J. Parallel Distrib. Comput. 79–80 (2015) 67–79

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Adaptive, scalable and reliable monitoring of big data on clouds

Mauro Andreolini [*,1], Michele Colajanni [2], Marcello Pietri [2], Stefania Tosi [2]

University of Modena and Reggio Emilia, Italy

HIGHLIGHTS

- Real time monitoring of cloud resources is crucial for system management.
- We propose an adaptive algorithm for scalable and reliable cloud monitoring.
- Our algorithm dynamically balances amount and quality of monitored time series.
- We reduce monitoring costs significantly without penalizing data quality.

ARTICLE INFO

Article history:
Received 15 November 2013
Received in revised form
8 May 2014
Accepted 18 August 2014
Available online 26 August 2014

Keywords:
Adaptivity
Monitoring
Cloud computing
Big data
Scalability

ABSTRACT

Real-time monitoring of cloud resources is crucial for a variety of tasks such as performance analysis, workload management, capacity planning and fault detection. Applications producing big data make the monitoring task very difficult at high sampling frequencies because of high computational and communication overheads in collecting, storing, and managing information. We present an adaptive algorithm for monitoring big data applications that adapts the intervals of sampling and frequency of updates to data characteristics and administrator needs. Adaptivity allows us to limit computational and communication costs and to guarantee high reliability in capturing relevant load changes. Experimental evaluations performed on a large testbed show the ability of the proposed adaptive algorithm to reduce resource utilization and communication overhead of big data monitoring without penalizing the quality of data, and demonstrate our improvements to the state of the art.

1. Introduction

An increasing number of applications deployed over the cloud operates on big data that we consider a collection of data sets so large and complex that it becomes difficult to gather, store, analyze and visualize through traditional approaches [19,23]. To effectively manage large-scale data centers and cloud systems, operators must understand the behavior of systems and applications behavior producing big data. This requires continuous real-time monitoring integrated with on-line analyses that can be related to performance, prediction, anomaly detection and SLA satisfaction. In similar contexts, the monitoring system presents all the features that are typical of an application producing and working on big data: volume, variety, velocity, veracity (the so called "4 Vs" of IBM scientists). Hence, a key challenge of a scalable monitoring infrastructure is to balance the monitoring and analysis costs incurred with the associated delays, against the benefits attained from identifying and reacting timely to undesirable or non-performing system states such as load spikes.

Previous attempts at reducing the overhead of a real-time monitoring infrastructure present several drawbacks that make them inapplicable to a context of cloud-based applications handling big data. Some proposals for reducing the data set dimension operate on the whole time series (e.g., [8,32]). Others use fixed sampling intervals and do not consider that in highly heterogeneous systems the statistical characteristics of the monitored time series change [12,4]. Another class of work focuses on specific classes of performance indexes and it is not generalizable (e.g., [15]). Finally, there are well designed architectures that do not scale to the volume of data requested by big data applications [6].

This paper introduces a novel real-time adaptive solution for scalable and reliable monitoring of applications producing big data. It strives to achieve this goal by reducing the amount of monitoring

[*] Corresponding author.
E-mail addresses: mauro.andreolini@unimore.it (M. Andreolini), michele.colajanni@unimore.it (M. Colajanni), marcello.pietri@unimore.it (M. Pietri), stefania.tosi@unimore.it (S. Tosi).
[1] Department of Physics, Computer Science and Mathematics, Modena 41125, Italy.
[2] Department of Engineering "Enzo Ferrari", Modena 41125, Italy.

http://dx.doi.org/10.1016/j.jpdc.2014.08.007
0743-7315/© 2014 Elsevier Inc. All rights reserved.

data produced. By adapting sampling intervals to continuously changing data characteristics, it reduces computational and communication costs without a penalization on the reliability of monitored data. The main idea that drives the algorithm adaptivity is simple. When the system behavior is relatively stable, our solution settles for large sampling intervals so that the quantity of data that is gathered and sent for further analysis is reduced. When significant differences between samples occur, the sampling interval is reduced so to capture relevant changes in system performance. The proposed solution automatically chooses the best settings for monitoring parameters, and it updates such settings so to adapt to data characteristics. Moreover, monitoring settings are adapted to the preference of the administrator.

The proposed algorithm gives system administrators the possibility of choosing the best trade-off between reducing computational and communication overhead and preserving the reliability of monitored data. However, this reduction comes at the cost of penalizing the reliability of monitored data because discarding samples keeps the monitoring overhead low but limits the possibility of capturing load changes promptly. Moreover, monitoring solutions should adapt to the frequent changes in the statistical characteristics of monitored datasets. An effective monitoring solution has to support dynamic data acquisition from heterogeneous sources, and to be adaptive to data characteristics and operational needs [27].

This work extends our preliminary findings published in [22] in three directions. We formulate the problem of real-time monitoring in the big data field, where requirements of scalability and reliability are mandatory. We improve the definitions of the proposed adaptive monitoring solution and of its parameters, with detailed descriptions of the algorithm phases. We add an extensive evaluation of the algorithm performance and a comprehensive comparison with respect to state-of-the-art solutions. Experiments show that the proposed adaptive algorithm is able to improve the ability of capturing relevant load changes in up to 55% more than static solutions; in our experiments, the misdetection of load spikes has been also very low (less than 5%). Implementations of adaptive versions for existing solutions do not achieve the performance of our proposal, that benefits an effective tuning of monitoring parameters according to data characteristics and administrator preferences. These results represent a major improvement with respect to the state-of-the-art techniques which either are reliable and resource intensive or tend to be highly scalable by worsening the reliability of sampled data [12,19,13,32].

The remainder of this paper is organized as follows. Section 2 defines the problem of real-time monitoring for big data on clouds. Section 3 presents the proposed adaptive monitoring algorithm. Section 4 introduces the experimental testbed used for the evaluations. Section 5 analyzes experimental results achieved on real scenarios involving big data applications. Section 6 compares our proposal against the state-of-the-art monitoring solutions. Section 7 concludes the paper with some final remarks.

2. Problem definition

In large data centers hosting big data applications the only way to build a scalable monitoring infrastructure is to reduce the amount of monitoring data without sacrificing its statistical properties that are at the basis of any post-gathering analysis. Any monitoring algorithm can be characterized according to this trade-off. To this purpose, we introduce two parameters.

The first parameter G (*Gain*) is defined as one minus the ratio between the number of samples collected by the considered monitoring algorithm and the number of samples collected by the baseline monitoring algorithm that samples data at the highest possible frequency t^0 (e.g., 1 s). Both monitoring algorithms are supposed to operate over the same time interval that must be sufficiently long to be statistically relevant. G assumes values in the [0, 1] interval. Higher values of G denote algorithms aiming to reduce the computational and communication overhead due to monitoring.

$$G = 1 - \frac{N(t)}{N(t^0)}. \tag{1}$$

The second parameter, Q (*Quality*) quantifies the ability of an algorithm to accurately represent load changes in system resources (e.g., load spikes). A comprehensive metrics for estimating Q must take into account two factors: the error introduced by monitors using sampling intervals larger than t^0 (that is, the distance between the original monitored dataset and the reduced one) and the ability to evidence load spikes in the monitored dataset. For this reason, we define Q as a combination of the *NRMSE* (Normalized Root Mean Square Error) [10], and the *Fmeasure* as the weighted average of precision and recall in spike detection [30]:

$$Q = \frac{\mathit{Fmeasure} + (1\text{-}\mathit{NRMSE})}{2}, \tag{2}$$

where *Fmeasure* and *NRMSE* take values $\in [0, 1]$. Q assumes values in the [0, 1] interval. A further motivation for combining two parameters into Q instead of one is due to the fact that datasets in the considered scenarios are highly variable. As stated in [25], the *NRMSE* measure alone is unable to guarantee an accurate quality measure when the statistic characterization of the dataset is highly variable. For this reason, we integrate *NRMSE* with *Fmeasure*, that measures the ability of the monitoring algorithm to identify significant load spikes.

$$\mathit{Fmeasure} = \frac{2 \cdot \mathit{precision} \cdot \mathit{recall}}{\mathit{precision} + \mathit{recall}}. \tag{3}$$

As detailed in [7], *recall* is the fraction of spike detections that are successfully identified, while *precision* is the fraction of relevant detections over the total number of spike detections. In this paper, we consider as a "false positive" (FP) a load spike detected by the monitoring algorithm when the original time series does not exhibit one. This can happen because a generic monitoring algorithm modifies the original time series and can accidentally introduce load spikes in the representation of system load. It can happen that precision is lower than 1. By combining the two metrics, *Fmeasure* gives a global estimation of the detection quality. An *Fmeasure* value close to 1 denotes a good detection quality, while it is lower for algorithms with worse capability in capturing load changes.

The trade-off of a monitoring algorithm can be expressed as a weighted mean of G and Q through the E parameter (*Evaluation*):

$$E = w \cdot G + (1 - w) \cdot Q, \tag{4}$$

where $w \in (0, 1)$ is a tuning constant chosen by the administrator. As the amount of saved data impacts on the quality of the representation, we must allow the system administrator to decide how to regulate the trade-off between overhead reduction and information reliability. Values of $w > 0.5$ put more emphasis on G than on Q; the opposite is true for $w < 0.5$; while $w = 0.5$ gives equal importance to both parameters.

