads: the first user scenario is characterized by relevant changes in the number emulated browser at samples $\{$50, 150, 200, 250, 315, 440, 475, 540$\}$; the second one by any relevant change; and the last one by a single relevant change, at sample 300, in which both the user requests type and the number of emulated browser are changed. We consider that all detected changes signaled correctly by the detection algorithm are called *true positives* (TP). If an algorithm does not detect one relevant state change, the related sample is classified as *false negative* (FN). Analogously, the detection of a change is classified as *false positive* (FP) if it occurs when the time series is in a stable state.

Figure 4.2(b), Figure 4.3(b) and Figure 4.4(b) show the results of the gain function g function referring to the Cusum algorithm. In all of these figures, the horizontal dotted lines represent the upper and lower H thresholds set by the model, that signals a relevant state change every time the gain function g overcomes these limits. Each circle at the bottom and at the top of the figures denotes a false positive detection, that is, a signaled change that does not correspond to a real state change. A cross denotes a correct detection of a state change, that is a true positive.

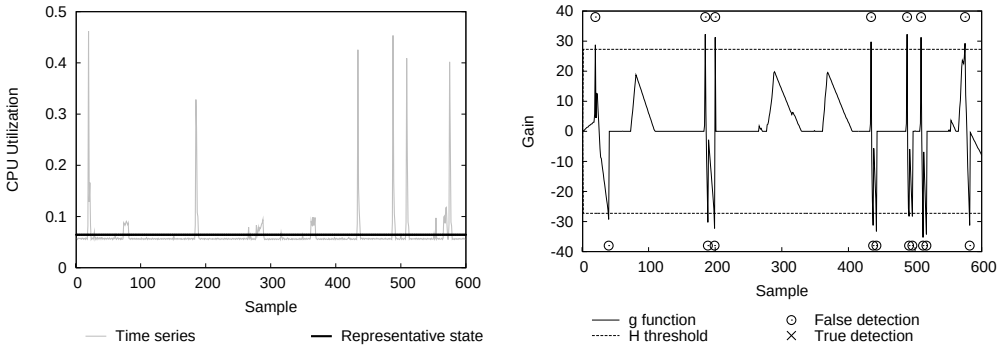
Figure 4.2(a) reports a time series characterized by non-stationary, highly variable and unpredictable data. These statistical properties of the data stream make it difficult to the model to identify intervals of stability in which the statistical properties ϑ remain constant. This reflects in oscillating values of the gain function that cause a lot of signals not corresponding to real state changes: there are eleven false positive detections and two false negative detections that are the undetected changes at sample 315 and 540.

Figure 4.3 reports a typical result achieved by existing algorithms by the Cusum algorithm applied to spiky time series. Since the g function accumulates the deviations of the monitored data from the current state, it records also the contributions of instantaneous spikes and anomalous perturbations departing from the current data behavior. These contributions are usually of high intensity and cause an immediate exceeding of the H threshold. For this reason, we have false positive detections corresponding to the peaks of the time series. Removing out-of-scale



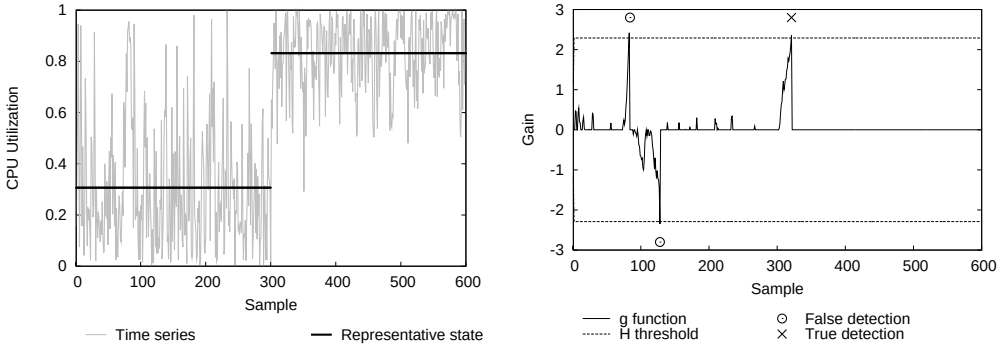
(a) Time series (b) Results of the Cusum algorithm

Figure 4.2: Time series characterized by non stationarity, high variability and unpredictability



(a) Time series (b) Results of the Cusum algorithm

Figure 4.3: Time series characterized by spikes



(a) Time series (b) Results of the Cusum algorithm

Figure 4.4: Time series characterized by variable variance

values through filtering algorithms before applying the detection rule should solve this problem. For this reason, instead of using the original time series $\{y\}$, we suggest the use of an online *data representation* $\{x\}$, that is a rectified version of the monitored samples. More details will be given in Section 4.1.2.

In Figure 4.4 we report an example where the Cusum detection model is applied to a time series characterized by a variable variance that passes from a higher value in the interval $[0 : 300]$ to a lower value after the state change. Here, the detector exhibits two problems: false positive detections at samples 90 and 112 during the interval of high variance and excessive delay because the relevant state change is signaled 21 samples after its occurrence at sample 300. A late detection is equivalent to an incorrect detection when the delay is incompatible with the temporal constraints imposed by the online context. These problems are the consequence of a static estimation of the time series variance used by the models for detecting changes in the mean [137]. For this reason, an efficient state change detection model in the context of Internet-based systems must adapt its detection policies to the variable variance of the time series.

This preliminary analysis evidences the main problems that may affect an online state change detection algorithm when the time series show the main statistical properties characterizing data streams related to Internet-based servers: false detections and lack of signals. We conclude that novel online detection models are required in the context of data streams related to modern Internet-based architectures.

4.1.2 Adaptive model for state change detection

The performance of change detection algorithms is highly dependent on the statistical characteristics of the measured data [45]. Because of the inherent variability and non-stationary behavior of the monitored processes related to Internet-based servers, existing algorithms achieve poor results.

We propose a new methodology that extends and integrates two well known models (Cusum [138] and Wavelet [146]) in two directions: adaptability and application to continuous data streams. The integrated version can be efficiently implemented and is therefore suitable to online detections. The motivation for our choice comes from the observation that the Cusum change detection algorithm has been proven to be optimal in terms of detection delay and minimum false alarm probability [154]. Nevertheless, its direct application leads to poor results when the data stream is characterized by high variability. To this end, we combine it with an online Wavelet filter. We adopt Wavelet filters for their ability to remove random errors from time series (operation which is known as denoising or rectification depending on the context [147]) without affecting the relevant features of the original data stream. In this respect, they have proven to be optimal with

respect to various error norms and smoothness property [147].

The baseline Cusum algorithm

Cusum is a widely used model for detecting changing points in otherwise stationary time series with known statistical characteristics. Proposed by Page in 1954 [138], this technique has been extensively studied and extended since [135, 155]. The Cusum algorithm has been shown to be optimal when the variance of the time series does not change, in that it guarantees minimum mean delay to detection in the asymptotic regime when the mean time between false alarms goes to infinity [154].

In this section, we describe the baseline Cusum algorithm; we use it in the experiments as a comparison testbed and to illustrate the benefits of our online Adaptive Cusum algorithm.

Given a time series $\{y_s, \ldots, y_i\}$ with known mean μ_s, the one-sided Cusum detects an increase in the mean through the following g_i gain function:

$$g_0^+ = 0 \tag{4.5}$$
$$g_i^+ = \max\{0, g_{i-1}^+ + y_i - (\mu_s + K^+)\} \tag{4.6}$$

which measures positive deviations of the monitored time series y_i from the state value μ_s. The gain function g_i^+ accumulates deviations of y_i from the state value μ_s that are greater than a pre-defined threshold K^+, and resets to 0 on becoming negative. The term K^+, which is known as the allowance or slack value, determines the minimum deviation that the statistics g_i^+ accounts for, and depends on the choice of the minimum shift to be detect, Δ. According to the detection rule common to all detection models, a positive change is signaled when g_i^+ exceeds a design chosen threshold H^+.

The one-sided Cusum test for detecting negative deviations is defined similarly, as:

$$g_0^- = 0 \tag{4.7}$$
$$g_i^- = \max\{0, g_{i-1}^- + (\mu_s - K^-) - y_i\} \tag{4.8}$$

A negative change is signaled when g_i^- exceeds a threshold H^-.

A two-sided test to detect both increases and decreases is obtained by applying the two tests simultaneously. As in this chapter we are interested in detecting both increases and decreases, we will consider the two-sided test. For the sake of simplicity we will consider the symmetric case whereby $K^+ = K^- = K$ and $H^+ = H^- = H$.

When a relevant shift is detected, the Cusum test also provides an estimate of the new system state μ_{s+1} through the following equations:

$$\mu_{s+1} = \begin{cases} \mu_s + K + \frac{g_i^+}{N^+} & \text{if } g_i^+ > H \\ \mu_s - K - \frac{g_i^-}{N^-} & \text{if } g_i^- > H \end{cases} \tag{4.9}$$

where N^+ (N^-) denotes the number of samples elapsed since the last time g_i^+ (g_i^-) was set to zero, that is $N^+ = i - \inf\{j \mid g_j^+ = 0\}$ and similarly for N^-.

The performance of the Cusum test is expressed in terms of the so called *Average Run Lengths* (ARL): ARL_0 denotes the average number of samples between false detections when no change has occurred; ARL_1 denotes the average number of samples to detect a change when it does occur. Ideally, ARL_0 should be large, while ARL_1 should be small. Both ARL measures are affected by the design parameters H and K. To achieve good performance, the suggested values in stationary time series are $K = \frac{\Delta}{2}$, where Δ is the minimum shift to be detected, and $H = 5\sigma_y$, where σ_y is the time series standard deviation [45]. In these conditions characterized by stationarity and constant variance of time series, the choice of $K = \frac{\Delta}{2}$ has been shown to provide near minimal ARL_1 for a wide range of threshold values H, while $H = 5\sigma_y$ guarantees $ARL_0 = 470$ for a shift of $\Delta = \sigma_y$, which is typically considered as a reference value.

There are several techniques to compute ARL_0 and ARL_1. In this chapter, we consider the Siegmund approximation that combines simplicity and efficacy [156].

In Figure 4.5 we plot ARL_0 and ARL_1 ($\Delta = \sigma_y$, $K = \Delta/2$). As shown in this figure, the performance of the Cusum algorithm is heavily dependent on the ratio H/σ_y. Let us consider first a fixed standard deviation σ_y and let us vary the threshold value H. As expected, for larger values of the threshold H, it is more difficult to incur in false detections (that is, it is more difficult that perturbations of g_i exceed the threshold) at the cost of an increasing detection delay ARL_1 since g_i needs to attain larger values for detection. From the same figure, we can also observe that, by increasing the threshold H, the false detection rate (which is inversely proportional to ARL_0) decreases exponentially fast at the expense of higher ARL_1.

Let us now fix H and let us vary the standard deviation σ_y. The key observation here is that the performance of the Cusum rapidly degrades as σ_y increases (smaller values of H/σ_y), since the false detection rate increases exponentially fast. Observe that this could be compensated by using large value of H which, on the other hand, has a negative impact on detection delay which, in on-line operation, must be kept as small as possible. As a consequence, in a non-stationary context whereby the variance can exhibit significant variation over time, we should not rely on a fixed set of parameters.

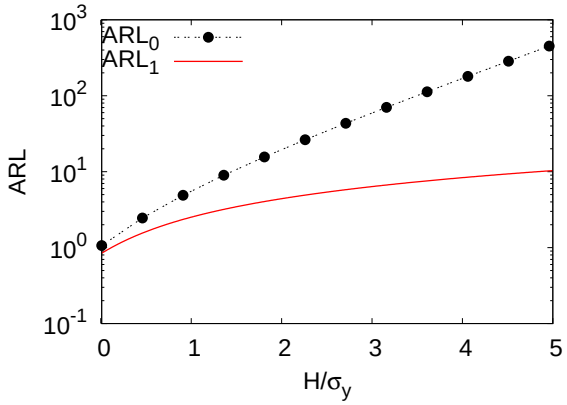


Figure 4.5: Average Run Length, $\Delta = \sigma_y$.

The online Adaptive Cusum detector

The basic Cusum algorithm is the best choice for statistical quality control of many processes characterized by stationary time series [45]. Unfortunately, it cannot be directly applied to online state change detection in Internet-based systems. First of all, the Cusum algorithm requires knowledge of the stationary state value μ_s against which measuring the time series deviations. Our experience shows this does not apply to computer system resource dynamics: CPU, disk and network usage dynamics are clearly non-stationary with respect to all relevant statistics, where the mean value and standard deviation vary over time. Second of all, the non-stationary behavior prevents the possibility of setting the design parameters H and K that can guarantee good performance over time. This is a critical issue because, as we have seen, for any fixed value of H the Cusum performance rapidly degrades as the time series standard deviation σ_y increases.

In this section, we propose a version of the Cusum model that we call *Adaptive Cusum*, capable of detecting state changes in face of the non-stationary characteristics of the continuous data stream. The proposed algorithm aims to solve the limitations of the baseline Cusum algorithm outlined above as follows:

1. we replace the reference state value μ_s by an adaptive estimation μ_i of the time series mean;

2. we replace the standard deviation σ_y by an online estimation σ_i of it ;

3. we dynamically adjust the threshold H as to provide a target ARL_0 performance in spite of high and possibly time varying variances.

The proposed Adaptive Cusum-based detector consists of two components: an Exponential Weighted Moving Average (EWMA) filter that tracks the slow

varying mean, and a two-sided Cusum test with varying thresholds for detecting relevant state changes. We consider the following tracking EMWA filter:

$$\mu_i = \alpha y_i + (1-\alpha)\mu_{i-1} \tag{4.10}$$

where $0 < \alpha \leq 1$ is typically set to $1/(1+2\Pi * cf)$ and cf is the cutoff frequency of the EWMA filter [157].

The time series variance σ_y is in general unknown and varying over time. We resort to a widely adopted approximation of variance that, for the sake of simplicity necessary in an online context, basically replaces the standard deviation σ_y with the mean deviation $E[\mid y - E[y] \mid]$ computed over time [158]:

$$\sigma_i = \alpha \mid y_i - \mu_i \mid + (1-\alpha)\sigma_{i-1} \tag{4.11}$$

where $0 < \alpha \leq 1$ is the same as in (4.10).

By setting a suitable value for ARL_0 so to guarantee good performance in terms of low false detections, we are able to dynamically adjust the value of the threshold H^* to reflect the variation of the data stream variance. This keeps a desired performance of the state change detector. The computation of H^* for the desired ARL_0 is carried out by a numerical inversion of the the Siegmund approximation [156], where the parameters are set as following: $K = \frac{\Delta}{2}$, where Δ is the smallest shift we want to detect (state changes smaller than Δ are accounted for by μ_i which tracks the state). Hence, to evaluate H^* we use:

$$ARL_0^{+/-} = \frac{e^{\frac{\Delta}{\sigma_i}(\frac{H^*}{\sigma_i}+1.166)} - \frac{\Delta}{\sigma_i}(\frac{H^*}{\sigma_i}+1.166) - 1}{\frac{\Delta^2}{2\sigma_i^2}}. \tag{4.12}$$

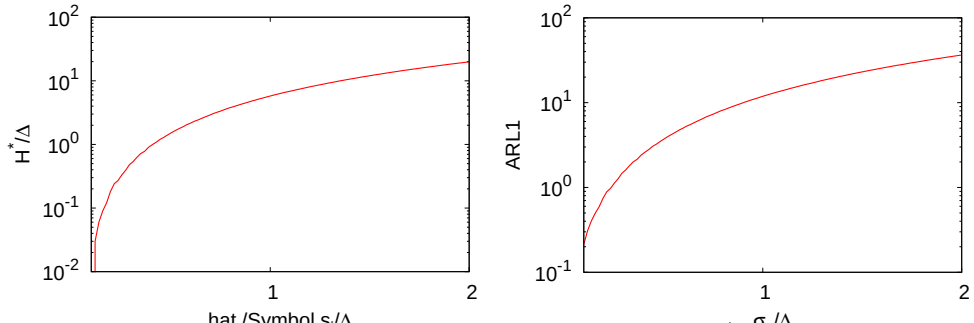
The resulting curve is plotted in Figure 4.6(a) where the target run length is set to $ARL_0 = 1000$. As expected, H^* increases significantly as a function of σ_i. We can compute a closed form approximation of the curve through a polynomial approximation. For $ARL_0 = 1000$, we obtained a good approximation with the following 3rd degree polynomial:

$$H^*(\sigma_i) = -1.3645\sigma_i^3 + 10.3031\sigma_i^2 + 3.1860\sigma_i - 0.2882 \tag{4.13}$$

The detection rule is provided by modifying (4.6) and (4.8) as following:

$$g_0^+ = 0 \tag{4.14}$$
$$g_i^+ = \max\{0, g_{i-1}^+ + y_i - (\mu_i + K^+)\} \tag{4.15}$$

$$g_0^- = 0 \tag{4.16}$$
$$g_i^- = \max\{0, g_{i-1}^- + (\mu_i - K^-) - y_i\} \tag{4.17}$$



(a) Adaptive threshold H^* (b) ARL_1

Figure 4.6: Adaptive threshold H^* and Average Run Length ARL_1 as a function of the time series variance (target: $ARL_0 = 1000$)

where μ_s is replaced by the adaptive estimation of the state μ_i.

A state change is detected whenever g_i^+ or g_i^- exceeds the threshold H^*. When a change occurs, the online estimation of the state μ_i is updated through the online implementation of (4.10):

$$\mu_i = \begin{cases} \mu_{i-1} + K + \frac{g_i^+}{N^+} & \text{if } g_i^+ > H^* \\ \mu_{i-1} - K - \frac{g_i^-}{N^-} & \text{if } g_i^- > H^* \end{cases} \tag{4.18}$$

This adaptive version of the Cusum algorithm we have presented is able to deal with non-stationary time series. It does so by adapting the threshold H^* to the online estimate of the time series standard deviation, to guarantee acceptable level of false detections. Clearly this comes at a cost. In Figure 4.6(b) we also plot the ARL_1 of the Adaptive Cusum as function of σ_i. ARL_1 is a function of H^*, which in its turn, in our scheme, is function of σ_i. As expected, ARL_1 grows for increasing values of σ_i. This can be explained by observing that for higher values of σ_i the algorithm requires higher values of the threshold H^* to guarantee a desired level of false detection rate. The higher values of the threshold H^* translate into higher detection delays (see Figure 4.5).

As confirmed by our experiments, because of the high variance that characterizes time series related to Internet-based servers, the Adaptive Cusum algorithm alone might not be sufficient to guarantee good detection performance. To overcome these limitations, we combine our novel change detection algorithm with a data representation obtained by the *rectification* Wavelet algorithm that should reduce the noise/perturbations from the measurements and improve the change detection performance. We discuss it below.

Wavelet-based denoising

The performance of the adaptive Cusum algorithm is negatively affected by high variability in the original time series. Hence, we have to reduce the amount of noise/perturbation through some rectification algorithm. There are many existing techniques which can be broadly classified into linear and non-linear methods. Linear filter-based methods are widely adopted for their ease of use and computational efficiency. Unfortunately, they are single scale by nature, that is, they are simple low pass filters. As a consequence, if a time series contains features at multiple scales, linear filters must tradeoff the extent of noise removal with the quality of the retained features [49]. In our context, this would either result in reduced noise removal and too many false detections or in excessive smoothing which would nullify the effectiveness of the overall change detection scheme. Non-linear methods, inherently multiscale, exhibit better performance than the linear counterparts. In particular, Wavelet-based methods have been shown to be nearly optimal for various error norms and smoothness of the resulting time series [147]. Wavelet denoising exploits the use of a orthonormal basis localized both in space and frequency which allows us to reduce/remove noise without smoothing the time series features.

In this section we describe the rectification phase we adopt before the application of our adaptive state change detection model. In other words, to improve the performance of the change detection algorithm, we find it better to consider a *rectified data representation* $\{x\}$ [49] than the original data stream $\{y\}$. Here, $\{x\}$ retains the significant features of the original data but removes (most of) the variability that can be ascribed to short-term perturbations. We regard $\{x\}$ as the data representation of the monitored process obtained through an online version of the Wavelet model, and we apply state change detection rules based on the adaptive version of the Cusum algorithm to the rectified time series $\{x\}$. We refer to an online version of the Wavelet-based denoising/rectification [49], that is a powerful method for filtering/rectification, and use it as a basis of our online data representation. Rectification based on the Wavelet transform [146] is able to isolate and remove the perturbations that affect the monitored processes. As observed above, the reason of this result is that it uses an orthonormal basis localized in space and frequency while the traditional solutions based on exponential smoothing techniques work only in the frequency domain.

We start giving a definition of Wavelet transform, commonly used in the offline contexts, that represents a time series as the sum of a shifted and scaled version of a base Wavelet function ψ and a shifted version of a low-pass scale function ϕ. With a proper choice of the Wavelet and scale functions, the resulting families of

functions are:

$$\psi_{mk}(n) = \sqrt{2^{-m}}\psi(2^{-m}n - k) \tag{4.19}$$

$$\phi_{mk}(n) = \sqrt{2^{-m}}\phi(2^{-m}n - k) \tag{4.20}$$

where m and k are the dilation and translation parameters, respectively, from an orthonormal basis. A time series $\{y\}$ can be conveniently rewritten as follows:

$$y_i = \sum_{k=1}^{n2^{-L}} a_{Lk}\phi(i) + \sum_{m=1}^{L}\sum_{k=1}^{n2^{-m}} d_{mk}\psi_{mk}(i) \tag{4.21}$$

where a_{Lk} is the k-th scaling function coefficient at the coarsest scale L, d_{mk} is the k-th Wavelet coefficient at scale k, and n is the time series length. The coefficients m and k are computed by the inner product of $\{y\}$ with the base functions. Computation of the transform and its inverse can be done in $O(n)$. As indicated in [49], in our implementation we set the coarsest scale L equal to 5 if the time series is perturbed by white noise, $L = 4$ otherwise. A key feature of this representation is that the Wavelet decomposition captures significant signal features in few relatively large coefficients, while perturbations result uncorrelated. As a result, perturbations - and perturbations only - can be effectively removed by setting equal to zero the Wavelet coefficients smaller than a threshold specified below.

Summing up, we obtain the data representation $\{x\}$ of the original time series $\{y\}$ through the following steps:

1. compute the Wavelet transform of the original time series $\{y\}$. We use the standard Haar function [159] as a base Wavelet, which consists of a simple rectangular impulse function;

2. set to zero the Wavelet coefficients which are lower than a suitable threshold t_m where m is the dilation parameter. As indicated in [49], we set the threshold $t_m = \sigma_m\sqrt{2\log n}$ where $\sigma_m = \frac{1}{0.6745}\text{median}\{|d_{mk}|\}$;

3. compute the inverse Wavelet transform to obtain $\{x\}$.

This rectification technique has been proved to be superior to other approaches [147] but it can only be applied on offline operations. Since this would be not acceptable for real-time operations, we consider the online version proposed in [49] where, at each step i, the rectified value x_i is computed using only past values as follows:

a) consider the sub-sequence $\{y_i\} = (y_{i-M+1}, \ldots, y_i)$ of maximum dyadic length, *e.g.*, with $M = \lfloor \log_2 i \rfloor$;

b) compute the rectified sequence $(z_1, \ldots, z_M)$ of the sub-sequence $\{y_i\}$ using the steps 1-3 above;

c) set $x_i = z_M$, *e.g.*, set the actual rectified value equal to the last value of the rectified sequence computed at step *b)*.

This online version can be computed in $O(n \log n)$ steps (see [49] for details) and it is therefore suitable to efficient implementations that are required by real-time management systems.

The overall scheme which combines the Wavelet-based rectification phase described in this subsection and the adaptive change detection phase described in Section 4.1.2 is summarized in Figure 4.7. The original measurements $\{y_i\}$ are fed to the rectification phase which produces the *rectified data representation* x_i. This is passed to the change detection phase implemented by the Adaptive Cusum algorithm, which consists of a standard deviation estimator that adaptively computes the Cusum threshold H^*, an EWMA filter to estimate the time series mean μ_i, and the Cusum algorithm itself which outputs the change detection events.

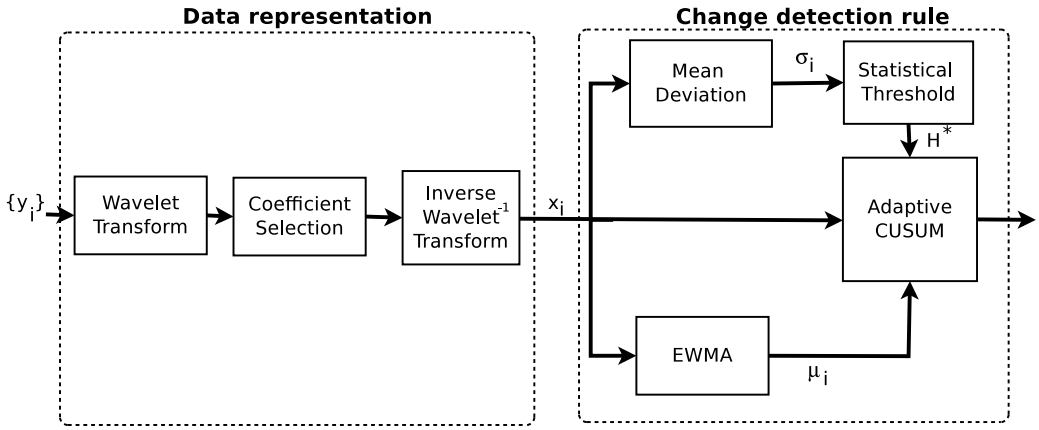


Figure 4.7: Schematic view of the proposed methodology

4.1.3 Other detection algorithms

The choice of the most appropriate model for state change detection depends on several factors, among which the application context and the statistical properties of the time series are the most important. These factors guide the choice of the detection rule and the data representation technique most appropriate to the state change detection model. As the focus of this chapter is on state change detection algorithms working online in the context of Internet-based systems, many existing models cannot be considered for comparison because they can work just offline.

Hence, as a term of comparison, we consider three popular classes of models that can be applied to online contexts: threshold-based model [149], Exponential Weighted Moving Average (EWMA) [45], baseline Cusum [138].

The second important element is that the considered data streams are characterized by a high variability to the extent that, for a fair comparison, all considered algorithms work on a filtered data representation instead of raw data. To this purpose, we consider three rectification techniques: Exponential Weighted Moving Average (EWMA) [45], Kalman filter [142] and Wavelet [49]. The combination of state change detection algorithms and filtering methods allows us to consider the following detectors:

- EWMA$_n$-Threshold (Section 4.1.3);
- EWMA$_n$-EWMA (Section 4.1.3);
- Kalman-baseline Cusum (Section 4.1.3);
- EWMA$_n$-baseline Cusum (Section 4.1.2);
- Wavelet-baseline Cusum (Section 4.1.2):
- Wavelet-Adaptive Cusum (Section 4.1.2).

EWMA$_n$-Threshold detector

Threshold-based models are commonly applied to Internet-based systems to support many real-time management algorithms [160–162]. The threshold-based detector uses a data representation x_i of the measurements that filters them through an exponential weighted moving average. It is defined as: $x_i = \lambda y_i + (1-\lambda)x_{i-1}$, where $0 \leq \lambda \leq 1$ (a value is $\lambda = \frac{2}{n+1}$ [45]) and the starting representative state value is the mean of n time series values, that is, $\mu_s = \sum_{i=1}^{n}(y_i)/n$. On the basis of the chosen number of n past values, we denote the data representation technique as EWMA$_n$. We consider a detection based on a single threshold [149]. The detection rule compares the deviation among the value of x_i and the estimated mean μ_s to a statistical threshold set equal to Δ, that is, the smallest shift to be detected. Therefore, the considered detection rule is:

$$\text{detection rule} = \begin{cases} \text{state change}, & \text{if} \quad |\mu_s - x_i| \geq \Delta \\ \text{no state change}, & \text{otherwise.} \end{cases} \tag{4.22}$$

When a state change is detected, μ_s is updated to the current value of the data representation x_i . The performance of the threshold-based detectors depends on the choice of the threshold value. There is no theoretical support for it, and the right

choice of the threshold value completely depends on the application context and the workload characteristics. The quality of the threshold-based detectors tends to worsen when working on time series with high variances, that are responsible for many false positive detections.

EWMA$_n$-EWMA detector

The Exponential Weighted Moving Average (EWMA) is largely used to detect state changes in control charts [45] that are tools to determine whether a process is in a stable statistical control. The data representation x_i is an exponential weighted moving average computed on n past values. This algorithm detects a relevant state change each time the shift among μ_s and x_i overcomes a threshold depending on Δ, on the standard deviation σ_y of the time series and on the cut-off frequency λ of the EWMA data representation. The detection rule is defined as:

$$\text{detection rule} = \begin{cases} \text{state change}, & \text{if} \quad |\mu_s - x_i| \geq M\sigma_y\sqrt{\frac{\lambda}{2-\lambda}} \\ \text{no state change}, & \text{otherwise.} \end{cases} \tag{4.23}$$

where M is the length of the control limits and depends on the minimum shift Δ to be detected. μ_s is set to the current value of the data representation x_i each time the model detects a state change.

The EWMA detector is a good model when small shifts have to be detected [45]. In these conditions, the performance of the EWMA is similar to that of the baseline Cusum. Otherwise, when the state change consists of a large shift, the quality of the EWMA tends to decrease [45]. Moreover, the EWMA-based detectors are characterized by a tradeoff between the false positive detections and the number of false negative detections. A detector using a small set n of past values offers a more reactive data representation and tends to maximize the number of true positive detections at the cost of some false positive detections. Increasing the number n of considered past values, the data representation becomes smoother and should decrease the number of false positive detections at the cost of some false negatives. For this reason, in our context it is hard to find an n value able to provide reliable and efficient performance for time series with time-varying characteristics.

Kalman-baseline Cusum detector

State change detection models with the Kalman filter as data representation and the baseline Cusum as the detection rule are widely used in literature [137, 163]. The traditional implementation of the Kalman filter requires the knowledge of the system state model, which is impossible to define in the Internet-based context due to its non-stationary and unpredictable behavior [11]. For this reason, we consider

a simplified version of the Kalman filter, as presented in [142]. The considered data representation algorithm, namely BART, combines a version of the Kalman model for filtering the bandwidth measurements with a change detection model based on the Cusum test. The Kalman filter estimates the data representation x_i as follows:

$$x_i = x_{i-1} + G_i(y_i - Dx_{i-1}) \tag{4.24}$$

where G_i is the Kalman gain, y_i is the measured quantity and the D matrix represents its one step model. A Cusum based algorithm is then applied to x_i to update quickly the system state value when a state change is detected. This makes it feasible to overcome the tradeoffs regarding speed of adaptation to changes versus stable estimation. The BART algorithm can be directly applied in our contexts. We use the model parameter values that are tailored in [142].

The output of this algorithm is used as the data representation for the baseline Cusum detection rule. The quality of this detection model is conditioned by the ability of the BART algorithm to maintain a stable estimation of the state, through the choice of the statistical threshold H and of the slack value K that represents the minimum deviation that the detection rule of the baseline Cusum accounts for. The BART data representation tends to reduce significantly its quality in non-stationary conditions and when the time series variance increases because it updates continuously its state also during periods of stability. This tends to cause a high number of false detections.

4.1.4 Performance results

In this section, we present the main performance metrics . Then, we evaluate the quality of the proposed online detection model in different time series conditions carried out by modifying several parameters of the original time series in terms of variance and perturbations .

Performance metrics

The detection quality of the state change detection algorithms is evaluated in terms of the well known metrics *recall, precision* [164] and *mean delay* for detection [137].

We define *recall* as the fraction of detections that are relevant to the time series and that are successfully retrieved:

$$\textit{recall} = \frac{TP}{TP + FN} \tag{4.25}$$

This measure can be considered as the probability that a change is successfully detected by the algorithm. To achieve a recall value equal to 1, the detection algorithm must signal all relevant changes. The value of the recall alone is insufficient, because it must be supported by some information related to the number of false detections. This is measured by the *precision*, that is the fraction of relevant detections:

$$\textit{precision} = \frac{TP}{TP + FP} \tag{4.26}$$

where $(TP + FP)$ is the total number of detections. The precision gives information on the ability of a detection algorithm to limit false state change detections. A precision equal to 1 means that the algorithm detects only relevant changes, while low precision values are caused by a detection algorithm that signals many false state changes.

A tradeoff between recall and precision values exists. These two metrics can be combined into one measure, namely the *F-measure*, that gives a global estimation of the detection quality. The F-measure is the weighted harmonic mean of the precision and recall, that is,

$$\textit{F-measure} = 2\frac{precision * recall}{precision + recall} \tag{4.27}$$

An F-measure value close to 1 denotes a good detection quality, while it is lower for detection algorithms affected by false positive and false negative detections.

The *mean delay* for detection is related to the ability of an algorithm to detect a state change when it actually occurs. It quantifies the time required for the detection of a new state through the distance between the sample at which the model signals a state change and the actual sample of change in the state representation, and computes the mean over all the state changes. For example, let us consider a time series with Y state changes. Let $[s_1, \ldots, s_Y]$ be the actual samples of change and $[\hat{s}_1, \ldots, \hat{s}_Y]$ the samples at which the model detects the changes. The mean delay for detection is defined as:

$$\textit{mean delay} = \frac{\sum_{i=1}^{Y}(\hat{s}_i - s_i)}{Y} \tag{4.28}$$

Good detection algorithms should minimize the mean delay for detection.

