Real-Time Adaptive Algorithm for Resource Monitoring

Mauro Andreolini, Michele Colajanni, Marcello Pietri, Stefania Tosi
University of Modena and Reggio Emilia
{mauro.andreolini,michele.colajanni,marcello.pietri,stefania.tosi}@unimore.it

***Abstract*—In large scale systems, real-time monitoring of hardware and software resources is a crucial means for any management purpose. In architectures consisting of thousands of servers and hundreds of thousands of component resources, the amount of data monitored at high sampling frequencies represents an overhead on system performance and communication, while reducing sampling may cause quality degradation.**

We present a real-time adaptive algorithm for scalable data monitoring that is able to adapt the frequency of sampling and data updating for a twofold goal: to minimize computational and communication costs, to guarantee that reduced samples do not affect the accuracy of information about resources.

Experiments carried out on heterogeneous data traces referring to synthetic and real environments confirm that the proposed adaptive approach reduces utilization and communication overhead without penalizing the quality of data with respect to existing monitoring algorithms.

***Index Terms*—Adaptive Sampling; Monitoring; Cloud Computing; Large-Scale; Scalability.**

I. Introduction

Resource management, anomaly detection and prevention in large scale architectures require continuous monitoring of system and software resources [1], [2]. The problem with frequently gathered and updated system information is that it may cause computational and traffic overhead [3], [4]. Hence, it is an open challenge to solve the trade-off intrinsic in monitoring tens of thousands of resources subject to real-time management objectives: keeping the overheads low and the quality of extracting system information high. Existing techniques are inefficient, unsuitable to real-time monitoring purposes, or are strongly coupled with specific architectures (e.g., [5]). For example, techniques based on adaptive dimensionality [6] require the whole data series, hence they cannot be used in real-time contexts; delta-encoding techniques based on static thresholds [7], [8] do not work because they do not take into account that in system monitoring the dataset tends to be highly variable.

This paper introduces a novel real-time adaptive algorithm that can be used as a monitoring mechanism for large architectures because it guarantees low computational and communication costs and high quality of data. It reaches this goal by adapting sampling intervals to data characteristics and state changes. When system behavior is relatively stable, it settles large sampling intervals so that the quantity of data that is gathered and sent to the analyzer is reduced. When significant differences between sampled data occur, the sampling frequency is augmented so to capture relevant changes in system performance. The proposed algorithm automatically chooses the best settings for monitoring sampling intervals, and it updates such settings through adaptive frequencies that aim to minimize the error of gathered data.

Experiments show that the proposed algorithm is able to reduce the computational and communication costs of monitoring up to 75% with respect to fine-grained static sampling, with an introduced error, with respect to complete sampling, that is below 8%. A further advantage of the proposed algorithm is that it leaves open the choice between reducing overhead and reducing quality. Better/worse quality in state representation can be achieved by reducing/augmenting overhead saving. These results represent a major improvement with respect to state-of-the-art techniques which either are accurate and resource expensive or they reduce computational and communication costs by worsening too much the quality of sampled data [3], [7], [9], [10].

The remainder of this paper is organized as following. Section II defines the problem of real-time monitoring for large-scale systems. Section III presents the proposed adaptive monitoring algorithm. Section IV analyzes experimental results achieved on synthetic and real scenarios. Section V compares this paper against the state-of-the-art. Section VI concludes the paper with some final remarks.

II. Monitoring challenges

In large scale architectures, the aim of real-time monitoring is to collect, store and process large volumes of data in order to obtain two types of views: the behavior of the whole system, and the details of critical or interesting components [2], [11].

We refer to a typical monitoring architecture (e.g., [2]–[4], [12]–[14]) shown in Figure 1 consisting of two logical layers: 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 (*sending phase*). The algorithms considered in this paper can be applied at the collection and at the sending phase. The details about other components, such as data storage, data analyzer for system knowledge, and management decisions are beyond the scope of this paper.