Existing monitoring methods (e.g., [12,15]) collect data at fixed sampling intervals and forward new information to analysis modules only if it differs from the previous collected one by some static numeric threshold. Although this approach can achieve high values of G by reducing the amount of gathered and transmitted data, it may lead to highly inaccurate results and very low Q values due to the missing of most of spikes in data. As an example, Fig. 1 reports two scenarios. Fig. 1 offers a detailed

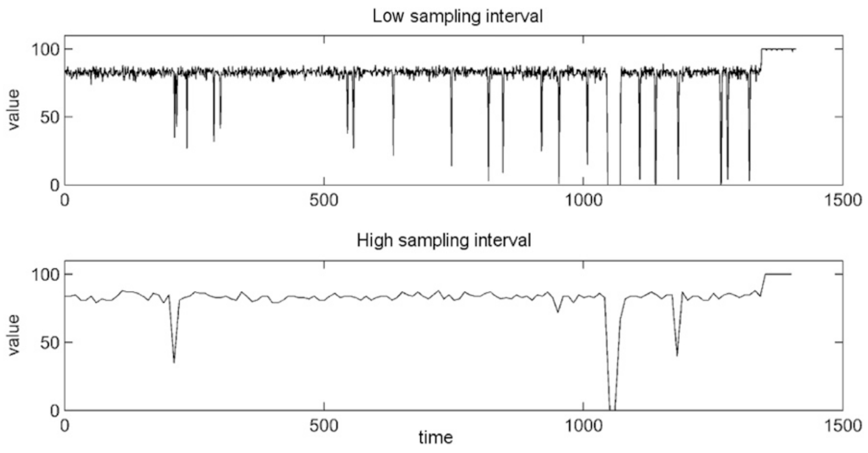


Fig. 1. Resource state representation for methods using low and high sampling intervals.

representation of the system behavior where samples are gathered at the lowest possible sampling interval ($t^0 = 1$ s in the reported example): here, we have maximum quality combined with very high overhead (i.e., low G and high Q values). The bottom figure denotes a system representation where overhead savings are preferred with respect to information quality: here, using a high sampling interval reduces the amount of collected data but at the same time it loses evidence of the majority of load spikes (i.e., high G and low Q values).

Even though there is no single best value of E for any context, the example points out the need of a solution that is flexible enough to adapt its monitoring parameters to data characteristics, so to preserve the most useful information of the real system traces but also to reduce the amount of collected data. Moreover, it is mandatory for the monitoring solution to adapt the best choice of its parameters in an autonomic way. The reason is that in cloud environments monitored data flows are heterogeneous, typically highly variable, and the volume is typically huge. These issues make it impossible to anticipate the best monitoring parameters settings for each scenario, and make it impracticable to rely on human intervention. Hence, any static choice of the threshold values and/or sampling intervals risks to be a significant source of errors. In order to guarantee the best trade-off between mitigation of monitoring overhead and information quality, we propose a scalable and reliable real-time monitoring algorithm that adapts its parameter settings to the characteristics of monitored data and to administrator preferences.

3. Adaptive monitoring

The real-time adaptive monitoring algorithm proposed in this paper consists of two phases: a *training phase* for the evaluation of the best parameters setting for monitoring and an *adaptive monitoring phase* that is the core algorithm of the proposed monitoring solution. In Section 3.1, we define the parameters that are used by the two phases of the algorithm, which are described in Sections 3.2 and 3.3.

3.1. Parameters definition

The real-time adaptive algorithm analyzes monitored data and distinguishes periods of relative stability from periods of high variability. The idea is to reduce the quantity of monitored data when a resource is relatively stable, and to increase it during periods of high variability. In this way, we limit the computational and communication overhead, and at the same time we guarantee that important system changes are not missed. This algorithm operates by dynamically setting two key variables: sampling interval t, and variability Δ. The sampling interval t determines the time interval that elapses from the collection of two consecutive samples. The lower the sampling interval, the higher the number of data to gather and to transmit. We dynamically evaluate the minimum sampling interval t_m as the lowest value that the sampling interval t can assume, and the maximum sampling interval t_M as the highest value of t. Clearly, $t_M \geq t_m$. The variability Δ represents the deviation among consecutive samples and it is used to discriminate stable from variable states. When Δ is low, the monitored resource is considered to be in a stable state; when Δ is high, the resource is considered as highly variable. We evaluate two parameters related to Δ.

1. The peak variability Δ_p is the threshold above which a deviation among consecutive samples determines a spike identification that is, there is a spike when $\Delta \geq \Delta_p$.
2. The tolerable variability Δ_q is defined as the deviation that must occur between consecutive samples for the monitored time series to be considered highly variable, even more than in presence of a load spike ($\Delta_q \geq \Delta_p$). When the variability of the monitored data becomes too high, the sampling interval must be reset to its minimum value t_m in order to be able to capture the statistical properties of the time series.

Setting the t and Δ variables and related thresholds determines the dynamic behavior of the monitoring algorithm, that aims to find out the best settings for t_m^*, t_M^*, Δ_p^*, Δ_q^* in order to solve in the most convenient way the trade-off between monitoring scalability and data reliability.

3.2. Training phase

The adaptive monitoring algorithm consists of an initial training phase that chooses adaptively the parameters t_m^*, t_M^*, Δ_p^*, Δ_q^* that are related to the minimum and maximum sampling intervals and to the variability thresholds. The training phase evaluates the best values in the ranges by choosing those that maximize E in Eq. (4) over a subset of λ data samples used for training. This search has a combinatorial complexity because, in the basic form, it has to train all combinations of parameters over the number of data used for training. This does not represent a real problem because training is done occasionally. However, in Section 5 we show how to choose λ and parameters ranges so to reduce computational costs. Moreover, we can further reduce training computational cost by adopting a binary search [16] that is able to pass to a logarithmic complexity of the search.

The Algorithm 1 reports the pseudo-code of naive training and the ranges of parameters used for training. Over a subsets of λ

training samples, it evaluates gain, quality, and E for all combinations of tested parameters. In the initialization phase, the optimal value of E (E^*), is set to 0 (line 1), the minimum sampling interval t_m is set to the lowest possible value t^0 and X_{tmp} contains the time series monitored at an interval of t^0. The following for loop (lines 4–12) iterates over the spectrum of values for the monitoring interval t and the variations Δ. It invokes the data monitoring algorithm (line 6), storing the resulting data series in the X variable. It computes the quality parameter Q (line 7), the gain parameter G (line 8) and evaluates the tradeoff E (line 9). Finally, it updates the maximum value of E (line 11) and the corresponding parameters (line 12). We use this setting for the core phase of the monitoring algorithm.

Algorithm 1 Training phase

```
1: $E^* \Leftarrow 0$
2: $t_m \Leftarrow t^0$
3: $X_{tmp} \Leftarrow [1{:}t^0{:}\lambda]$
4: for $t_{\{m,M\}} \rightarrow [t^0, 15 \cdot t_m]$ do
5:     for $\Delta_{\{p,q\}} \rightarrow [\sigma^0, 1.5 \cdot \Delta_p]$ do
6:         $X$ = AdaptiveMonitoring($t_m, t_M, \Delta_p, \Delta_q$)
7:         Q = QualityEvaluation($X$,$X_{tmp}$)
8:         G = 1 - length($X$) / $\lambda$
9:         E = $w \cdot G + (1-w) \cdot Q$
10:        if $E > E^*$ then
11:            $E^* \Leftarrow E$
12:            $\{t_m^*, t_M^*, \Delta_p^*, \Delta_q^*\} \Leftarrow \{t_m, t_M, \Delta_p, \Delta_q\}$
```

Algorithm 2 computes the value of the Q parameter. In the initialization phase, the initial point of the data series is determined (line 1), the true and false positive counts are set to 0 (line 2), the *NRMSE* value is set to 0 (line 3) and the distance mean-outliers in the original time series X_{tmp} is computed (line 4). Then, a loop through all the points of the time series is executed (line 6–24) to compute the quality parameter Q. Lines 6–11 extract the time series value x_0 and the time i, while line 12 updates the sum of square errors necessary to compute *NRMSE*. The true and false positive counts are updated in line 13. Line 16, 18 and 20 compute *NRMSE*, Precision and Recall, respectively. Line 22 computes the *Fmeasure* and, finally, line 23 computes the Q parameter.

3.3. Adaptive monitoring phase

Algorithm 3 reports the pseudo-code of the adaptive monitoring phase. In the initialization phase (lines 1–6), the monitoring interval t is initially set to the minimum value t_m^* (line 1) and the variability Δ is set to zero (line 2). The ϵ_{inc} variable, set to an initial value of $\epsilon = 10$ in line 3, is used to trigger the sampling with the lowest time interval possible (t^0) of the time series X used to retrain the parameters $\{t_m, t_M, \Delta_p, \Delta_q\}$. After $\epsilon \cdot \lambda$ samples, X will be sampled. The variable k counts the length of the time series X used to retrain the parameters. When $k > \lambda$, sampling of X is complete retraining can start. Initially, k is set to 0 (line 4). Finally, the previous and the current time interval are initialized in line 5 and 6, respectively.