Detection quality

As we are interested in online detections of changes in non-stationary, unpredictable time series arising from real application contexts, in this section we evaluate the detection quality of the proposed algorithm for a wide range of time series

based on emulated profiles deriving from real measures. We consider times series with a relevant increment of their values followed by a proportional decrement as in [165]. To facilitate the analysis and algorithms comparison, the profile is normalized so that state increases/decreases are denoted by a unit value and the model parameter values are set to provide the best results in every considered time series.

As expected, all detection algorithms tend to diminish their detection quality for increasing variability of the time series. It is important to apply the detection algorithms on time series characterized by different levels of variability. As described in [166] and [167], the most important statistical properties that characterize a time series are the standard deviation σ_y, and the data correlation index ρ_y. σ_y measures the dispersion of data while ρ_y measures the dependence of perturbations on the behavior of time series values. To this purpose we evaluate the performance metrics of the detection algorithms as a function of σ_y and for ρ_y set to 0. The recall and precision results for all considered algorithms and for several σ_y values are reported in Tables 4.1 and 4.2, respectively.

σ_y	**EWMA$_5$ Threshold**	**EWMA$_5$ EWMA**	**EWMA$_{10}$ EWMA**	**EWMA$_5$ baseline Cusum**	**Wavelet baseline Cusum**	**Kalman baseline Cusum**	**EWMA$_5$ Adaptive Cusum**	**Wavelet Adaptive Cusum**
0.1	1	1	1	1	1	1	1	**1**
0.2	0.97	1	1	1	1	1	1	**1**
0.3	0.96	1	1	1	1	1	1	**1**
0.4	0.92	1	1	1	1	1	1	**1**
0.5	0.89	1	0.98	0.99	1	1	1	**1**
0.6	0.83	0.99	0.87	1	1	1	1	**1**
0.7	0.85	0.95	0.31	1	1	1	1	**1**
0.8	0.92	0.88	0.02	1	1	1	0.99	**1**
0.9	0.94	0.71	0.01	0.99	1	1	0.99	**1**
1	1	0.64	0.01	1	1	1	1	**1**

Table 4.1: Recall - $\rho_y = 0$

Table 4.1 shows that when the dispersion is low ($\sigma_y \leq 0.5$) all the considered algorithms achieve recall values close to 1. This means that the algorithms are able to detect all relevant state changes, with the exception of the threshold-based algorithm. When the dispersion increases ($\sigma_y > 0.5$), the algorithms using EWMA as a detection rule (i.e., EWMA$_5$-EWMA and EWMA$_{10}$-EWMA) worsen significantly, as can be seen from their recall values. They risk being completely unreliable in highly variable contexts, as they do not detect many state changes. Even in time series with an intense data dispersion, the threshold and Cusum-based algorithms have good recall values (> 0.9), that is, they detect all relevant state changes. However, if we also consider the precision results, we notice in Table 4.2 that the threshold-based method is very sensitive to the data variations

σ_y	**EWMA$_5$ Threshold**	**EWMA$_5$ EWMA**	**EWMA$_{10}$ EWMA**	**EWMA$_5$ baseline Cusum**	**Wavelet baseline Cusum**	**Kalman baseline Cusum**	**EWMA$_5$ Adaptive Cusum**	**Wavelet Adaptive Cusum**
0.1	1	0.23	0.31	1	1	1	1	**1**
0.2	0.33	0.43	0.50	0.99	1	1	1	**1**
0.3	0.13	0.50	1	0.95	1	1	1	**1**
0.4	0.08	0.97	1	0.89	1	0.99	1	**1**
0.5	0.06	0.98	1	0.77	0.99	0.98	0.99	**0.99**
0.6	0.05	0.81	1	0.65	0.90	0.87	0.92	**0.96**
0.7	0.05	0.7	1	0.50	0.85	0.72	0.87	**0.96**
0.8	0.05	0.61	1	0.37	0.80	0.58	0.73	**0.93**
0.9	0.04	0.53	1	0.29	0.73	0.49	0.64	**0.87**
1	0.04	0.52	1	0.25	0.67	0.39	0.55	**0.84**

Table 4.2: Precision - $\rho_y = 0$

of the time series: although it detects all state changes, this is accompanied by a high number of false positives, as confirmed by the low precision values of the EWMA$_5$-Threshold. All Cusum-based algorithms are able to achieve good precision in time series characterized by a data dispersion $\sigma_y \leq 0.5$. In more variable contexts (e.g., $\sigma_y > 0.5$), only the proposed Wavelet-Adaptive Cusum provides a high detection quality by achieving always a precision ≥ 0.84.

Figure 4.8 reports the F-measure as a function of the standard deviation σ_y for all considered detection algorithms. This shows the combined effect of recall and precision. With the exception of algorithms based on the EWMA detection rule, the algorithm performance worsens for increasing values of σ_y. Figure 4.8 shows that the Wavelet-Adaptive Cusum algorithm achieves the best F-measure values for every σ_y. For example, when $\sigma_y = 1$, the F-measure of Wavelet-Adaptive Cusum remains consistently above 0.9, even though EWMA Adaptive Cusum, the best existing online detection algorithm, has an F-measure of only 0.7. The threshold-based method is characterized by an exponential decay of the detection quality for increasing values of σ_y. This behavior reveals that a similar algorithm cannot be applied in non-stationary and non deterministic contexts. For small standard deviations, the EWMA$_5$-EWMA and EWMA$_{10}$-EWMA improve the F-measure until $\sigma_y = 0.5$; beyond this, their F-measure decreases. This is due to their *inertia* limit that, together with the F-measure degradation for high σ_y values, highlights that the performance of the EWMA-based algorithms are unacceptable because too much sensitive to the statistical characteristics of the time series and to the choice of the algorithm parameters. Existing Cusum-based methods (EWMA$_5$-Cusum and Kalman-Cusum) are characterized by a small decay of the F-measure for low values of σ_y. On the other hand, the F-measure decreases faster when $\sigma_y > 0.5$. These results confirm that popular Cusum-based algorithms do not work well in online contexts related to system resource measures

of Internet-based systems. Instead, combining the Cusum detection rule with a data representation based on the Wavelet, it is possible to contain this limit as confirmed by the slow decay of the F-measure for high σ_y values.

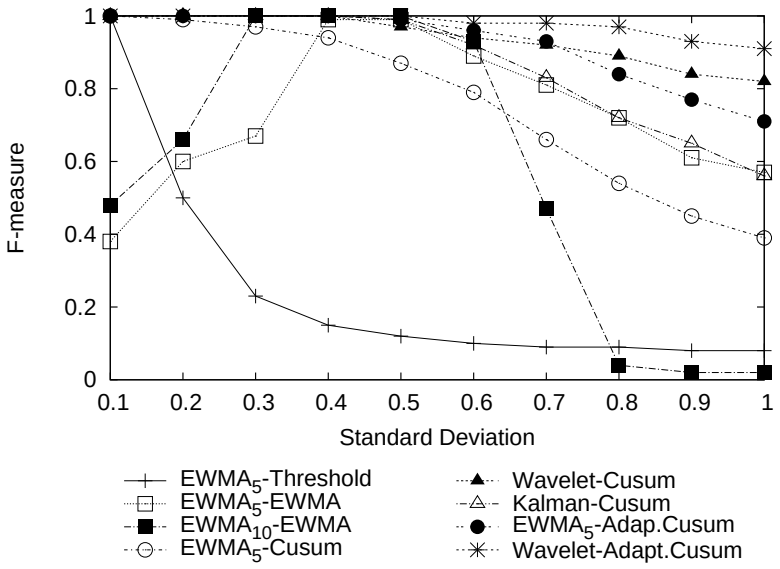


Figure 4.8: F-measures

We have to consider also the mean delay results, that are reported in Figure 4.9 for all the considered algorithms. In Figure 4.9, we plot the mean delay for detection for increasing values of σ_y. An expected trait of all the state change detection algorithms is the increase of the mean delay for detection, as a consequence of higher values of σ_y . All the algorithms but three are characterized by similar and low mean delay values. The EWMA_{10}-EWMA, EWMA_5-EWMA model and Kalman-Cusum, instead, are characterized by significant higher delays. In particular, the EWMA_{10}-EWMA is affected by a mean delay of 60 samples in the worst case. This result shows that this algorithm is inadequate to support real-time management decisions. Moreover, it confirms that the choice of the data representation technique is crucial for online state detection models. In highly variable contexts, just the proposed Wavelet-Adaptive Cusum provides an efficient tradeoff between the detection quality, expressed in terms of F-measure, and the delay.

Next, we examine the effects of the correlation ρ_y of the data component on the detection quality by considering different correlation indexes ($\rho_y = \{0, 0.1, 0.2, 0.3\}$) for two dispersion values ($\sigma_y = \{0.6, 0.9\}$). Figure 4.10 reports the F-measure values of the four Cusum-based algorithms that in Figure 4.8 show the best results (Wavelet-Adaptive Cusum, EWMA_5-Adaptive Cusum, Kalman-Cusum and Wavelet-Cusum) and of the two algorithms based on EWMA detection rule (EWMA_5-

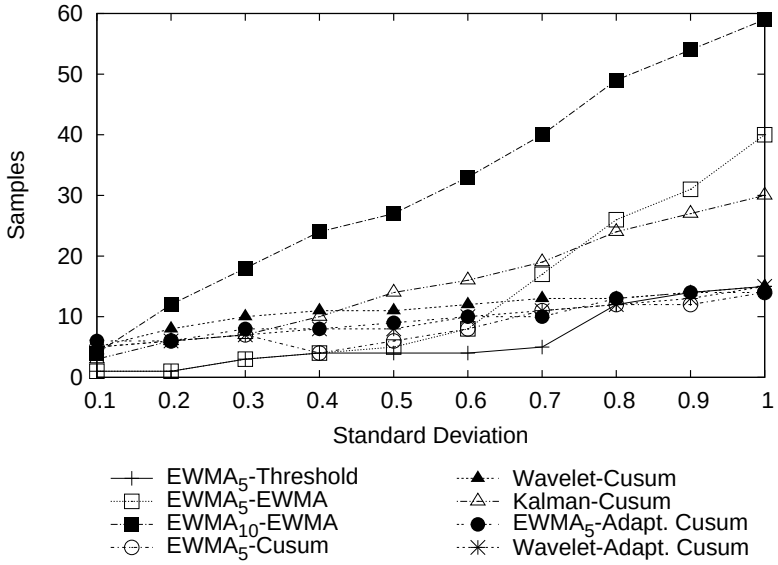
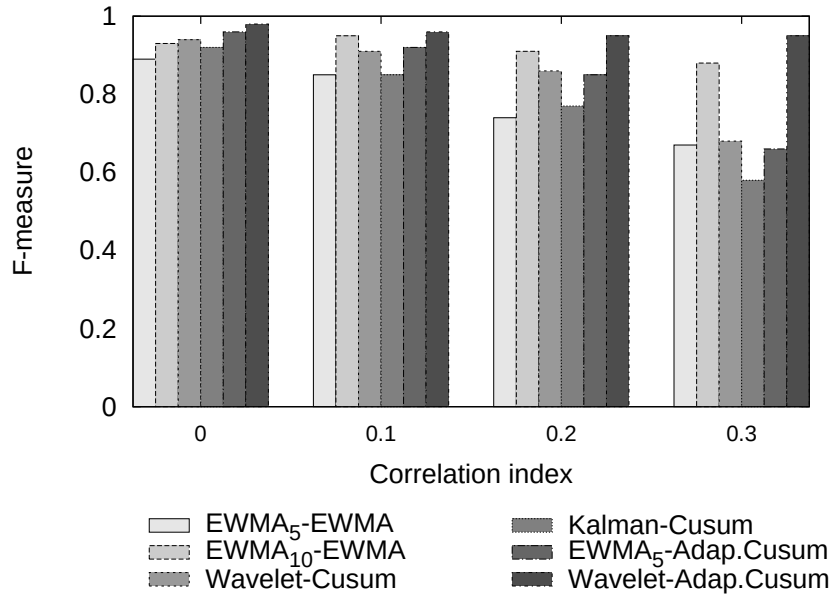


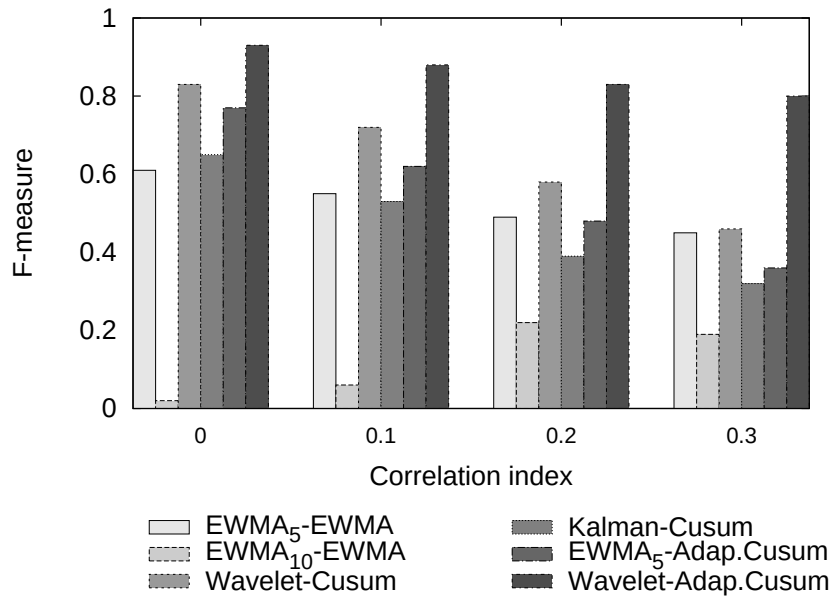
Figure 4.9: Mean delay for detection

EWMA, EWMA_{10}-EWMA). The results confirm that the proposed Wavelet-Adaptive Cusum algorithm improves the performance of all existing detectors for every correlation index and any σ_y characterizing the time series.

For quite high values of noise dispersion ($\sigma_y = 0.6$), the Wavelet-Adaptive Cusum algorithm always provides F-measures higher than 0.95. Its performance remains acceptable also when dealing with more variable time series, to the extent that in the most chaotic context of intense variance and strong correlation of the data component ($\sigma_y = 0.9$, $\rho_y = 0.3$) it improves of more than 50% the performance of the best existing algorithm Wavelet-Cusum. This result confirms the importance of using an adaptive detection rule in non-stationary and highly variable contexts. A second important result is that the Wavelet-Adaptive Cusum algorithm is less sensitive to the statistical characteristics of the time series with respect to any other detection algorithm. For a fixed value of σ_y, the performance of the proposed model degrades slowly with the increase the correlation index, thus demonstrating that the detection quality of the Wavelet-Adaptive Cusum is less affected by ρ_y than all the other Cusum-based algorithms. This stiffness is quite useful in all real contexts characterized by highly variable, non-stationary and non deterministic behaviors of the measured data.



(a) $\sigma_y = 0.6$



(b) $\sigma_y = 0.9$

Figure 4.10: F-measures

4.1.5 Experimental results

In this section, we apply the online detection models on measures related to system resources of real systems. In these experiments, since for real traces we do not

actually have a *ground-truth* which to refer to, we need a mean to define the representative states of time series and the occurrence of state changes. We adopted the following simple methodology [1], based on the *cluster analysis*. We start with an offline pre-processing of the monitored time series in which we remove all perturbations by means of a non-causal low-pass filter. We then apply cluster analysis to compute the representative states. We use a distance-based clustering based on the Quality Threshold clustering algorithm [168] and set a distance measure among clusters/states equal to Δ, that corresponds to the minimum relevance of the change that a model has to capture. For each state, we compute the mean value μ. The sequence of the mean values of the states $\{\mu\}$ and the respective samples $\{s\}$ to change to another state define the *representative states* of the time series.

In Figure 4.11, we give an example of a real time series, representing the CPU utilization of a database server, and its representative states and samples of change. By applying cluster analysis with Δ equal to 0.2 to the time series of Figure 4.11(a), we identify four representative states, defined by the following means values $\mu_1 = 0.50$, $\mu_2 = 0.75$, $\mu_3 = 0.52$ and $\mu_4 = 0.30$ and samples of change $s_1 = 50$, $s_2 = 100$ and $s_3 = 160$.

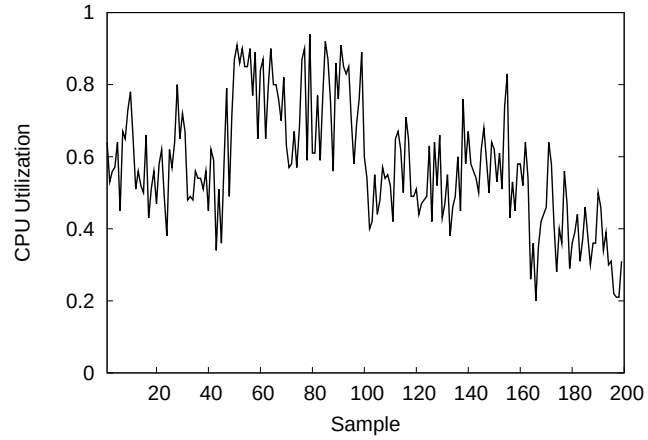
We evaluate the proposed state change detection algorithm on time series coming from server measures of an Internet-based system hosting Web sites with static, dynamic and secure content. For evaluation purposes, here we report the results related to the measures shown in Figure 4.12 referring to the CPU utilization of a machine hosting multiple guest servers. The considered measures are 1-minute averages of the CPU utilization on a single server.

We evaluate the representative state for the state change detection algorithms through the offline methodology described above that determines relevant state changes at samples 50, 125, 200, 275, 350, 400, 475, and 550, evidenced by the horizontal line in Figure 4.12.

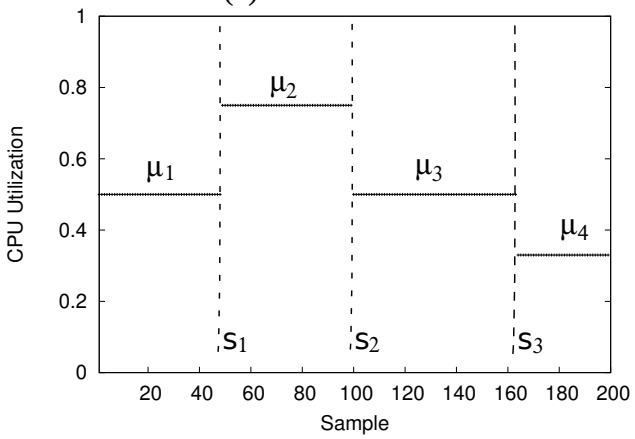
We report the results of the proposed Wavelet-Adaptive Cusum and of the selected detection algorithms (EWMA_5-Cusum, Kalman-Cusum, EWMA_5-EWMA and EWMA_{10}-EWMA) described in Section 4.1.3.

Figures 4.13(a), 4.14(a), 4.15(a), 4.16(a) and 4.17(a) show the curves of the online data representation generated by each of the five algorithms. The respective (b) figures on the right report the occurrence of false positive detections (vertical lines with a circle at the top) and true positive detections (vertical lines with a cross at the top). In this benchmark, we can appreciate that the Wavelet-Adaptive Cusum algorithm is able to achieve a data representation more smoothed than the corresponding representations of the EWMA_5-baseline Cusum, the Kalman-baseline Cusum and the EWMA_5-EWMA and similar to the EWMA_{10}-EWMA.

[1] We remark that in absence of a ground truth, any definition of representative state is arbitrary. Nevertheless, the proposed technique provided us *qualitatively* very good results.



(a) Time series



(b) Results of the Quality Threshold Clustering

Figure 4.11: Representative state of a real time series

The EWMA_5-Cusum, the Kalman-Cusum and the EWMA_5-EWMA shown in Figures 4.14(b), 4.15(b) and 4.16(b), detect all state changes but their precision decreases due to a high number of false positive detections. This is a consequence of their inability to maintain a stable data representation when the time series are non-stationary and highly variable. A different behavior is shown by the EWMA_{10}-EWMA in Figure 4.17(b): it misses more than 60% of state changes. This is due to the so called *inertia limit* [45], that is, the inability to react quickly to time series changes when the size of the smallest shift to detect is significantly higher than time series variance. As a consequence, the inertia limit strongly affects the recall quality. These results evidence and confirm the tradeoff problem between the false positive detections and the number of false negative detections that characterizes the EWMA-based detection rule.

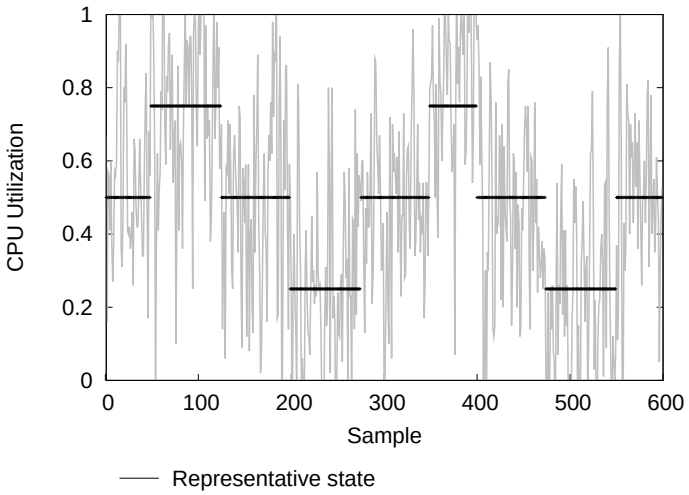
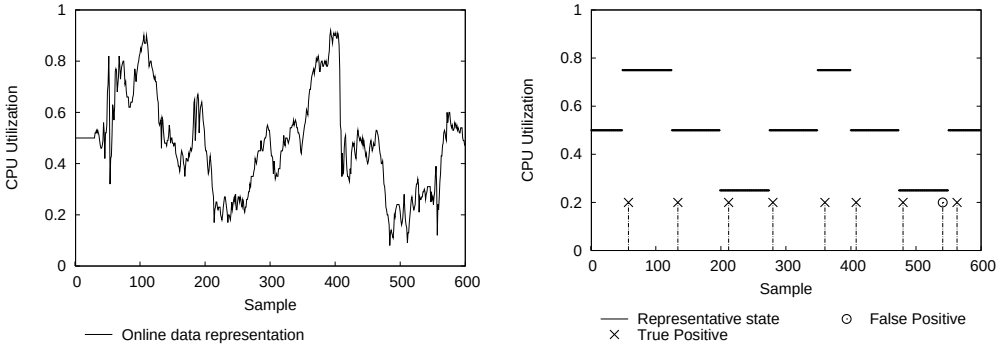
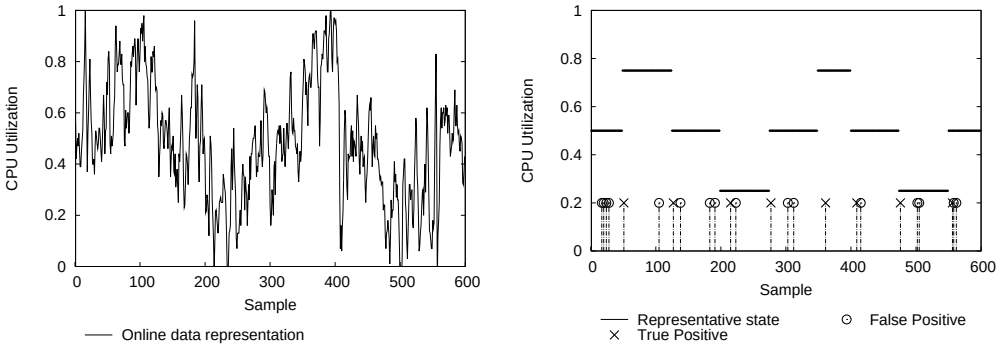


Figure 4.12: CPU utilization of an Internet-based server



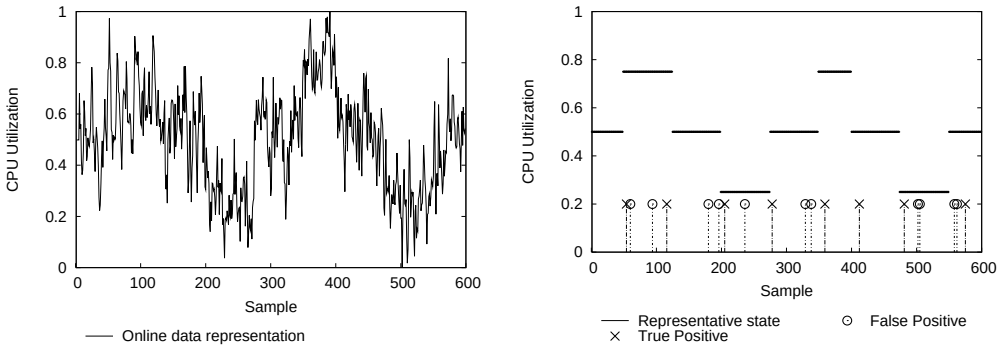
(a) Data representation (b) State change detections

Figure 4.13: Wavelet-Adaptive Cusum detector



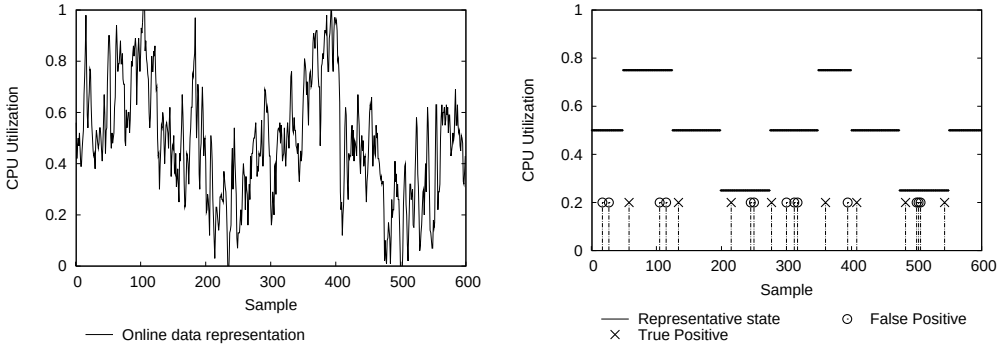
(a) Data representation (b) State change detections

Figure 4.14: $EWMA_5$-Cusum detector



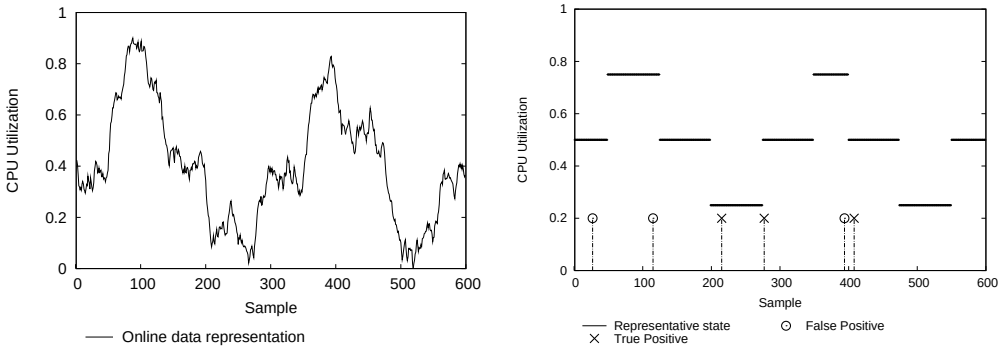
(a) Data representation (b) State change detections

Figure 4.15: Kalman-Cusum detector



(a) Data representation (b) State change detections

Figure 4.16: $EWMA_5$-EWMA detector



(a) Data representation (b) State change detections

Figure 4.17: $EWMA_{10}$-EWMA detector

On the other hand, the Wavelet-Adaptive Cusum algorithm in Figure 4.13(b) exhibits a precise detection also in contexts characterized by multiple relevant changes and high variability. The proposed algorithm detects timely the state

changes and it is affected by just one false detection at sample 540. Nevertheless, after the false state change detection, the Wavelet-Adaptive Cusum algorithm is able to adapt immediately its data representation to the right stable state. This capacity of self recovery is one of the most important property of the proposed algorithm that allows us to achieve always the best results. We report a summary of the results shown in the previous figures in Table 4.3.

Model	**TP**	**FP**	**FN**
Wavelet-Adaptive Cusum	8	1	0
EWMA$_5$-Cusum	8	16	0
Kalman-Cusum	8	11	0
EWMA$_5$-EWMA	8	13	0
EWMA$_{10}$-EWMA	3	3	5

Table 4.3: Summary experimental results

We now consider the problem of spiky time series. As example, we consider the time series shown in Figure 4.18, coming from the data streams related to the CPU utilization of a Web server. The measures exhibit a stable behavior without relevant state changes and are characterized by a low variance and by some spikes, such as at samples 170, 184, 259. In Table 4.19, we report the number of false positives signaled by the considered state change detection models. A reliable detection model applied to this time series should not detect any relevant state change. However, in these conditions, only the Wavelet-Adaptive Cusum avoids wrong detections. This result is the consequence of the usage of a reliable data representation rectified by spikes and perturbations and of the capacity of the proposed algorithm to preserve the stable state. On the other hand, the traditional state change detection models, such as the EWMA$_5$-Cusum, Kalman-Cusum and EWMA$_5$-EWMA, detect many state changes in occurrence of the spike values, as

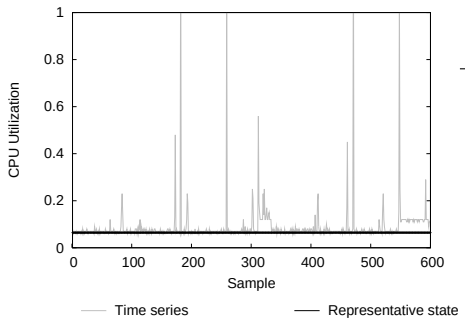


Figure 4.18: Spike time series

Detector	False Positive
Wavelet-Adaptive Cusum	**0**
EWMA$_5$-Cusum	**10**
Kalman-Cusum	**4**
EWMA$_5$-EWMA	**8**
EWMA$_{10}$-EWMA	**2**

Figure 4.19: Results

shown in Table 4.19. The $EWMA_{10}$-EWMA is able to limit the number of false detections to 2 because of the inertia limit.

4.2 Behavioral change detection

The identification of behavioral changes in some system resources is mandatory for an efficient management of Internet systems. As the dimension of modern Internet systems increases, the evaluation of state change detections through traditional algorithms becomes computationally intractable.

Existing models and frameworks that signal relevant changes in server operations by working on the entire data sets of monitored resources [140, 142] are inadequate to support system management decisions. We should consider that, even in a medium size infrastructure, huge amounts of data streams referring to different system resources may reach a change detector that should analyze, model and treat them at different temporal scales (from days to weeks) and should support prompt reconfigurations motivated by changes in system and workloads [28]. On the other side, solutions that avoid the analysis of all monitored measures by working on aggregated visions of subsets of heterogeneous servers [28] prevent both reliable change detections and the identification of servers that experienced that changes. Therefore, two main problems limit the applicability of present approaches for change detection and motivate this chapter: *dimensionality* and *heterogeneity* of the data set.

By taking advantage of the novel approach for big data treatment and server characterization introduced in Chapter 3, we characterize the statistical properties of the resource measures coming from system monitors, classify them, and evaluate the dynamic evolution of such resource classification. The periodic evaluation of this approach allows us to signal in a simple and accurate way the most significant behavioral changes without the need of considering all collected data streams bur analyzing just the most important ones.

The proposed approach has three main advantages: it does not require to model and analyze all resource measures of each server; it makes no a-priori assumptions about the statistical characteristics of data; it is flexible enough to detect behavioral changes at different time scales. These features guarantee that our method is appropriate for behavioral change detection in large enterprise data centers where continuous changes and high dimensionality of data are the norm.

4.2.1 Problem definition and fundamental choices

Most management systems referring to enterprise Internet systems and private cloud architectures require algorithms that are able to decide whether a rele-

vant change has occurred in the profile of some monitored system resources. In this context, change detection should reduce the operator involvement in favor of model-driven decision systems. Existing models and frameworks are inadequate to support management decisions that have to gather, filter and analyze huge amounts of heterogeneous and variable data streams, because they work on the entire data set of measures coming from server monitors.

The goal of the methodology of interest for this chapter is to detect *behavioral changes* in server resource measures, in terms of a collection of related data instances that behaves anomalously with respect to a behavior observed so far on the data stream [35]. The detection of behavioral changes requires the identification of a set of *behavioral attributes* and to find whether and when these attributes change. We identify the behavioral attributes in the *statistical properties* of monitored measures (e.g., correlation, standard deviation, mean value), that represent the type of relationship between data instances. When the observed statistical properties of a data stream significantly differ from the expected behavior, a change is declared.

Let us evidence an example of behavioral change in Figure 4.20. We consider the CPU utilization measures of a server hosting five virtual machines. The data show a periodic behavior during the first 1000 samples and then suddenly change to a stable behavior. This is a typical example of behavioral change that, for an efficient management of the enterprise Internet systems, the detection algorithm should promptly signal.