In the collection phase, system metrics are collected on the basis of the proposed adaptive algorithm and then sent to the collectors: we can adapt sampling intervals by choosing the best trade-off between quality and computation/communication overheads. In the sending phase, the al-

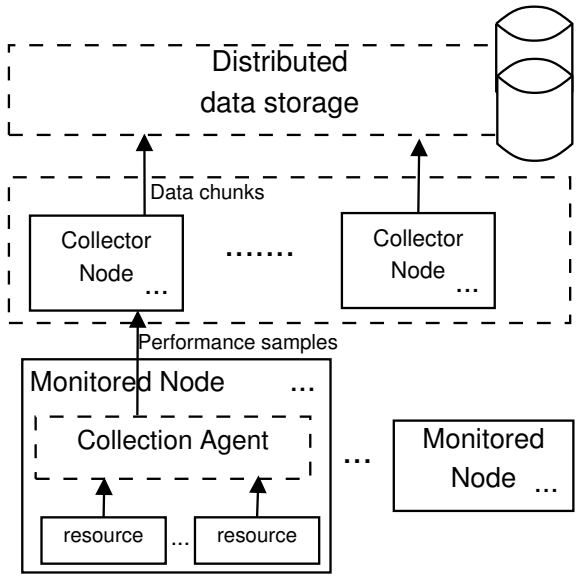


Fig. 1: Monitoring Architecture

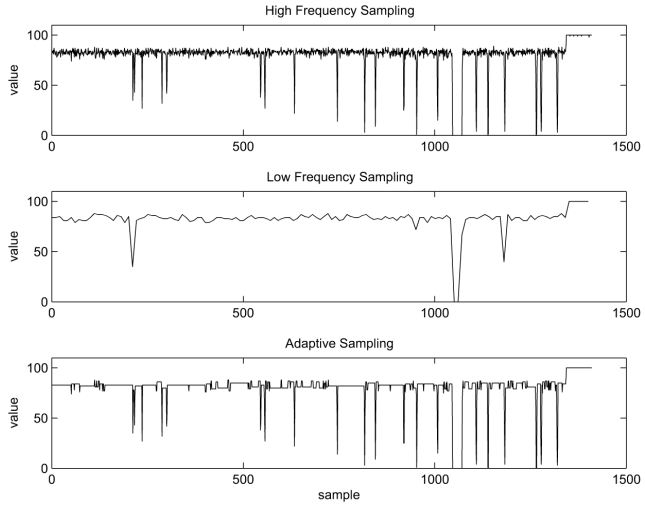


Fig. 2: Comparison of high, low and adaptive monitoring

gorithm is applied to already collected samples, hence the margins of adaptivity operate on quality vs. communication savings. On the positive hand, it is possible to adapt the parameters of the algorithm by comparing the recently sampled data with previous collected, analyzed and forwarded data.

Several other methods (e.g., [7], [8]) are based on data collection at fixed sampling time intervals and forwarding of new information only if it differs by some static numeric threshold. Although this approach can reduce the amount of gathered and transmitted data, it may lead to highly inaccurate results. The proposed algorithm allows system managers to choose the preferred trade-off between overhead reduction and information quality.

Figure 2 reports three scenarios. The top figure represents the real system behavior where samples are gathered at maximum frequency (e.g., one second). The middle figure denotes a system representation where overhead savings are preferred with respect to information quality; this is achieved through a low frequency sampling method that guarantees a reduction of the amount of collected data but also a loss of evidence of the majority of load spikes. The bottom figure refers to the proposed adaptive algorithm that aims to preserve the most useful information of the real system trace but also to reduce the amount of collected data with respect to continuous sampling.

We should also consider that the quality of sampled and transmitted data may be affected by several sources of error.