The monitoring algorithm operates in an endless loop (lines 7–38). In line 8, the deviation Δ is updated with the difference of the current and previous sample value. If no appreciable variability is detected ($|\Delta| \leq \Delta_p^*$ in line 8, jumping to line 19), the current sample is not stored and the sampling interval can be further increased, while avoiding that it exceeds the maximum possible value t_M^* chosen for the sampling interval (line 20). This upper bound over the sampling interval is necessary to avoid missing of isolated spikes on time series that present long periods of stability. In this context, sampling intervals would tend to become higher

Algorithm 2 Quality Evaluation

```
1: i $\Leftarrow t^0$
2: { TP, TN, FP, FN} $\Leftarrow$ 0
3: NRMSE $\Leftarrow$ 0
4: range $\Leftarrow$ avg($X_{tmp}$) $\pm$ 1.5$\cdot$ std($X_{tmp}$)
5:
6: while i $< \lambda \cdot t^0$ do
7:     pi $\Leftarrow$ i
8:     i $\Leftarrow$ i + $t^0$
9:     $x_o \Leftarrow X[pi]$
10:    if $\exists X[i]$ then
11:        $x_o \Leftarrow X[i]$
12:    NRMSE $\Leftarrow$ NRMSE + $\left(\frac{X_{tmp}[i]-x_o}{\max\{|X_{tmp}[i]|,|x_o|\}}\right)^2$
13:
14:    {TP, TN, FP, FN} $\Leftarrow$ {TP, TN, FP, FN}
       + ChangeDetector(range,$X_{tmp}[i]$,$x_o$)
15:
16: NRMSE $\Leftarrow \sqrt{\frac{NRMSE}{\lambda}}$
17:
18: precision $\Leftarrow \frac{TP}{TP+FP}$
19:
20: recall $\Leftarrow \frac{TP}{TP+FN}$
21:
22: fmeasure $\Leftarrow \frac{2 \cdot precision \cdot recall}{precision+recall}$
23:
24: Q $\Leftarrow \frac{fmeasure+NRMSE}{2}$
```

and higher, and isolated spikes would be easily missed by the algorithm if no maximum sampling interval t_M is set.

On the other hand, if the deviation is greater than the peak variability ($|\Delta| > \Delta_p^*$ in line 10, jumping to line 11) a spike (or, even worse, a highly variable series) has been detected. In this case, the monitored time series is augmented with the current sample (line 11), since spikes must be included in the load representation. If the deviation is higher than the maximum tolerable variability (line 13), then the sampling interval must be reduced to the lowest possible value t_m^* chosen by the administrator (line 14); otherwise (line 15), the interval is reduced accordingly. In the end (line 18), the deviation is set back to 0 since the new sample has been incorporated in the monitored time series. Lines 22–23 update the previous and next time interval.

In highly variable contexts, the best settings of the parameters $\{t_m, t_M, \Delta_p, \Delta_q\}$ may become rapidly obsolete. Lines 25–38 of the algorithm are responsible for the retraining of the parameters in this case. After $\epsilon \cdot \lambda$ samples have been collected by the adaptive algorithm (line 25), a time series X_{tmp} will be sampled at the lowest time interval t^0. If a new time series has to be collected from scratch (line 26), it is initialized (lines 27 and 28). Lines 30 and 31 update the time series X_{tmp} and the next time interval, respectively. If a sufficient number of samples has been collected (at least λ in line 32) the quality parameter is evaluated for this series (line 33) and, if it is bad (line 34) the parameters are retrained (line 35). Finally, ϵ is incremented and k is reset to 0.

4. Experimental testbed and training

4.1. Testbed

As experimental testbed, we refer to a typical monitoring architecture (e.g., [19,23,31,18,3,2]) consisting of multiple logical layers as in Fig. 2: at the lowest layer, a set of resources on the monitored nodes are scanned from some probes attached to a collection agent (*collection phase*); then, the sampled metrics are sent to the higher layer called collector nodes (*sending phase*). The algorithm proposed in this paper can be applied at both phases. In the collection phase, the system metrics are gathered on the basis

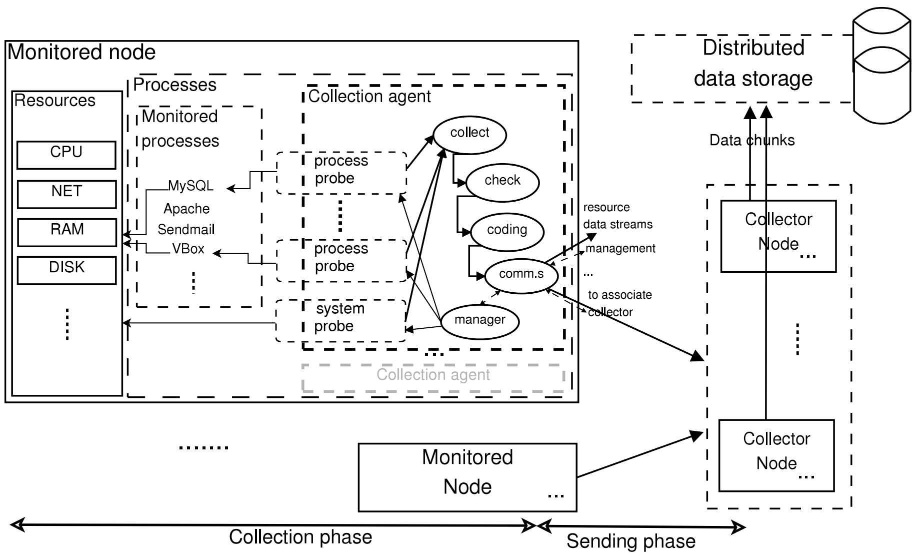


Fig. 2. Monitoring architecture.

Algorithm 3 Adaptive monitoring phase

```
1: $t \Leftarrow t^*_m$
2: $\Delta \Leftarrow 0$
3: $\epsilon_{inc} \Leftarrow \epsilon$
4: $k \Leftarrow 0$
5: $pi \Leftarrow \lambda$
6: $i \Leftarrow pi+t$
7: while True do
8:     $\Delta \Leftarrow \Delta + (x_i - x_{pi})$
9:
10:    if $|\Delta| > \Delta^*_p$ then
11:        $X \Leftarrow [X, x_i]$
12:
13:        if $|\Delta| > \Delta^*_q$ then
14:            $t \Leftarrow t^*_m$
15:        else
16:            $t \Leftarrow \max(t^*_m, t - t^*_m)$
17:
18:        $\Delta \Leftarrow 0$
19:    else
20:        $t \Leftarrow \min(t^*_M, t + t^*_m)$
21:
22:    $pi \Leftarrow i$
23:    $i \Leftarrow i + t$
24:
25:    if $i > \epsilon_{inc} \cdot \lambda$ then
26:        if $k = 0$ then
27:            $X_{tmp} \Leftarrow []$
28:            $ri \Leftarrow pi$
29:
30:        $X_{tmp} \Leftarrow [X_{tmp}, x_{pi}{:}t^0{:}x_i]$
31:        $k \Leftarrow k + (i-pi)/t^0$
32:        if $k > \lambda$ then
33:            Q$\Leftarrow$QualityEvaluation($X[ri, \dots, pi], X_{tmp}$)
34:            if $Q < (1 - w)$ then
35:                $\{t^*_m, t^*_M, \Delta^*_p, \Delta^*_q\} \Leftarrow$ TrainingPhase($X_{tmp}$)
36:
37:            $\epsilon_{inc} \Leftarrow \epsilon_{inc} + \epsilon$
38:            $k \Leftarrow 0$
```

of the proposed adaptive algorithm and then sent to the collector nodes: we adapt sampling intervals by choosing the best trade-off between data quality and computation/communication overheads. In the sending phase, the algorithm is applied to collected samples, hence it is possible to adapt the parameters of the algorithm by comparing the recently sampled data with previously collected, analyzed and forwarded data.

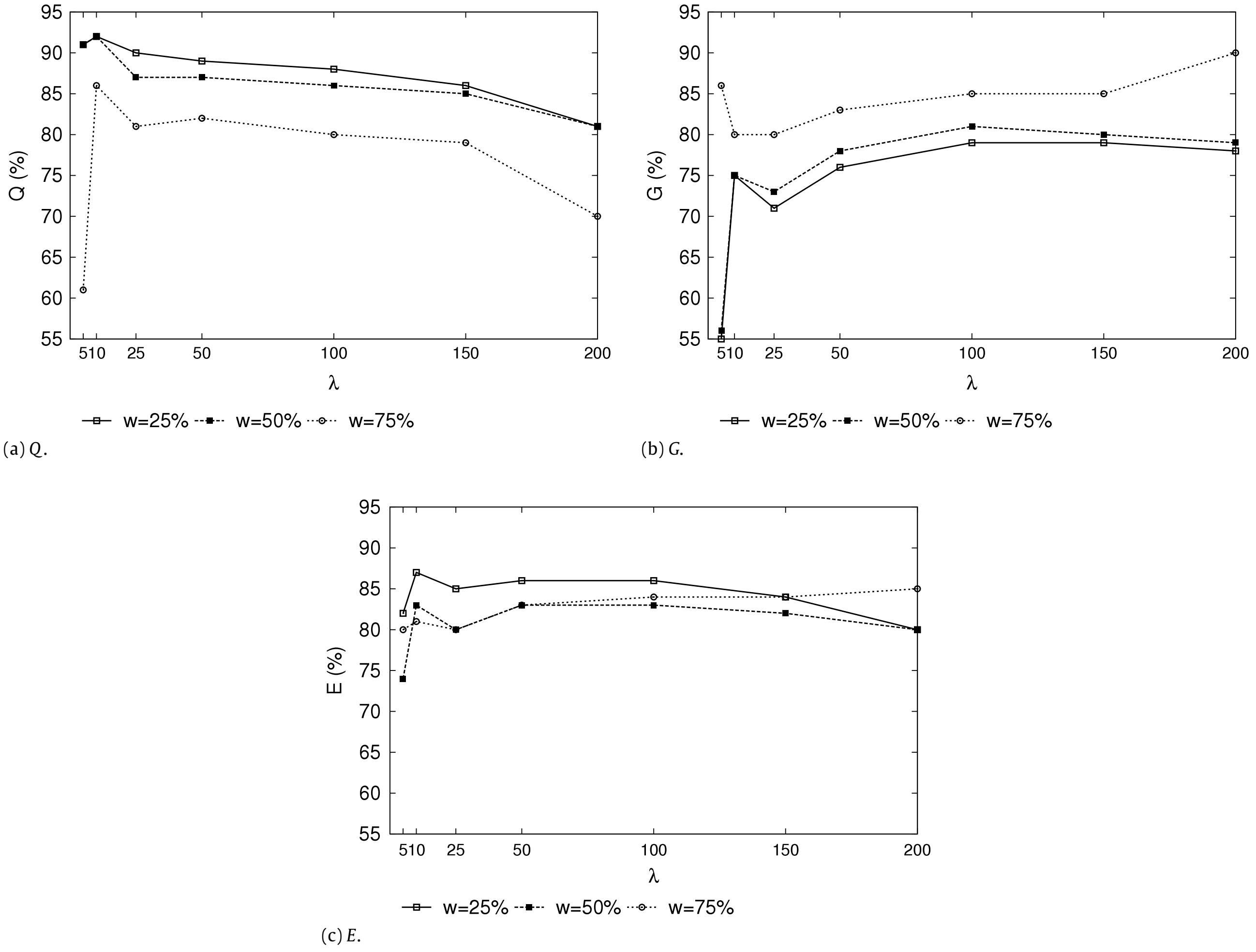
In our experiments, we use the monitoring platform deployed on Amazon EC2 [1] and Emulab [29], as described in [2]. Details of the monitored nodes are the following: micro instance, 613 MB memory, up to 2 EC2 Compute Units (Dual-Core AMD Opteron(tm) Processor 2218 HE, cpu 2600 MHz, cache size 1024 kB), EBS storage, and dedicated network bandwidth of 100 Mbps per node. In the considered testbed, the monitored nodes execute different applications such as Apache2, MySQL, Java and PHP programs subject to TCP-W [28] and RUBiS [20] workload models. MapReduce jobs and MongoDB queries are used for data distribution and analysis. For each node, we monitor 25 different performance indicators (e.g., CPU, memory, network, and disk) at system level and an average of 20 different performance indicators at process level (for each running application). The amounts of metrics associated to different performance indicators are different: we monitor 20 K series related to the CPU, 26 K to the memory, 46 K to the network, 11 K to the disk, and 37 K to other metrics. Moreover, we have 12 specific statistics related to Apache users, MySQL and MongoDB queries, MapReduce jobs. The total data set consists of about 140 K monitored samples. A detailed description of the monitoring infrastructure and the metrics involved can be found in [21]. In the next section, we present the results obtained from experiments lasting for about 12 h and performed over more than 1000 nodes.