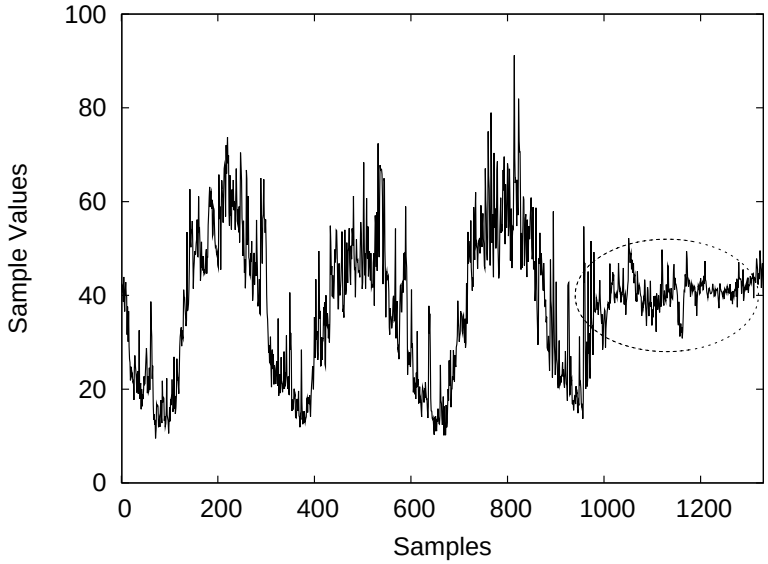


Figure 4.20: Example of a behavioral change.

This section provides a methodology for detecting behavioral changes in large Internet systems. In this context, our methodology addresses the challenge of detecting behavioral changes in suitable subsets of servers, interacting and competing for the same components at spatial, functional or application level. Our proposal works on performance measures sampled at suitable temporal intervals, ranging from seconds to few minutes in relation to the different time scales of interest.

It is mandatory to remark that the proposed change detector is oriented to enterprise Internet systems, to the so called private cloud-based infrastructures, and to any context where the system manager has full control on the resources assigned to different services. On the other hand, in the present version, the proposed approach cannot be applied to cloud data centers where the cloud provider can assign and move virtual machines from one host/cluster to another one without informing the cloud consumer.

Our methodology signals changes when the behavior of a server departs from normality, by using two types of supports for facing the dimensionality and heterogeneity problems: the analysis of a small set of all and only informative data; the classification of streams among negligible, deterministic and random streams so to evidence their statistical behavior.

The first stage is to evaluate the impact of each data stream on the system activity, so to distinguish important and not-relevant sources of information. Eliminating the less important data streams diminishes the system management complexity. Since the resource measures in an Internet system may be highly variable and this characteristics influences the change detection, we consider variability as the main statistical property to discriminate between relevant and not-relevant information. The variability is typically expressed in terms of process variance or in terms of coefficient of variation, and it quantifies the degree of activity of a process even as a measure of the system health. Indeed, the variability of processes is used to characterize the system behavior [109] in many different natural and non-natural processes, such as the flow of water through a river basin, the traffic of highway systems, or the Internet traffic. In [9], the process variance is used to classify heterogeneous nodes for revealing the impact of external requests on the system. In this chapter, we define *energy* the index of variation of the monitored processes, and we use this energy information in order to characterize the statistical nature of server behaviors. We reduce the dimensionality of the data set by using the strategy we introduced in Chapter 3.

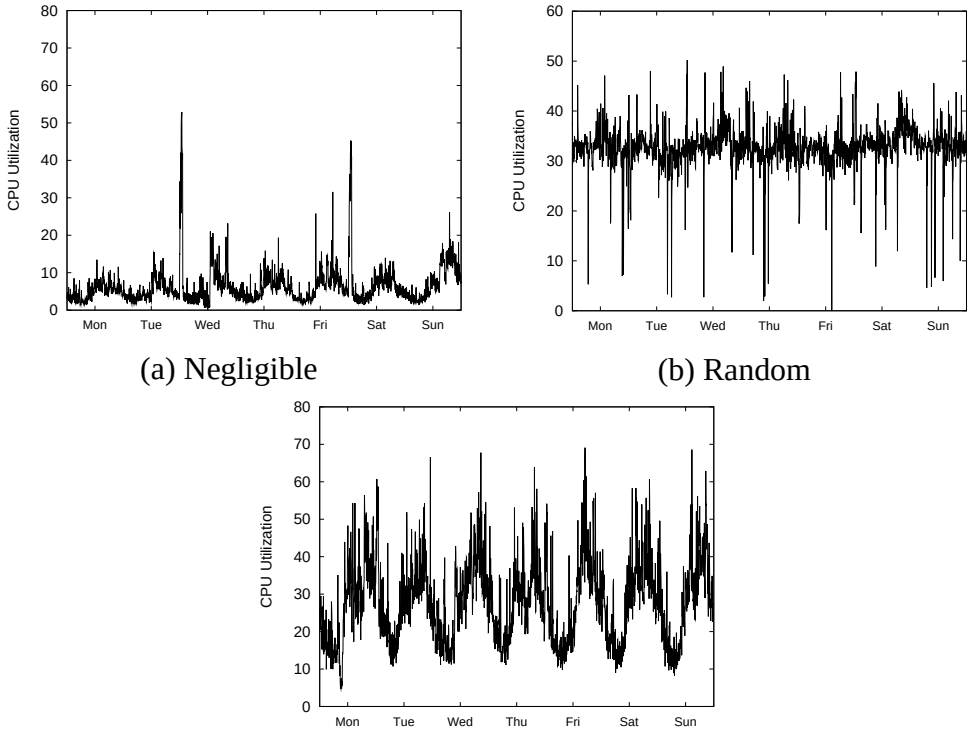
As a second stage, we characterize the resource behavior emerging from the monitored data in three main categories:

- **deterministic** behavior, whose behavioral attributes are systematic trends and periodic patterns that are predictable and possible to model;

- **random** behavior, presenting isolated and occasional bursts and dips, spikes and noises;

- **negligible** behavior, that gives a little contribution to the overall system behavior.

Figures 4.21(a), (b) and (c) report an example of the three mentioned statistical behaviors in the CPU utilization sampled from a Web server, an application server and a database server of an Internet system, respectively.



(a) Negligible

(b) Random

(c) Deterministic

Figure 4.21: Examples of server behaviors.

This classification is quite important because it allows us to identify the actual behavior of a server and to assign it to one of the three categories. The change in time in the characterization of a server is symptomatic of a significant behavioral change in its functioning. The identification of this type of changes is obtained through our methodology without the need of analyzing all numerous and heterogeneous data streams and without the loss of relevant information.

4.2.2 Methodology

The proposed detection method employs statistical analyses of data developed at consecutive intervals of time in order to signal a change in the behavior of some servers.

At time interval t, the set of collected sampled measures originates a matrix X_t that forms a high dimensional multivariate structure. It is a $n \times p$ matrix, where n is the number of samples collected among two consecutive intervals (for example, $n = 2016$ if we consider the five minutes samples enclosed in an one-week interval), and p is the number of considered system resource measures. To each resource matrix X_t we associate a vector $B_t = [b_{t,1}, \ldots, b_{t,p}]$ containing, the characterization of each resource measure at time interval t, that is, $b_{t,k} \in \{Deterministic, Random, Negligible\}$ for $1 \leq k \leq p$.

At the next time interval $t + 1$, we collect the new set of observations X_{t+1} and check whether the B_{t+1} vector characterizes the server behaviors as at time t. When a relevant change affects some servers, their statistical characterization changes and $B_t \neq B_{t+1}$. This deviation is symptomatic of a relevant change that we are able to capture simply by looking at the differences in the elements of two vectors. It is important to remark that the proposed approach shows only major changes through the analysis of a small set of data. However, a lack of change in the statistical characterization does not mean that a server did not experience any change.

We now detail how to build the B_t vectors. At each time interval t, our methodology implements two steps on collected data: *dimensionality reduction* and *heterogeneity inspection*.

The first step is required since the data sets collected from the server log files of modern enterprise Internet systems form a multivariate structure characterized by thousands of dimensions and more. In these structures and associated coordinated spaces, any reliable decision requires a preliminary identification of the most relevant information for management purposes. A common approach is to find a new coordinate space consisting of a lower dimension that is representative of the original space [120]. When a structure can be approximated through a smaller number of dimensions in a way that minimizes the error and the loss of information, we can refer to the smaller number of dimensions as the *structure intrinsic dimensionality*. In literature [112] several models exist. In our implementation, we choose the PCA [110] since we are mainly interested to measure the system energy in term of variance, and the PCA is the best model to reveal the intrinsic structure of data set on the basis of its variance.

As data sets of resource measures are statistically heterogeneous, before applying PCA to the collected X_t matrix it is important to normalize them to time series characterized by zero mean and unit variance, as suggested in [121]. PCA

maps X_t on a new set of axes [120] called components. Calculating the components is equivalent to solving the symmetric eigenvalue problem for the matrix $\Psi_t = X_t^T X_t$, that is a measure of the covariance of the time series deriving from sampled measures. In practice, at sample i each component v_i is the i-th eigenvector computed from the spectral decomposition of Ψ_t [121]:

$$\Psi_t v_i \quad = \quad \lambda_i v_i \qquad i = 1, \ldots, p \tag{4.29}$$

where λ_i is the eigenvalue corresponding to the eigenvector v_i and represents the magnitude of the variation along each component v_i. We quantify the *energy*, σ_i, associated to the component v_i as the percentage of variation related to its eigenvalue λ_i, that is, $\sigma_i = \frac{\lambda_i * 100}{\sum_{j=1}^{p} \lambda_j}$. As Ψ_t is symmetric positive definite, its eigenvectors are orthogonal and the corresponding eigenvalues are non-negative real numbers. By convention, the eigenvectors are unit norm and the eigenvalues are arranged from large to small, so that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_p$.

Once mapped the data set into their component space, we have to compute the transformed data considering one component at a time. In this new dimensional space, a *dimension* u_i ($i = 1, \ldots, p$) is defined as a vector of size p obtained by the contribution of the data set matrix X_t and of the component v_i. As suggested in [121], this vector is normalized to unit length by dividing it by $\sqrt{\lambda_i}$. Hence, for each principal component v_i we have:

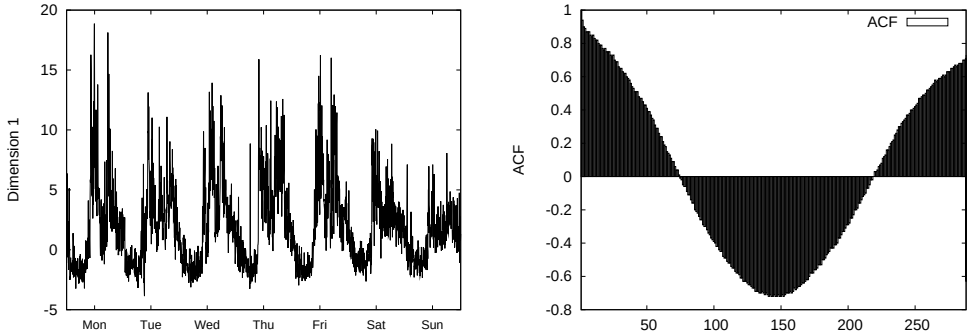
$$u_i \quad = \quad \frac{X_t v_i}{\sqrt{\lambda_i}} \qquad i = 1, \ldots, p \tag{4.30}$$

The above equation shows that all server behaviors, when weighted by v_i, produce one dimension of the transformed data. Thus the vector u_i captures the temporal variation common to all server measures along the component v_i. Since the components are ordered with respect to their contribution to the overall energy, u_1 captures the strongest temporal trend common to all server measures, u_2 captures the next strongest, and so on.

The energy characterizing each component can be used to reduce the data dimensionality by ignoring the less variable components of the structure. In order to choose how many component it is useful to retain, we pursue the criterion of the percentage of variability expressed by the principal components [125]. We determine in advance the amount of residual variability that we want tolerate and then exclude the residual components. Finding that only a small set of r dimensions are relevant implies that X_t can be mapped on a r-dimensional subspace of $\mathbb{R}^p$ and that $r \ll p$ is the intrinsic dimension of X_t. At every time interval t, this result allows the method to distinguish between the *relevant dimensions* $U_t = \{u_1, \ldots, u_r\}$ that are contributions shared by all resource measures on the r principal components, and the *not-relevant dimensions* $\overline{U_t} = \{u_{r+1}, \ldots, u_p\}$ corresponding to the less important components in terms of system energy.

The second step investigates data heterogeneity with the goal of characterizing server behaviors at time interval t and register in the B_t vector their deterministic, random or negligible nature. After the classification in relevant U_t and not-relevant $\overline{U_t}$ dimensions, we distinguish the relevant dimensions between *correlated* U_t^C and *low-correlated* U_t^L by evaluating the autocorrelation function (ACF) of each relevant dimension U_t as in [127] and many other contexts [11, 119].

Figure 4.22(a) reports an example of dimension exhibiting correlated periodicity and seasonal fluctuations because of diurnal activity, as well as the difference between weekday and weekend activity. The high values of the autocorrelation function of Figure 4.22(b) confirm that the dimension is correlated.



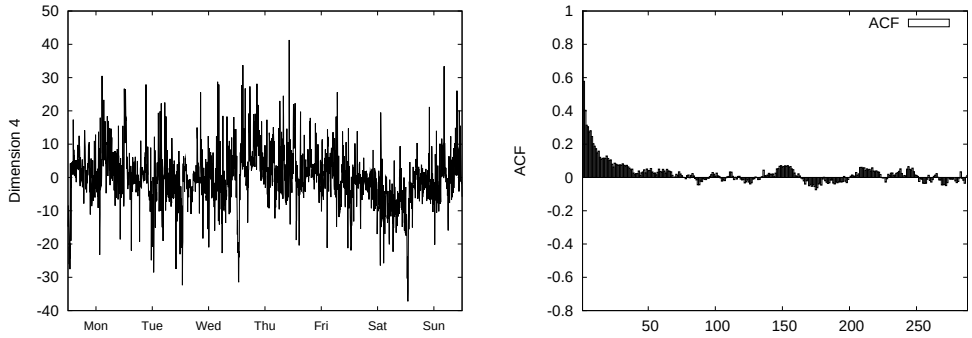
(a) Dimension (b) ACF

Figure 4.22: Example of correlated dimension.

Figure 4.23(a) reports an example of a dimension that is classified as low-correlated, since it presents the quick decay of the ACF values shown in Figure 4.23(b). This dimension is characterized by random perturbations and/or spikes varying in time and intensity. We use the classification of the relevant dimensions between correlated and low-correlated to evaluate three behavioral values for each data stream referring to one server resource.

At every time interval t and for every server resource measure k ($1 \leq k \leq p$), the idea is to add all the contributions for each of the three types of dimensions and then to evaluate the maximum.

- $D_{t,k} = \sum_j v_j, \forall j \mid u_j \in U_t^C$ denotes the contributions provided by the relevant correlated dimensions;
- $R_{t,k} = \sum_j v_j, \forall j \mid u_j \in U_t^L$ denotes the contributions provided by the relevant low- correlated dimensions;
- $N_{t,k} = \sum_j v_j, \forall j \mid u_j \in \overline{U_t}$, accumulates the contributions of the not-relevant dimensions.



(a) Dimension (b) ACF

Figure 4.23: Example of low-correlated dimension.

Now, it is possible to compute the highest value among the three that is, $M_{t,k} = max\{D_{t,k}, R_{t,k}, N_{t,k}\}$, and to estimate the characterization, $b_{t,k}$, of the k-th server resource measure on the basis of its main behavioral value:

$$b_{t,k} = \begin{cases} \textit{Deterministic} & \text{if } M_{t,k} = D_{t,k} \\ \textit{Random} & \text{if } M_{t,k} = R_{t,k} \\ \textit{Negligible} & \text{if } M_{t,k} = N_{t,k} \end{cases} \tag{4.31}$$

The characterization of each server is recorded in the B_t vector and represents the term of comparison for the detection of a behavioral change in the characterization at two consecutive time intervals. When $b_{t,k} \neq b_{t+1,k}$, a behavioral change affecting the k-th monitored server is detected.

4.2.3 Results

In this section, we present the experimental results of the proposed approach on data traces collected from an enterprise Internet system consisting of 514 servers and supporting different types of applications, including Web sites, databases, access controls, CMS, mail server, management software. In this testbed, system monitors collect resource measures every minute referring to CPU utilization, primary and secondary memory-related metrics, network activities. Our system is organized hierarchically, as suggested in [8], with subsystems of almost half a hundred servers in order to support an efficient orchestration of real-time management mechanisms.

For the sake of presentation, we consider two data sets, X_t and X_{t+1}, referring to the CPU utilization of a subsystem of 50 servers hosting e-Commerce websites with dynamic and interactive content over two consecutive days, November 8 and 9, 2010, respectively. X_t and X_{t+1} are matrices with $n = 1440$ rows and $p = 50$ columns. This section presents the results of the server characterization and of the

Order	Behavioral class	Energy
1	**Correlated**	51.03%
2	**Correlated**	11.99%
3	**Correlated**	8.64%
4	**Low-correlated**	4.8%
5	**Low-correlated**	3.27%
6	**Correlated**	3.14%
7	**Correlated**	3%
8	**Low-correlated**	2.87%
9	**Low-correlated**	2.42%

Table 4.4: Classification of relevant dimensions.

behavioral change detection. Then, we analyze the computational complexity of the methodology by considering several sizes of the data set.

Server characterization and behavioral changes

We initially consider the matrix X_t and evaluate the energy σ_i carried on by each dimension u_i in order to identify how many dimensions actually bring information to this data set. Considering the 90-percentile of the energy of the overall system, we estimate that it is captured just by the first nine dimensions as shown in Table 4.4. These dimensions contribute to most of server variability. Our approach evaluates the first 9 dimensions as *relevant*, while the other 41 dimensions are marked as *not-relevant* from the point of view of the energy. This is an important result of the proposed methodology applied to a real Internet system because it confirms that, in terms of energy, the resource measures all together form a structure with an intrinsic dimensionality of $r = 9$ that is much lower than the original number (50) of the considered set of time series.

The application of the ACF to the $r = 9$ relevant dimensions gives that 5 dimensions are *correlated* and 4 are *low-correlated* (Table 4.4).

The goal is now to determine which dimensions actually bring information to which CPU traces of the data set. In the left half of Table 4.5, we report the results of the server characterization on the data set X_t. The columns D_t, R_t and N_t denote the three categories of our methodology at time interval t. Since it is important to validate the proposed approach, we compare its results against those obtained by applying to each data stream the traditional mechanisms (that we name *Baseline characterization*) requiring statistical methods for the analysis of server behavior, including the correlation among values [127] and probability

	X_t				X_{t+1}			
	Baseline characterization	**Methodology**			Baseline characterization	**Methodology**		
CPU	Behavior	D_t	R_t	N_t	Behavior	D_{t+1}	R_{t+1}	N_{t+1}
1	Random	0.2211	**0.2379**	0.0181	Random	0.1155	**0.2592**	0.0481
2	Negligible	0.0006	0.0009	**0.1297**	Negligible	0.0008	0.0012	**0.1373**
3	Negligible	0.0443	0.0334	**0.1866**	Negligible	0.0537	0.0454	**0.1936**
4	Random	0.0391	**0.3012**	0.1185	Random	0.0409	**0.3120**	0.1115
5	Negligible	0.0186	0.0116	**0.1955**	Negligible	0.0190	0.0163	**0.2051**
6	Negligible	0.0162	0.0128	**0.1938**	Negligible	0.0183	0.0141	**0.2182**
7	Random	0.0615	**0.2379**	0.0181	Random	0.0753	**0.2622**	0.0201
8	Negligible	0.0492	0.0370	**0.2030**	Negligible	0.0505	0.0401	**0.2112**
9	Random	0.1160	**0.2206**	0.1236	Random	0.1724	**0.2310**	0.1311
10	Negligible	0.0005	0.0005	**0.1309**	Negligible	0.0008	0.0018	**0.1413**
11	Deterministic	**0.3539**	0.1612	0.0540	Random	0.2073	**0.2648**	0.0208
12	Negligible	0.0099	0.0049	**0.1960**	Negligible	0.0133	0.0054	**0.2006**
13	Negligible	0.0139	0.0122	**0.2052**	Negligible	0.0105	0.0166	**0.2922**
14	Deterministic	**0.3158**	0.2091	0.0150	Deterministic	**0.3801**	0.1696	0.0117
15	Deterministic	**0.2302**	0.0792	0.0860	Random	0.0434	**0.2309**	0.0345
16	Deterministic	**0.3057**	0.2396	0.0176	Deterministic	**0.3113**	0.2601	0.0158
17	Deterministic	**0.3368**	0.1016	0.0402	Random	0.1556	**0.2635**	0.0198
18	Random	0.0357	**0.3350**	0.0712	Random	0.0370	**0.3048**	0.0525
19	Random	0.0685	**0.2085**	0.0071	Random	0.0484	**0.2391**	0.0111
20	Deterministic	**0.2755**	0.1612	0.0251	Deterministic	**0.2444**	0.1220	0.0288
21	Random	0.1170	**0.2858**	0.0239	Random	0.1333	**0.2510**	0.0244
22	Random	0.0837	**0.3412**	0.0213	Random	0.0745	**0.3287**	0.0255
23	Deterministic	**0.3955**	0.0846	0.0927	Deterministic	**0.3446**	0.0821	0.1018
24	Deterministic	**0.3163**	0.3154	0.0374	Deterministic	**0.3131**	0.3987	0.0345
25	Negligible	0.0002	0.0006	**0.1037**	Negligible	0.0012	0.0005	**0.1198**
26	Negligible	0.0569	0.0402	**0.1912**	Negligible	0.0722	0.0550	**0.1624**
27	Negligible	0.0071	0.0186	**0.2028**	Negligible	0.0113	0.0225	**0.2888**
28	Negligible	0.0220	0.0268	**0.2112**	Negligible	0.0245	0.0211	**0.2009**
29	Random	0.0432	**0.1261**	0.1127	Random	0.0669	**0.1174**	0.0990
30	Random	0.0207	**0.2498**	0.0137	Random	0.0193	**0.2330**	0.0115
31	Deterministic	**0.3955**	0.0846	0.0927	Deterministic	**0.3566**	0.0866	0.1217
32	Deterministic	**0.4183**	0.2324	0.0277	Deterministic	**0.4033**	0.2431	0.0071
33	Deterministic	**0.3037**	0.2695	0.0293	Deterministic	**0.3169**	0.2244	0.0331
34	Random	0.2114	**0.2388**	0.0166	Random	0.2070	**0.2388**	0.0166
35	Negligible	0.0478	0.0410	**0.1858**	Negligible	0.0552	0.0670	**0.2004**
36	Negligible	0.0387	0.0353	**0.2114**	Negligible	0.0307	0.0564	**0.2145**
37	Random	0.0275	**0.3535**	0.0616	Random	0.0222	**0.3114**	0.0601
38	Negligible	0.0236	0.0842	**0.1214**	Negligible	0.0215	0.0667	**0.1552**
39	Random	0.0611	**0.2059**	0.0287	Random	0.0021	**0.2003**	0.0224
40	Deterministic	**0.3810**	0.2160	0.0421	Deterministic	**0.3911**	0.2022	0.0126
41	Deterministic	**0.2462**	0.0784	0.0815	Deterministic	**0.2132**	0.0663	0.0115
42	Deterministic	**0.3902**	0.0892	0.0960	Deterministic	**0.3888**	0.0531	0.0770
43	Negligible	0.0003	0.0008	**0.1094**	Negligible	0.0005	0.0011	**0.1677**
44	Negligible	0.0039	0.0100	**0.1755**	Negligible	0.0023	0.0332	**0.1810**
45	Deterministic	**0.3879**	0.1695	0.0458	Deterministic	**0.3552**	0.1565	0.0221
46	Deterministic	**0.3002**	0.0716	0.0866	Deterministic	**0.3441**	0.0651	0.0733
47	Negligible	0.0090	0.0198	**0.1659**	Negligible	0.0110	0.0153	**0.1977**
48	Negligible	0.0221	0.0237	**0.2081**	Negligible	0.0319	0.0451	**0.2231**
49	Negligible	0.0006	0.0003	**0.1388**	Negligible	0.0007	0.0004	**0.1874**
50	Deterministic	**0.2032**	0.0898	0.0844	Deterministic	**0.2243**	0.0711	0.0491

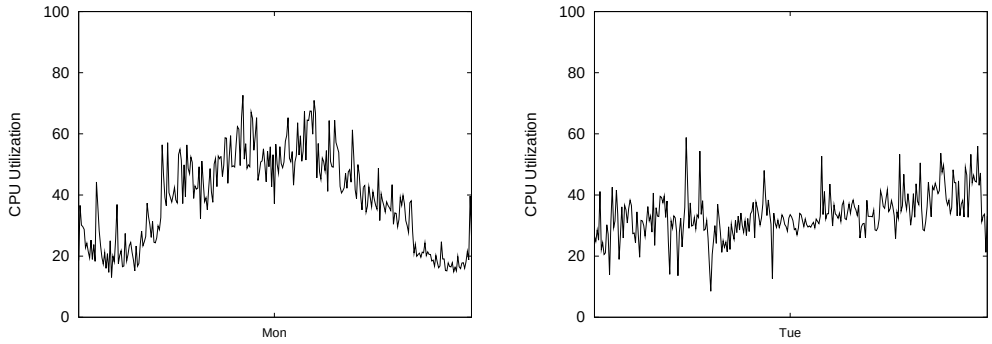
Table 4.5: Server characterization and behavioral changes.

distributions [128], and mean load analysis [129]. Due to the high variability of system resource measures, all these analyses must provide a pre-filtering step to remove noise from data [59]. The classification results achieved by the Baseline characterization are reported in the second column. In the same table, we report the results obtained through the proposed approach in the three *Methodology* columns. These columns highlight in bold text the M_t value for each server. This value denotes the behavioral characterization that is assigned by the proposed methodology. For example, baseline analyses at time interval t of server 3 compute a mean load utilization of 6.39% that sets it as a negligible server. Our methodology reaches the same characterization through the computation of only three behavioral values: a predominant $N_{t,3}$ value of 0.1866 characterizes the server as under-loaded, avoiding the analysis of the entire data stream.

If we compare the classes with bold indexes to the server characterization arising from the Baseline characterization reported in the second column, we see a perfect correspondence of results. Indeed, our methodology is able to identify the main statistical behavior of 50 servers through a classification and a statistical analysis of only the most relevant 9 dimensions of the data set. By working on dimensions that capture the common patterns of data variation and isolate random perturbations, our methodology avoids to apply any filtering techniques on the small set of dimensions.

We apply the proposed methodology on the second data set X_{t+1} in order to obtain the system view of the next day. During the night, a new business application connected to a database was installed and caused a relevant change in server activities. The results are shown in the right half of Table 4.5. By comparing the characterization results of X_t and X_{t+1}, we are able to evidence the behavioral changes of servers. In particular, the new application modifies the activity of servers 11, 15 and 17 from deterministic to random, as denoted by their prevailing D_t and R_{t+1} values. In such a way, we are able to evaluate the behavioral implications of the new application in the activity of servers due to the effect of non-determinism incurred by the new requests. A more detailed analysis is reported in Figure 4.24. Figure 4.24(a) shows the CPU utilization trace of the server 11 before the introduction of the new application. An evident deterministic behavior follows the diurnal activity of users. With the insertion of the new application, server 11 changes its behavior showing random dips and bursts of CPU utilization as in Figure 4.24(b).

Changes in server behaviors guide system operators to adapt the system to changing environments. Besides that, the identification of changes in the behavior of some servers does not imply a change in overall system functioning: the transition of a handful of servers from the deterministic to the random class does not mean a necessary change in the overall system operations.



(a) Time interval t (b) Time interval $t + 1$

Figure 4.24: Behavioral change of server 11.

Computational complexity of the methodology

In this section, we evaluate the computational complexity of the proposed methodology in order to assess its feasibility to Internet systems dimensionality. We compare the number of computations required by our proposal for characterizing server behavior to that needed by traditional approaches working on the all set of monitored data. The two computational complexities are reported in Table 4.6 (we remind n as the number of samples, p the number of considered resource measures and r the number of relevant dimensions.)

Proposed strategy		**Analysis of each data set**	
Major processing steps	Computational complexity	Major processing steps	Computational complexity
PCA analysis of a nxp matrix	$\mathcal{O}(p^3 + n * p^2)$	Filtering of p sets of length n	$\mathcal{O}(p * nlog(n))$
Behavioral analysis of r sets	$\mathcal{O}(r * n^2)$	Behavioral analysis of p sets	$\mathcal{O}(p * n^2)$
Total	$\mathcal{O}(p^3 + n * p^2 + r * n^2)$	Total	$\mathcal{O}(p * n^2)$

Table 4.6: Computational complexity.

The PCA requires around $\mathcal{O}(p^3 + n * p^2)$ computations [29] for the eigenvalue decomposition of a covariance matrix, while the analysis of the statistical attributes of a stream may provide a number of operations spanning from logarithmic to exponential in the number of samples. In this evaluation, we consider the application of behavioral analyses having at most a quadratic computational cost. Hence, the total computational complexity of the proposed methodology is $\mathcal{O}(p^3 +$

$n*p^2 + r*n^2$). On the other side, the baseline characterization needs a pre-filtering step on all monitored data before applying behavioral analyses. Also for filtering techniques, there are a lot of models with different properties and complexities. In order to provide reliable characterizations, we decide to apply Fast Fourier Transform [59] for filtering data by making $\mathcal{O}(p * nlog(n))$ computations. Then, quadratic (or less) behavioral analyses are applied to all filtered data. Hence, the total computational complexity of the baseline approach is $\mathcal{O}(p*nlog(n) + p*n^2)$ $\approx \mathcal{O}(p * n^2)$.

In Figure 4.25 we show the tendencies of the two computational complexities, when the two approaches characterize different number of streams of fixed length $n = 1440$. By varying the dimensionality p of the data set up to one thousand resource measures, the computational complexity of the proposed methodology (line with filled circles in the figure) maintains always lower than that of the baseline characterization (line with squares). The dimensionality reduction lowers the number of data streams to analyze, thus lightening the expense of server characterization.

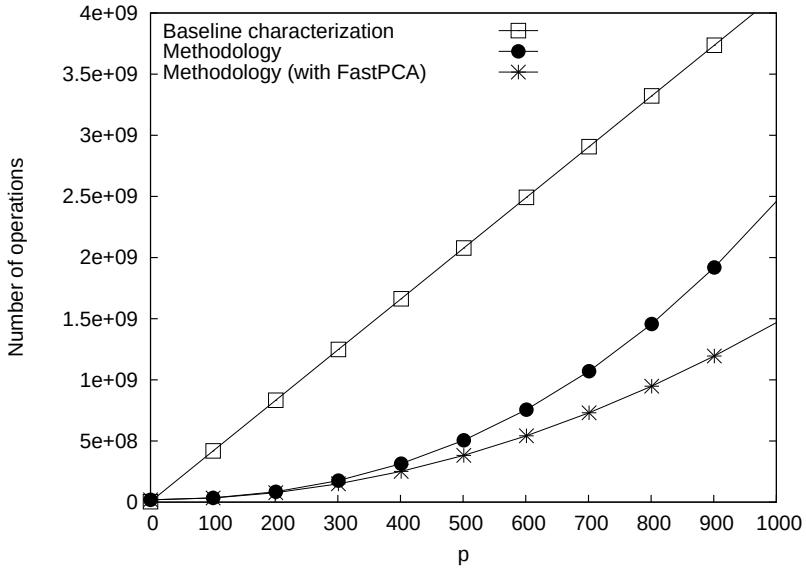


Figure 4.25: Analysis of the computational complexity.

We expect that, for higher p values, there is a size of the data set after which the baseline characterization is more efficient than the proposal. However, it is mandatory to consider that, in large Internet systems, it is necessary to work hierarchically on subsets of interacting components, typically comprising hundreds of servers hosting tens of virtual machines. For proper system design and management, it is basilar to select and work on significant parts of the Internet system,

that is on the whole so huge that all measures are useless. Moreover, the performance of the proposed methodology can be improved by the application of less consuming implementations of the Principal Component Analysis. For example, FastPCA [30] can lower the dimensionality reduction complexity to $\mathcal{O}(p^2)$ and its adoption inside our methodology brings to a computational complexity that follows the curve with star points in Figure 4.25.

Chapter 5

Correlation analysis

Correlation models are applied to various scientific fields as bases for several statistical analyses, such as forecasting [32], state aggregation [33], anomaly [34, 35] and event detection [70]. For computer systems management, capturing the correlation between time series monitored in a system allows us to discover groups of resources with similar behavior, to reflect changing relationships and, as a global effect, to manage the system in a more efficient way.

There are several correlation models showing robust and accurate results when applied to low variable domains. However, they fail in finding correlation between time series collected from system monitors, where relationships between data are often hidden by high variability. In these contexts, the most popular models, such as the Pearson coefficient [68], the Spearman rank [69], the Kendall rank [39], and the Local Correlation index [70], are unable to capture correlations even when they exist. The approach of filtering highly variable time series and then applying correlation models does not solve the problem and opens other issues.

We propose a new correlation model that is able to capture both linear and non-linear dependences even among time series characterized by high variability. The accuracy and robustness of the proposed solution is achieved through an original approach that separates trend patterns from perturbation patterns, and evaluates correlation by computing the similarity of trend patterns because they reveal the way in which a time series may depend on another one. From this perspective, the idea we pursue is based on the following steps:

1. we extract from time series the information about their trend patterns by removing the perturbations that mask the presence of possible relationships between data;

2. we measure how close the extracted trend patterns of two time series are, thus capturing the presence of linear dependency and of non-linear dependency.