- **Delay.** When data are gathered at fixed time intervals, there are time windows in which significant changes may occur while system collectors and analyzers do not notice them. This delay affects the quality of monitoring approaches especially when they tend to use large sampling intervals in order to limit the quantity of collected and transmitted data.
- **Static sampling.** In large scale architectures consisting of several resources, it is impossible to anticipate the best monitoring frequency for each resource because they are heterogeneous and typically highly variable, and it is impracticable to rely on human intervention. Hence, any static choice about threshold values and/or sampling interval risks to be a significant source of errors.

We can conclude that real-time adaptive sampling is mandatory in large scale architecture. The goal now is to introduce an effective real-time monitoring algorithm that must guarantee the best trade-off between the amount of collected and transmitted data and the effects of sampling reduction on the quality of system representation.

III. Adaptive monitoring

The real-time adaptive monitoring proposed in this paper consists of three adaptive algorithms: one for the collection phase (Adaptive Sampling), that is used for the mainstream description of the overall approach; one for the sending phase (Adaptive Forwarding); and one for the continuous check and re-evaluation of the parameters of the algorithms after the training phases (Adaptive Supervision). We recall that any monitoring system must solve a basic trade-off that is, reducing the quantity of collected data reduces the capacity of achieving a realistic representation of a system resource behavior. Our algorithms do not attempt to find the solution that fits for all scenarios because this goal is impossible. On the other hand, we leave open such trade-off through the parameters $Gain$ denoting the reduced overhead, and $Quality$ denoting the ability of representing the real system behavior. The trade-off between $Gain$ and $Quality$ is represented through the $Eval$ metric as:

$$Eval = w \cdot Gain + (1 - w) \cdot Quality \qquad (1)$$

where $w \in (0, 1)$ is a parameter of choice. As the quantity of saved data impacts on the quality of the representation, the

proposed algorithms allow the system administrator to decide how to regulate this trade-off. Values of $w > 0.5$ put more emphasis on $Gain$ than $Quality$; the vice versa is true for $w < 0.5$, while $w = 0.5$ gives equal importance to both metrics. We advance that for reason of space we leave the discussion about the tuning of this parameter for a future paper.

There are several possible measures for $Gain$ and $Quality$, but we find useful to adopt the following choices. In our algorithms, $Gain$ is the ratio between the number of data collected by the proposed algorithms and the number of data collected by the baseline algorithm that samples data at the highest frequency. Higher $Gain$ values denote algorithms aiming to reduce the computational and communication overhead of monitoring.

$Quality$ denotes the ability of an algorithm to represent the state changes of a system resources in the most accurate way. In our algorithm, $Quality$ in its turn is a combination of the Normalized Root Mean Square Error (NRMSE) [15], and the F-measure that is the weighted average of the precision and recall.

$$Quality = \frac{Fmeasure + (1 - NRMSE)}{2} \tag{2}$$

where we recall that $Fmeasure$ and NRMSE denote values $\in (0, 1)$. The motivation for using two measures instead of one is that the dataset in the considered scenarios are highly variable, hence the NRMSE alone is unable to guarantee an accurate quality measure (see for example the discussions in [16]). For this reason, we consider also $Fmeasure$ [17] as the measure of the capacity of the monitoring algorithm to identify significant load spikes. Hence, by referring to [17], a true positive is counted when a real spike is identified, while a false positive denotes the identification of a spike that has no correspondence in the reality; a true negative indicates the correct absence of spikes, and a false negative denotes a lack of identification of a spike.

A. Parameters of the adaptive algorithms

The proposed real-time adaptive algorithms work by distinguishing periods of relative stability from periods of high variability of the monitored resources. The basic idea is to limit computational and communication overhead, and at the same time to guarantee that important system changes are not missed . To this purpose, we reduce the quantity of monitored data when a resource is relatively stable, and we increase it during periods of high variability.