4.2. Parameter setting

The proposed algorithm requires a phase of training over λ samples for the estimation of the best parameters $\{t^*_m, t^*_M, \Delta^*_p, \Delta^*_q\}$. We evaluate some performance with the goal of identifying the best settings for training set size and training parameters and to limit training costs.

4.2.1. Training set size

The training set size λ has a strong impact on the computational cost of the algorithm. Hence, it is important to limit these costs while guaranteeing high quality of the monitoring algorithm. Initial experiments showed that the optimal setting of λ is influenced by the emphasis given to gain and quality in the determination of E in Eq. (4). Our algorithm does not attempt to



(a) Q.

(b) G.

(c) E.

Fig. 3. Evaluation of the adaptive algorithm using different weights.

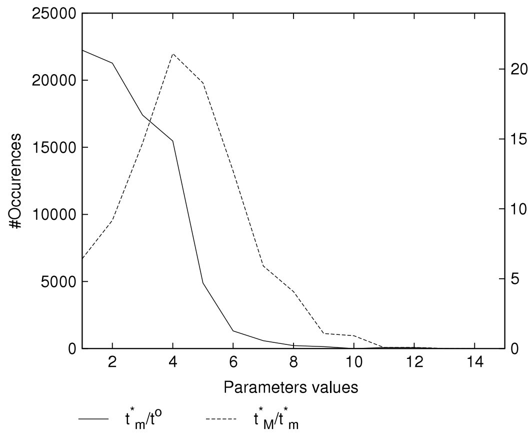
find the best trade-off scalability vs. reliability, but it leaves to the system administrator the possibility to express a preference for this trade-off by choosing how to set w in Eq. (4). We recall that the higher the w, the higher the interest on gain G (i.e., reducing overheads) than on quality Q, and vice versa. According to the choice of w, we want to find the best training set size that maximizes the trade-off. To this purpose, we evaluate the performance of the proposed algorithm with respect to different weights. We apply the adaptive monitoring algorithm to the workload described in Section 4 by considering different sizes of data used for training, spanning from $\lambda = 5$ to $\lambda = 200$ samples. Fig. 3(a)–(c) report the average Q, G and E values of all the tests, respectively. The results consider different trade-off preferences that is, $w = 0.25$, $w = 0.5$, and $w = 0.75$.

Fig. 3(a) evidences that $w = 0.25$ gives the best quality results Q with respect to other w settings. On the other hand, Fig. 3(b) shows that the highest G results are obtained for $w = 0.75$, while lower w values lead to worse gain results. The reason is that, over short training sets, the values that differ by a small quantity could be easily identified as load spikes. This leads to the choice of short sampling intervals that are the cause of high quality and low gain results. Vice versa, larger training sizes lead to choose wider sampling intervals which determine a lower quality of data and a higher gain.

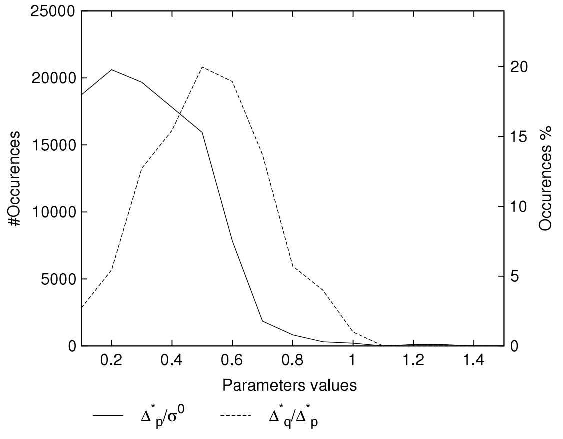
By combining these results, Fig. 3(c) shows that the choice of different training set sizes influences only slightly the algorithm performance. For each w setting, the E values place between 80% and 88% if we choose any λ between 10 and 200. This negligible dependence of the adaptive algorithm on the training set size is important because it evidences that the algorithm is robust enough to adapt its parameters in order to obtain high quality results despite the size of data sets used for training. These characteristics allow us to limit the training set size to few samples (e.g., $\lambda = 10$) thus reducing the costs during the training phase. Besides that, a simple rule can be extracted from these results for the choice for the best training set size with respect to the w setting. When w is low (i.e., $w = 0.25$), it is preferable to use low training set sizes (i.e., $\lambda = 10$). When w is high (i.e., $w = 0.75$), the training phase tends to prefer high λ sizes (i.e., $\lambda = 200$). When an administrator does not specify a preference between gain and quality (i.e., $w = 0.5$), any value between 50 and 150 is a good choice for the training set size. Where not otherwise specified, the experimental results are related to scenarios where G and Q values have the same importance that is, $w = 0.5$, and $\lambda = 100$.

4.2.2. Distribution of training parameters

Limiting the range of tested values for the parameters t_m, t_M, Δ_p, Δ_q is a way to further reduce the computational cost of training. Given t^0 as the minimum sampling interval (e.g., 1 s), we initially choose to evaluate t_m and t_M in the range $[t^0, 15 \cdot t^0]$ and $[t_m, 15 \cdot t_m]$. Given σ^o as the standard deviation computed over the first λ data, we start evaluating Δ_p and Δ_q in the ranges $[\sigma^o, 1.5 \cdot \sigma^o]$ and $[\Delta_p, 1.5 \cdot \Delta_p]$, respectively. Since it would be expensive to train all possible values in these ranges, we have to choose a step between two consecutive values to train. In our data sets, we could observe that a difference of 10% between two



(a) t.



(b) Δ.

Fig. 4. Distribution of parameters.

consecutive values in the ranges produces high quality results with limited computational costs.

After this initial setting, we evaluate algorithm performance in order to reduce those ranges to subsets of values having high probability to maximize E. To this purpose, we compute the distribution of the parameters for sampling intervals and thresholds that maximizes the E value of each series. As shown in Fig. 4, the results are Gaussian distributions in which more than 90% of the parameters take small values. We see that t^*_m ranges between t^0 and $5 \cdot t^0$, while t^*_M ranges between t^*_m and $8 \cdot t^*_m$. The thresholds Δ^*_p and Δ^*_q range between σ^o and $0.5 \cdot \sigma^o$, and $0.2 \cdot \Delta^*_p$ and $0.8 \cdot \Delta^*_p$, respectively.

The Gaussian distributions support the reduction of parameters ranges to smaller subsets and help us to further reduce the cost of the algorithms during the training phase, in which large parameter ranges seem not to be required. On the basis of these results, in the following section we set $[t^0, 5 \cdot t^0]$ and $[2 \cdot t^*_m, 10 \cdot t^*_m]$ for training set sizes, and $[\sigma^o, 0.5 \cdot \sigma^o]$ and $[\Delta^*_p, 0.9 \cdot \Delta^*_p]$ for thresholds.

A further reduction of training overhead can be obtained by adopting an adaptive binary search [16] of the best parameters inside the subsets of ranges. The basic binary search is made adaptive to series characteristics in order to find most peaks in the series. By comparing the binary search results to those obtained by the complete search, we obtain the same results in 93.08% cases, but the computational cost of the training phase is reduced to logarithmic. Similar results ($\pm 5\% E$) between the two searches are achieved in the 6.51% of cases, while only in the 0.40% of cases the differences between the results are higher than 5%. We can conclude that an adaptive binary search allows us to reduce the computational cost of the training phase while maintaining a precision level higher than 99.2% in more than 99.6% of cases with respect to the naive search.