The proposed method represents a step ahead of the literature because the Pearson correlation moment does not cover non-linear dependency [70], the Spearman and Kendall ranks are conditioned by the data distributions [68]. The LoCo model would be able to capture some non-linear relationships, but it infers that the main patterns of a time series are only trend patterns without considering the impact of perturbations. However, system and software resources are characterized by several perturbations due to I/O operations, synchronizations, context switching. We extend the correlation analysis and its applications to domains that are characterized by perturbations and highly variable time series [11, 40]. Time series related to system resources may have linear and non-linear relationships. For example, in Internet services, network traffic volume and service times change in accordance with the volume of user requests [41]. These relationships are typically masked by perturbations intrinsically related to the nature of the applications, but when correlations exist our model is able to identify them.

We illustrate the proposed method applied to real and synthetic time series. We discuss its quantitative and qualitative interpretation, compare it against existing solutions and demonstrate its accuracy and robustness. Moreover, we evaluate that the proposed correlation model is robust and accurate even as a support to different applications, such as tracking analysis, anomaly detection, and prediction analysis.

5.1 Problem definition

We consider data obtained from system and software monitors through the periodic sampling of resource measures. As evidenced in many works [11, 40, 42], these measures are extremely variable even at different time scales. We transform these samples in time series. We denote the two considered time series as $\mathbf{x} \equiv [x_1, \ldots, x_n]$ and $\mathbf{y} \equiv [y_1, \ldots, y_n]$, where each one is a vector collecting a time-ordered discrete sequence of data points that can be sampled once.

Existing correlation models do not work on time series exhibiting a high degree of variability, hence we are interested to propose a new *correlation index*, typically denoted as ρ, measuring the similarity between **x** and **y**, as in [76]. The absolute value of the correlation index ranges between 0 and 1. When $\rho = 0$, there is no relationship between the two time series, while $\rho = 1$ indicates a complete correlation between **x** and **y**. The literature offers several guidelines for the best interpretation of the value of the correlation index [38, 68, 76], but all criteria depend on the context and purposes of the analysis. Providing rules for interpreting the meaning of correlation indexes is out of the scope of this chapter. In our context, we decide to adopt the common interpretation indicating a *strong correlation* when $\rho > 0.5$, and a *weak correlation* for $\rho \leq 0.5$ (e.g., [76]). Different

choices for the threshold may lead to different results but do not impact the main conclusions of this chapter.

Many models for capturing correlation are robust and accurate (e.g., [39, 68–70]) except when they are applied to time series characterized by high variability that is typical of system resource metrics. In accordance with [75], high variability is a phenomenon by which a set of observations takes values that vary over orders of magnitude, with most observations taking values around the time series trend (i.e., *trend pattern*), and some observations departing from it with appreciable frequency, even taking extremely large values with non-negligible probability (i.e., *perturbation pattern*). Trend patterns represent the tendency of a time series that may be related to the other time series, while perturbation patterns consist of random observations hiding trends. A high standard deviation is the most typical trademark of a highly variable time series. This characteristic implies a trend pattern that is hard to identify because it is masked by perturbations. In this chapter, we use standard deviation as a measure of data variability.

The ability of a correlation model in detecting dependency among correlated time series is measured in terms of *accuracy*, while the ability in guaranteeing a stable correlation index when conditions do not change is measured in terms of *robustness*. Accuracy measures how close the correlation index is to the effective level of correlation between two time series, while robustness evaluates the variability of the correlation index among different evaluations carried on under unchanged conditions.

In the case of highly variable time series, the most popular correlation models are affected by two main problems:

1. low accuracy, since they are unable to detect linear and non-linear dependences even among correlated time series;

2. low robustness, since they do not guarantee a stable evaluation of the correlation index, even when the relationships between the time series do not change.

Let us give an example of the above problems. We refer to datasets coming from the resource monitoring of a multi-tier system, whose architecture is illustrated in Figure 5.1. The four application servers are deployed through the Tomcat servlet container and are connected to two MySQL database servers. The Web switch node, running a modified version of the Apache Web server as in [11], assigns the same amount of requests to the two Apache-based HTTP servers, that run identical applications on identical hardware.

In Figure 5.2(a), we report 1050 data of the CPU utilization of the two Web servers sampled every 5 minutes. The data are highly variable, although there is visual evidence of a high correlation between the two time series. At system

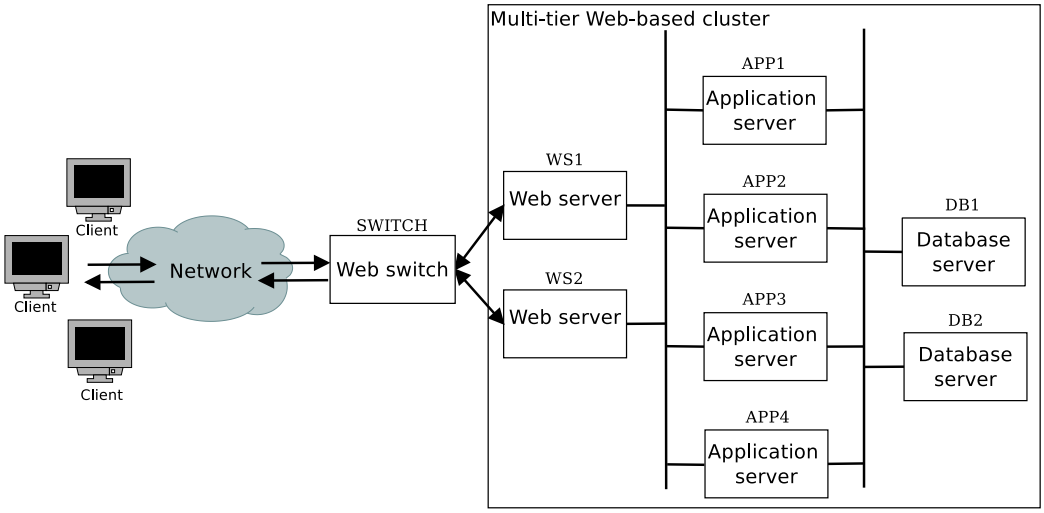
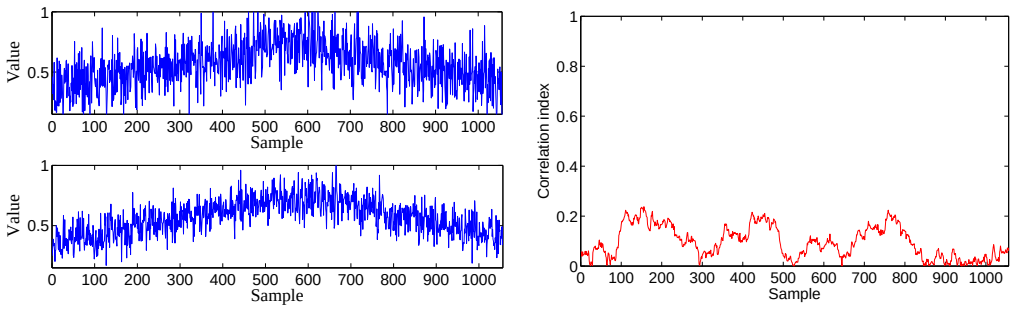


Figure 5.1: Architecture of a multi-tier Web cluster.

level, this correlation is confirmed by the fact that the two hosts receive an analogous amount of requests assigned by the front-end load balancer dispatcher. We evaluate the correlation index between the two time series through the Pearson model [68]. (The Pearson model is used as an example, but other existing models do not change the results).

The results of the correlation index are reported in Figure 5.2(b). Despite the clear relationship between the datasets referring to the two hosts, Figure 5.2(b) shows that the Pearson model is affected by the two anticipated problems: its results are characterized by low accuracy because the Pearson correlation index remains lower than 0.25 during the entire interval of observation; its results are characterized by low robustness because the correlation index presents marked oscillations even when the correlation between the times series does not change.

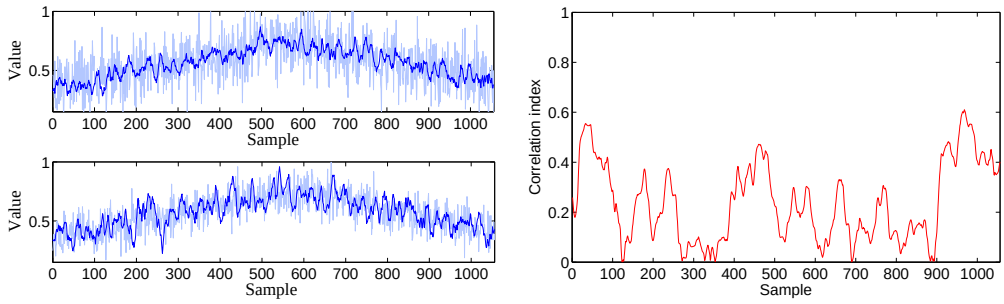


(a) CPU utilization samples (b) Pearson correlation index

Figure 5.2: Result of correlation models applied to highly variable data.

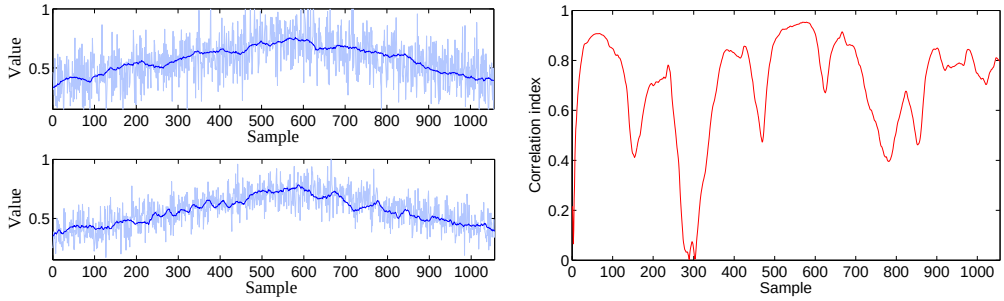
A typical approach addressing issues related to highly variable time series is to refer to some rectification algorithms (e.g., [45]). Unfortunately, we will see that filtering data and then applying an existing correlation model does not work and, even worse, opens other issues by moving the problem to the dimension of finding the "right" filter and its parameters. This chapter cannot discuss all details of the extensively investigated and hard-to-solve problems related to filters. Interested readers can refer to the huge literature on this field (e.g., [11, 45, 146]). Basically, we have two alternatives that are equally useless: to apply a weak or a strong filter. A weak filter removes small variability, hence underlying relationships among time series remain undetectable by existing correlation models; a strong filter may remove important information about trend patterns and thus prevent the possibility of finding correlations.

As example, let us consider again the time series represented in Figure 5.2(a). We choose a popular filter such as the Exponential Weighted Moving Average (EWMA) [45] as a basis, and apply different parameters in order to obtain a weak and a strong filter. The filtered time series are shown in Figure 5.3(a) and Figure 5.4(a), respectively.



(a) Weak filtered CPU utilization (b) Pearson correlation index

Figure 5.3: Weak filtering applied before correlation analysis.



(a) Strong filtered CPU utilization (b) Pearson correlation index

Figure 5.4: Strong filtering applied before correlation analysis.

In Figure 5.3(b), we report the results of the Pearson model applied to the weakly filtered data. As expected, the results are improved with respect to the not filtered case, but the conclusion remains unchanged: the two time series are considered uncorrelated because the correlation index remains lower than 0.5 for almost the entire interval of observation. By increasing the filter strength, most perturbations are discarded, as shown in Figure 5.4(b). The problem is that a strong filter cancels also information about trend patterns characterizing the time series. The final result is that the correlation index becomes even less robust as it continuously passes from evaluating the time series as correlated, then uncorrelated, correlated again, and so on.

All these results evidence that high variability represents a limit of existing correlation models even when they are integrated with filtering techniques. Detecting correlations among highly variable data as those obtained by resource monitoring requires some novel approach. The model proposed in the following section has several benefits: it does not require any assumption about statistical properties, any pre-analysis of time series characteristics, and any integration with pre-filtering techniques; it is able to adapt its parameters to data characteristics and to capture linear and non-linear dependences.

5.2 Correlation model for highly variable data

In this section, we present a novel correlation model, namely *CoHiVa* (*Co*rrelation for *Hi*ghly *Va*riable data), that is able to evaluate similarity between time series that are characterized by high variability. This model may be viewed as an improvement of the LoCo algorithm [70] that proposes the evaluation of correlation through the analysis of pattern similarity. CoHiVa extends this idea to the correlation analysis in highly variable domains. Unlike the Pearson model [68] that works well only if time series are linked by a linear relationship, CoHiVa does not assume any data dependency and it is appropriate to discover both linear and non-linear dependencies. Moreover, CoHiVa does not assume any data distribution, as required by the Spearman and Kendall models [39, 69].

The CoHiVa algorithm is based on the following four main steps:

1. we extract from **x** and **y** the trend patterns and the perturbation patterns;
2. we remove errors contaminating the time series;
3. we select the trend patterns by discarding the perturbation patterns containing highly variable information;
4. we compute the CoHiVa correlation index between the two time series by evaluating the similarity between their trend patterns.

Each step is detailed in the following subsections.

5.2.1 Pattern extraction

The first goal is to identify the main patterns that are present in the time series $\mathbf{x} \equiv [x_1, \ldots, x_n]$, where patterns correspond to trends (i.e., periodic and seasonal components) and perturbations. To this end, we apply the Singular Value Decomposition (SVD) [70] to the auto-covariance matrix of the time series. Among the spectral decomposition algorithms, SVD is considered as the baseline technique for separating existing patterns without any assumption about the statistical characteristics of the data [79, 131]. In practice, we estimate the full auto-covariance matrix of the time series $\mathbf{x}$ that is defined as:

$$\mathbf{\Phi}(x) = \mathbf{x} \otimes \mathbf{x}, \tag{5.1}$$

where $\mathbf{\Phi}(x)$ is the auto-covariance matrix of $\mathbf{x}$.

Then, we compute the SVD of the auto-covariance matrix $\mathbf{\Phi}(x)$ as follows:

$$\mathbf{\Phi}(x) = \mathbf{U}(x)\mathbf{\Sigma}(x)\mathbf{V}(x)^T, \tag{5.2}$$

where $\mathbf{U}(x)$, $\mathbf{\Sigma}(x)$ and $\mathbf{V}(x) \in \mathbb{R}^{n \times n}$.

The columns $\mathbf{v}_i$ of $\mathbf{V}(x) \equiv [\mathbf{v}_1, \ldots, \mathbf{v}_n]$ are the right singular vectors of $\mathbf{\Phi}(x)$. Similarly, the columns $\mathbf{u}_i$ of $\mathbf{U}(x) \equiv [\mathbf{u}_1, \ldots, \mathbf{u}_n]$ are the left singular vectors of $\mathbf{\Phi}(x)$. Finally, $\mathbf{\Sigma}(x) \equiv \text{diag}[s_1, \ldots, s_n]$ is a diagonal matrix with positive values s_i, called the singular values of $\mathbf{\Phi}(x)$.

5.2.2 Removing errors

The singular vectors corresponding to singular values often contain some approximation errors [50] that usually contaminate the measured variables. The contribution of these errors must be discarded by eliminating the singular vectors corresponding to the smallest singular values [81]. By retaining just the principal vectors corresponding to the highest k singular values $(k < n)$ we can reconstruct a *k-dimensional approximation* of the correlation matrix:

$$\bar{\mathbf{\Phi}}(x) \equiv \bar{\mathbf{U}}(x)\bar{\mathbf{\Sigma}}(x)\bar{\mathbf{V}}(x)^T, \tag{5.3}$$

where $\bar{\mathbf{U}}(x) \equiv [\bar{\mathbf{u}}_1, \ldots, \bar{\mathbf{u}}_k]$, $\bar{\mathbf{V}}(x) \equiv [\bar{\mathbf{v}}_1, \ldots, \bar{\mathbf{v}}_k]$ and $\bar{\mathbf{\Sigma}}(x) \equiv \text{diag}\,[\bar{s}_1, \ldots, \bar{s}_k]$.

Literature on SVD gives little importance to the problem of dynamically selecting the appropriate number of principal vectors that capture the patterns (e.g., [70, 131]). A common approach is to choose a fixed number of principal vectors independently of data characteristics, but this choice is unsuitable to time-varying contexts where the statistical properties of data continuously change in time. Hence,

we choose a threshold-based method that takes into account the characteristics of considered data [52]. We select the principal vectors contributing to 90% of variation, and discard singular vectors contributing for less than 10%. This number of principal vectors capturing the main patterns of the time series is variable and dependent to the amount of variation in data. The variable number of principal vectors capturing the main patterns of the time series is denoted by k.

5.2.3 Selection of trend patterns

We now analyze the main patterns of **x** in order to understand the information they carry. The goal is to retain just trend patterns but, in the context of interest for this chapter, we can expect that the main patterns may include also some perturbation patterns [82]. As these last patterns prevent the identification of correlation between time series, we have to discard them. The idea is to build a new matrix that is based on a subspace of the $\hat{k} \leq k$ principal vectors that capture trend patterns.

To remove perturbation patterns from $\bar{\mathbf{U}}(x)$, we compute the Hurst exponent H of the k principal vectors by means of the R/S analysis of Hurst [83]. The Hurst exponent measures whether the data have pure random variability or are characterized even by some underlying trends [84]. For each principal vector $\bar{\mathbf{u}}_i \in \bar{\mathbf{U}}(x)$ of length n, we first define its cumulative deviate series:

$$Z_t = \sum_{j=1}^{t} (\bar{u}_j - m), \tag{5.4}$$

where $t = 1, 2, \ldots, n$, $\bar{u}_j$ is the $j-$th element of the $i-$th principal vector and m is the mean of $\bar{\mathbf{u}}_i$.

To define the rescaled range $\frac{R(n)}{S(n)}$ of Hurst, we compute the range as:

$$R(n) = \max\left(Z_1, Z_2, \ldots, Z_n\right) - \min\left(Z_1, Z_2, \ldots, Z_n\right), \tag{5.5}$$

and the standard deviation as:

$$S(n) = \sqrt{\frac{1}{n} \sum_{j=1}^{n} \left(\bar{u}_j - m\right)^2}. \tag{5.6}$$

The *Hurst exponent* H is defined in terms of the asymptotic behavior of the rescaled range as a function of the time span of a time series as follows [83]:

$$\mathrm{E}\left[\frac{R(n)}{S(n)}\right] = Cn^H \quad \text{as } n \to \infty, \tag{5.7}$$

where $E[\frac{R(n)}{S(n)}]$ is the expected value of the rescaled range, n is the number of observations in a time series, and C is a constant.

If the estimated Hurst exponent H_i of a principal vector $\bar{\mathbf{u}}_i \in \bar{\mathbf{U}}(x)$ is close to 0.5, then we can conclude that $\bar{\mathbf{u}}_i$ contains perturbations. On the other hand, if the Hurst exponent remains far from 0.5, then we can assume that $\bar{\mathbf{u}}_i$ is a trend principal vector.

Our model builds up a new $\hat{\mathbf{U}}(x)$ matrix containing only the $\hat{k}$ trend principal vectors $\hat{\mathbf{u}}_j$, $j = 1, \ldots, \hat{k}$ as follows: $\forall \bar{\mathbf{u}}_i \in \bar{\mathbf{U}}(x)$, $i = 1, \ldots, k$:

$$\hat{\mathbf{u}}_j \equiv \bar{\mathbf{u}}_i \quad \text{if} \quad H_i < 0.5 - \frac{\delta}{2} \quad \text{or} \quad H_i > 0.5 + \frac{\delta}{2}, \tag{5.8}$$

where δ is a two-sided 95% confidence interval empirically computed depending on the number of samples as in [84].

This separation approach allows us to remove perturbation patterns in the time series and to focus only on trend patterns. By focusing on the $\hat{k} \ll n$ trend patterns, we are able to construct a new approximation of the $\mathbf{\Phi}(x)$ matrix that we name *trend approximation*. Given $\hat{\mathbf{U}}(x) \equiv [\hat{\mathbf{u}}_1, \ldots, \hat{\mathbf{u}}_{\hat{k}}]$, the corresponding singular values and the right singular vectors form the matrices $\hat{\mathbf{\Sigma}}(x) \equiv \text{diag}\,[\hat{s}_1, \ldots, \hat{s}_{\hat{k}}]$ and $\hat{\mathbf{V}}(x) \equiv [\hat{\mathbf{v}}_1, \ldots, \hat{\mathbf{v}}_{\hat{k}}]$, respectively. Through these matrices, the trend approximation of the correlation matrix using only trend patterns is given by:

$$\hat{\mathbf{\Phi}}(x) \equiv \hat{\mathbf{U}}(x)\hat{\mathbf{\Sigma}}(x)\hat{\mathbf{V}}(x)^T. \tag{5.9}$$

The matrix $\hat{\mathbf{\Phi}}(x)$ approximates the trend behavior of $\mathbf{\Phi}(x)$ by removing both error information and perturbation patterns that affect the identification of correlations. This approach retains trend information, as well as seasonal and oscillatory behavior in the time series.

5.2.4 Computation of the CoHiVa index

After the extraction of the main trends, we can evaluate whether they are correlated or not by computing how close their trend patterns are. As example, we compute the correlation index of the time series **x** and **y** by measuring their trend similarity. When the matrices $\hat{\boldsymbol{U}}(x)$ and $\hat{\boldsymbol{U}}(y)$ are similar, the time series **x** and **y** follow similar (linear or non-linear) trends, and we can consider that the original time series are correlated. In geometric terms, if two time series are correlated, then the trend principal vectors of each time series should lie within the subspace spanned by the trend principal vectors of the other time series.

For this reason, we project the trend principal vectors of the time series **x** into the trend principal vectors of **y**, as following:

$$\rho(\mathbf{x}, \mathbf{y}) = \frac{1}{\hat{k}(x) + \hat{k}(y)}(\|\hat{\mathbf{U}}(x)^T \hat{\mathbf{U}}(y)\| + \|\hat{\mathbf{U}}(y)^T \hat{\mathbf{U}}(x)\|), \tag{5.10}$$

where $\hat{k}(x)$ and $\hat{k}(y)$ are the numbers of trend principal vectors of $\mathbf{\Phi}(x)$ and $\mathbf{\Phi}(y)$, respectively, while $\hat{\mathbf{U}}(x)$ and $\hat{\mathbf{U}}(y)$ are the trend principal vectors matrices collecting them.

5.3 Performance evaluation

We evaluate the performance of the proposed correlation model, and we compare it against the results of several state-of-the-art alternatives. As terms of comparison, we consider the following correlation models: the Pearson product moment (Pearson) [68], the Spearman rank (Spearman) [69], the Kendall rank (Kendall) [39], and the Local Correlation index (LoCo) [70]. Moreover, we consider the performance of a model that is integrated with a pre-filtering technique, namely Pearson with filtering. To evaluate the accuracy and robustness of all models, we initially refer to synthetic time series that allow us to have full control over their actual degree of correlation. Then, in the following section we will consider real time series.

In Section 5.3.1, we first evaluate the impact of linear and non-linear correlations on the models performance. Then, in Section 5.3.2 we evaluate how the performance of the correlation models changes in case of time series affected by different levels of variability.

5.3.1 Linear and non-linear correlations

Here, we consider three types of time series: correlated with linear dependence, correlated with non-linear dependence, not correlated. The time series of each scenario take values in the range $[0, 1]$. In order to evaluate the ability of the correlation models to capture different types of dependency for different levels of variability, we introduce perturbations from $N(0, \sigma)$, where $\sigma \in \{0.01, 0.05, 0.1, \ldots, 0.5\}$ is the standard deviation that quantifies the intensity of perturbations added to data [166, 167]. The performance of the correlation models is evaluated in terms of accuracy and robustness over 1000 independent generations of data for each σ value in each scenario.

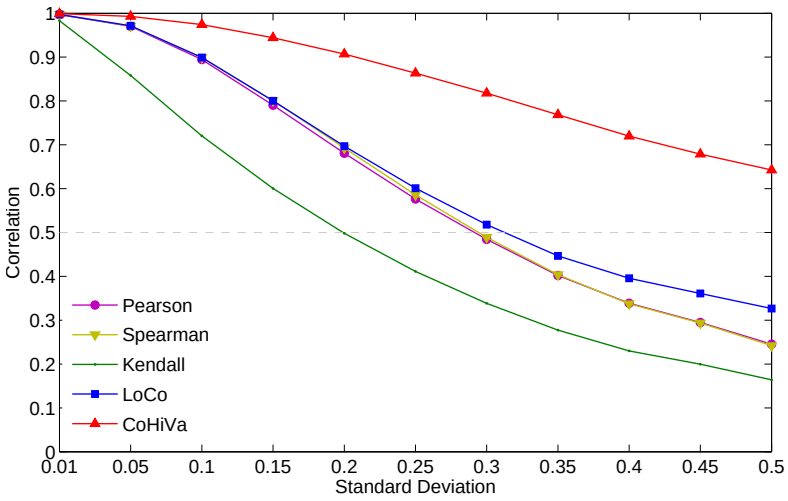
Accuracy

We define the *accuracy* of a correlation model as its ability to capture correlation when data present some linear or non-linear relationships, and in

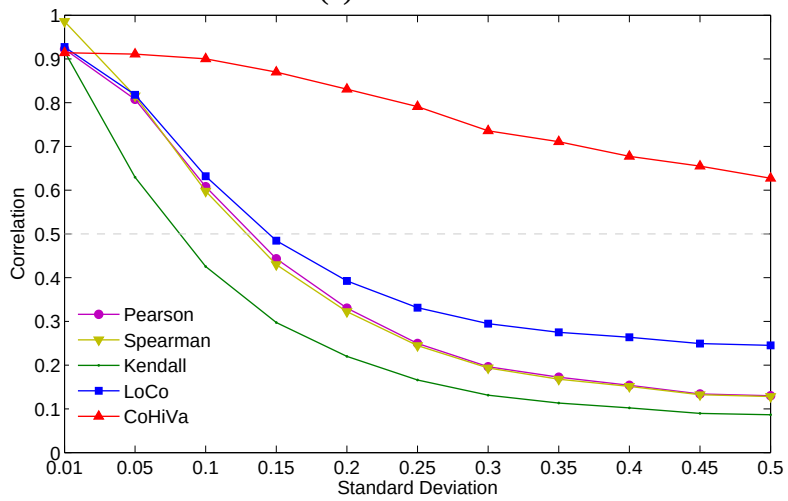
categorizing as not correlated time series having no dependence. For example, an accurate model should obtain a correlation index close to 1 in both linear and non-linear scenarios, and an index close to 0 in the uncorrelated scenario.

The first set of experiments evaluates the accuracy of the correlation models when time series are characterized by different intensities of perturbations. In Figure 5.5, we report the mean correlation index of all considered models computed over 1000 generations of correlated data with different σ values. We remind the reader that we consider a strong correlation when $\rho > 0.5$, and a weak correlation for $\rho \leq 0.5$ [76]. The results of Figure 5.5(a) refer to the linear scenario. As expected, we see a decrease of all correlation indexes for increasing values of σ, but the impact of perturbations is different for the considered models. When the dispersion is low ($\sigma \leq 0.2$), all models are able to capture the strong correlation among data. When the dispersion increases ($\sigma > 0.2$), the Kendall model is the first to lose its ability to detect data correlation. In higher variable contexts ($\sigma > 0.3$), only the CoHiVa model captures the strong data correlation for each variability level, because its index is always higher than 0.65. The accuracy of all the models deteriorates when we pass to a scenario where the correlation between data is non-linear. A comparison between Figure 5.5(a) and Figure 5.5(b) gives a first idea about the overall results. Only the CoHiVa model is able to detect a strong correlation for any σ when the relationship between data is non-linear. On the other hand, all existing models are affected by a low accuracy for increasing values of σ. (They estimate a weak correlation even when data are perturbed by very low levels of dispersion, such as $\sigma = 0.15$.) It is interesting to observe that the Spearman rank, which is expressly oriented to capture non-linear dependencies [69], exhibits the best accuracy when the dispersion is very low (that is, $\sigma < 0.05$), but it loses its capacity as soon as the time series are characterized by higher perturbations.

To address issues related to high variability, the state-of-the-art models may increase their accuracy by working on a filtered representation of the original time series. We anticipated in Section 5.1 that this approach does not work, but for the sake of an exhaustive comparison we compare the performance of CoHiVa against a Pearson model combined with a pre-filtering technique. We have to specify that the choice of the best filtering model and of its parameters is a serious issue by itself, and is out of the scope of this chapter. We integrate the Pearson correlation model with an EWMA filter that we have experimentally evaluated as giving good results. We do not claim that we are applying an optimal filter with optimal parameter setting, even because the definition of optimum is improper in this context.



(a) Linear scenario

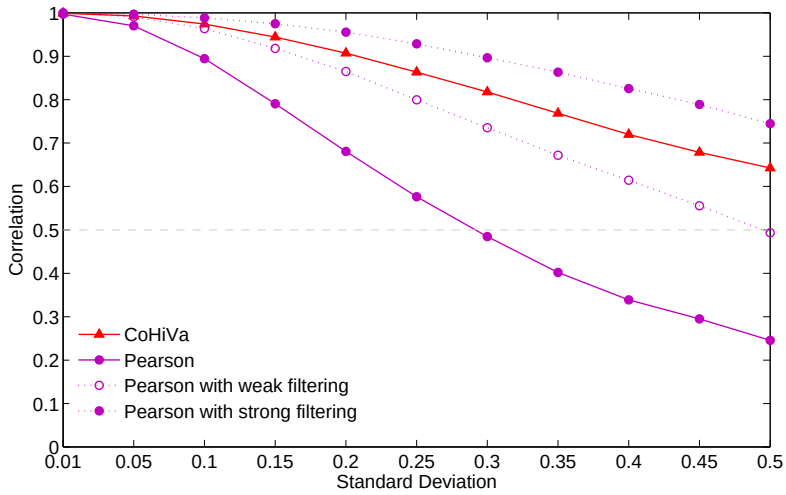


(b) Non-linear scenario

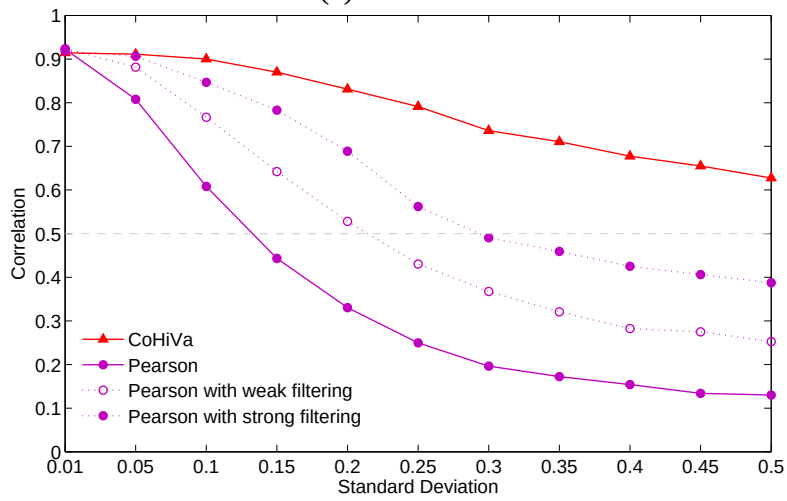
Figure 5.5: Analysis of accuracy of correlation models without data filtering.

Figure 5.6 shows the results obtained by applying the Pearson model to data filtered through a weak and a strong filter. If we compare the results of Pearson without filter to the results of Pearson with any kind of filtering, we can appreciate that the filter in fact improves accuracy: the correlation value is higher for every σ and for linear and non-linear scenarios. In the linear scenario shown in Figure 5.6(a), the Pearson model with filtering is able to detect a correlation index higher than 0.5 for any σ. On the other hand, if we consider the non-linear scenario shown in Figure 5.6(b), there is an evident decrease of the correlation index. Both weak and strong filtering

are useless because they estimate a $\rho \leq 0.5$ when $\sigma > 0.2$ and $\sigma > 0.3$, respectively.



(a) Linear scenario



(b) Non-linear scenario

Figure 5.6: Analysis of accuracy of correlation models with data filtering.

These results demonstrate that filters do not guarantee accurate results, without considering the further problems related to the choice of the filter and its parameters in a highly variable context.

To complete the accuracy evaluation of the models, we report some results obtained in the third scenario characterized by time series with no dependence. Despite the level of variability, we see in Figure 5.7 that all the models are accurate and detect a weak correlation between time series having

no dependence. This result shows that CoHiVa joins the high performance of capturing linear and non-linear correlations to the ability of detecting the absence of correlation.

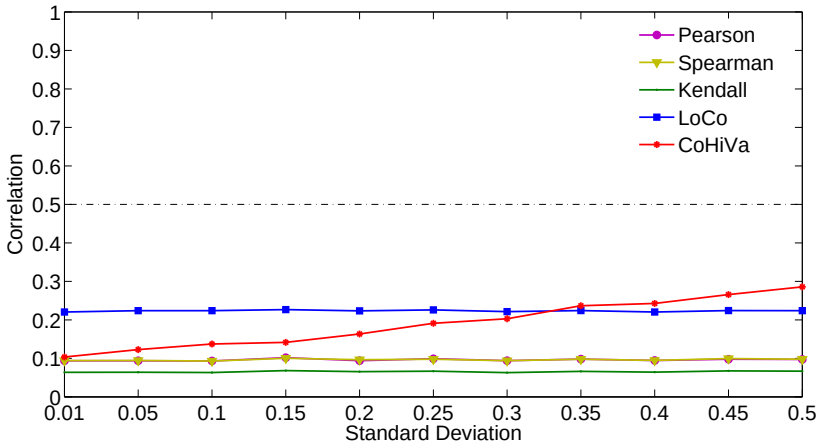


Figure 5.7: Analysis of accuracy in a not correlated scenario.