The adaptivity of the algorithms is based on the choice of the following two main parameters:

1) The sampling interval t determines the time interval that elapses from the collection of one sample to the successive. The lower the sampling interval, the higher the number of data to gather and transmit.
2) The tolerable variability Δ discriminates stable from variable states by determining the value of deviation between consecutive samplings. When the difference among samples is lower than Δ, the monitored resource is considered to be in a stable state; when this difference is higher than Δ, the resource is considered as highly variable. The lower the tolerable variability Δ, the higher the probability of the resource to incur some relevant data changes.

We should consider that the algorithms can be applied to resource states characterized by different ranges of values (e.g., the CPU utilization goes from 0 to 100%, the network bandwidth from 0 to say 10Gbps). Hence, in the training phase we use a percentage $\Delta_{\%}$ that is independent of the specific monitored values. After the training phase, we use Δ that is related to the values of each monitored resource. It is obtained by evaluating the first quartile $Q1$ and third quartile $Q3$ of all points collected during the training phase, and by referring to percentage $\Delta_{\%}$ through the following equation [18]:

$$\Delta = \Delta_{\%} \cdot 1.5 \cdot (Q3 - Q1) \tag{3}$$

B. Adaptive sampling algorithm

The adaptive sampling algorithm is characterized by an initial training phase that is used for the self-choice of all parameters: the minimum t_m^s and the maximum t_M^s of the sampling period, the threshold for the identification of a peak Δ_c^s, and the threshold that identifies the lowering of sampling interval at minimum Δ_t^s.

At the beginning of the training phase, we set the minimum and maximum possible values for each of these parameters. The training phase evaluates the best values for them by choosing those that maximize $Eval$ in (1). This search has an exponential cost in terms of complexity because in the naive form we have to evaluate all combinations of the representations of data sampled during the training phase. This does not represent a real problem because this training is done once, on λ^s data points x of the series X. However, we reduce this initial computational cost by adopting a binary search that is able to pass to a log_2 complexity of the naive search, with respect to the parameters t_m^s,t_M^s,Δ_c^s and Δ_t^s. For reason of space we postpone the details of the computational costs of the algorithms in a future paper, but we advance that the average cost of the proposed algorithm (during many twelve-hour experiments) is between the cost of static algorithms using t=t_m and t=t_M. At the end of the training phase, we have the best values associated to each parameter as $\{\bar{t_m^s}, \bar{t_M^s}, \bar{\Delta_c^s}, \bar{\Delta_t^s}\}$.

After the training phase, the sampling phase uses the Adaptive Sampling Function (Alg. 1) for the next samples x of X. It initially sets Δ^s to zero (no error), and its sampling interval t^s to $\bar{t_m^s}$. Until there are data to collect, Δ^s is increased by the current error $\delta_i = x_i^s - x_{pi}^s$, where x_i^s is the current sample and x_{pi}^s is the last sampled one. When the absolute value of Δ^s exceeds the threshold $\bar{\Delta_c^s}$, and $\bar{\Delta^s}$ exceeds $\bar{\Delta_t^s}$ (high difference between the two points), then the sampling interval t^s is set to t_m^s. Otherwise, if Δ^s exceeds $\bar{\Delta_c^s}$ but not $\bar{\Delta_t^s}$, then the sampling interval t^s is decreased. In both cases, Δ^s is set to zero because the last sampled point becomes x_i

and the error is set again to null. If Δ^s has not exceeded $\bar{\Delta}^s_c$, this means that the values of the two points are similar and the state is stable, so the sampling interval t^s can be increased.