5. Performance evaluation

In this section we report the most significant experimental results for evaluating the performance of the proposed adaptive monitoring algorithm. Evaluations are carried out with the goal of analyzing the performance of the algorithm with respect to different training parameters and types of series, and of comparing results against those of existing monitoring methods.

5.1. Robustness to training parameters

The goal is to evaluate the impact of different values for the sampling intervals t_m and t_M, and for the threshold parameters Δ_p and Δ_q. We would like the performance of our algorithm to be slightly influenced by changes in the parameters setting. For all series we choose the values of t^*_m, t^*_M, Δ^*_p, Δ^*_q that maximize E. Then, we test the antecedent and subsequent values, and we calculate the new E results. This iterative phase continues until the difference between the previous E value and the new calculated one overcomes the difference between the best and the currently tested value.

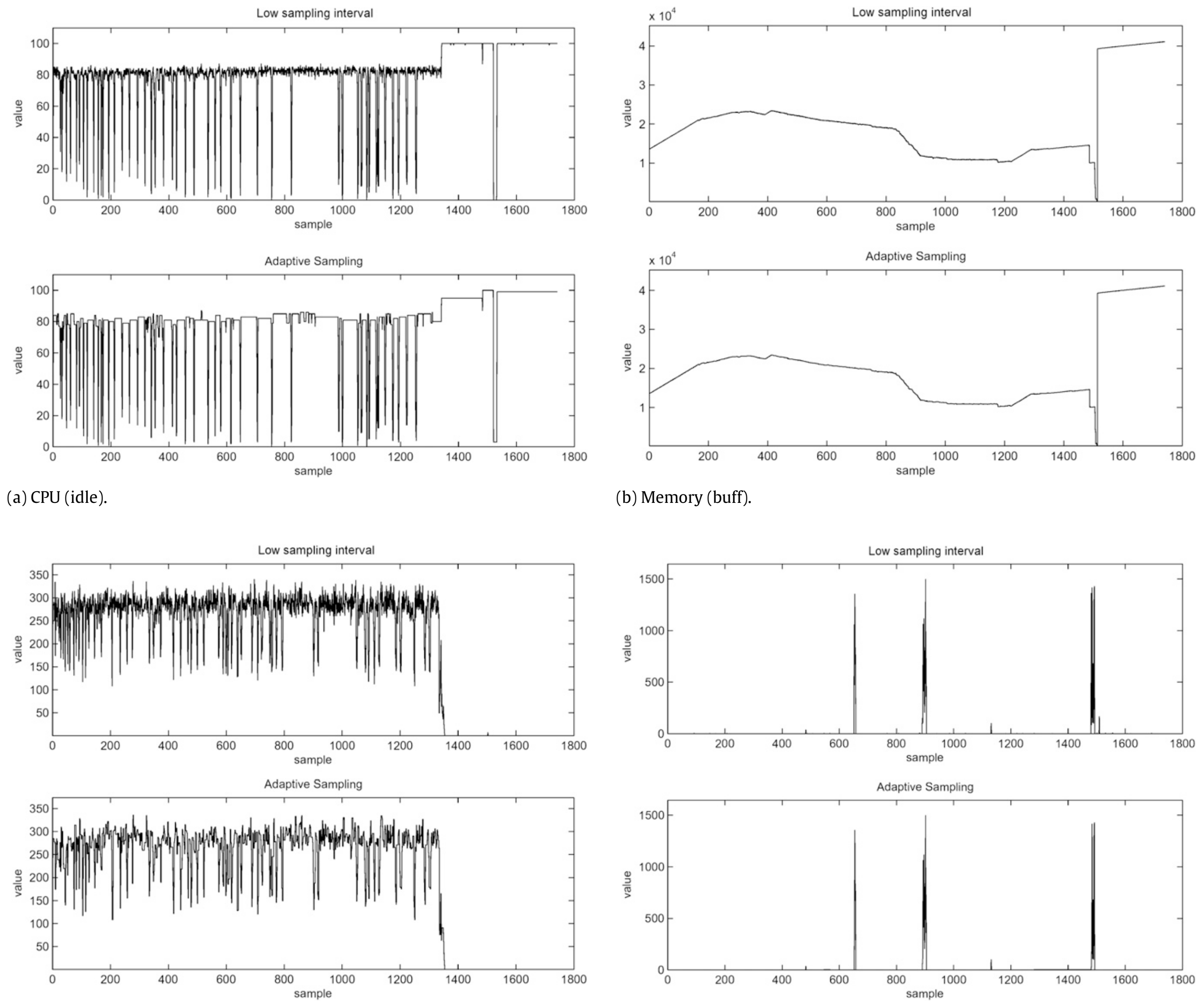
In our testbed, the proposed algorithm achieves high robustness levels with respect to small changes in parameters settings. In 96.51% of cases, introducing a difference of 20% between tested parameter values leads only to a slight degradation (less than 15%) in the algorithm performance. Only in 3.47% of cases, the performance decrease is between 15% and 25% with less than 0.02% of cases where robustness is low.

5.2. Impact of series characteristics

Series belonging to different performance indicators present various statistical characteristics that may lead to different results of the adaptive monitoring algorithm. Hence, we evaluate the performance of the adaptive monitoring algorithm against different types of series, that is, CPU, memory, network, and disk related series. In Fig. 5, we present some examples for each performance indicator. At the top, each figure shows the series sampled at the lowest sampling interval (i.e., $t^0 = 1$ s in this example), while the series at the bottom are sampled with our adaptive monitoring algorithm.

Fig. 6 reports the average values of G, Q and E over all tested series for each performance indicator. The results show that values of Q tend to be high for any series type and characteristics. Q values are above 80% both for spiky series (such as CPU in Fig. 5(a)), for series with high variability (such as network in Fig. 5(c)), for series presenting a recurrent linear trend (such as memory in Fig. 5(b)), and for stable series (such as disk in Fig. 5(d)). The high quality results (i.e., $\geq 95\%$) when dealing with linear trends come from the ability of our solution to adapt its parameters to capture even very small shifts in series with low standard deviation. Adaptivity to series standard deviation allows also to achieve high Q values in series where variability is high, by capturing most of load spikes while saving samples during stable periods.

The amount of saved samples can be appreciated by looking at gain results in Fig. 6. G is high in series presenting stable states, with its best values on disk related series characterized by long periods of stability. On the contrary, G is low (i.e., 54%) on series presenting recursive trends, since data always change and stable periods are rare and short.



(a) CPU (idle).

(b) Memory (buff).

(c) Network (txkBs).

(d) Disk (block in).

Fig. 5. Examples of resource series sampled by using low intervals and the adaptive monitoring algorithm.

The combined effect of Q and G can be appreciated through the E bars in Fig. 6, showing high performance results for all types of series. E values are always higher than 78%, thus showing that the proposed adaptive algorithm guarantees high performance for any time series related to systems or processes performance indicators.

5.3. Performance comparison

In this section, we compare the results of our adaptive monitoring algorithm to those obtained by the state-of-the-art real-time algorithms. As terms of comparison, we consider: (1) the static frequency sampling algorithm, (2) the delta encoding and static thresholds algorithm, and (3) a hybrid algorithm which uses our adaptive core and delta-encoding and static thresholds technique.

5.3.1. Static frequency sampling algorithm

While our solution adaptively sets the best parameters values, all state-of-the-art algorithms require the static setting of the sampling interval t and/or of the variability Δ. These static choices, in turn, influence the performance of the monitoring algorithms.

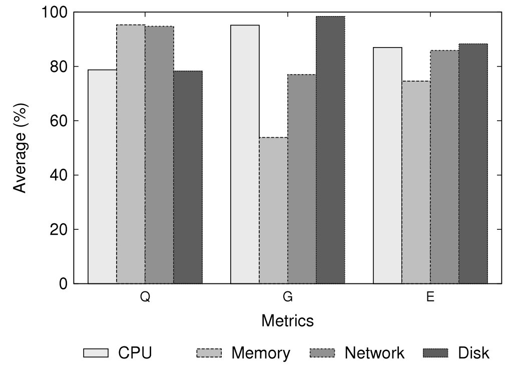


Fig. 6. Results for different performance indicators.

The static frequency sampling algorithm depends upon the a-priori choice of t. Table 1 reports the performance of the static frequency sampling algorithm obtained from experiments over all the 140 K series for different sampling intervals t. The gain value is equal to $100 - (100/t)\%$ (e.g., 50% when $t = 2$, 80% when $t = 5$), thus making high t values preferable in order to improve

Table 1
Algorithm based on static frequency sampling.

t	*Fmeasure* (%)	1-*NRMSE* (%)	Q (%)	G (%)	E (%)
2	72.37	89.15	80.76	50.00	65.38
5	53.22	81.86	67.54	80.00	73.77
10	44.33	78.14	61.24	90.00	75.62
20	33.29	72.63	52.96	95.00	73.98

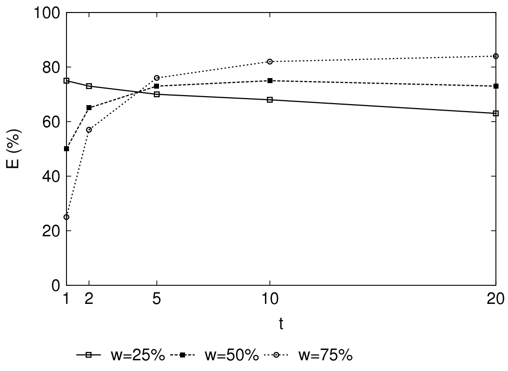


Fig. 7. Performance of static frequency sampling algorithm by using different weights.

scalability. Despite that, for $t \geq 5$ the quality of the static frequency sampling algorithm becomes low, as well as its ability of capturing spikes. *Fmeasure* ranges from 53% when $t = 5$ to 33% when $t = 20$. Moreover, also the error increases when the sampling interval increases. It is interesting to observe that the static frequency sampling algorithm never achieves an E value above 76%.