Robustness

The accuracy of a correlation model must be combined with information about its *robustness*, that assesses the reliability of correlation model results across different evaluations. We quantify the robustness in terms of the *coefficient of variation* (CoV) of the correlation indexes computed over the different data generations. The coefficient of variation is defined as the ratio of the standard deviation to the mean of the correlation index over all the experiments. A lower CoV denotes a better robustness of the correlation model.

We evaluate the robustness of the results obtained in Section 4.2.3. Table 5.1 reports the CoV of each considered correlation model computed over 1000 generations of time series in a linear scenario. The columns refer to the increasing values of perturbations intensity σ, while the rows report the correlation models. The CoV of all correlation models increases when σ increases. Compared to existing models, the CoHiVa model is able to keep the lowest CoV for any σ value. Thanks to a CoV always lower than 0.15, the proposed correlation model guarantees a high robustness in capturing linear correlations also among highly variable data.

As expected, the non-linear context worsens the robustness of all the models. This main conclusion is confirmed by the CoV values reported in Table 5.2.

	σ					
	0.01	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**
Pearson	0.0232	0.0299	0.0914	0.1992	0.3437	0.4817
Spearman	0.0227	0.0304	0.0905	0.2036	0.3496	0.4874
Kendall	0.0371	0.0486	0.1098	0.2170	0.3606	0.4936
LoCo	0.0220	0.0284	0.0835	0.1653	0.2452	0.2735
Pearson with weak filtering	0.0001	0.0069	0.0324	0.0785	0.1206	0.1787
Pearson with strong filtering	0.0001	0.0123	0.0497	0.0917	0.1318	0.1736
CoHiVa	0.0073	0.0086	0.0217	0.0498	0.0888	0.1343

Table 5.1: Coefficient of Variation in the linear scenario.

	σ					
	0.01	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**
Pearson	0.0274	0.1296	0.3704	0.5843	0.6838	0.7052
Spearman	0.0266	0.1457	0.3894	0.5919	0.6892	0.7104
Kendall	0.0391	0.1632	0.4014	0.5997	0.6960	0.7194
LoCo	0.0258	0.1136	0.2550	0.2923	0.3073	0.2924
Pearson with weak filtering	0.0070	0.0835	0.2026	0.2215	0.2224	0.2236
Pearson with strong filtering	0.0083	0.0944	0.2120	0.2468	0.2434	0.2473
CoHiVa	0.0380	0.0172	0.1467	0.1614	0.1711	0.1926

Table 5.2: Coefficient of Variation in the non-linear scenario.

These results demonstrate that only the CoHiVa model is able to guarantee a CoV lower than 0.2 for any perturbation intensity. On the other hand, state-of-the-art models show poor results even for medium-low values of σ ($\sigma \leq 0.2$). With the exception of the LoCo and the Pearson model integrated with filters, all the other correlation models are totally unreliable in highly variable contexts because they reach CoV values around 0.7. These results confirm that they cannot be used to capture non-linear relationships among highly variable time series.

Our analyses confirm that the most popular correlation models without filtering are affected by low accuracy and robustness when data exhibit high variabilities and/or non-linear dependency. Filtering the time series and then applying a

correlation model is not sufficient to solve the problems. The use of the proposed CoHiVa model allows to guarantee good performance for any considered type of correlation and variability in the time series.

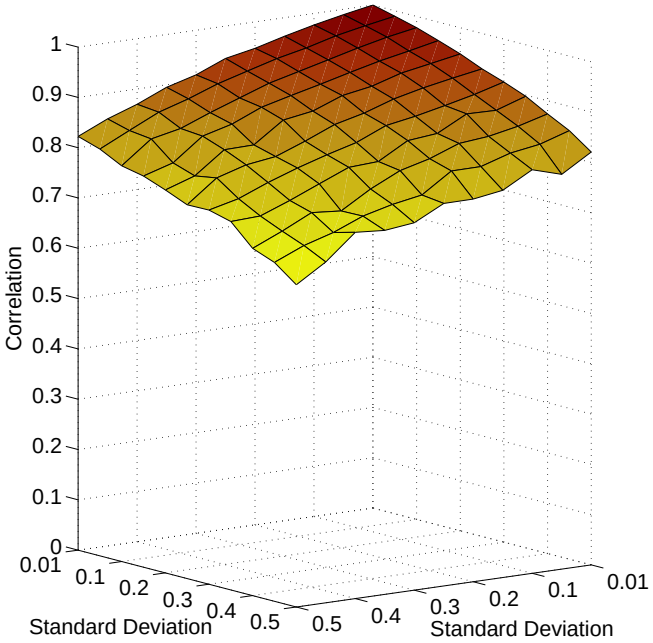
5.3.2 Different variability levels

So far, we considered a variability level σ and simulated two time series both affected by that same level of variability. Now, we consider the case in which the variability level that affects the two time series is different. The goal is to evaluate how the difference in time series variability levels impacts on the models performance. Again, we evaluate the models performance in terms of both accuracy and robustness over 1000 generations of linear correlated data taking values in the range $[0, 1]$. We use this set to test the models over all possible combinations of time series with variability levels in $\{0.01, 0.05, 0.1, \ldots, 0.5\}$.

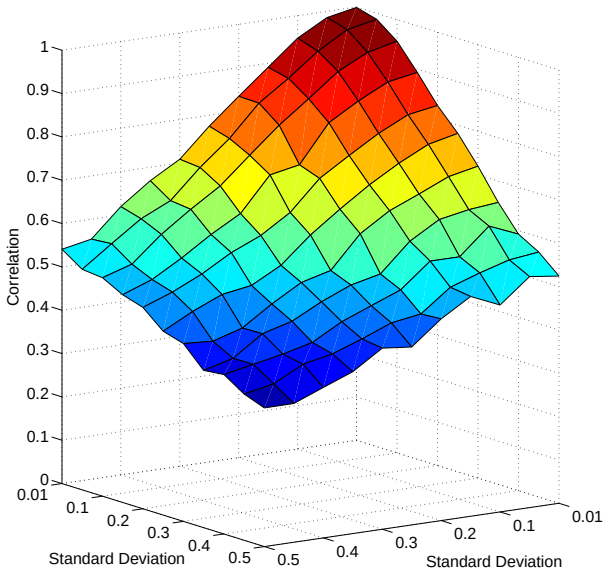
Figure 5.8 shows the accuracy results of the CoHiVa and LoCo correlation models over all the combinations of variabilities in the time series. (We selected LoCo as term of comparison, since it has shown to be our best competitor).

As expected, both indexes have better performance when working on sets where one or both time series have low variability. As the variability of one time series increases, the model performance decreases. Besides that, the increasing of variability has very different impact over the performance of the two models. In Figure 5.8(a), we see that the performance of CoHiVa slightly depend on the different variability levels for the two time series. Despite the fact that a time series has low ($\sigma = 0.01$) or high ($\sigma = 0.5$) variability, the correlation level captured by CoHiVa always maintains high, with an index that never goes below 0.64. On the contrary, to combine different variability levels has evident impact on LoCo performance in Figure 5.8(b). In particular, the increase in variability of one of the two time series strongly decreases the LoCo correlation index. For example, if one time series has $\sigma = 0.1$, LoCo gives very different results in case that the other time series is low variable ($\sigma = 0.01$) or highly variable ($\sigma = 0.5$). In the first case, the LoCo index sees a high correlation between the two time series with $\rho = 0.94$; in the latter case, ρ drops to 0.47 and the two time series are seen as not correlated. This drastic difference (≈ 0.5) in LoCo accuracy results due to the change of variability in only one time series are due to the inability of LoCo to adapt the choice of number of principal components to the characteristics of the time series. Through CoHiVa and its adaptive selection of trend patterns, accuracy is maintained high despite the level of variability of the time series.

To complete the evaluation, Figure 5.9 shows the robustness results of the CoHiVa and LoCo correlation models over all the combinations of variabilities in the time series. Again, changes in the variability levels of the two time series have small effects on the robustness of CoHiVa, as we can see in Figure 5.9(a).

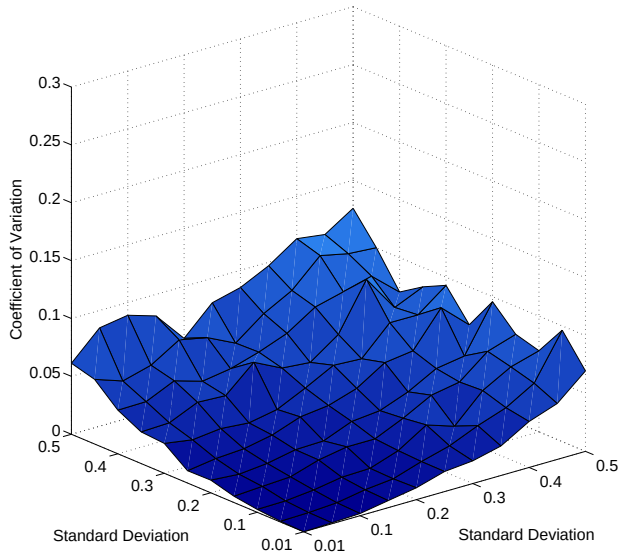


(a) CoHiVa

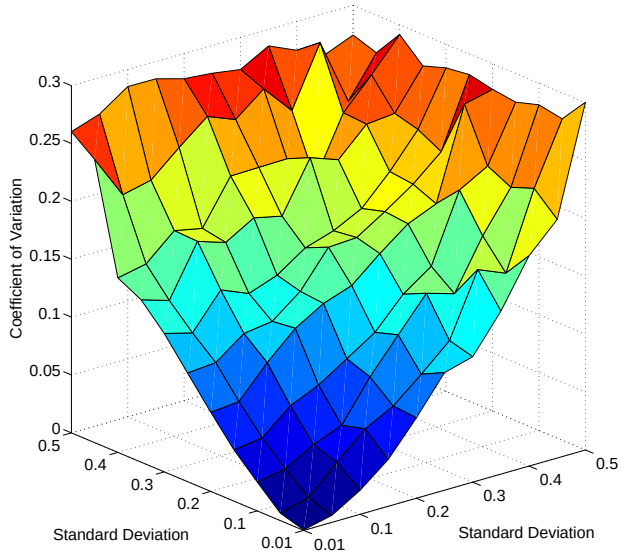


(b) LoCo

Figure 5.8: Analysis of accuracy at different variability levels.



(a) CoHiVa



(b) LoCo

Figure 5.9: Analysis of robustness at different variability levels.

CoV values rise above 0.1 only in case that both time series are affected by high variability levels (i.e., $\sigma \geq 0.4$) and they never exceed 0.12. By comparing Figure 5.9(b) to Figure 5.9(a), we see an evident increase in both values and instability of LoCo CoVs with respect to CoHiVa ones. This means that LoCo robustness, as well as its accuracy, is strongly dependent to the variability of the two time series. For example, if one time series has $\sigma = 0.01$, the LoCo robustness can span from a CoV value of 0 to a CoV value of 0.3 on the basis of the variability level of the other time series.

From these analyses, we see that both different types of correlation (i.e., linear and non-linear) and different levels of variability strongly affect the accuracy and robustness of existing correlation models. The main result is that the CoHiVa model that we propose for correlation is able to guarantee good performance for any considered scenario.

5.4 Complexity

This section estimates the computational complexity and memory space requirements of the considered models: Pearson, Spearman and Kendall, LoCo, CoHiVa, and the possibility of using a filter model.

The majority of these models are characterized by a linear computational complexity as a function of the components n of the time series window.

The Spearman and Kendall ranks, and the Pearson index computing correlation between two time series of n data points requires $\mathcal{O}(n)$ operations. Both them require $\mathcal{O}(n)$ space for storing the time series data.

The LoCo index requires $\mathcal{O}(n^2k)$ operations to compute the k largest eigenvectors of the auto-covariance matrix of a time series of length n. LoCo authors [70] set the parameter k to a small value ($k = 4$) in all their experiments, hence the complexity remains quadratic. For the same reason, the memory requirements can be considered linear, because LoCo needs $\mathcal{O}(nk)$ space for storing k eigenvectors and $\mathcal{O}(n)$ space for storing the time series values, for a total of $\mathcal{O}(nk + n)$ = $\mathcal{O}(nk)$ space.

The CoHiVa model has an approach different from LoCo, because CoHiVa does not set the number of patterns, but it selects k and $\hat{k}$ according to the characteristics of the time series. For this reason, we can distinguish two phases in CoHiVa: startup and operation. The startup phase, during which we evaluate k and $\hat{k}$, requires the SVD decomposition of the auto-covariance matrix that has a complexity in the order of $\mathcal{O}(n^3)$. After this startup phase, we incrementally update the CoHiVa index in a streaming setting. By employing the subspace tracking algorithms as in [58], the update of the trend approximation matrix requires $\mathcal{O}(n\hat{k})$ operations. The space requirement is in the order of $\mathcal{O}(n\hat{k})$. In all our

experiments on real datasets, we observed that typically $\hat{k} \ll n$. Hence, the cost of the operation phase of CoHiVa tends to be linear as a function of the length of the time series in terms of time and space. These values are comparable to the complexity of existing correlation solutions.

As example, Table 5.3 reports the average runtime for the different techniques to compute correlation between linear correlated time series over one of the experiments. The experimental setting used as example is $\sigma = 0.3$ and $n = 100$.

	Pearson	**Spearman**	**Kendall**	**LoCo**	**CoHiVa**	
					startup	**operation**
Time (s)	0.01	0.02	0.04	0.37	0.51	0.06

Table 5.3: Average runtime over an example experiment.

For highly variable time series, we can also consider the possibility of the combined use of filtering data and applying a correlation algorithm. In this case, the overall computational complexity depends on the filtering technique. Some filtering techniques, such as EWMA, are characterized by a linear complexity. Other more efficient filters, such as Fast Fourier Transform [59], require $\mathcal{O}(nlog(n))$ computations. As a consequence, the total cost can span from linear to quadratic and even more. In terms of space requirements, pre-filtering data and then computing correlation requires at least $\mathcal{O}(n^2)$ space for storing the original and the filtered time series.

5.5 System management applications

We evaluate the performance of the CoHiVa model for three types of system management applications that can benefit from an accurate and robust evaluation of correlation: tracking analysis (Section 5.5.1), anomaly detection (Section 5.5.2), and prediction studies (Section 5.5.3).

5.5.1 Tracking analysis

For system management, it is useful to evaluate how the correlation between system resource measures evolves. To this purpose, correlation is evaluated on a

sliding window containing the most recent samples of time series. Several correlation models can be adapted to tracking correlation analysis. We compare the performance of four correlation models: Pearson, LoCo, Pearson integrated with a filter, and CoHiVa. The right size for the sliding window and the progression rule that eliminates older samples and incorporates new ones depend on the application context. For the sake of a fair comparison we choose the same window size and the same progress rule for all considered models.

As a summary of a larger set of experiments, we consider the time series referring to the CPU utilization of the two Web servers in the architecture described in Section 5.1. The results are reported in the four graphs of Figure 5.10. The CoHiVa model (Figure 5.10(a)) maintains a stable index around 0.85 during the entire interval of observation. As the two time series are correlated but characterized by a high variability that tends to mask their correlation, the CoHiVa results confirm that this model is accurate and robust even for tracking analysis. On the other hand, the indexes of Pearson (Figure 5.10(b)) and of LoCo (Figure 5.10(c)) are affected by low accuracy and low robustness. They oscillate continuously and tend to conclude that the two time series are weakly correlated for most of the time, while the opposite is true. The integration of the Pearson model with a filter (Figure 5.10(d)) improves the accuracy of the Pearson results but not their robustness, since the variability of the correlation index remains too high.

5.5.2 Anomaly detection

There are several strategies for anomaly detection [35], but in this chapter we are interested to those founded on tracking correlation analysis (e.g., [34]) that are based on an intuitive idea. If a correlation between two time series exists and, at a certain point, this relationship disappears, we can assume that an anomaly occurs in the observed system. Similar conclusions can be achieved when two unrelated time series become correlated.

A visual exemplification of the transition from a normal to an anomalous behavior is presented in Figure 5.11 and Figure 5.12, respectively. Let us consider the two servers in the architecture receiving the same number of requests through a load balancer.

Figure 5.11(a) shows the network packet rate during two days of normal behavior, and Figure 5.11(b) displays the results of the CoHiVa model (continuous line) and the Pearson model (dotted line) in tracking the correlation between the two time series. This figure reveals that the CoHiVa index is able to capture the correlation between the two metrics despite the high variability perturbing the data. On the other hand, the Pearson index remains lower than 0.5, thus concluding that there is no correlation between the two servers.

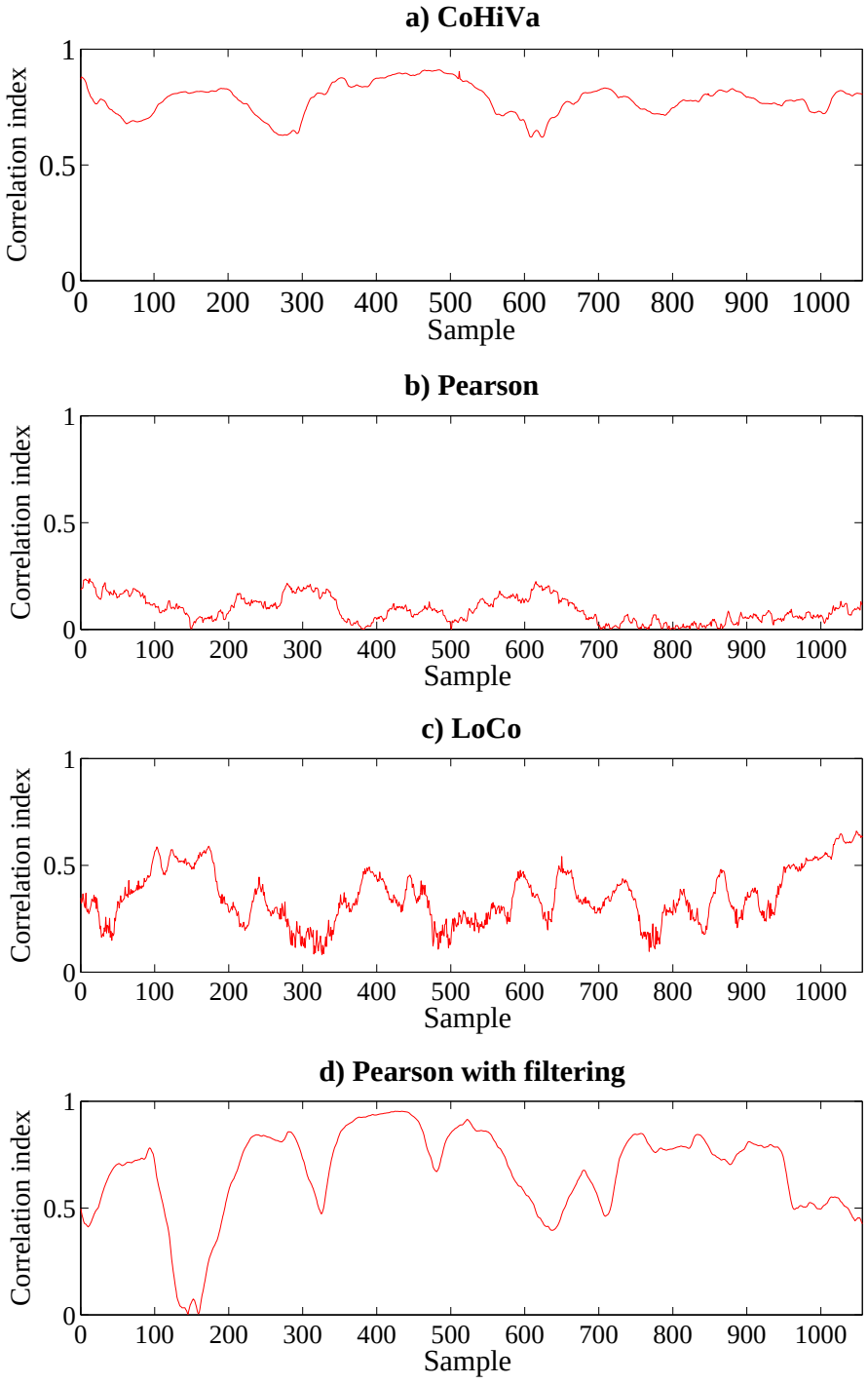
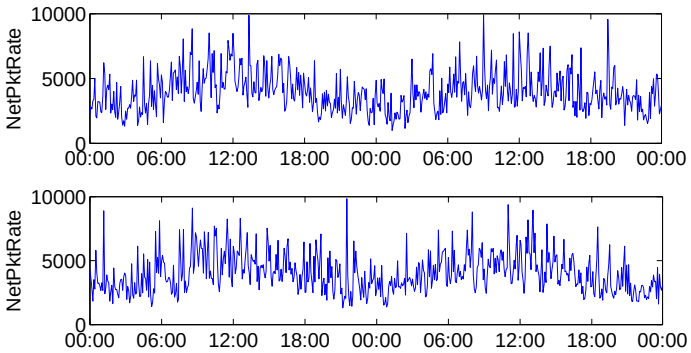
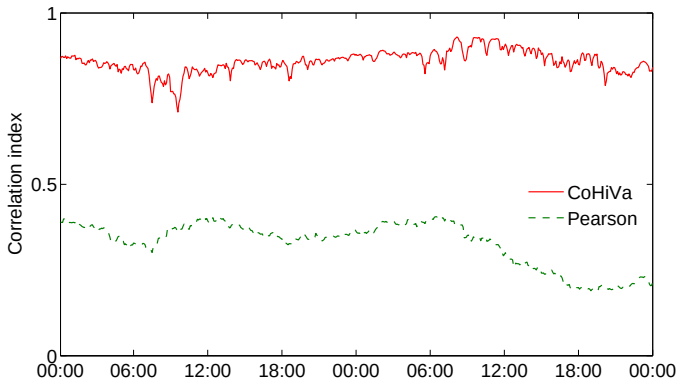


Figure 5.10: Performance of four correlation models: CoHiVa, Pearson, LoCo, and Pearson integrated with a filter.



(a) Network Packet Rate of two servers



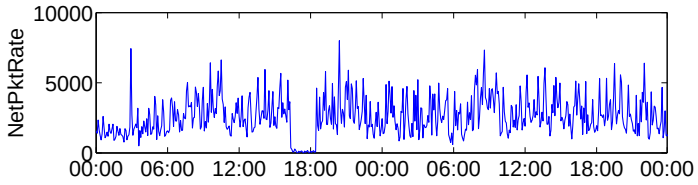
(b) Correlation indexes

Figure 5.11: Normal behavior.

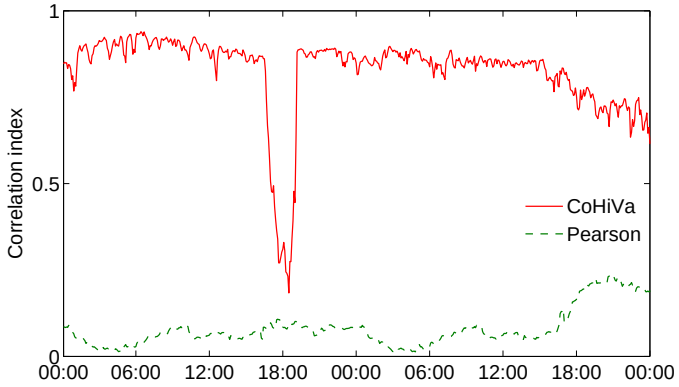
During the following two days reported in Figure 5.12(a), a system problem in the load balancer causes one of the two serves to not receive requests. The problem was temporary and normal service resumed after two days. In correspondence with this event, the CoHiVa correlation index (continuous line in Figure 5.12(b)) shows a progressive drop to the extent that the two time series are evaluated as no longer correlated. On the other hand, the Pearson index (dotted line in Figure 5.12(b)) remains lower than 0.5 during the entire period of sampling, thus resulting useless as a basis for tracking anomalies.

5.5.3 Time series prediction

Time series prediction is often adopted in system management contexts (e.g., [60]). In order to establish whether a time series is predictable or not, we measure its autocorrelation, that is, the correlation among values of a time series at different lags in time. This information determines the presence of a statistical dependency among the values of a time series. Prediction models are considered applicable



(a) Network Packet Rate of two servers

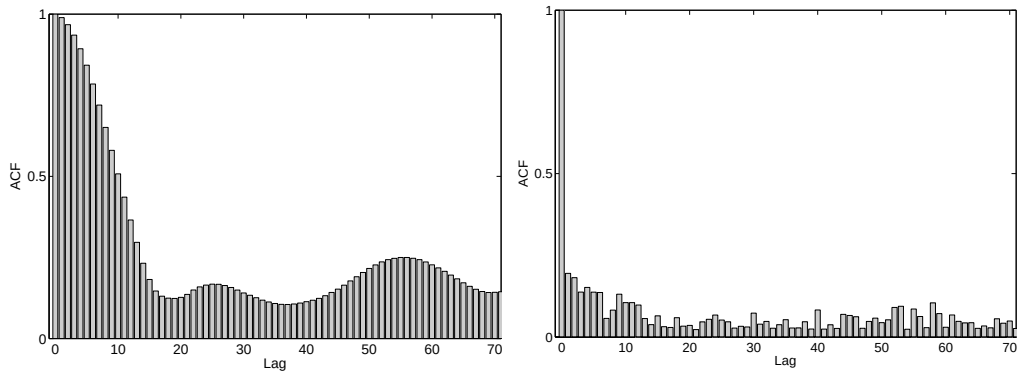


(b) Correlation indexes

Figure 5.12: Anomalous behavior in one server.

if the decay in the autocorrelation function (ACF) of the time series is exponential [32]. The lag at which the autocorrelation function becomes negligible determines how many past values it is convenient to include for the estimation of future values [167].

Let us consider as an example the time series at the bottom of Figure 5.11(a) to illustrate the importance of a robust evaluation of the autocorrelation even for highly variable data. To determine if the considered time series is predictable, we compute its ACF through the CoHiVa and Pearson correlation indexes. Figure 5.13(a) shows the ACF obtained by computing the CoHiVa correlation index at different lags ranging in the interval $[1, 72]$. The exponential decay of the curve means that a relationship between the samples exists and therefore, by following CoHiVa, we can conclude that the time series can be predicted. On the other hand, by computing the ACF on the basis of the Pearson index, we obtain the quickly decaying curve of Figure 5.13(b) suggesting that the time series cannot be predicted.



(a) CoHiVa index (b) Pearson index

Figure 5.13: Autocorrelation functions.

To validate that the considered time series is actually predictable, we refer to an autoregressive model (AR) that is a weighted linear combination of p past values. These values are weighted by p linear coefficients that are the first p values of the ACF function evaluated on time series. The p order of the AR model is defined by a statistical test based on the partial auto-correlation function that is described in [167].

We set AR parameters according to the ACF results obtained through CoHiVa and on the basis of this evaluation we conclude that the AR(12) is the best autoregressive model for the considered time series. In Figure 5.14, we show the results of applying an AR(12) model for predicting 6 hours of network packet rate on the basis of past data.

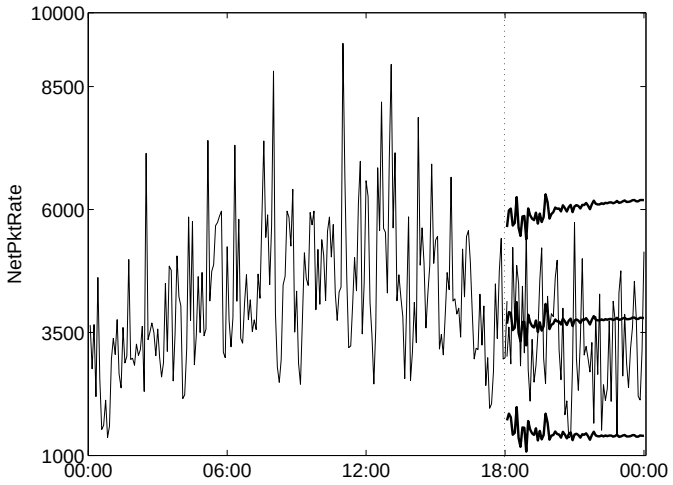


Figure 5.14: Predicted time series.

The line before 18:00 represents past samples. After that hour, the line represents the future values that we aim to predict. The three bold lines represent the predicted values including their confidence interval. We can appreciate that the actual values are always contained in the prediction interval, hence the time series is predictable as anticipated by the ACF evaluation through CoHiVa.

Chapter 6

Data clustering

Clustering is a widely adopted approach for augmenting the level of knowledge on rough data. The goals of clustering applied to computer and network datasets can be different, going from Web sites characterization [62], classification of users navigation patterns [65], network traffic classification and management [63]. For example, many network management goals such as flow prioritization, traffic shaping and policing, and diagnostic monitoring as well as many network engeneering problems, such as workload characterization and modeling, capacity planning, and route provisioning may benefit from traffic clustering [63].

In this chapter, we are interested in correlation-based clustering algorithms applied to highly variable time series. This set of algorithms (e.g., Pearson product moment [68], Spearman and Kendall ranks [39, 69]) consider that time series are similar if they exhibit some degree of statistical inter-dependency, and differ from other popular approaches using some geometrical distance (e.g., Euclidean distance [67], cosine distance [66]) as their *similarity measure*. The reason of focusing on correlation similarity measures is distance functions are not always adequate in capturing dependencies among the data. In fact, strong dependencies may exist between time series even if their data samples are far apart from each other as measured by distance functions [64]. In the next section, we will support this statement through a network related example.

The choice and the performance of the similarity measure impact the quality of any clustering algorithm. The better the accuracy and robustness of the measure in finding similarity, the better the quality of the clustering model. As we broadly demonstrated in Chapter 5, existing correlation indexes are accurate and robust in disclosing similarity except when time series exhibit high variability. This is the case of most traffic data that are highly variable in terms of number of connections, request inter-arrivals, flow sizes (e.g., [72, 73, 75]). In these scenarios, popular correlation indexes, such as the Pearson coefficient [68], the Spearman rank [69], the Kendall rank [39], and the Local Correlation index [70], show poor results

because they are unable to capture correlations even when they exist.

By basing on the CoHiVa correlation index presented in Section 5.2 as similarity measure, in this chapter we present a clustering model that can group time series presenting similarity also when characterized by high variability, such as network traffic [75], workloads [74], and data center resource metrics [11]. Such data may have strong correlations that are masked by perturbations. When some correlations exist, CoHiVa is able to identify them and clustering algorithms can group time series accordingly. The improvements with respect to the state of the art are shown on synthetic and real datasets characterized by high variability.

6.1 Problem definition

We define the clustering process based on similarity by considering a dataset $\boldsymbol{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$. For example, it contains all time series of a monitored network, where each time series $\mathbf{x}_j = [x_{j1}, \ldots, x_{jn}]$ is a vector containing a time-ordered discrete sequence of traffic data sampled once. We are interested in partitioning the N time series into K clusters $\mathcal{C} \equiv \{\mathcal{C}_1, \ldots, \mathcal{C}_K\}$ $(K \leq N)$, such that:

1. $\mathcal{C}_i \neq \varnothing$, $i = 1, \ldots, K$;
2. $\bigcup_{i=1}^{K} \mathcal{C}_i = \boldsymbol{X}$;
3. $\mathcal{C}_i \cap \mathcal{C}_j = \varnothing$, $i, j = 1, \ldots, K$ and $i \neq j$.

The clustering algorithm requires the choice of a similarity measure determining groups of time series so that the similarity between time series within a cluster is larger than the similarity between time series belonging to different clusters. As a similarity measure, we adopt the *correlation index* ρ between two time series $\mathbf{x}_i$ and $\mathbf{x}_j \in \boldsymbol{X}$, where the absolute value of ρ ranges between 0 and 1. When $\rho = 0$, there is no correlation between the two time series, while $\rho = 1$ indicates a complete correlation between $\mathbf{x}_i$ and $\mathbf{x}_j$. The literature offers several guidelines for the best interpretation of the value of the correlation measure [68, 76], but all criteria depend on the context and purposes of the analysis. In this chapter, we do not refer to a specific traffic scenario, hence we can adopt the most general interpretation indicating a *strong correlation* when $\rho > 0.5$, and a *weak correlation* for $\rho \leq 0.5$ (e.g., [76]). Different choices for the threshold do not impact the main conclusions of this chapter.

This chapter proposes a new similarity measure that is able to determine correlation clustering even in datasets exhibiting a high degree of variability, where existing correlation indexes (e.g., [39, 68–70]) are not accurate. High variability

is a typical phenomenon in network-related time series [75] in which most observations take values around the time series trend (*trend pattern*) and some observations depart from it with appreciable frequency, even by assuming extremely large values with non-negligible probability (*perturbation pattern*). Trend patterns represent the tendency of a time series that may be related to the other time series, while perturbation patterns consist of random observations hiding trends. In this chapter, we use the standard deviation as the measure of time series variability because a high standard deviation is the most typical trademark of highly variable network measurements [75]. For our purposes, this feature causes trend patterns that are hard to identify because masked by perturbations.

Figure 6.1 illustrates some examples of highly variable time series derived from network monitors measuring the number of active connections, active clients and transferred bytes during a day period. In each time series, we can see the presence of different trend patterns during working hours and during the night. These patterns are masked by perturbation patterns. However, there is an evident dependency between the number of active connections and the amount of transferred bytes. In fact, when the number of active connections increases (decreases), the amount of transferred bytes increases (decreases) as well.