Algorithm 1 Adaptive sampling function

$t^s \Leftarrow t^{\bar{s}}_m$
$\Delta^s \Leftarrow 0$
pi$\Leftarrow \lambda^s$
i$\Leftarrow$ pi+t^s
while True **do**
  $\Delta^s \Leftarrow \Delta^s$ + (x_i - x_{pi})
  if $|\Delta^s| > \bar{\Delta}^s_c$ **then**
    $X^s \Leftarrow \{X^s, x_i\}$
    if $|\Delta^s| > \bar{\Delta}^s_t$ **then**
      $t^s \Leftarrow t^{\bar{s}}_m$
    else
      $t^s \Leftarrow \max(t^{\bar{s}}_m, t^s - t^{\bar{s}}_m)$
    $\Delta^s \Leftarrow 0$
  else
    $t^s \Leftarrow \min(t^{\bar{s}}_M, t^s + t^{\bar{s}}_m)$
  pi$\Leftarrow$ i
  i$\Leftarrow$ i + t^s

C. Adaptive forwarding algorithm

The Adaptive Forwarding algorithm is similar to the Adaptive Sampling algorithm. The main difference is that now we can dynamically evaluate the quality of collected data X^s with respect to the state representation X^r and we can decide whether to forward to the collector all sampled data or just a subset of them because the quality would not degrade and we would limit overhead.

Even in this phase, we have a training phase (λ^r data points) for the self-choice of the parameters of the algorithm: t^r_m and t^r_M are the minimum and the maximum values of the sampling intervals, respectively; Δ^r_c and Δ^r_t are the threshold that identifies a peak and the threshold after which t^r must be set to t^r_m, respectively. The best choices $\{t^{\bar{r}}_m, t^{\bar{r}}_M, \bar{\Delta}^r_c, \bar{\Delta}^r_t\}$ are identified through a binary search that evaluates the best $Eval$ for all combinations of feasible parameters.

After the training phase, the Adaptive forwarding algorithm uses the Adaptive Forwarding function described in (Alg. 2). In this case, each collected point in X_c is forwarded ($X^r \Leftarrow \{X_r, x^c_i\}$) when the absolute value of Δ^r exceeds $\bar{\Delta}^r_c$.

Even more important, this algorithm is able to adapt dynamically its parameters $\{t^{\bar{r}}_m, t^{\bar{r}}_M, \bar{\Delta}^r_c, \bar{\Delta}^r_t\}$ on the basis of the comparison between collected and new data. The goal is to avoid a degradation of the $Quality$ measure through the so called *Adaptive Supervision algorithm*. It evaluates the error between the collected series X^s and the forwarded series X^r. The error evaluation is performed each λ^r samples through the $Quality$ function defined in (2). When the evaluated $Quality$ is lower than w (1), then the algorithm re-evaluates the parameters $\{t^{\bar{r}}_m, t^{\bar{r}}_M, \bar{\Delta}^r_c, \bar{\Delta}^r_t\}$, by adapting them on the basis of the last sampled data. This Adaptive Supervision algorithm allows the monitoring model to adapt its parameters to time series that are characterized by different trends with respect to the dataset analyzed during the training phase.

Algorithm 2 Adaptive Forwarding function

$t^r \Leftarrow t^{\bar{r}}_m$
$\Delta^r \Leftarrow 0$
pi$\Leftarrow \lambda^r$
lpi$\Leftarrow$ pi
i$\Leftarrow$ pi+t^r
while i < length(X^s) **do**
  $\Delta^r \Leftarrow \Delta^r$ + (x^s_i - x^s_{pi})
  if $|\Delta^r| > \bar{\Delta}^r_c$ **then**
    $X^r \Leftarrow \{X_r, x^c_i\}$
    if $|\Delta^r| > \bar{\Delta}^r_t$ **then**
      $t^r \Leftarrow t^{\bar{r}}_m$
    else
      $t^r \Leftarrow \max(t^{\bar{r}}_m, t^s - t^{\bar{r}}_m)$
    $\Delta^r \Leftarrow 0$
  else
    $t^r \Leftarrow \min(t^{\bar{r}}_M, t^s + t^{\bar{r}}_m)$
  pi$\Leftarrow$ i
  i$\Leftarrow$ i + t^r
  if (pi - lpi) > λ^r **then**
    $\{t^{\bar{r}}_m, t^{\bar{r}}_M, \bar{\Delta}^r_c, \bar{\Delta}^r_t\} \Leftarrow$AdaptiveSupervision($X^s$[lpi,. . . ,pi], X^r[lpi,. . . ,pi])
    lpi$\Leftarrow$ pi

IV. Performance evaluation

In this section we report the most significant experimental evaluations that demonstrate the ability of the proposed algorithms in satisfying the requirements of adaptivity, quality and quantity introduced in Section II.