The most important result of these experiments is the difficulty in statically choosing a t value that can guarantee a good trade-off between gain and quality over all types of series. Moreover, this difficulty is enhanced by the dependency of results on different values of w in Eq. (4). Fig. 7 plots the E values obtained for different w settings and for increasing t values. It is evident that when the administrator sets high w values (i.e., >50%), wrong static choices of t may lead to very poor results, with E values even lower than 25%.

5.3.2. *Delta-encoding and static threshold algorithm*

Also for the algorithm based on delta-encoding and static thresholds, the sampling interval t is fixed, as well as the Δ parameter. Hence, these values must be chosen a-priori.

Table 2 reports the algorithm performance with respect to different settings of t and Δ. As expected, low t values bring to high quality results and low gains, while high sampling intervals cause low quality and high gains. Increasing the sampling interval t increases the E value, with the best performance when $t = 10$ and $\Delta = 10$. Despite that, with this setting *Fmeasure* is extremely low (i.e., ∼35%) thus meaning that most spikes in the series are missed. These results demonstrate that different choices of the algorithm parameters strongly affect performance results.

As we know, performance also depends on the emphasis given by system administrators to G and Q in Eq. (4). Fig. 8 shows the behavior of E when using different w values. Results are reported with respect to different static choices of t and Δ. By giving more emphasis on quality (i.e., $w = 0.25$), E values are fairly stable, with a standard deviation of results equal to 3.4. By using higher values of w, the standard deviation increases to 7.4 when $w = 0.5$ and to 17.3 when $w = 0.75$.

This evaluation shows that the choice of parameters cannot be independent of the setting of w and stresses the need and potential of adaptive algorithms over big data sizes and heterogeneous environments.

Table 2
Delta encoding and static thresholds algorithm.

t	Δ	*Fmeasure* (%)	1-*NRMSE* (%)	Q (%)	G (%)	E (%)
1	0	100.00	100.00	100.00	5.93	52.97
	1	90.86	99.58	95.22	12.89	54.06
	2	89.94	99.16	94.55	21.47	58.01
	5	83.94	96.43	90.19	43.82	67.01
	10	63.18	87.50	75.34	72.48	73.91
2	0	72.36	89.14	80.75	53.26	67.01
	1	65.16	89.12	77.14	56.47	66.81
	2	64.68	89.11	76.90	61.13	69.02
	5	62.92	87.85	75.39	70.88	73.14
	10	52.68	86.62	69.65	82.14	75.90
5	0	53.21	81.86	67.54	81.14	74.34
	1	48.05	81.84	64.95	82.40	73.68
	2	48.04	81.83	64.94	83.74	74.34
	5	47.02	81.72	64.37	87.29	75.83
	10	45.17	81.50	63.34	91.88	77.61
10	0	44.34	78.14	61.24	90.52	75.88
	1	40.12	78.14	59.13	91.21	75.17
	2	39.64	78.14	58.89	91.95	75.42
	5	39.11	78.14	58.63	93.88	76.26
	10	35.73	78.16	56.95	96.29	76.62

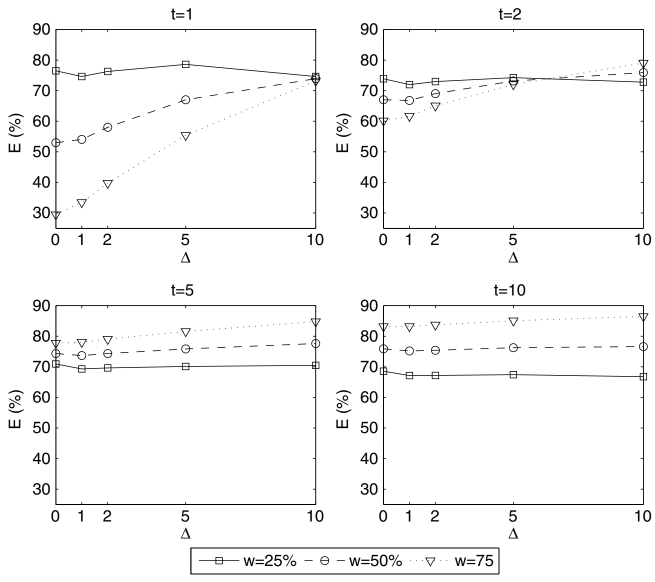


Fig. 8. Performance of delta encoding and static thresholds algorithm by using different weights.

5.3.3. *Adaptive version of delta-encoding and static thresholds algorithm*

Since fixed sampling intervals and variability levels influence the performance of the algorithm based on delta-encoding and static thresholds, we decide to improve its implementation and to make its parameter choice adaptive. To do this, we apply the two phases of our adaptive monitoring algorithm to the naive version of the delta-encoding and static thresholds algorithm. A training phase over λ samples is used for the finding of the best parameter values for t and Δ, and the monitoring phase is performed to evaluate the algorithm performance. Table 3 reports the best results obtained by the adaptive version. Experimental results show that E values span from 76.54% when $\lambda = 5$ to 79, 37% when $\lambda = 100$. If we compare these results to those in Table 2, we see that the performance of the adaptive version is always higher than that achievable through the static version of the algorithm. Moreover, *Fmeasure* never falls below 65%, thus improving the ability of the algorithm in capturing load spikes.

Table 3
Performance summary of the adaptive version of delta-encoding and static thresholds algorithm.

λ	*Fmeasure* (%)	1-*NRMSE* (%)	*Q* (%)	*G* (%)	*E* (%)
5	75.45	81.39	78.42	69.56	73.99
10	74.92	81.25	78.09	77.87	77.98
25	74.14	81.11	77.63	76.11	76.87
50	71.64	81.30	76.47	81.69	79.08
100	68.90	80.49	74.70	84.04	79.37
150	69.53	78.26	73.90	83.31	78.60
200	65.14	75.12	70.13	83.80	76.97

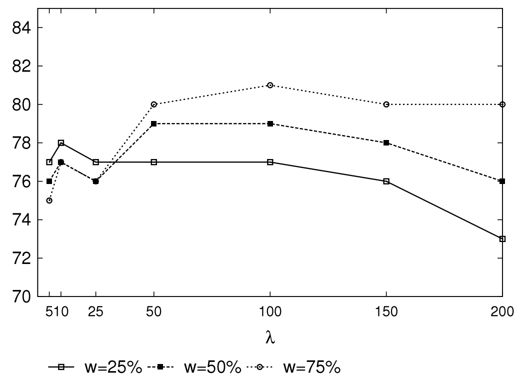


Fig. 9. Performance of adaptive version of the delta encoding and static thresholds algorithm by using different weights.

The performance improvement is also clear when looking at the behavior of the adaptive and static versions of the delta-encoding and static thresholds algorithm with respect to different settings of w. By comparing Fig. 9 to Fig. 8, we see that making the algorithm adaptive strongly increases *E* values and makes results stable. In particular, the average *E* value of the adaptive version is equal to 76.77%, 77.91% and 79.06% by using $w = 0.25$, 0.5 and 0.75, respectively. This is an improvement of 5.07%, 7.26% and 9.47% in performance with respect to the static version. Moreover, adaptivity makes sure that wrong static parameter settings cannot make the monitoring performance drop, with *E* values that never go below 75%. Stability of the results is due to the adaptivity of the algorithm that allows it to find the best parameter setting despite of the value of w.

5.3.4. *Algorithms comparison*

We finally compare the results obtained by the state-of-the-art algorithms with the results obtained by the proposed adaptive monitoring algorithm. Since both the static frequency sampling algorithm and the delta-encoding and static threshold algorithm require the a-priori choice of their parameters, it becomes difficult to directly compare their results to those of our adaptive solution with respect to different parameter settings. A general comparison that we can provide is related to the best performance achieved by each algorithm, despite of the parameter setting that is used to achieve it.

Table 4 summarizes the best results obtained by all the considered algorithms. For the sake of a fair comparison, the best results for all algorithms refer to a setting of $w = 0.5$. As we see from the table, the adaptive monitoring algorithm overcomes the performance of state-of-the-art algorithms, thanks to both low errors and high *Fmeasure* results. The ability of limiting errors and capturing most of spikes in series allows our algorithm to obtain higher *E* values with respect to any other state-of-the-art solution.

A direct comparison can be done between the performance of our solution to that of the adaptive version of the delta-encoding and static thresholds algorithm, since both methods implement our adaptive core algorithm. Fig. 10 compares the performance of the two adaptive solutions in terms of *Fmeasure*, *NRMSE*, and *E* metrics with respect to different λ sizes of the training set.

By looking at Fig. 10(a), we see that the adaptive solution guarantees high performance in capturing significant load changes, with *Fmeasure* values always higher than 65%. Besides that, our adaptive solution maintains an improvement of 20% in *Fmeasure* results for all training set sizes. This result shows the importance of our solution to adapt its parameter setting to the variability in data. Thanks to this feature, our adaptive algorithm represents a significant improvement in monitoring reliability. Reliability is improved, also thanks to lower errors in sampling. Fig. 10(b) shows that errors of adaptive solutions never overcome 25%, while our solution reduces sampling errors with an improvement from 2% to 7% with respect to the adaptive version of an existing solution.