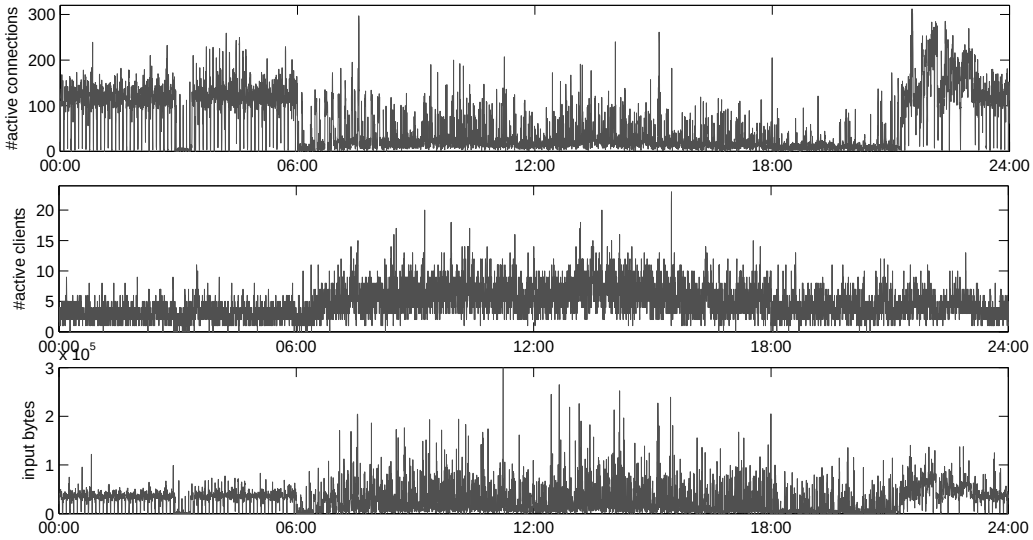


Figure 6.1: Examples of network-related time series.

Despite the variability, we want that a good similarity measure can detect this dependency so to group the two time series in the same cluster. This goal cannot be achieved through distance-based similarity measures because the distance between the sample values of the two time series is not always close, hence the

two time series cannot be clustered together through a traditional distance-based clustering model. For this reason, we prefer to consider correlation as the similarity measure, and we propose a correlation-based clustering model that is able to disclose dependency even in highly variable scenarios where distance-based clustering models do not work.

As demonstrated in Chapter 5, the most popular correlation indexes are affected by two main problems when working on highly variable time series:

1. low accuracy, since they are unable to detect similarities even among correlated time series;

2. low robustness, since they do not guarantee a stable evaluation of the correlation index, even when the relationships between the time series do not change.

These problems of existing correlation indexes affect the quality of the results of clustering models based on similarity when applied to highly variable datasets. In these contexts, the risk is that time series characterized by strong correlation may be clustered in different groups, and uncorrelated time series may be assigned to the same cluster. These reasons motivate the use of our correlation index (CoHiVa) presented in Chapter 5 as new similarity measure that is able to disclose correlations in an accurate and robust way in highly variable contexts. CoHiVa has several benefits: it does not require any assumption about statistical properties, any pre-analysis of time series characteristics, and any integration with pre-filtering techniques. Moreover, it is able to adapt its parameters to data characteristics, to capture linear and non-linear dependences, and to cluster time series basing on their statistical correlation in an accurate and robust way.

6.2 Clustering of highly variable datasets

In this section, we present how the novel correlation index, CoHiVa, that we presented in Chapter 5 may be used as the similarity measure of different clustering algorithm(s) applied to highly variable time series.

The described clustering algorithm adopts CoHiVa to modify the complete-linkage clustering model [77], but other clustering algorithms using similarity measures can be considered as well. The proposed algorithm denotes N singleton clusters $\{\mathcal{C}_1, \ldots, \mathcal{C}_N\}$, each corresponding to a monitored time series ($\mathcal{C}_i \equiv \mathbf{x}_i$, $i = 1, \ldots, N$). It then computes the NxN similarity matrix $\mathcal{D}$ containing the CoHiVa correlation indexes $\rho(\mathbf{x}_p, \mathbf{x}_q)$ between all pairs of time series $\mathbf{x}_p$, $\mathbf{x}_q$ in $\boldsymbol{X}$ through the procedure described in Section 5.2.

Let us remind the main steps of the procedure. First, we identify the main patterns in $\mathbf{x}_p$ that correspond to trends (i.e., periodic and seasonal components) and perturbations by applying the Singular Value Decomposition (SVD) [70] to the auto-covariance matrix of the time series. We compute:

$$\mathbf{\Phi}(x_p) = \mathbf{U}(x_p)\mathbf{\Sigma}(x_p)\mathbf{V}(x_p)^T, \tag{6.1}$$

where $\mathbf{\Phi}(x_p)$ is the auto-covariance matrix of $\mathbf{x}_p$, and $\mathbf{U}(x_p)$, $\mathbf{\Sigma}(x_p)$ and $\mathbf{V}(x_p) \in \mathbb{R}^{n\times n}$.

We then reconstruct a *k-dimensional approximation* of the correlation matrix by retaining just the principal vectors corresponding to the highest k singular values $(k < n)$ obtained through SVD. We select the principal vectors accounting for 90% of the variation and compute the *k-dimensional approximation* matrix through the following equation:

$$\bar{\mathbf{\Phi}}(x_p) \equiv \bar{\mathbf{U}}(x_p)\bar{\mathbf{\Sigma}}(x_p)\bar{\mathbf{V}}(x_p)^T, \tag{6.2}$$

where $\bar{\mathbf{U}}(x_p) \equiv [\bar{\mathbf{u}}_1, \ldots, \bar{\mathbf{u}}_k]$, $\bar{\mathbf{V}}(x_p) \equiv [\bar{\mathbf{v}}_1, \ldots, \bar{\mathbf{v}}_k]$ and $\bar{\mathbf{\Sigma}}(x_p) \equiv \text{diag}\,[\bar{s}_1, \ldots, \bar{s}_k]$.

Then, we retain just trend patterns among the main patterns of $\mathbf{x}_p$ by removing perturbation patterns from $\bar{\mathbf{U}}(x_p)$ in 6.2. To this end, we compute the Hurst exponent H of the k principal vectors by means of the rescaled range analysis of Hurst (see Equation 5.7) and we consider only the principal vectors $\bar{\mathbf{u}}_i \in \bar{\mathbf{U}}(x_p)$ having an estimated Hurst exponent H_i far from 0.5. Such vectors builds up a new *trend approximation* matrix by applying:

$$\hat{\mathbf{\Phi}}(x_p) \equiv \hat{\mathbf{U}}(x_p)\hat{\mathbf{\Sigma}}(x_p)\hat{\mathbf{V}}(x_p)^T. \tag{6.3}$$

For each pair of time series $\mathbf{x}_p$ and $\mathbf{x}_q$, we can now compute the CoHiVa correlation index as following:

$$\rho(\mathbf{x}_p, \mathbf{x}_q) = \frac{1}{\hat{k}(x_p) + \hat{k}(x_q)}(\|\hat{\mathbf{U}}(x_p)^T\hat{\mathbf{U}}(x_q)\| + \|\hat{\mathbf{U}}(x_q)^T\hat{\mathbf{U}}(x_p)\|), \tag{6.4}$$

where $\hat{k}(x_p)$ and $\hat{k}(x_q)$ are the amounts of trend principal vectors of $\mathbf{\Phi}(x_p)$ and $\mathbf{\Phi}(x_q)$, respectively, while $\hat{\mathbf{U}}(x_p)$ and $\hat{\mathbf{U}}(x_q)$ are the trend principal vectors matrices collecting them.

Through this procedure, our algorithm finds out the CoHiVa correlation indexes $\rho(\mathbf{x}_p, \mathbf{x}_q)$ between all pairs of time series in $\boldsymbol{X}$ and uses them to fill the NxN similarity matrix $\mathcal{D}$. These correlation indexes represent the measures of similarity between all the clusters.

We then proceed through the following steps:

1. Find the most correlated pair of clusters in the similarity matrix:

$$\rho(\mathbf{x}_r, \mathbf{x}_s) = \max_{1 \leq p,q \leq N, p \neq q} \rho(\mathbf{x}_p, \mathbf{x}_q). \tag{6.5}$$

2. Combine cluster $\mathcal{C}_r$ and cluster $\mathcal{C}_s$ to form a new cluster, denoted as $\mathcal{C}_{(r,s)}$.

3. Update the similarity matrix $\mathcal{D}$ by deleting the rows and columns corresponding to clusters $\mathcal{C}_r$ and $\mathcal{C}_s$ and by adding a row and a column corresponding to the early created cluster. The similarity between the new cluster $\mathcal{C}_{(r,s)}$ and a generic old cluster $\mathcal{C}_i$ is defined as:

$$\rho(\mathbf{x}_i, \mathbf{x}_{(r,s)}) = \max(\rho(\mathbf{x}_i, \mathbf{x}_r), \rho(\mathbf{x}_i, \mathbf{x}_s)). \tag{6.6}$$

4. Repeat steps 1)-2)-3) until all objects are in a cluster.

This procedure results as an organized tree built up according to the similarity matrix computed through CoHiVa. Cutting the tree at a given height gives a partition clustering at a selected precision ϕ. For example, if we discriminate between weak and strong correlation through the value of $\phi = 0.5$, we cut the tree when the highest correlation value found in the similarity matrix goes below $\phi = 0.5$. The time series forming a sub-tree after this cut are considered part of the same cluster [80].

6.3 Experimental results

We evaluate the quality of the proposed clustering algorithm by referring to two network-related datasets characterized by high variability: Abilene network traffic, and measurements collected at the border router of our university.

- **Abilene network.**
  The publicly available Abilene dataset contains aggregate data based on measurements of origin-destination (OD) flows on the Abilene network [87]. We consider sampled data from every router over a seven-day period, starting December 12, 2003 [1]. At sampling period of five minutes, each link produces 2016 samples a week.

- **University network.**
  The dataset is obtained from a monitor attached to a border router of the university and contains flows characterized by different metrics, such as total number of packets (excluding ack packets), packet size statistics (mean,

[1] Data available at: http://math.bu.edu/people/kolaczyk/datasets.html

minimum, maximum, quartiles), number of bytes transferred in each direction, number of active connections and number of active clients. These network metrics are aggregated every 10 seconds. The presented results refer to one day characterized by 8640 samples for each network metric.

The clustering algorithms applied to these two datasets are used for two different purposes: for traffic clustering of Abilene network data (Section 6.3.1), and for server clustering of university network data (Section 6.3.2).

6.3.1 Traffic clustering

The identification of the main statistical properties of traffic flows and the clustering of flows based on such properties are crucial to many network management tasks and network engeneering problems [63]. Due to the high variability of OD traffic flows [75], clustering solutions must be able to group data even in highly variable contexts. Our algorithm represents a good solution for traffic clustering because it is able to identify correlation between OD flow traffic patterns and to cluster together flows presenting similar statistical properties.

We carried out a preliminary analysis of the Abilene dataset with the goal of extracting the main statistical properties shared by the OD traffic flows that we expect to be shared by the time series clustered together. This analysis evidences the presence of the following six main statistical properties:

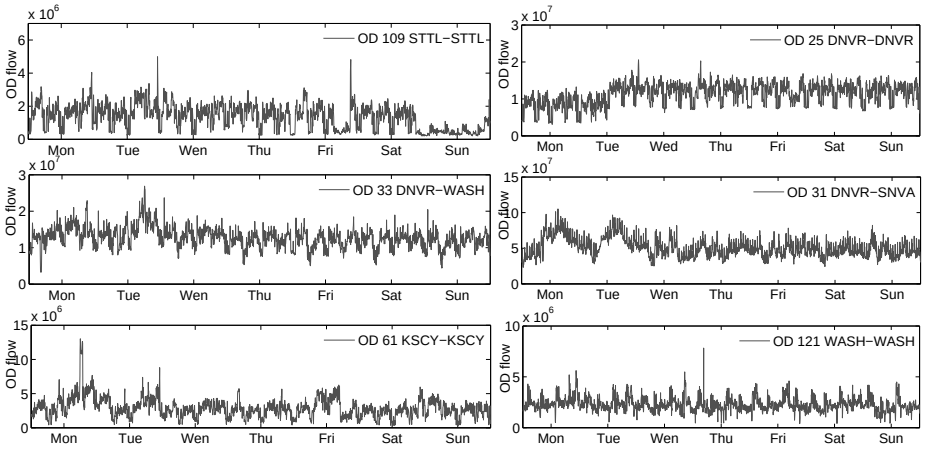
1. *Periodic pattern of 6 hours*: some OD flows have fluctuations of request volume over a time period of 6 hours, typically related to hourly human behaviors. Some intra-daily periodic patterns are shown in Figure 6.2(a).

2. *Periodic pattern of 24 hours*: some OD flows present the typical increase of user requests during working hours and a decrease during the night. Some examples of daily patterns are reported in Figure 6.2(b).

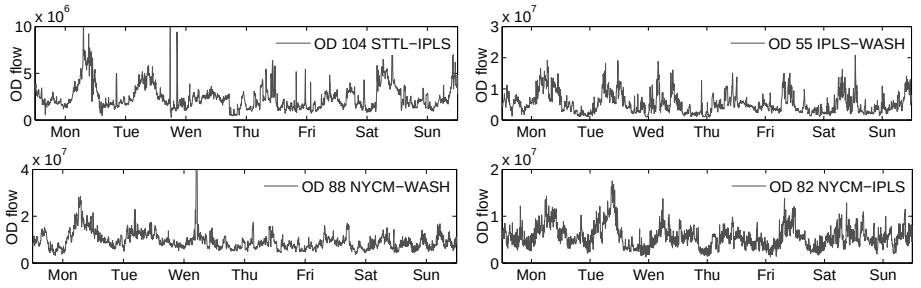
3. *Repeated spikes every 45 minutes*: some OD flows manifest bursts of requests repeated every 45 minutes. Figure 6.2(c) shows this behavior during a time interval of 2 days.

4. *Repeated spikes every 90 minutes*: some OD flows exhibit regular spikes every hour and a half as in Figure 6.2(d).

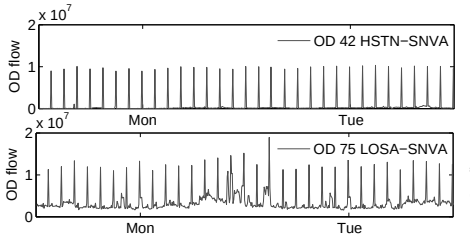
5. *Aperiodic trend*: some OD flows are characterized by increasing or decreasing trends even thought they do not manifest any periodicity. Some flows with these characteristics are presented in Figure 6.2(e).



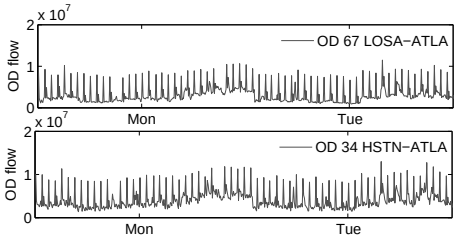
(a) Periodic pattern of 6 hours



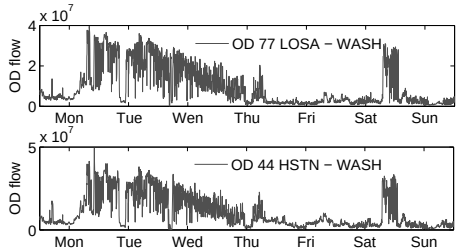
(b) Periodic pattern of 24 hours



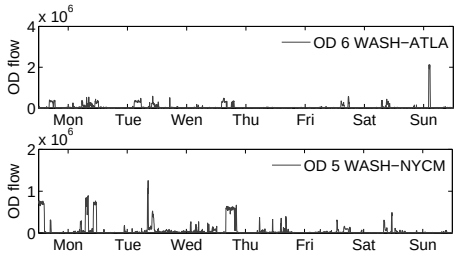
(c) Periodic spikes every 45 minutes



(d) Periodic spikes every 90 minutes



(e) Aperiodic trends



(f) Stochastic patterns

Figure 6.2: Characteristics of Abilene traffic flows.

6. *Stochastic pattern*: some OD flows are characterized by minor bursts and irregular behavior as those in Figure 6.2(f).

This preliminary analysis gives us the ground truth. In other words, we can expect that the most effective clustering algorithm is able to find out 6 clusters, each one including all and only the OD flows sharing just one of the identified statistical properties. Thanks to this term of comparison, we can compute the *recall* and the *precision* [88] of the other clustering algorithms.

The *recall* measures the ability of an algorithm to cluster a flow presenting a statistical property together with other flows having that statistical property (e.g., a flow with a 24-hour periodic pattern is clustered together with the flows having a daily period). To achieve a recall of 100%, the clustering algorithm must insert each flow in the cluster representing the corresponding statistical property.

The *precision* gives information about the ability of the clustering algorithm to limit the number of flows that present a statistical property but are clustered together with flows characterized by a different statistical property (e.g., a flow with a 24-hour periodic pattern is clustered together with 6-hours periodic flows). A precision of 100% means that the algorithm inserts into a cluster only the flows with the corresponding statistical property.

We compare the results of the complete-linkage clustering algorithm using CoHiVa against those obtained through the Pearson product moment and the LoCo score, that achieved the best trade-off between accuracy and robustness on synthetic settings. Table 6.1 reports the recall and precision values for the three considered algorithms.

Cluster		**CoHiVa**	**Pearson**	**LoCo**
Periodic pattern of 6 hours	*Recall*	100%	100%	100%
	Precision	89%	56%	89%
Periodic pattern of 24 hours	*Recall*	91%	0%	81%
	Precision	83%	0%	78%
Repeated spikes every 45 minutes	*Recall*	75%	100%	0%
	Precision	60%	40%	0%
Repeated spikes every 90 minutes	*Recall*	100%	100%	0%
	Precision	100%	100%	0%
Aperiodic trend	*Recall*	100%	100%	0%
	Precision	100%	67%	0%
Stochastic pattern	*Recall*	81%	49%	67%
	Precision	91%	100%	83%

Table 6.1: Recall and precision

This table evidences three main results.

- The ability of CoHiVa in disclosing the presence of all 6 clusters, since it obtains high recall and precision values over all the clusters. The null values obtained by Pearson in both precision and recall over the cluster with daily periodic flows mean that this index does not reveal the presence of correlated flows having daily patterns. LoCo is unable to identify spiky and aperiodic flows because it is characterized by null recall and precision values over these three clusters. These results are in accord to those achieved on synthetic settings in the previous section: Pearson is unable to identify correlation among highly variable data (e.g., flows with a 24-hours period), while LoCo fails in finding spiky and aperiodic flows because it considers just the first principal component not including information of those irregular patterns.

- The ability of CoHiVa in discriminating between correlated and not correlated flows. This algorithm guarantees high recall and precision values in the identification of correlated flows belonging to the first 5 clusters and the best performance for the identification of uncorrelated stochastic flows. On the other hand, Pearson and LoCo insert in the same cluster many flows that are correlated with other flows, as their lower recall values over the stochastic cluster demonstrate.

- The ability of CoHiVa in guaranteeing the best compromise between the capacity of clustering together flows having similar statistical properties and the capacity of limiting the number of flows that are wrongly classified.

This last result can be appreciated by introducing a further performance measure. Since a trade-off between recall and precision values exists, these two metrics can be combined into one measure, namely the *F-measure* [88], that gives a global estimation of the quality of the clustering algorithm through the weighted harmonic mean of precision and recall, that is:

$$\textit{F-measure} = 2\frac{precision * recall}{precision + recall} \tag{6.7}$$

The closest the F-measure to 1, the highest the quality of the clustering algorithm.

The combined effect of recall and precision can be appreciated in Figure 6.3 showing that CoHiVa achieves the best F-measure in all cluster identification. This is an important result about the robustness of CoHiVa: independently of the flow statistical properties, CoHiVa guarantees the best compromise between the number of correctly and wrongly clustered flows. Pearson and LoCo achieve some good results but they show also unacceptable performance in other cases. In

particular, the null F-measure values of Pearson and LoCo indexes in the identification of flows with highly variable, spiky and aperiodic flows limit their applicability as similarity measures for correlation-based clustering algorithms.

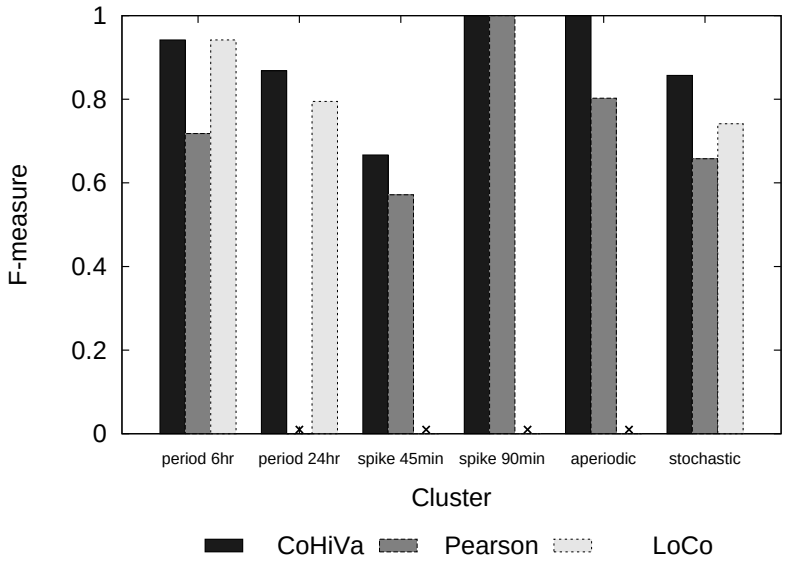


Figure 6.3: F-measure.

6.3.2 Server clustering

As a further test case, we apply the clustering algorithms to the identification of network-based servers having similar statistical behavior. The idea is to group servers performing similar tasks through the analysis of the network traffic flows they generate. Even these scenarios are characterized by highly variable measures gathered by a network monitor connected to the border router of our university. We select 21 destination servers by considering the destination address and the port number of the traffic flows. Table 6.2 reports the different classes of servers and the number of monitored servers for each class. We expect that a good clustering algorithm is able to group together hosts supporting same server processes. In other words, we expect that Web servers are grouped in the same cluster, and that this cluster is separated from the cluster containing FTP servers, as well as from that containing DNS servers, and so on.

We apply a K-means clustering model [67] to the 21 datasets referring to the monitored network metrics of each considered server. We set $K = 7$ because we have seven classes of servers. We consider clustering in a 2-dimensional space based on the correlation among input/output bytes and input/output packets of the

	Web	DNS	FTP	Samba	Oracle	MySql	Microsoft Sql
# servers	7	2	2	5	1	2	2

Table 6.2: Server classes

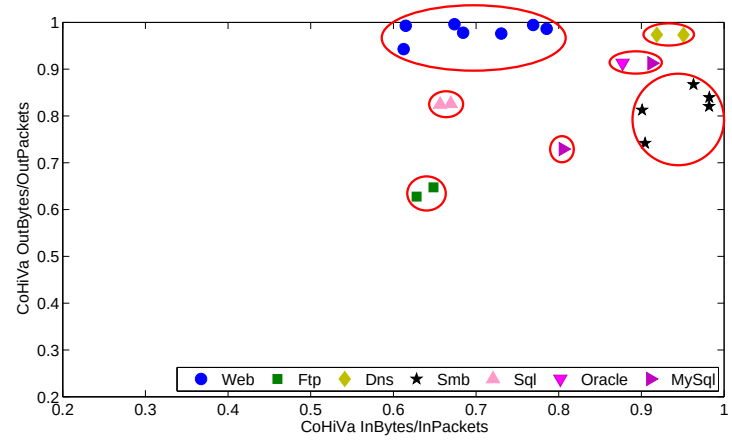
monitored traffic flows, supported by previous studies on correlation of flow characteristics (e.g., [89]). By considering these two pairs of metrics, for each dataset related to one of the 21 servers we compute three similarity matrices containing the pair-wise similarity measures computed through CoHiVa, Pearson, and Loco. These measures allow us to place each server in the spaces generated through the indexes. In each space, the K-means model groups servers so as to minimize the within-cluster distance [67].

Figure 6.4 shows the results related to the CoHiVa index (Figure 6.4(a)), the Pearson product moment (Figure 6.4(b)), and the LoCo score (Figure 6.4(c)).

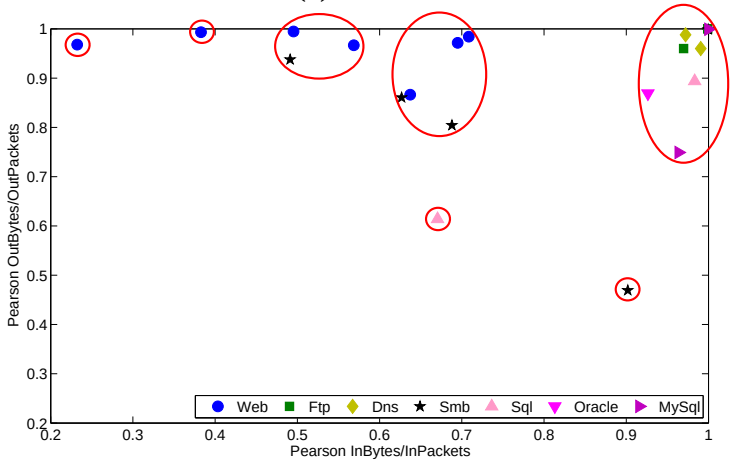
For this dataset, CoHiVa is able to group all the Web servers in a cluster, that is separated from the cluster containing the two DNS servers, from that referring to the two FTP servers, from the one for all the Samba servers, and so on. We should note that the chosen metrics and similarity measure do not allow us to discriminate between the Oracle and MySql database servers, that consequently are grouped together.

The results related to Pearson and LoCo are much poorer: they cluster different types of servers in the same cluster, and some servers are in singleton clusters despite their expected correlation. These results are caused by the high variability of some monitored metrics and to the low mean value of others. As a consequence, Pearson and LoCo correlation indexes see as uncorrelated time series that actually present some dependency, while they see high correlations between time series that are actually independent.

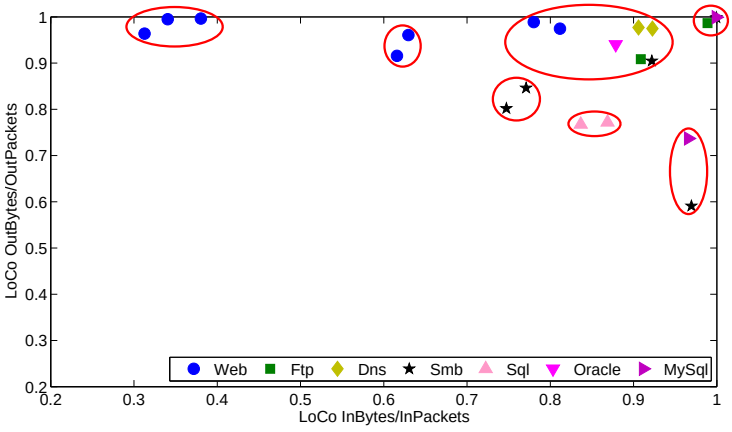
The reported results show that CoHiVa represents an effective solution for clustering data in highly variable contexts where state-of-the-art similarity measures are affected by poor results.



(a) CoHiVa



(b) Pearson



(c) LoCo

Figure 6.4: Server clustering results.

Chapter 7

Predictive analytics

A user of an infrastructure cloud service such as EC2 has the option of selecting an availability zone where their resources are provisioned. An availability zone (AZ) is a data center that is physically isolated from other availability zones. Cloud providers offer several of such availability zones in various geographies. For any cloud provider, availability zones are not identical - their hardware infrastructure, type and version of management stack, and load characteristics differ. As a result, services offered by different availability zones will vary.

In this chapter, we present an adaptive and predictive method for automatically selecting availability zones. In the proposed model a user provides a set of desired quality of service attributes for each instance deployment request and their importance. Based on this input, we construct a utility function. QoS measurements for previously deployed instances by the same or other users are available with their associated utility value. We use these observed utility values to train prediction models for each availability zone. In essence, the prediction models help us learn unpublished attributes of a service. We accommodate time varying changes in service attributes by reconstructing the models based on continuously changing input data. Given the prediction model and the requirement specified in a new request, we formulate and solve an optimization problem to select the optimum availability zone for that request. We further use prediction models to extract performance insights for availability zones based on user satisfaction and compare the performance of zones with respect to different request types. This additional use of prediction models helps service providers to understand the performance of their availability zones for different request types.

The proposed solution moves cloud service users from the common practice of manually selecting a cloud availability zone, by relying on community knowledge, to the automatic selection of customized solutions, focused on their needs. It is important to note that our solution considers the possibility of a deployment across multiple provider deployment, i.e., the resources can be placed in different zones

of different providers if such a deployment is supported by other important factors such as availability of suitable images and automation, similarity of service types, etc. In a multi-cloud setting our solution would benefit cloud brokering services which become increasingly popular.

7.1 Problem definition

Some differences in services offered by different availability zones are widely known as they are published by the service provider. Such well known attributes are types of offered instances, their sizes, and prices. As an example, as of writing this thesis, EC2 does not offer Cluster GPU instances in Sao Paulo zone while they are available in North Virginia zone. However, cloud users have observed a number of differences in the quality of service provided by availability zones that are not captured in those advertised attributes.

Benchmarking studies performed on EC2 report on variation between performance of same type instances deployed in different availability zones. Shad et al. [104] measured significant differences in instance disk read/write performance, as well as in instance network performance between availability zones in US and EU locations. They conjecture that divergent disk performance is due to different HDD technologies used in US and EU. They also stipulate that differences in network performance result from different virtual machine placement algorithms applied in different availability zones. Furthermore, Life Scaling blog1 reports up to 100% difference in network latency between pairs of instances deployed within different availability zones in the US East region. Farley et al. [94] and Ou et al. [103] measured substantial performance variation (up to 280%) between same-type instances deployed on different processor architectures in EC2 with benchmarks stressing various types of resources. Furthermore, Ou et al. [103] estimated up to 9.5 times higher probability of acquiring an instance on a specific processor architecture between two availability zones indicating that, due to infrastructure differences, the expected instance performance in these two zones should vary considerably. Outside of scientific studies, users of EC2 report particularly high probability of instance deployment rejection due to resource contention in certain availability zones (mostly in the US East region).

The above data suggest that considerable difference in quality of service may be observed by cloud users with different availability zones, even when offered by the same provider. This observation is not surprising as different data centers are likely made of different generations of hardware and software technologies. Moreover it should be expected that those characteristics will change over time. From a user perspective, such inconsistency in service availability and behavior is an unwelcome challenge as availability zone selection may have a rather dras-

tic impact on the availability, performance, and cost-efficiency of the deployed workload. A tool that would help a user to automatically select an availability zone suitable for their requirements would alleviate these concerns. As the QoS attributes needed to make a decision are not usually advertised as part of a service description, in addition to the fact that they may vary over time, a suitable tool will have to estimate such attributes based on prior observations.

7.1.1 Formulation of the problem

Request attributes

A user request is represented by a requirement vector. Let the i^{th} user request be represented by the vector

$$\mathbf{r}_i = \{r_{i1}, r_{i2}, \ldots, r_{im}\}, \tag{7.1}$$

where r_{ij} , $j = 1, \ldots, m$ specifies the j^{th} requirement of user i that is expected to be satisfied by the selected availability zone. User requirements may include: (i) resources, as the resource amounts required by the user (e.g., CPU, memory etc); (ii) QoS criteria, as quality of service objective that a user wants to achieve (e.g., highest reliability, minimum execution time, highest performance); (iii) constraints, as restrictions on possible service provisioning (e.g., locality constraints, service type constraints, load balancing constraints); (iv) user instance types, as the type of instance the user wants to run; and (v) user machine types, as the type of machine that the user requires the availability zone to provide.

Availability zone model

Availability zones differ in characteristics depending on their implementation. Attributes such as availability zone size, hardware infrastructure, or management stack (including in- stance placement policies) result in different levels of reliability and performance. Attribute values that influence the QoS offered by an availability zone for a particular instance type are not known. Generally, quality of service data for any particular instance type in a given availability zone are not known a priori. It is, however, possible to measure the QoS parameters after an instance has been deployed. To obtain such a measurement, one can use a monitoring service provided by the cloud owner, e.g., Cloud Watch offered by EC2, or deploy a proprietary tool to monitor the deployment and runtime characteristics of provisioned instances.

Therefore, in our model of an availability zone, we assume that either the availability zone manager, or a monitoring tool deployed within it, can provide us with a measurement to evaluate each requirement specified in the request type.

For example, such measurements would identify certain architectural features of a node on which the instance was deployed, and include notifications of their failures and recovery as well as runtime performances such as throughput of various resources, delays, etc. Such measurements may be evaluated against the requirements specified in the re- quest, as given by Equation 1. We call the results of such an evaluation a requirement satisfaction level.

Let $c_{ik} \in [0, 1]$ denote the satisfaction level of requirement r_{ik} . If the requirement r_{ik} is fully satisfied, then $c_{ik} = 1$, otherwise $0 \leq c_{ik} \leq 1$. To keep the method general, we are not prescribing a method to measure how much a user requirement is satisfied after deployment. Rather, we are stating that such an evaluation can be done using an arbitrary method to produce the vector of satisfaction levels, $\mathbf{C}_i^T = [c_{i1}, c_{i2}, \ldots, c_{im}]$, for each incoming request, r_i , deployed to an availability zone. $\mathbf{C}_1^T$ is the only input we need for each availability zone.