Finally, adaptivity ensures that *E* results are always greater than 74%, as we can see in Fig. 10(c). When using training set sizes $\lambda > 5$, our adaptive monitoring algorithm has an average *E* value $> 82\%$, with an increase of more than 4% with respect to the other adaptive solution. These results point out the benefits gained from adaptation of parameters to data characteristics, that allows our algorithm to capture relevant load changes in data and to achieve high performance results even on highly variable time series.

5.3.5. *Resource consumption*

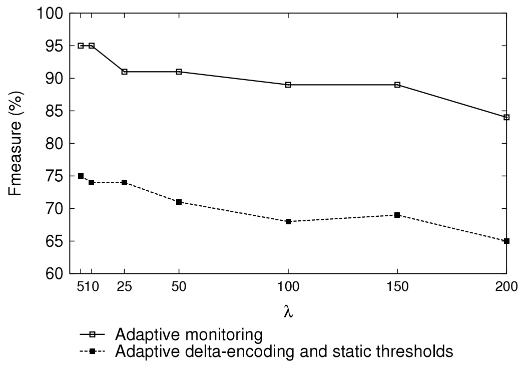
We finally compare the algorithms in terms of resource consumption when collecting about 3 K metrics for each monitored node. A detailed analysis of the impact on cost of data collection and transmission can be found in [21]. Here, we refer to the CPU utilization. Fig. 11 shows a comparison of typical behaviors in terms of CPU consumption when running different monitoring algorithms.

Fig. 11(a) reports the CPU consumption of the static frequency sampling algorithm and the delta-encoding and static threshold algorithm, both in case of setting low (e.g., $t = 1$) and high (e.g., $t = 1$) sampling intervals. The static sampling frequency algorithm has an average CPU consumption of 0.92% when using $t = 1$, and of 0.09% when using $t = 10$, while the algorithm based on delta encoding and static thresholds has an average CPU consumption of 1.21% when using $t = 1$, and of 0.13% when using $t = 10$. For the two static algorithms, these results show that CPU consumption depends on the choice of the sampling interval t. The CPU consumption is linear with respect to t, with different coefficients in the two static algorithms. The algorithm based on delta encoding and static thresholds has an overhead of 30% with respect to the CPU consumption values of the static sampling frequency algorithm.

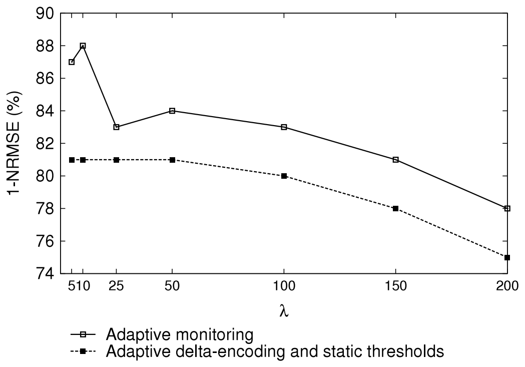
When running our adaptive monitoring algorithm, experiments cause almost the same RAM consumption ($9.4\% \pm 0.5\%$) of the collection agents and have a negligible impact on the disk, while the impact on network depends on the communication overhead of the algorithms. If considering CPU consumption, the typical behavior caused by our adaptive algorithm is reported in Fig. 11. We can see three phases in the trend of the CPU consumption. In the first phase, λ training set values must be collected. During this phase, the CPU consumption shows a growing trend until the end of training set collection. Then, the adaptive monitoring algorithm selects the best parameters and this selection motivates the initial high spike of CPU utilization. After the selection, the algorithm consumption stabilizes around a fixed amount of CPU, that ranges between those of the static frequency sampling algorithms using $t = t_m$ and $t = t_M$. These results show that the adaptivity of our solution does not impact significantly on resource consumptions because CPU costs are comparable to those of existing static solutions.

Table 4
Best performance comparison.

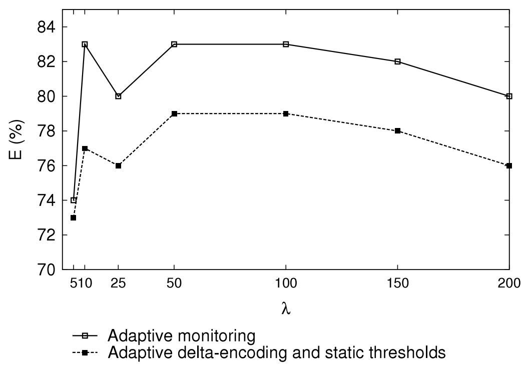
Algorithm	*Fmeasure* (%)	1-*NRMSE* (%)	*Q* (%)	*G* (%)	*E* (%)
Static frequency sampling	44.33	78.14	61.24	90.00	75.62
Delta-encoding and static th.	35.73	78.16	56.95	96.29	76.62
Adaptive monitoring	89.99	83.48	86.74	81.09	83.92



(a) *Fmeasure*.

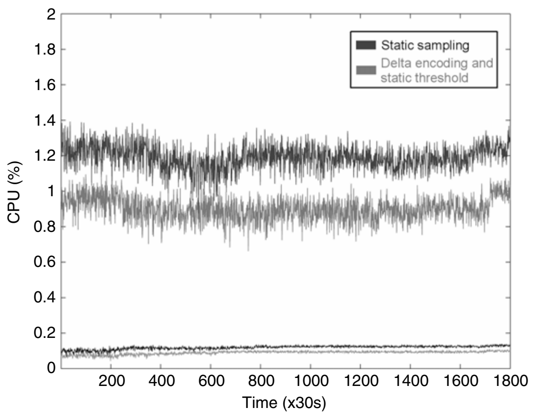


(b) 1-*NRMSE*.

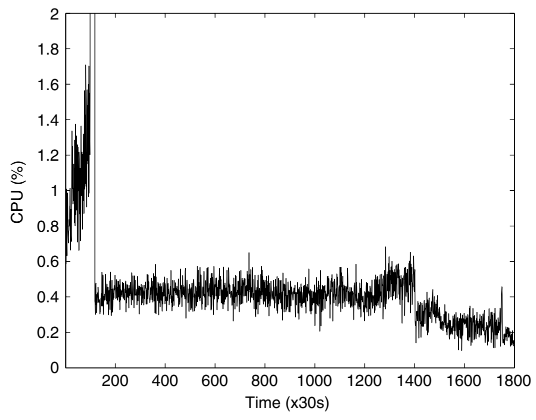


(c) *E*.

Fig. 10. Comparison between the proposed adaptive monitoring algorithm and our adaptive version of the delta-encoding and static thresholds algorithm.



(a) Static algorithms.



(b) Adaptive algorithm.

Fig. 11. Average CPU consumption of the monitoring algorithms.

6. Related work

Techniques for improving the scalability of data acquisition in big data monitoring can be distinguished between lossy and lossless. Lossless techniques that do not try to reduce the complexity of the monitored data cannot achieve a scalability level as high as methods which are customized to the nature of time series, but they can be applied to data independently [32]. On the other hand, lossy techniques strive to reduce the data set without compromising its fidelity; they can be distinguished between offline and online schemes. Offline techniques need to obtain the whole series while the online techniques process resource samples on the fly. Other techniques, such as adaptive dimensionality [8], differ from the proposed approach because they aim to reduce the dimensionality of series.

The field of offline lossy techniques is rich in proposals, such as Fast/Discrete Fourier [24] and Wavelet [9] Transform, Piecewise Aggregate Approximation [14], Singular Value Decomposition [26], Symbolization [17] and Histograms [5]. Some recent studies achieve probabilistic or deterministic error bounds on individual data samples [11], but these algorithms work by assuming the knowledge of the entire time series [32]. Hence, they are unsuitable to real-time monitoring contexts considered in this paper. Other approaches based on linear segmentation (e.g., PLA [13], sliding window [4]) impose a wait time during the reconstruction of the series, because they use the first data point of a time series as the starting data sample of a segment and use the next data samples to evaluate the approximation error. The three main variants of this algorithm improve the quality for specific datasets, but they are not robust if we consider arbitrary datasets [13,32].

The state-of-the-art in the field of online lossy techniques can be distinguished between delta-encoding techniques based on static thresholds [12,15], and architecture-related aggregation techniques [6]. The latter methods are strongly coupled with the architectural layer and are inapplicable to a generic monitor infrastructure. Furthermore, they can be efficient for specific datasets, but they do not guarantee any robustness and quality level with respect to datasets originated by different distribution nor to different application scenarios as the proposed algorithms do. The delta-encoding techniques based on static thresholds are applicable to any monitor infrastructure, but the choice of a static sampling frequency does not allow to solve the trade-off between scalability and reliability that represents the main novelty of the proposed approach. In [8] Keogh et al. introduce a novel dimensionality reduction technique called Adaptive Piecewise Constant Approximation (APCA) that approximates a known time series by a set of constant value segments of varying lengths such that their individual reconstruction errors are minimal. This solution is unsuitable to our context because it operates on the whole time series, hence it cannot be used for real-time analyses involving millions of streams. In [12], Kamoshida et al. implement a strategy for low cost data gathering: they collect resource performance indexes only when they differ significantly from previous values. However, their sampling interval is fixed and does not take into account that, in highly heterogeneous systems such as large data centers, sampling frequencies should be adapted to accomodate changes in the statistical characteristics of the monitored time series. In [15], Keralapura et al. address the problem of monitoring aggregate frequency counts of events and raising alarms based on static and adaptive threshold. In particular, they characterize the overhead of monitoring as a function of threshold values and are able to find the optimal thresholds that guarantee minimal resource consumption by the monitoring infrastructure. Of course, this approach is limited to a particular class of performance indexes (frequency counts) and is not generalizable. In [6], Boehm et al. discuss a hierarchical, tree-based overlay network of monitoring processes where the upper layer nodes analyze and reduce the monitoring samples received by the lower layer nodes. In this way, a trade-off between accuracy of the results and scalability of the monitoring process can be achieved. However, according to the authors themselves, the prototype currently does not scale to the volume of data requested by big data applications.