7.1.2 Model construction

In this Section, we explain how to build a prediction model [90], by learning the behavior of each availability zone based on the historical data of past requests. Our prediction model maps the requirement vector of an incoming request, $\mathbf{r}_i$ = $\{r_{i1}, r_{i2}, \ldots, r_{im}\}$, to a customer satisfaction measure defined by a utility function, $f(r_i) \in [0, 1]$. The utility function reaches its maximum value of one when there is complete satisfaction. The value of $f(\mathbf{r}_i)$ depends on how much the requirements of an incoming request is satisfied by the avail- ability zone where the request is placed. If the requirement rik is fully satisfied, then $c_{ik} = 1$, otherwise $0 \leq c_{ik} \leq 1$. The vector of satisfaction levels, $\mathbf{C}_i^T = [c_{i1}, c_{i2}, \ldots, c_{im}]$, for i each incoming request r_i is observed after the request is deployed. The satisfaction of some requirements may be more crucial than others, therefore the satisfaction level of each requirement may have different significance. The weight vector $\mathbf{W}_i^T = [w_{i1}, w_{i2}, \ldots, w_{im}]$ denotes the significance levels i for requirements r_i. The more the value of w_{ik}, the stronger the significance of requirement r_{ik} is. One possible way of defining the utility function $f(\mathbf{r}_i)$ is to take a linear combination of the satisfaction level for each incoming request $\mathbf{C}_i^T$ and the associated weights $\mathbf{W}_i^T$ multiplied by an indicator i function $\phi(\mathbf{r}_i)$. The indicator function is used to set the satisfaction level to zero when the request is rejected. Hence, we define the utility function as,

$$f(\mathbf{r}_i) = \phi(\mathbf{r}_i) \sum_{j=1}^{m} w_{ij} c_{ij} = \phi(\mathbf{r}_i) \mathbf{W}_i{}^T \mathbf{C}_i, \tag{7.2}$$

where:

$$\phi(\mathbf{r}_i) = \begin{cases} 0 & \text{if request is rejected} \\ (\sum_j w_{ij})^{-1}, & \text{otherwise} \end{cases}$$

Term $c_{ik} \in [0, 1]$ is the satisfaction level for user i against the requirement r_{ik} and the weight $w_{ik} \in \mathbb{R}_{\geq 0}$ indicates the importance of satisfying a particular requirement. Note that the selection of $\phi(\mathbf{r}_i) = (\sum_j w_{ij})^{-1}$, in the case of no rejection, normalizes that weight vector and limits the maximum possible value of $f(\mathbf{r}_i)$ to 1.

7.1.3 Example

Let $\mathbf{r}_i$ =$[r_S, r_L, r_I, r_A, r_M]$ be the requirement vector of an incoming request. The request contains requirements related to the size, supported instance type, infrastructure type, and reliability of an availability zone. The description of the requirement attributes are listed below:

r_S**:** Requested CPU and RAM resources where $r_S \in \{$*micro, small, medium, large, xlarge*$\}$

r_L**:** Level of reliability where $r_L \in \{$*Low, Medium, High*$\}$

r_I**:** Tolerance to interruption where $r_I \in \{$*0, 1*$\}$

r_A**:** Requested application type where $r_A \in \{$*Compute, Storage, Memory Instance*$\}$

r_M**:** Requested machine type where $r_M \in \{$*Intel Xeon, AMD Opteron series* $\}$

The satisfaction level of these requirements can be measured through an agent deployed along with the request to the availability zone. The measured satisfaction for each requirement is captured by vector $\mathbf{C}_i^T$. Let's assume that for an incoming request with a requirement vector $\mathbf{r}_i$ = ["large", "medium", "1", "Compute intense", "Intel Xeon series"], the satisfaction vector is observed as,

$$\mathbf{C}^T = [0, 1, 0, 1, 1]. \tag{7.3}$$

Note that partial satisfaction levels are not considered to simplify the example. This means that the size and tolerance to interruption requirements of the incoming request, $r_S = \{"large"\}$ and $r_I = \{"1"\}$, are not satisfied while other requirements are fully satisfied. If the associated weight vector is:

$$\mathbf{W}^T = [0.2, 0.3, 0.3, 0.1, 0.1], \tag{7.4}$$

then the utility function for $\mathbf{r}_i$ is computed as:

$$f(\mathbf{r}_i) = \begin{cases} \phi(\mathbf{r}_i)\mathbf{W}_i{}^T\mathbf{C}_i = 0.5, & \text{if request is placed;} \\ 0 & \text{if request is rejected,} \end{cases}$$

where $\phi(\mathbf{r}_i) = 1$ if the request is placed and 0 otherwise. Note that due to the weights associated with each requirement, the satisfaction level did not exceed 0.5 while more than half of the requirements are satisfied.

7.1.4 Predicting the utility function

If there are multiple availability zones providing the same services, it is natural that the availability zone which returns the maximum utility value $f(\mathbf{r}_i)$ for the incoming request is likely to satisfy the requirements most. The exact value of the utility function can be computed only after deployment. It can be, however, predicted based on the utility values of deployed requests in the past. In this chapter, we propose to build a prediction model for every availability zone and use the predicted utility values, $\tilde{f(\mathbf{r}_i)}$, in selecting an optimum availability zone.

Figure 7.1 shows how we create the training tables and the associated prediction models for each availability zone. First, the incoming requests are placed into the availability zones by using a random selector. This way the incoming requests are distributed uniformly to each zone. After the placement of a request, the utility value for that request is computed and stored along with the associated requirement vector in a training table for the zone that the request is placed. Each row of a training table corresponds to a single placement instance. Once the training tables are built from the corresponding placement instances for each zone, the associated prediction models are generated by using machine learning techniques.

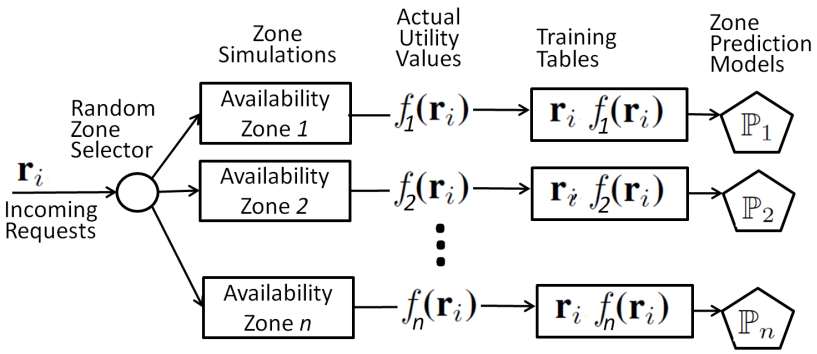


Figure 7.1: Generating prediction models.

Let $\mathbb{P}_n$ denotes the predictive model, such as a classifier, for the satisfaction level of an incoming request placed in availability zone n. $\mathbb{P}_n$ is trained by sample records or instances characterized by the tuple ($\mathbf{r}_i$, $f(\mathbf{r}_i)$) for $i = 1 \ldots, M$, where M is the size of the training set, or the number of the instances used for training. Here, $f(\mathbf{r}_i)$ is the empirical value of the utility function associated with the requirement vector $\mathbf{r}_i$. It is also called the target value or the satisfaction category. Classification models assume that utility values are discrete. In cases that the utility value is continuous, regression models should be used for prediction. Table 7.1 depicts the structure of the training data set used by $\mathbb{P}_n$ to learn the behavior of availabilIty zone n.

Case	Attributes				Target
1	r_{11}	r_{12}	$\ldots$	r_{1m}	$f(\mathbf{r}_1)$
2	r_{21}	r_{22}	$\ldots$	r_{2m}	$f(\mathbf{r}_2)$
3	r_{31}	r_{32}	$\ldots$	r_{3m}	$f(\mathbf{r}_3)$
.	.	.	$\ldots$	.	.
M	r_{M1}	r_{M2}	$\ldots$	r_{Mm}	$f(\mathbf{r}_M)$

Table 7.1: Training data set

After the training phase, $\mathbb{P}$ learns how to predict the satisfaction level for the requirement vector $\mathbf{r}_\ell$ as depicted in the following equation:

$$\mathbb{P}(\mathbf{r}_\ell) = \tilde{f}(\mathbf{r}_\ell) \tag{7.5}$$

Here $\tilde{f}(\mathbf{r}_\ell)$ is the predicted satisfaction level. The estimated mean squared prediction error, $\bar{e}(\mathbb{P})$ for model $\mathbb{P}$ is given by:

$$\bar{e}(\mathbb{P}) = \frac{1}{M} \sum_{\ell=1}^{M} [f(\mathbf{r}_\ell) - \tilde{f}(\mathbf{r}_\ell)]^2 \tag{7.6}$$

In order to find an unbiased estimation of the predicted error, the trained model is tested against a set of requirement vectors that are not part of the training set. Cross-validation [97] is a technique often used in machine learning to evaluate the models by dividing the sample data into training and validation segments. The first segment is used to learn the model and the second one is to validate. Equation (7.7) shows how to estimate the testing error by using M test data. Figure 7.2 depicts the steps of calculating error at every placement instance. In typical cross validation, the training and validation sets must cross-over in successive rounds such that each data point has a chance of being validated. We used k-fold cross validation to measure the accuracy of our predictive models.

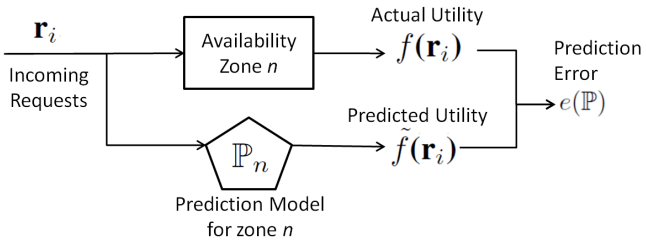


Figure 7.2: Prediction error

7.1.5 Selecting an availability zone

Had the value of the utility function been known before deployment, the availability zone that satisfies the customer requirements most would have been an obvious choice. The utility values, however, strongly depend on unpublished zone properties and they are not publicly available. Regardless, we can predict them by using the machine learning techniques and the historical data as explained above and use predicted utility values for availability zone selection. One possible selection policy uses the maximum predicted utility value to select the availability zone. Hence, if there are n availability zones, the zone that satisfies the requirement vector of the ℓ^{th} incoming request most is represented with the equation:

$$\mathbb{P}_S(\mathbf{r}_\ell) = \max_i\{\mathbb{P}_i(\mathbf{r}_\ell)\} \quad \forall i \leq n. \tag{7.7}$$

Here the utility function is maximized when $i = S$. Therefore, the zone selector assigns zone S as the optimal zone for the incoming request provided that zone S has enough capacity. Figure 7.3 shows the process of selecting the best availability zone. The prediction models are obtained as depicted in Figure 7.1.

If the best zone that maximizes the utility function cannot accommodate the associated request due to shortage in zone capacity, then the second best zone is selected for placing the incoming request. This procedure repeats until the request is placed (or finally rejected). The left hand side of Figure 7.4 shows how the zone selection procedure loops until the best available zone that can accommodate the request is determined. The right hand side of Figure 7.4 shows how the prediction models are updated based on the new training data set obtained as described in the following section.

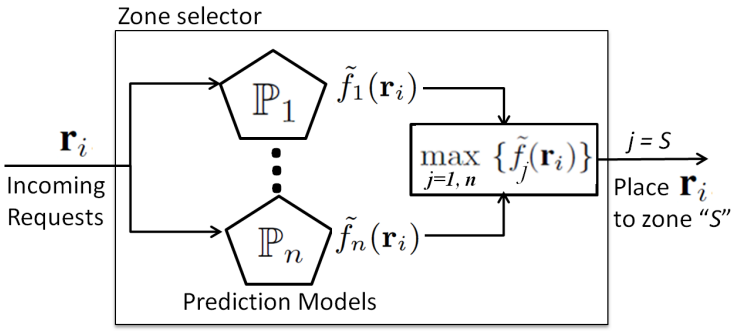


Figure 7.3: Availability zone selector.

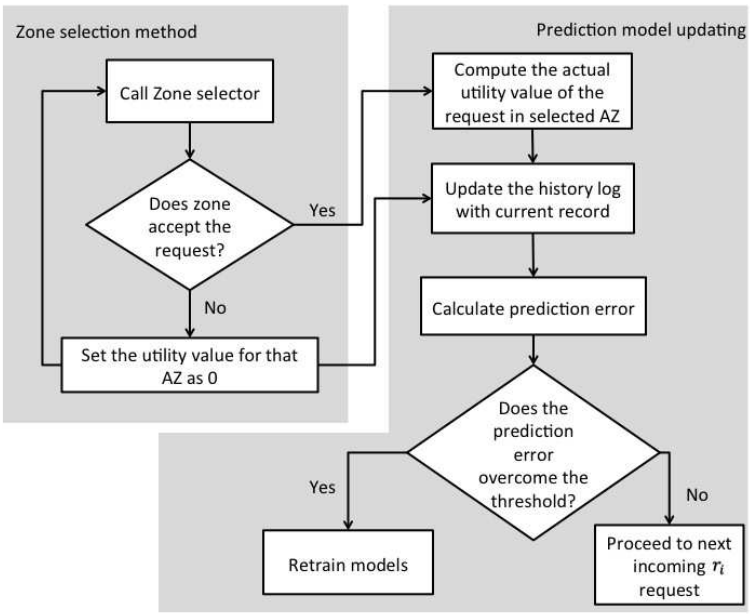


Figure 7.4: Zone selection method and updating prediction models.

7.1.6 Updating the prediction models

Due to dynamic nature of the cloud environment, the accuracy of the prediction models may decrease over time. Decline in prediction accuracy happens when the availability zone features change significantly. These changes may not be publicized or become available to the placement policy. Therefore, as the availability zone features change, the prediction models need to be retrained to learn the new cloud behavior. We monitor the prediction error by using equation (7.6) and run

a k-fold cross validation after a number of new requests are placed. After each placement, the training data set in the history log is updated. Prediction models are retrained when the average prediction error increases above the preset threshold.

Let $\bar{e}_L(\mathbb{P})$ denote the new mean squared prediction error after replacing the L oldest training samples with the most recent ones. It is given by:

$$\bar{e}_L(\mathbb{P}) = \frac{1}{M} \sum_{\ell=L}^{M+L} [f(\mathbf{r}_\ell) - \tilde{f}(\mathbf{r}_\ell)]^2 \tag{7.8}$$

If the prediction model does not successfully capture the most recent behavior of the availability zone, then $\bar{e}_L(\mathbb{P}) \geq \bar{e}(\mathbb{P})$ is expected. The prediction models are updated if the increase in the newly estimated error is more than a preset threshold, as:

$$\bar{e}_L(\mathbb{P}) \geq \gamma \bar{e}(\mathbb{P}), \tag{7.9}$$

where $\gamma > 1$ is the threshold value.

The right hand side of Figure 7.4 shows how the new training samples are collected dynamically to retrain the prediction models when the prediction error exceeds the threshold value. After the zone selector places the request to the best available zone, its actual utility value along with the requirement vector are stored in the history log as a new training data set instance. In case the zone selector cannot place the request in the first attempt, the utility value for the zone that rejects the request is set to 0 and is stored in the history log as well. Once the preset threshold for prediction error is exceeded, the models are retrained with the most updated training data set.

7.1.7 Deriving insight for availability zones

Prediction models for the availability zones enable us to make the best placement decision for the incoming requests, based on the learned behavior from the historical data. Learning the zone behavior without requiring a priori knowledge is a powerful technique when certain characteristics of the zone are not published. The results that we present in Section 7.3 show that the placement decision based on predictive modeling works as good as some other placement methods that require a priori knowledge of zone attributes.

Another aspect of predictive modeling is to derive insight for the performance of the cloud availability zones for planning purposes. This is particularly important for cloud managers. Zone prediction models can identify the availability zones that perform better for particular application types. As an example, a high-performance database along with strong reliability requirement can perform better

in some specific zones while a video encoding application under low avail- ability can perform better in some other zones. By clustering the requirements of incoming requests according to their workload characteristics and labeling them based on where they perform best, zone preferences for a particular instance characteristics are identified.

The steps to derive such conclusions are composed of creating most popular instance types as clusters, identifying the zones that the clusters perform best and labeling the clusters based on the high-performing availability zones. As a result, the most recent behavior of the zones are learned and used to plan ahead for infrastructure development by the cloud owners.

7.2 Description of setup

In this section, we describe the simulation environment and experimental setting that we developed for evaluating the predictive method for selecting cloud availability zones.

7.2.1 Simulation environment

We used a Java based simulation environment to simulate various availability zones with different characteristics and an inter-arrival process for incoming heterogeneous user requests. Figure 7.5 shows the components of our simulation infrastructure.

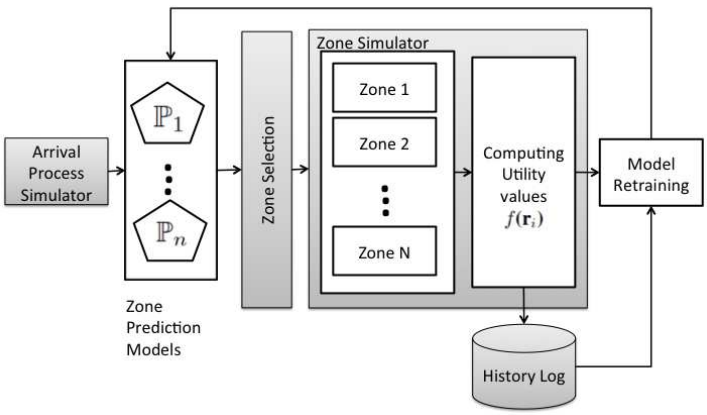


Figure 7.5: Simulation environment.

It has mainly three components. The first component is the *zone simulator* which simulates availability zones with various CPU and memory capacities, re-

liability levels, failure types, internal placement algorithms, supported instance and/or infrastructure types. The second component is the *arrival process generator* where incoming requests with various requirements are simulated based on specified inter-arrival and lifetime distributions. The desired load factor is controlled by the parameters of the inter-arrival distribution in the arrival process simulator. The third component is the *zone selection layer* which uses predicted utility values to select the availability zone that would best satisfy an incoming request. After the zone models are generated, by using the machine learning techniques as explained in Section 7.1, the utility value for an incoming requests is predicted for each zone separately.

After every incoming request generated by the *arrival process* is placed into an availability zone by the *zone selection layer*, the zone simulator checks whether or not the requirements are satisfied. Hence, the implemented satisfaction vector is a binary vector. The request requirements and the associated utility value constitute a training sample for the zone that the request is placed into. In order to generate the training data, the same simulation infrastructure is used except that the zone selector and the prediction models are replaced by a random zone selector as depicted in Figure 7.1. After generating a sufficient number of training data sets for each zone, the training data sets are passed to IBM SPSS Modeler [96], which offers a variety of modeling methods to develop prediction models for the utility value of an incoming request. We generate and compare several prediction models for each zone. The predictive models we consider include Neural Networks, Decision Trees, and Support Vector Machines. Among various models, we choose the one with the highest accuracy.

One other aspect of the zone simulator is that it compares the predicted utility value with the actual utility value observed after the placement. If the difference between actual and predicted values increases, the initial prediction accuracy is reduced. In this case, the models are trained again with the latest data stored in the history log.

7.2.2 Experimental setting

Simulated zone attributes

We simulated 12 availability zones with different capabilities that are listed in Table 7.2. Most of these attributes, especially the ones related to performance, are not published and known a priori (e.g., Reliability levels, failure types, placement algorithms). Each availability zone is composed of 10 racks, each rack contains 2 blade centers and each blade center contains 5 physical machines (PMs). An availability zone is simulated to support a heterogeneous hardware infrastructure, composed of a mix of processor models. The used CPU models are: Intel Xeon

series ”E5507”, ”E5430”, ”E5645”, ”X5500”, ”X5570” and AMD Opteron series ”2218HE”, and ”270”. We define 3 different heterogeneous infrastructure types depending on the percentage of hardware mix in a zone. Table 7.3 shows the exact percentage of CPU model mixes per infrastructure type.

#AZ	Reliability level	Failure type	Placement algorithm	Support for high performance istance(s)	Infrastructure type
1	low	racks	random	High Storage	Type I
2	low	racks	random	High Compute	Type I
3	low	racks	random	High Memory Speed	Type I
4	medium	blades	random	High Storage	Type II
5	medium	blades	random	High Compute	Type II
6	medium	blades	random	High Memory Speed	Type II
7	medium	racks	LeastFull	High Storage	Type III
8	medium	racks	LeastFull	High Compute	Type III
9	medium	racks	LeastFull	High Memory Speed	Type III
10	high	blades	LeastFull	High Network I/O + SSD	Type I
11	high	blades	LeastFull	High Network I/O + SSD	Type II
12	high	blades	LeastFull	High Network I/O + SSD	Type III

Table 7.2: Cloud availability zones attributes

Infrastructure type	Processor brand	CPU model	% of CPU models
Type I	Intel Xeon	E5507	50%
	Intel Xeon	E5430	30%
	Intel Xeon	E5645	20%
Type II	AMD Opteron	2218HE	70%
	AMD Opteron	270	30%
Type III	Intel Xeon	X5500	60%
	Intel Xeon	X5570	40%

Table 7.3: Cloud availability zones attributes

The admission controller monitors the available capacity in all zones. Incoming requests are rejected in a zone only if there is not enough capacity to accommodate the request in the targeted PM for placement. For the sake of simplicity,we simulated each zone with the same CPU and memory capacity with 32 cores and 32 GB memory, respectively. In order to create performance variations, we introduce two types of internal placement algorithms: random and load balancing. For availability zones 1-6, a random placement algorithm where the instances are placed randomly to PMs is implemented. For the rest, a placement algorithm

which balances the load by selecting the least full PM is implemented. As for the unpublished characteristics of the zones,each availability zone is simulated to support different high performance instance types. For instance, high network instances with bandwidth requirement of 4KB performs best in zones 10, 11, and 12 (see Table 7.2). Even though instance types are advertised as a selection to user, their performance per zone is not published by the cloud provider. The failures are simulated such that either entire rack or entire blade center fails with either low or high probabilities. De-pending on the failure probability, blade center/rack failure and availability information, reliability levels are classified as low, medium and high. The mean time between failures is Gamma distributed, while lifetime of failures (i.e., time to recovery) follows a LogNormal distribution.

Simulated request attributes

We generated training data sets by simulating multiple instances of user requests with different attributes. As in the example reported in Section 7.1, the requirement vector of an incoming request has the form$\mathbf{r}_i = [r_S, r_L, r_I, r_A, r_M]$. Each attribute in the vector takes values among those listed in Table 7.4.

The size of the instances is represented by r_S. The smallest instance size is 1 core with 0.6 GB memory, while the largest instance is 16 cores and 15 GB memory. We simulated 5 different instance sizes where the details of other instance sizes are depicted in Table 7.5. Reliability level is denoted by r_L and users are allowed to specify either low, medium or high reliability zones depending on their application type. Users also specify the interruption tolerance that is the sensitivity for being interrupted during their application lifetime. Quantity r_I represents the interruption tolerance level, ranging from 0 to 1. Value 0 is selected for the instances that are very tolerant to interruption, whereas value 1 is selected for the instances that are very sensitive to interruption. Users also indicate the desired instance type. The types of the instances (e.g., high storage, high compute, etc.) that a user can specify are listed in Table 7.4 under the r_A column. Lastly, users indicate their hardware infrastructure preference. r_M in Table 7.4 lists the simulated user hardware choices.

We set the weight vector for utility function as:

$$\mathbf{W}^T = [w_S, w_L, w_I, w_A, w_M], \tag{7.10}$$

where $w_i = 0.2$ for $i \in \{S, L, I, A, M\}$. As the weight vector indicates, for our experimental settings, all the preferences have equal weights. We use normalizing factor of 1.0 for this particular set up that yields the maximum value of utility function, when all the requirements are satisfied as 1, while the minimum value is 0 when the request is rejected.

r_S	r_L	r_I	r_A	r_M
micro	0.2 (low)	0	High Storage	Intel Xeon E5507
small	0.5 (medium)	1	High Compute	Intel Xeon E5430
medium	0.8 (high)		High Memory	Intel Xeon E5645
large			High Network	Intel Xeon E5500
xlarge			SSD	Intel Xeon E5570
				AMD Opteron 270
				AMD Opteron 2218HE

Table 7.4: User request attributes

	$micro$	$small$	$medium$	$large$	$xlarge$
CPU	1	2	4	8	16
Memory (GB)	0.6	1.7	3.5	7.5	15

Table 7.5: Instance sizes (r_S)

Request arrival pattern

We simulated all combinations of 5 user attributes (in Table 7.4) summing up to more than 450 different instance request types with their unique set up. The inter-arrival time of each request type follows an exponential distribution whose parameters depend on the desired load factor (i.e., 95% for our set up), while the request life time follows a uniform distribution. The mix of request types is evenly distributed and this set up is simulated for a 4-week time window.

7.3 Numerical results

In this section, we demonstrate numerical results for different applications of prediction models with the set up described in Section 7.2. First, we show the efficiency of predictive method for selecting availability zones over non-predictive methods then we show deriving insight for availability zones by using prediction models.

7.3.1 Availability zone selection

The placement approach introduced in this chapter does not require a priori knowledge of availability zone properties. In other words, the attributes of zones are not available during placement. Hence, placement is made solely based on the predicted behavior of the availability zones which is learned from past behavior. In this Section, we compare the effectiveness of prediction-based placement policy against policies that perform placement by utilizing the state of the availability

zone and its properties. Our goal is to show that the impact of availability zone features to user satisfaction can be learned, and that placement policies based on the learned behavior perform better.

In our experiments, we compare our *Prediction Based* policy that selects the availability zone with best utility to other policies making optimal zone selections based on attributes other than utility. We compare the policy classes listed below.

1. **Random policy**. Incoming requests are placed to availability zones randomly. This policy is used as a baseline. Any non-random policy that considers the characteristics of the zones and the requirements of the incoming requests is expected to perform better than the random policy.

2. **Load-based policies**.Availability zone selection is made merely based on the utilization of the zone at the time of request arrival. If the zone with the highest available capacity is selected for load balancing, we call the policy *Least-Utilized*. On the other hand, if the zone with the lowest availability is selected for lower energy consumption, we call the policy *Most-Utilized*.

3. **Attribute-based policies**. Zone selection is made by matching some of the advertised attributes of the zone with the preferences of incoming requests. In practice, cloud providers publish supported instance types per availability zone. However, they do not publish the performance of the instance types per availability zone. Even though it is not yet practical to publish the zone performances for instance types, we assume that it is hypothetically available to the user. One example of such a policy, *Match Instance Type* policy, makes a selection based on the advertised performance levels of instance types (see Table 7.2) by each zone. If the candidate zone supports the performance level of the instance types required by the incoming request then it is selected. Similarly, it is not practical for cloud providers to publicize the hardware configuration of their infrastructure hence, the machine types (e.g., CPU models) that instances are running on are not available to the user. For experimental purposes, we assume that such information is available to simulate the case where the machine types are known by the user. The policy, *Match Machine Type*, selects availability zones based on the supported CPU models (see Table 7.3) in different infrastructure types defined in Table 7.2.

Figure 7.6 reports the average utility value obtained by the different policies for best availability zone selection at different levels of availability zone utilization. The expected utility value for our *Prediction Based* policy is close to 0.85 for all utilization levels. This is significantly higher than the expected utility values achieved by load-based and attribute-based policies. The utility values are less

than 0.7 for *Random*, *Least Utilized* and the *Most Utilized* policies, and less than 0.8 for attribute-based policies; *Match Machine Type* and *Match Instance Type*. As expected, *Random* policy achieves the lowest expected utility value. Figure 7.6 shows that learning the behavior of the availability zones based on historical data is better than actually knowing the zone utilization levels and certain zone attributes, such the type of machines or instances supported.

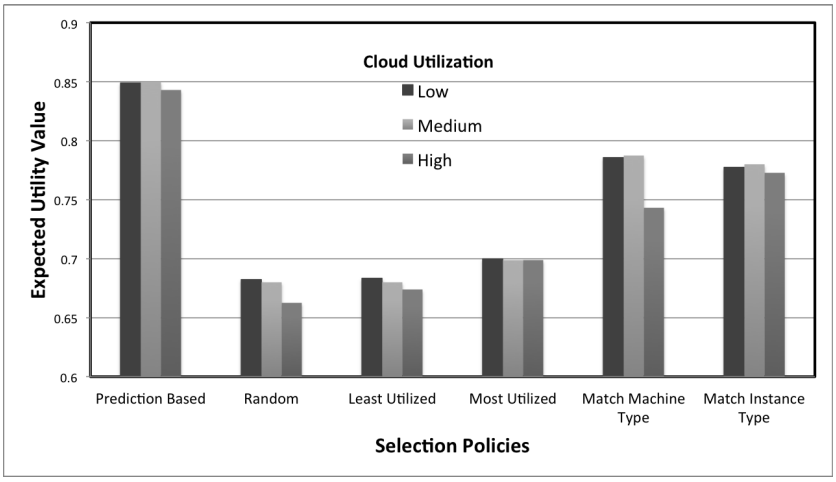


Figure 7.6: Comparison of utility values for different policies.

Figure 7.7 reports the percentage of user requests whose preferred instance types and preferred machine types are supported by the availability zones that they are assigned to. *Match Instance Type* policy satisfies the users application and *Match Machine Type* policy satisfies the users machine type requirements almost all the time. In both cases, the percentage of requests that are matched against their requirements is more than 95%. This is expected, since the *Match Instance Type* and the *Match Machine Type* policies have the prior information about instances and machine types supported by each zone, respectively. Policies other than *Prediction Based* policy match only 30% of the users machine or instance type requirements. As an example, while *Match Instance Type* policy can place to the requests to a zone where the required instance type is satisfied almost 100%, it performs very poor when it comes to matching the machine type requirement. This is because the prior information about the machine types is not available to *Match Instance Type* but it is only available to *Match Machine Type* policy. *Prediction Based* policy, on the other hand, can match both the instance and the machine type requirements significantly higher without any prior knowledge of the distributions of machines and the instance types for availability zones.

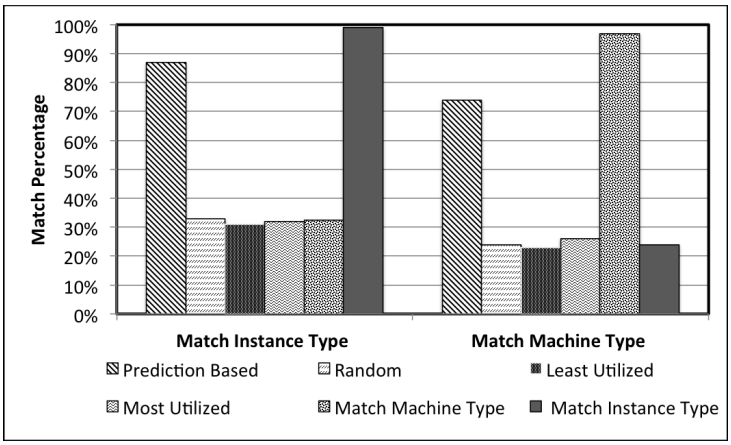


Figure 7.7: Percentage of matches.

Prediction Based policy aims to optimize user satisfaction by selecting the zone with the best utility value while maintaining a high resource utilization. In order to measure the impact of the *Prediction Based* policy to resource utilization, we compare the average zone CPU utilization values under different policies in Figure 7.8. This comparison shows that the *Prediction Based* policy is not reducing the utilization of the zones compared to other policies. Hence, Prediction based policy increases the utility value without under utilizing zone resources. It is seen that average zone utilization for CPU is around 50% for almost all of the policies. The reason for less CPU utilization than offered CPU load is due to high rejection probabilities (0.6) for large and xlarge instances.

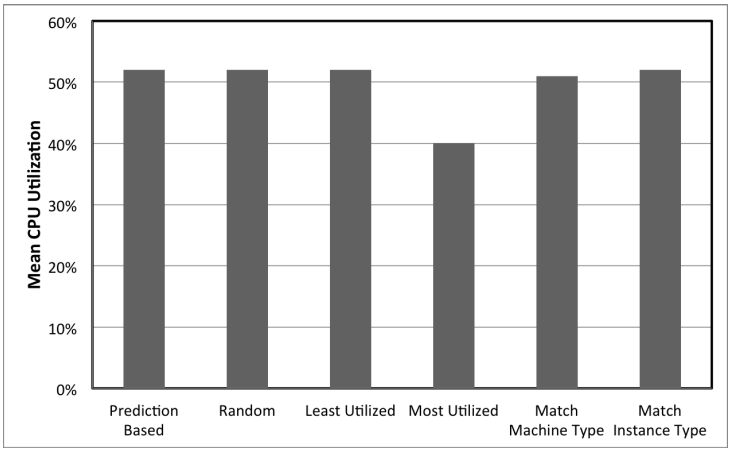


Figure 7.8: Mean cloud utilization

In our simulation, we allow every policy to try placing the request multiple times until it is accepted. Figure 7.9 reports the average number of trials for each policy in order to get accepted after the first one. We see that the number of additional attempts after the initial one for the *Prediction Based* policy is much less than 1. Therefore, the requests are placed to their best matching zones in their first trials most of the time. In contrast to *Prediction Based* policy, *Most Utilized* and *Match Machine Type* policies require almost 4 additional trials on the average. *Random*, *Least Utilized*, *Match Instance Type*, on the other hand, requires more than 2 additional trials on the average. Our *Prediction Based* policy reduces the number of trials significantly compared to other policies. Having less number of trials for placement enables instances to be provisioned faster.