7. Conclusions

We propose a real-time adaptive algorithm for scalable and reliable big data monitoring that is able to adapt sampling intervals and update frequencies in order to minimize computational and communication costs, while guaranteeing high accuracy in capturing relevant changes in system behavior. These qualities are mandatory when the system has to support collection, storing and analysis operations of big datasets coming from large and heterogeneous resource monitors. An extensive set of experiments performed on real time series shows that the proposed adaptive algorithm reduces the overheads of monitoring without penalizing the quality of data with respect to the state-of-the-art algorithms. We should consider that in a world where the exponential growth of system components and data sizes is the norm, even the scalability goal cannot take any rest. Our current research is evaluating the limits of the proposed method and when it is necessary to introduce quite alternative approaches.

References

[1] Amazon, Elastic Compute Cloud (EC2). http://aws.amazon.com/ec2.
[2] M. Andreolini, M. Colajanni, M. Pietri, A scalable architecture for real-time monitoring of large information systems, in: IEEE Second Symposium on Network Cloud Computing and Applications, London, UK, 2012, pp. 143–150.
[3] M. Andreolini, M. Colajanni, S. Tosi, A software architecture for the analysis of large sets of data streams in cloud infrastructures, in: IEEE 11th International Conference on Computer and Information Technology, CIT, Paphos, Cyprus, 2011, pp. 389–394.
[4] U. Appel, A.V. Brandt, Adaptive sequential segmentation of piecewise stationary time series, Inform. Sci. 29 (1) (1983) 27–56.
[5] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, 2002, pp. 1–16.
[6] S. Böhm, C. Engelmann, S.L. Scott, Aggregation of real-time system monitoring data for analyzing large-scale parallel and distributed computing environments, in: Proceedings of the 12th IEEE International Conference on High Performance Computing and Communications, HPCC, 2010, Melbourne, Australia, 2010, pp. 72–78.
[7] S. Casolari, S. Tosi, F.L. Presti, An adaptive model for online detection of relevant state changes in Internet-based systems, Perform. Eval. 69 (5) (2012) 206–226.
[8] K. Chakrabarti, E. Keogh, S. Mehrotra, M. Pazzani, Locally adaptive dimensionality reduction for indexing large time series databases, ACM Trans. Database Syst. 27 (2) (2002) 188–228.
[9] F.-P. Chan, A.-C. Fu, C. Yu, Haar wavelets for efficient similarity search of time-series: with and without time warping, IEEE Trans. Knowl. Data Eng. 15 (3) (2003) 686–705.
[10] J.R. Fienup, Invariant error metrics for image reconstruction, Appl. Opt. 36 (32) (1997) 8352–8357.
[11] M. Garofalakis, A. Kumar, Deterministic wavelet thresholding for maximum-error metrics, in: Proceedings of the 23rd ACM SIGMOD Symposium on Principles of Database Systems, 2004, pp. 166–176.
[12] Y. Kamoshida, K. Taura, Scalable data gathering for real-time monitoring systems on distributed computing, 2008, pp. 425–432.
[13] E. Keogh, S. Chu, D. Hart, M. Pazzani, An online algorithm for segmenting time series, in: Proceedings of IEEE International Conference on Data Mining, ICDM, 2001, pp. 289–296.
[14] E.J. Keogh, M.J. Pazzani, A simple dimensionality reduction technique for fast similarity search in large time series databases, in: Current Issues and New Applications on Knowledge Discovery and Data Mining, Springer, 2000, pp. 122–133.
[15] R. Keralapura, G. Cormode, J. Ramamirtham, Communication-efficient distributed monitoring of thresholded counts, in: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, Chicago, IL, USA, 2006, pp. 289–300.
[16] D.E. Knuth, Art of Computer Programming, Volume 4, Fascicle 4, The: Generating All Trees—History of Combinatorial Generation, Addison-Wesley Professional, 2006.
[17] J. Lin, E. Keogh, S. Lonardi, B. Chiu, A symbolic representation of time series, with implications for streaming algorithms, in: Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2003, pp. 2–11.
[18] M.L. Massie, B.N. Chun, D.E. Culler, The ganglia distributed monitoring system: design, implementation, and experience, Parallel Comput. 30 (7) (2004) 817–840.
[19] S. Meng, L. Liu, Enhanced monitoring-as-a-service for effective cloud management, IEEE Trans. Comput. 62 (9) (2013) 1705–1720.
[20] OW2 Consortium, RUBiS: Rice University Bidding System, 2013. http://rubis.ow2.org.
[21] M. Pietri, S. Tosi, M. Andreolini, A. Balboni, Monitoring large cloud-based systems, in: Proceedings of 4th International Conference on Cloud Computing and Services Science, CLOSER, Barcelona, Spain, 2014.
[22] M. Pietri, S. Tosi, M. Andreolini, M. Colajanni, Real-time adaptive algorithm for resource monitoring, in: Proceedings of 9th International Conference on Network and Service Management, CNSM, Zurich, Switzerland, 2013.

[23] A. Rabkin, R. Katz, Chukwa: a system for reliable large-scale log collection, in: Proceedings of the 24th International Conference on Large Installation System Administration, LISA'10, USENIX Association, Berkeley, CA, USA, 2010, pp. 1–15.
[24] D. Rafiei, A. Mendelzon, Efficient retrieval of similar time sequences using DFT, in: Proceedings of FODO Conference, Kobe, 1998, pp. 249–257.
[25] A.G. Ramakrishnan, S. Saha, Ecg coding by wavelet-based linear prediction, IEEE Trans. Biomed. Eng. 44 (12) (1997) 1253–1261.
[26] Ravi Kanth, K.V. Divyakant Agrawal, Ambuj Singh, Dimensionality reduction for similarity searching in dynamic databases, in: ACM SIGMOD Record. Vol. 27. No. 2, ACM, 1998, pp. 166–176.
[27] Y. Simmhan, S. Aman, A. Kumbhare, R. Liu, S. Stevens, Q. Zhou, V. Prasanna, Cloud-based software platform for data-driven smart grid management, Comput. Sci. Eng. (2013) 1–11.
[28] Transaction Processing Performance Council, TCP-W, 2013. http://www.tpc.org/tpcw.
[29] University of Utah, Emulab—Total Network Testbed, 2013. http://www.emulab.net.
[30] C.J. van Rijsbergen, Information Retrieval, Butterworths, London, 1979.
[31] C. Wang, K. Schwan, V. Talwar, G. Eisenhauer, L. Hu, M. Wolf, A flexible architecture integrating monitoring and analytics for managing large-scale data centers, in: Proceedings of the 8th ACM International Conference on Autonomic Computing, New York, USA, 2011, pp. 141–150.
[32] Z. Xu, R. Zhang, R. Kotagiri, U. Parampalli, An adaptive algorithm for online time series segmentation with error bound guarantee, in: Proceedings of the 15th International Conference on Extending Database Technology, Berlin, Germany, 2012, pp. 192–203.

Mauro Andreolini is currently an assistant professor at the Department of Physics, Computer Science and Mathematics of the University of Modena, Italy. He received his master degree (summa cum laude) at the University of Roma, Tor Vergata, January, 2001 and his Ph.D. in May, 2005 from the same institution. His research focuses on the design, implementation and evaluation of distributed and cloud-based systems, based on a best-effort service or on guaranteed levels of performance.

His reviewing activity includes several international peer-reviewed journals (including IEEE Transactions on Parallel and Distributed Systems and ACM Performance Review).

He has been Program Committee member of international Conferences (including ETNGRID 2006 and 2007) and has been in the organizing committee for the IFIP PERFORMANCE2002 Conference.

He is a member of the IEEE and the ACM.

Michele Colajanni is full professor in computer engineering at the University of Modena and Reggio Emilia since 2000. He received the Master degree in computer science from the University of Pisa, and the Ph.D. degree in computer engineering from the University of Roma in 1992. He manages the Interdepartment Research Center on Security and Safety (CRIS), and the Master in "Information Security: Technology and Law". His research interests include security of large scale systems, performance and prediction models, Web and cloud systems.

Marcello Pietri obtained the Master's Degree in Computer science on Oct. 2010, and the title of Engineer in Jan. 2012.

He worked some years for many companies, and in particular in the Embedded and Wireless solutions field.

He is currently a Ph.D. Student in Information and Communication Technology (ICT) and he is going to finish his Doctorate at the beginning of the 2014.

He is also working as Research Assistant at the University of Modena, since Feb. 2012.

He published many papers and book chapters, especially about Monitoring and Cloud Computing.

Stefania Tosi is a Ph.D. student in Information and Communication Technologies at the University of Modena and Reggio Emilia, Italy. She received her master degree (summa cum laude and solemn commendation) in Computer Science from the University of Modena and Reggio Emilia in July 2010.

Her research interests include performance evaluation of modern data centers and statistical models for data management.

As visiting research scholar at the IBM T.J. Watson Research Center in Yorktown Heights, New York, she has been (and still is) actively involved in research projects related to performance analytics in multi-cloud environments.

She has four publications in international journals and several proceedings of key international conferences. She received a best paper award at WWW/Internet 2010.