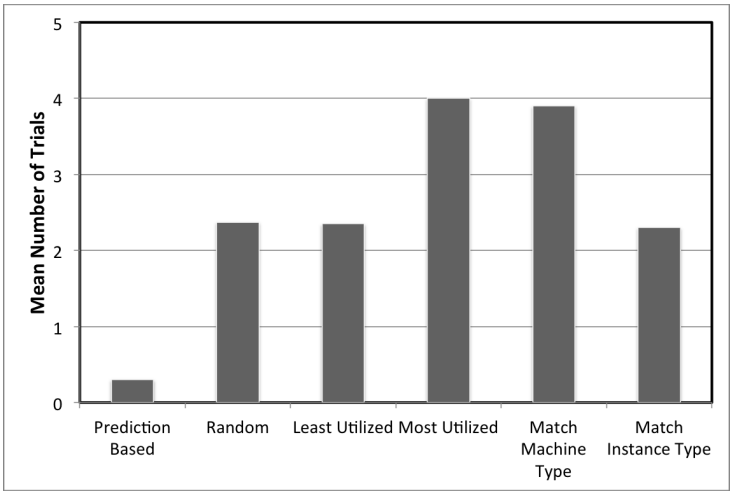


Figure 7.9: Average number of trials.

7.3.2 Deriving insight for availability zones

In order to demonstrate another use of our prediction models, we simply generate a sample cluster of workload and show which zone meets its requirements most by using predicted utility values. High-performance databases such as SAP, Microsoft SharePoint and other enterprise applications with very high reliability constraint can perform a cluster of high performance memory instances with high reliability constraint. Without referring to the inputs of Table 7.3, we generate average predicted utility rates for each zone with the use of prediction models in Figure 7.10.

Figure 7.10 shows relative average utility values per availability zone. By using predicted utility values of this cluster, we conclude that AZ 9 is the best

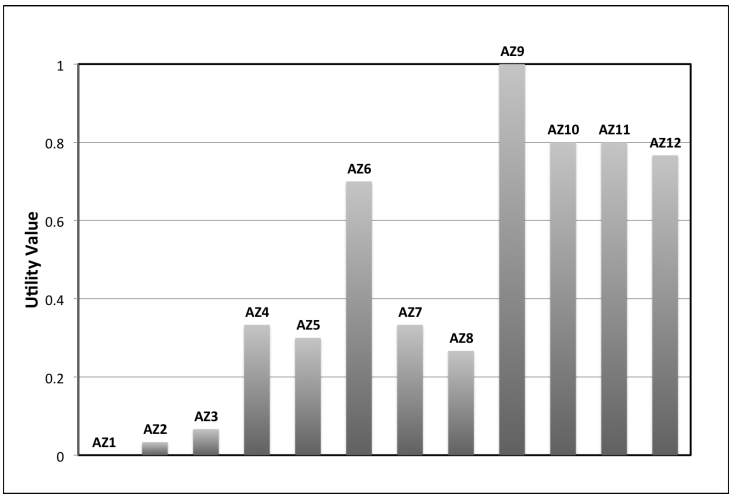


Figure 7.10: Zone preference for high performance memory instances with high reliability requirement.

zone followed by AZ 10, 11, and 12 with a decreasing performance, for satisfying high performance memory instances with high reliability constraint.

Assuming that performance of supported instance types are published, by only looking at the published properties, (see Table 2) AZ 3, 6, and 9 would have been good candidates for deploying high performance memory instances. However, our predicted utility values show that zone 3 is almost one of the weakest zones to satisfy this type of workload requirements due to only meeting instance type requirement but failing to meet the reliability level requirement. The prediction models capture the reliability level of zones from previous deployments and predict a low utility value for zone 3. It is also supported from the simulator that zone 3 is a low reliable zone with high probable rack level failures. Another interesting result that could not be derived from published attributes but can be extracted from our models is even though AZ 10, 11, and 12 are not the optimum zones for high performance memory instances in terms of not meeting the supported instance type, having high reliability on these zones make utility function almost as good as AZ 9 which is a medium level reliable zone supporting a high performance memory instances.

Chapter 8

Conclusion

Large Internet systems require management decisions that should continuously analyze huge volumes of variable and heterogeneous data sets coming from system and network monitors. Understanding the statistical characterization of data sets of each server and the evolution of the system dynamics is crucial for system management. However, the increasing size of monitored data, their high variability and their heterogeneity limit the applicability of existing approaches for performance modeling.

This thesis proposes a set of novel strategies for supporting decisions for system management that address the challenges of volume, variety, and variability typical of data monitored in large Internet systems.

First, we propose a novel strategy for big data analytics that, by taking system monitor data, is able to select the most relevant data sets, statistically characterize their behavior, and to evaluate the dynamics of the overall system. All experiments carried on real data sets demonstrate that the proposed strategy finds useful applications in system management related to modern large Internet systems. Future work in the direction of big data analytics is oriented to the characterization of server behavior and system dynamics through the combination of different metrics and resources.

Second, we present a set of methodologies for the detection of anomalies in modern Internet systems, where monitored data streams refer to heterogeneous and variable system resource measures. In particular, the models proposed in this thesis improve the state of the art in the fields of behavioral anomalies detection and state change detection. The proposed methods combine for the first time online models to achieve a continuously adaptive representation of the data flowing from the system monitors with online statistical tests as anomaly detectors. We demonstrate that the proposed methods are able to improve the performance of existing state of the art anomaly detector applicable at runtime. All experiments carried out on synthetic and real datasets demonstrate that the proposed solutions

are robust and effective: they signals just relevant anomalies with very low number of false detections, and provide the precision of detection for different statistical properties and variability levels of the input data. The proposed methods are characterized by low computational complexity and can be applied to thousands of data flows typical of large Internet systems.

Third, we propose a novel model for detecting correlations that is able to capture the presence of dependency also between highly variable time series as those coming from samples of monitored system resources. It is based on the extraction of the main patterns of the time series, the removal of perturbations, and the selection of just the trend patterns. Moreover, we use the proposed correlation index as similarity measure that can be applied to correlation-based clustering algorithms specifically tailored to the finding of related time series characterized by high variability. Evaluation carried out on datasets characterized by different levels of variability demonstrate that the proposed models improve the state of the art in terms of accuracy and robustness. Our promising results evidence the possibility of using the proposed models as a support for system management applications in all domains characterized by high variability where existing correlation models and clustering algorithms do not work.

Finally, an automated decision making mechanism for cloud availability zone selection is developed to optimize user satisfaction. Our method employs predictive analytics and machine learning techniques to learn the unpublished attributes of cloud availability zones. The models are dynamically updated to reflect the most recent performance changes in availability zones. Then, the optimum selection is made based on the learned behavior of the availability zones. The comparison of our selection method to non-predictive selection methods shows that the predictive approach performs better than the non-predictive ones in terms of user satisfaction, while maintaining high cloud performance. The prediction models can also be used to derive useful insights for availability zones, and can help cloud providers to plan ahead their operations. As future work, we intend to extend our experiments to more complicated workloads and test in a real environment. Another extension of this work, which we wish to pursue, is to investigate the confidence level of the prediction models and its influence on overall decision making results, in terms of user satisfaction and cloud performance.

Bibliography

[1] S. Tosi, S. Casolari, M. Colajanni, Supporting data center management through clustering of system data streams, Proceedings of the 2012 IEEE International Conference on Advanced Infocomm Technology, Paris, France, 2012.

[2] S. Tosi, S. Casolari, M. Colajanni, Data clustering based on correlation analysis applied to highly variable domains, Computer Networks, Elsevier, vol. 57(15), pp. 3025-3038, 2013.

[3] S. Tosi, S. Casolari, M. Colajanni, Detecting correlation between server resources for system management, accepted for publication in Journal of Computer and System Science, Elsevier.

[4] M. Unuvar, S. Tosi, Y. N. Doganata, M. Steinder, A. N. Tantawi, A predictive method for identifying optimum cloud availability zones, submitted to 7th International Conference on Cloud Computing (CLOUD2014).

[5] S. Casolari, M. Colajanni, S. Tosi, Selective resource characterization for evaluation of system dynamics, ACM SIGMETRICS Performance Evaluation Review, vol. 39(4), pp. 51-60, 2012.

[6] S. Casolari, M. Colajanni, S. Tosi, Detecting behavioral variations in system resources of large data centers, in: Proceedings of the 11th IEEE International Conference on Computer and Information Technology (IEEE CIT 2011), Cyprus, 2011.

[7] S. Casolari, F. Lo Presti, S. Tosi, An adaptive model for online detection of relevant state changes in Internet-based systems, Performance Evaluation, Elsevier, vol. 69(5), pp. 206-226, 2012.

[8] M. Andreolini, S. Casolari, S. Tosi, A hierarchical architecture for on-line control of private cloud based systems, in: Proceedings of WWW/Internet2010, Timisoara, Romania, 2010.

[9] S. Casolari, M. Villani, M. Colajanni, R. Serra, Separating internal and external fluctuation in distributed Web-based services, in: Proceedings of European Conference on Complex Systems, Warwick, UK, 2009.

[10] M. Andreolini, M. Colajanni, M. Nuccio, Scalability of contentaware server switches for cluster-based Web information systems, in: Proceedings of 12th International World Wide Web Conference, Budapest, Hungary, 2003.

[11] M. Andreolini, S. Casolari, M. Colajanni, Models and framework for supporting run-time decisions in web-based systems, ACM Transaction on the Web 2 (3), pp. 17:1–17:43, 2008.

[12] V. Cardellini, E. Casalicchio, M. Colajanni, P. Yu, The state of the art in locally distributed Web-server system. ACM Computing Surveys, vol. 34 (2), pp. 263z311, 2002.

[13] N. Tran, D. A. Reed, Automatic ARIMA time series modeling for adaptive I/O prefetching, TPDS, vol. 15(4), 2004.

[14] W. Xu, L. Huang, A. Fox, D. Patterson, M. Jordan, Online System Problem Detection by Mining Patterns of Console Logs, in: Proceedings of the 2009 Ninth IEEE International Conference on Data Mining, pp. 588-597, 2009.

[15] P. S. Vivek, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, E. Nahum, Locality-aware Request Distribution in Cluster-based Network Servers, in: Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 205-216, 1998.

[16] L. Cherkasova, P. Phaal, Session-based admission control: a mechanism for peak load management of commercial Web sites, IEEE Transactions on Computers, vol. 51 (6), pp. 669-685, 2002.

[17] D. A. Menasce, J. O. Kephart, Guest Editors' Introduction: Autonomic Computing, IEEE Internet Computing, vol. 11 (1), pp. 18-21, 2007.

[18] H. Chen, P. Mohapatra, Session-based overload control in QoS-aware Web servers, in: Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 2, pp. 516-524, 2002.

[19] V. Cardellini, M. Colajanni, P. S. Yu, Request redirection algorithms for distributed Web systems, IEEE Transaction on Parallel Distributed Systems, vol. 14 (4), pp. 335-368, 2003.

[20] IBM, P. Zikopoulos, C. Eaton, Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data, McGraw-Hill Osborne Media, 2011.

[21] D. Jeffrey, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, OSDI '04, pp. 137-150, 2004.

[22] Calder et al., Windows Azure Storage: a highly available cloud storage service with strong consistency, in: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, pp. 143-157, New York, NY, USA, 2011.

[23] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The Hadoop Distributed File System, in: IEEE/NASA Goddard Conference on Mass Storage Systems and Technologies, vol. 0, pp. 1-10, 2010.

[24] J. Gantz, D. Reinsel, The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east, IDC iView: IDC Analyze the Future, 2012.

[25] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, Vl2: a scalable and flexible data center network, in: Proceedings of the ACM SIGCOMM 2009 conference on Data communication, New York, NY, USA, pp. 51-62, 2009.

[26] V. Soundararajan and K. Govil, Challenges in building scalable virtualized datacenter management, ACM SIGOPS Operating Systems Review, vol. 44, pp. 95-102, 2010.

[27] B. Jia, T. W. Wlodarczyk, and C. Rong, Performance considerations of data acquisition in hadoop system, IEEE International Conference on Cloud Computing Technology and Science, pp. 545-549, 2010.

[28] X. Zhu et al., 1000 islands: an integrated approach to resource management for virtualized data centers, Cluster Computing, vol. 12(1), 2008.

[29] G. H. Golub, C. F. Loan, Matrix Computations (Johns Hopkins Studies in Mathematical Sciences), The Johns Hopkins University Press, 1996.

[30] A. Sharma, K. K. Paliwal, Fast principal component analysis using fixed-point algorithm, Pattern Recognition Letters, vol. 28(10), pp. 1151-1155, 2007.

[31] A. V. Oppenheim, R. W. Schafer, J. R. Buck, Discrete-time signal processing, Prentice Hall, 1999.

[32] C. Granger, P. Newbold, Forecasting Economic Time Series, Academic Press, 1986.

[33] T. Warren Liao, Clustering of time series data - a survey, Pattern Recognition, 38 (11), pp. 1857–1874, 2005.

[34] C. Kruegel, G. Vigna, Anomaly detection of web-based attacks, in: Proceedings of the 10th ACM conference on Computer and communications security, Washington D.C., USA, 2003.

[35] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM Computing Surveys, 41 (3), pp. 15:1–15:58, 2009.

[36] S. Papadimitriou, J. Sun, P. S. Yu, Local Correlation Tracking in Time Series, IEEE International Conference on Data Mining, Los Alamitos, CA, USA, 2006.

[37] J. Cohen, Applied multiple regression/correlation analysis for the behavioral sciences, L. Erlbaum Associates, 2003.

[38] F. Gorunescu, Data Mining: Concepts, Models and Techniques, Springer, 2011.

[39] M.G. Kendall, Rank correlation methods, Charles Griffin & Company Ltd., 1962.

[40] Q. Zhang, A. Riska, W. Sun, E. Smirni, G. Ciardo, Workload-Aware Load Balancing for Clustered Web Servers, IEEE Transaction on Parallel and Distributed Systems, 16 (3), pp. 219-233, 2005.

[41] S. Ghosh, M.S. Squillante, Analysis and control of correlated web server queues, Computer Communications, 27 (18), pp. 1771–1785, 2004.

[42] M. Dahlin, Interpreting Stale Load Information, IEEE Transaction on Parallel and Distributed Systems, 11 (10), pp. 1033–1047, 2000.

[43] A. Buda, A. Jarynowski, Life-time of correlations and its applications, Wydawnictwo Niezalezne, 2010.

[44] W. Willinger, D. Alderson, L. Li, A pragmatic approach to dealing with high-variability in network measurements, in: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, Taormina, Italy, 2004.

[45] D. C. Montgomery, Introduction to Statistical Quality Control, John Wiley and Sons, 2008.

[46] S. G. Mallat, A Theory of Multiresolution Signal Decomposition: The Wavelet Decomposition, IEEE Transaction on Pattern Analysis and Machine Intelligence, 11 (7), pp. 674–693, 1989.

[47] S. Papadimitriou, P.S. Yu, Optimal multi-scale patterns in time series streams, in: Proceedings of the 2006 ACM SIGMOD international conference on Management of data, Chicago, USA, 2006.

[48] S. Papadimitriou, S. Jimeng, C. Faloutsos, Streaming Pattern Discovery in Multiple Time-Series, in: Proceedings of the 31st International Conference on very large data bases, Trondheim, Norway, 2005.

[49] M. N. Nounou, B. Bakshi, On-Line Multiscale Filtering of Random and Gross Errors without Process Models, American Institute of Chemical Engineers Journal, 45 (5), pp. 1041-1058, 1999.

[50] B.R. Bakshi, Multiscale PCA with application to multivariate statistical process monitoring, AIChE Journal, 44 (7), pp. 1596-1610, 1998.

[51] B. Abrahao, A. Zhang, Characterizing application workloads on cpu utilization in utility computing, Technical Report HPL-2004-157, Hewllet-Packard Labs, 2004.

[52] R. Khattree, D. N. Naik, Multivariate data reduction and discrimination with SAS software, SAS Institute Inc., 2000.

[53] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, N. Taft, Structural analysis of network traffic flows, in: Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems, New York, USA, 2004.

[54] H. E. Hurst, Long-term storage capacity of reservoirs, Transaction of the American Society of Civil Engineers, 116, pp. 770-799, 1951.

[55] R. Weron, Estimating long range dependence: finite sample properties and confidence intervals, Physica A, 312 (1-2), pp. 285-299, 2002.

[56] M. Dobber, R. Van det Mei, G. Koole, A prediction method for job runtimes in shared processors: Survey, statistical analysis and new avenues", Performance Evaluation, 64 (7-8), pp. 755-781, 2007.

[57] B.L. Brockwell, R.A. Davis, Time Series: Theory and Methods, Springer-Verlag, 1987.

[58] B. Yang, Projection approximation subspace tracking, IEEE Transaction on Signal Processing, 43 (1), pp. 95-107, 1995.

[59] A.V. Oppenheim, R.W. Schafer, J.R. Buck, Discrete-time signal processing, Prentice Hall, 1999.

[60] G.E. Box, G.M. Jenkins, G.C. Reinsel, Time Series Analysis Forecasting and Control, Prentice Hall, 1994.

[61] H. Kargupta, K. Sivakumar, S. Ghosh, Dependency Detection in MobiMine and Random Matrices, in: Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery, London, UK, 2002.

[62] Z. Liu, M. Squillante, C. Xia, S. Yu, L. Zhang, Profile-based Traffic Characterization of Commercial Web Sites, in: Proceedings of the 18th International Teletraffic Congress, Berlin, Germany, 2003.

[63] J. Erman, M. Arlitt, A. Mahanti, Traffic classification using clustering algorithms, in: Proceedings of the 2006 SIGCOMM Workshop on Mining network data, New York, USA, 2006.

[64] H. Wang, W. Wang, J. Yang, P.S. Yu, Clustering by pattern similarity in large data sets, in: Proceedings of the 2002 ACM SIGMOD international conference on Management of data, Madison, Wisconsin, 2002.

[65] B. Hay, G. Wets, K. Vanhoof, Clustering navigation patterns on a website using a Sequence Alignment Method, in: Proceedings of 17th International Joint Conference on Artificial Intelligence, Washington, USA, 2001.

[66] M. Steinbach, G. Karypis, V. Kumar, A Comparison of Document Clustering Techniques, in: KDD Workshop on Text Mining, 2000.

[67] J. B. MacQueen, Some Methods for Classification and Analysis of Multi-Variate Observations, in: Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, 1, pp. 281-297, 1967.

[68] J. Cohen, P. Cohen, S.G. West, L.S. Aiken, Applied multiple regression/correlation analysis for the behavioral sciences, L. Erlbaum Associates, 2003.

[69] C. Spearman, The proof and measurement of association between two things, The American Journal of Psychology, 100 (3-4), pp. 441-471, 1904.

[70] S. Papadimitriou, J. Sun, P. S. Yu, Local Correlation Tracking in Time Series, IEEE International Conference on Data Mining, Los Alamitos, USA, 2006.

[71] S.G. Mallat, A Theory of Multiresolution Signal Decomposition: The Wavelet Decomposition, IEEE Transanction on Pattern Analysis and Machine Intelligence, 11 (7), pp. 674-693, 1989.

[72] P. Barford, M. Crovella, Generating representative Web workloads for network and server performance evaluation, SIGMETRICS Performormance Evaluation Review, 26 (1), pp. 151-160, 1998.

[73] M. Crovella, A. Bestavros, Self-similarity in World Wide Web traffic: evidence and possible causes, IEEE/ACM Transactions on Networking, 5 (6), pp. 835-846, 1997.

[74] M.N. Bennani, D.A. Menasce, Assessing the Robustness of Self-Managing Computer Systems under Highly Variable Workloads, in: Proceedings of the First International Conference on Autonomic Computing, Washington, USA, 2004.

[75] W. Willinger, D. Alderson, L. Li, A pragmatic approach to dealing with high-variability in network measurements, in: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, Taormina, Italy, 2004.

[76] A. Buda, A. Jarynowski, Life-time of correlations and its applications, Wydawnictwo Niezalezne, 2010.

[77] T. Sørensen, A Method of Establishing Groups of Equal Amplitude in Plant Sociology Based on Similarity of Species Content, Biologiske Skrifter, E. Munksgaard, 1948.

[78] S. Papadimitriou, P.S. Yu, Optimal multi-scale patterns in time series streams, in: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, Chicago, USA, 2006.

[79] S. Papadimitriou, S. Jimeng, C. Faloutsos, Streaming Pattern Discovery in Multiple Time-Series, in: Proceedings of the 31st International Conference on very large data bases, Trondheim, Norway, 2005.

[80] A. K. Jain, R. C. Dubes, Algorithms for clustering data, Prentice-Hall, Inc., 1988.

[81] B. Abrahao, A. Zhang, Characterizing application workloads on cpu utilization in utility computing, Technical Report HPL-2004-157, Hewlett-Packard Labs, 2004.

[82] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, N. Taft, Structural analysis of network traffic flows, in: Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems, New York, USA, 2004.

[83] H. E. Hurst, Long-term storage capacity of reservoirs, Transaction of the American Society of Civil Engineers, 116, pp. 770-799, 1951.

[84] R. Weron, Estimating long range dependence: finite sample properties and confidence intervals, Physica A, 312 (1-2), pp. 285-299, 2002.

[85] M. Dobber, R. Van det Mei, G. Koole, A prediction method for job runtimes in shared processors: Survey, statistical analysis and new avenues, Performance Evaluation, 64 (7-8), pp. 755-781, 2007.

[86] B.L. Brockwell, R.A. Davis, Time Series: Theory and Methods, Springer-Verlag, 1987.

[87] Abilene network, http://abilene.internet2.edu/

[88] D.L. Olson, D. Delen, Advanced Data Mining Techniques, Springer, 2008.

[89] K. Lan, J. Heidemann, On the correlation of Internet flow characteristics, Technical Report ISI-TR-574, USC/Information Sciences Institute, 2003.

[90] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, Classification and Regression Trees, CRC Press, New York, 1999.

[91] R. Buyya, R. Ranjan, and R. Calheiros, Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services, C.-H. Hsu, L. Yang, J. Park, and S. S. Yeo, editors, Algorithms and Architectures for Parallel Processing, vol. 6081 of Lecture Notes in Computer Science, pp. 13-31, Springer Berlin Heidelberg, 2010.

[92] 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(1), pp. 23-50, 2011.

[93] S. Chaisiri, B.-S. Lee, and D. Niyato, Optimal virtual machine placement across multiple cloud providers, Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific, pp. 103-110, 2009.

[94] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. D. Bowers, and M. M. Swift, More for your money: exploiting performance heterogeneity in public clouds, in: Proceedings of the Third ACM Symposium on Cloud Computing, SoCC '12, vol. 20, pp. 1-14, New York, NY, USA, 2012.

[95] I. Houidi, M. Mechtri, W. Louati, and D. Zeghlache, Cloud service delivery across multiple cloud platforms, in: 2011 IEEE International Conference on Services Computing (SCC), pp. 741-742, 2011.

[96] ibm.com/software/analytics/spss/products/modeler/, Ibm spss modeler overview, 2012.

[97] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, vol. 2(12), pp. 1137-1143, 1995.

[98] A. Li, X. Yang, S. Kandula, and M. Zhang, Cloudcmp: comparing public cloud providers, in: Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, IMC'10, pp. 1-14, New York, NY, USA, 2010.

[99] A. Li, X. Yang, S. Kandula, and M. Zhang, Comparing public-cloud providers, Internet Computing, IEEE, vol. 15(2), pp. 50-53, 2011.

[100] W. Li, J. Tordsson, and E. Elmroth, Modeling for dynamic cloud scheduling via migration of virtual machines, in: Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), pp. 163-171, 2011.

[101] J. L. Lucas Simarro, R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, Dynamic placement of virtual machines for cost optimization in multi-cloud environments, in: 2011 International Conference on High Performance Computing and Simulation (HPCS), pp. 1-7, 2011.

[102] B. Ofer, E. Hadad, E. K. Kolodmer, D. H. Lorenz, and Y. Moatti, Optimized placement of virtual machines in a network environment, June 6 2013. US Patent 20.130.145.368.

[103] Z. Ou, H. Zhuang, J. K. Nurminen, A. Yla-Jaaski, and P. Hui, Exploiting hardware heterogeneity within the same instance type of Amazon EC2, in: USENIX conference on Hot Topics in Cloud Computing, 2012.

[104] J. Schad, J. Dittrich, and J.-A. Quiane-Ruiz, Runtime measurements in the cloud: observing, analyzing, and reducing variance, in: Proceedings of the VLDB Endowment, vol. 3(1-2), pp. 460-471, 2010.

[105] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers, Future Generation Computer Systems, vol. 28(2), pp. 358-367, 2012.

[106] Z. ur Rehman, F. Hussain, and O. Hussain, Towards multi-criteria cloud service selection, in: 2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), pp. 44-48, 2011.

[107] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads, in: Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD), pp. 228-235, 2010.

[108] A. J. Oliner, A. Aiken, Online detection of multi-component interactions in production systems, in: International Conference on Dependable Systems and Networks, vol. 0, pp. 49-60, 2011.

[109] M. Argollo de Menezes, A. L. Barabasi, Separating Internal and External Dynamics of Complex Systems, Physical review letters, vol. 93(6), 2004.

[110] H. Abdi, L. J. Williams, Principal component analysis, Computational Statistics, vol. 2, pp. 433-459, 2010.

[111] M. Greenacre, Correspondence analysis in practice, Chapman & Hall/CRC, 2007.

[112] K.V. Mardia, J. T. Kent, J. M. Bibby, Multivariate Analysis, Probability and Mathematical Statistics, Academic Press, 1995.

[113] D. D. Lee, H.S. Seung, Learning the parts of objects by non-negative matrix factorization, Nature, vol. 401(6755), 1999.

[114] A. Hyvärinen, E. Oja, Independent Component Analysis: algorithms and applications, Neural Networks, Elsevier Science, vol 13(4-5), 2000.

[115] R. Van Der Krogt, J. Feldman, J. Little, D. Stynes, An integrated business rules and constraints approach to data centre capacity management, in: Proceedings of the 16th international conference on Principles and practice of constraint programming, Springer-Verlag, pp. 568-582, 2010.

[116] L. Xiaofei, J. Hai, Y. Xiaojie, ESPM: An optimized resource distribution policy in virtual user environment, Future Generation Computer Systems, Springer-Verlag, pp. 1393-1402, 2010.

[117] N. Singh, S. Rao, Energy Optimization Policies for Server Clusters, in: 6th Annual IEEE Conference on Automation Science and Engineering, 2010.

[118] D. Gmach, J. Rolia, L. Cherkasova, A. Kemper, Workload Analysis and Demand Prediction of Enterprise Data Center Applications, in: Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization, 2007.

[119] Y. Baryshnikov, E. Coffman, G. Pierre, D. Rubenstein, M. Squillante, T. Yimwadsana, Predictability of Web server traffic congestion, in: Proceedings of the 10th International Workshop on Web Content Caching and Distribution, Sophia Antipolis, FR, 2005.

[120] H. Hotelling, Analysis of a complex of statistica variables into principal components, Journal of Educational Psychology, vol. 24(7), 1933.

[121] L. Anukool, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, N. Taft, Structural analysis of network traffic flows, in: Joint International Conference on Measurement and Modeling of Computer Systems, 2004.

[122] I. Jolliffe, Principal Component Analysis, Encyclopedia of Statistics in Behavioral Science, 2005.

[123] R. B. Cattell, The scree test for the number of factors, Multivariate behavioral research,Psychology Press, 1966.

[124] H. F. Kaiser, An index of factorial simplicity, Psychometrica, vol. 39(1), 1974.

[125] J. E. Jackson, A user's guide to principal components, Wiley series in probability and mathematical statistics: Applied probability and statistics, Wiley, 1991.

[126] J. E. Gentle, Computational Statistics, Statistics and Computing, Springer, 2009.

[127] C. Chatfield, The Analysis of Time Series: An Introduction, Chapman and Hall, 1989.

[128] R. Gnanadesikan, M. B. Wilk, Probability plotting methods for the analysis of data, Biometrika, vol. 55(1), 1968.

[129] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, A. Kemper, An integrated approach to resource pool management: Policies, efficiency and quality metrics. in: Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN), 2008.

[130] G. Pacifici, W. Segmuller, M. Spreitzer, A. Tantawi, CPU demand for web serving: Measurement analysis and dynamic estimation, Performance Evaluation, vol. 65(6-7), 2008.

[131] S. Papadimitriou, J. Sun, C. Faloutsos, Streaming Pattern Discovery in Multiple Time-Series, VLDB, pp. 697-708, 2005.

[132] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking, IEEE Transactions on Signal Processing, vol. 50(2), 2002.

[133] C. K. Chui and G. Chen, Kalman filtering with real-time applications, Springer-Verlag New York, Inc., 1987.

[134] R. Y. Rubinstein and D. P. Kroese, Simulation and the Monte Carlo Method.

[135] F. Gustafsson, Adaptive Filtering and Change Detection, John Wiley and Sons, 2000.

[136] D. F. Allinger and S. K. Mitter, New results in innovations problem for nonlinear filtering, Stochastics, vol. 4, 1991.

[137] M. Basseville and I. Nikiforov, Detection of Abrupt Changes:Theory and Application. Prentice-Hall, 1993.

[138] E. S. Page, Estimating the point of change in a continuous process, Biometrika, vol. 44, 1957.

[139] J. Rolia, L. Cherkasova, M. Arlitt, A. Andrzejak, A capacity management service for resource pools, in: Proceedings of the 5th international workshop on Software and performance, New York, NY, USA, 2005.

[140] S. Casolari, F. Lo Presti, and S. Tosi, Real-time models supporting management decisions in highly variable systems, in Proceedings of the 29th IEEE International Performance, Computing and Communications Conference, 2010.

[141] P. Dinda and D. O'Hallaron., Host load prediction using linear models, Cluster Computing, vol. 3(4), pp. 265-280, 2000.

[142] E. Hartikainen and S. Ekelin, Enhanced network-state estimation using change detection, in: Proceedings of the 31st IEEE Conference on Local Computer Networks, 2006.

[143] V. Chandola, A. Banerjee, and V. Kumar, Anomaly detection: survey, ACM Computing Surveys, 2009.

[144] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang, Power and performance management of virtualized computing environments via lookahead control, in: Proceedings of the International Conference on Autonomic Computing, pp. 3 -12, 2008.

[145] N. Brenner and C. Rader, A new principle for fast fourier transformation, IEEE Acoustics, Speech & Signal Processing, vol. 24, pp. 264-266, 1976.

[146] S. G. Mallat, A theory of multiresolution signal decomposition: The wavelet decomposition, IEEE Transaction on Pattern Analysis and Machine Intelligence, vol. 11(7), 1989.

[147] D. L. Donovo, I. Johnstone, G. Kerkyacharian, and D. Picard, Wavelet shrinkage: Asymptotia?, Journal of the Royal Statistical Society, vol. 57(2), 1995.

[148] D. Lu, P. Mausel, E. Brondizio, and E. Moran, Change detection techniques, International Journal of Remote Sensing, 2004.

[149] N. A. Macmillan and C. D. Creelman, Detection Theory: a User's Guide. Lawrence Erlbaum Associates, 2005.

[150] S. Casolari, M. Colajanni, and F. Lo Presti, Runtime state change detector of computer system resources under non stationary conditions, in: Proceedings of 17th Interantional Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2009.

[151] A. S. Willsky and H. L. Jones, A generalized likelihood ratio approach to the detection and the estimation of jumps in linear systems, IEEE Transaction on Data and Knowledge Engineering, vol. 14(2), 2002.

[152] H. W. Cain, R. Rajwar, M. Marden, and M. H. Lipasti, An architectural evaluation of Java TPC-W, in: Proceedings of the 7th Symposium on High Performance Computer Architecture (HPCA2001), Monterrey, ME, 2001.

[153] TPC-W transactional Web e-commerce benchmark, 2004.

[154] G. Moustakides, Optimal stopping times for detecting changes in distribution, The Annals of Statistics, vol. 14(4), 1986.

[155] N. Vaswani, Additive change detection in nonlinear systems with unknown change parameters, IEEE Transactions on Signal Processing, vol. 55(3), 2007.

[156] D. Siegmund, Sequential Analysis. Tests and Confidence Intervals.

[157] M. Kendall and J. Ord, Time Series.Oxford University Press, 1990.

[158] V. Jaconson, Congestion avoidance and control, in: Proceedings of SIGCOMM'88, vol. 21, Stanford, CA, 1988.

[159] C. K. Chui, An Introduction to Wavelets. Academic Press, 1992.

[160] G. Khanna, K. Beaty, G. Kar, and A. Kochut, Application performance management in virtualized server environments, in: Proceedings of Network Operations and Management Symposium, 2006.

[161] N. Bobroff, A. Kochut, and K. Beaty, Dynamic placement of virtual machines for managing sla violations, in: Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management, 2007.

[162] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, Black-box and gray-box strategies for virtual machine migration, in: Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation, 2007.

[163] M. Severo and J. Gama, Change detection with kalman filter and cusum, in: Proceedings of 9th International Conference on Discovery Science, 2006.

[164] D. L. Olson and D. Delen, Advanced Data Mining Techniques. Springer, 2008.

[165] M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, and K. Walker, Agile application-aware adaptation for mobility, in: Proceedings of the 16th ACM International Symposium on Operating Systems Principles, 1997.

[166] M. Dobber, R. Van det Mei, and G. Koole, A prediction method for job runtimes in shared processors: Survey, statistical analysis and new avenues, Performance Evaluation, 2007.

[167] B. L. Brockwell and R. A. Davis, Time Series: Theory and Methods. Springer-Verlag, 1987.

[168] L. J. Heyer, S. Kruglyak, and S. Yooseph, Exploring expression data: Identification and analysis of coexpressed genes, Genome Research, 1999.