A. Experimental platforms

For the experiments we used the monitoring platform deployed on Amazon EC2 [19] and Emulab [20], that is also described in [14]. In the considered testbed, the monitored nodes execute different applications such ad Apache2, MySQL, Java and PHP programs subject to TCP-W [21] and RUBiS [22] workload models. MapReduce jobs and MongoDB queries are used for data distribution and analysis.

We present the results obtained from experiments performed over more than 1,000 nodes that generate about 140K series, each lasting for about 12 hours. For comparison purposes, we varied the sampling intervals from 1 to 30 seconds. Each node is characterized by 25 types of performance indicators and related series; each process is characterized by 20 time series; moreover, we have 12 specific statistics related to Apache users, MySQL and MongoDB queries, MapReduce jobs. The presented results are related to $w = 0.5$ that is, we consider that Gain and Quality have the same importance. Evaluations

related to different weights are not reported because of space reasons.

We initially consider the Adaptive Sampling algorithm that is related to the collection phase. The evaluations are carried out by comparing the results referring to the sampled series X_i^s against those of the series sampled every second X_i ($i = 1, \ldots, 140K$). This comparison can be achieved only at the end of the experiments when the latter series are completed.

In particular, we aim to find:

- the relations between training parameters and algorithm performance, and the impact of binary search applied to the choice of the $Eval$ parameter;
- the dependence of algorithm's performance on series related to different resources, and its robustness with respect to sampling intervals and threshold parameters;
- the performance of the proposed algorithms compared to existing methods.

B. Training phase

We apply the Adaptive Sampling algorithm to the whole series by considering different intervals for training: from 5 to 200 samples. We focus on four macro groups (CPU, memory, network, disk) because they include system and process performance. Due to the different amount of metrics associated to each group, the number of analyzed series are 20K for the CPU, 26K for the memory, 46K for the network, 11K for the disk and 37K for the other metrics (Other). Table I reports the Gain and the Quality divided by group and related to the dimension of the training set λ^s. The best value for $Eval$ is calculated as the average between Gain and Quality (1) shown in Section III. The results for $Eval$ as a function of the training set size λ^s are shown both in Figure 3 and in Table II. Figure 3 shows the best values for $Eval$ divided by group, while Table II shows the results summary for all the 140K series.

TABLE I: Performance evaluation

λ^s	Ev. Parameters	CPU	Memory	Network	Disk
5	Gain (%)	76,96	26,43	32,53	95,41
	Quality (%)	85,45	99,99	99,84	81,12
10	Gain (%)	92,75	48,84	62,42	97,97
	Quality (%)	84,28	99,06	98,34	93,30
25	Gain (%)	95,69	38,14	70,74	92,63
	Quality (%)	86,09	96,77	81,56	86,05
50	Gain (%)	95,37	46,56	75,71	98,32
	Quality (%)	85,53	98,95	82,42	84,19
100	Gain (%)	95,19	53,83	76,98	98,39
	Quality (%)	78,74	95,32	94,80	78,24
150	Gain (%)	96,77	51,37	75,61	98,91
	Quality (%)	81,10	93,17	89,49	78,61
200	Gain (%)	96,54	40,65	80,23	98,85
	Quality (%)	80,60	98,14	76,50	71,43

By using a small training set size, the parameters $t_m^{\bar{s}}$ and $t_M^{\bar{s}}$ tend to be small (e.g., from 1 to 3, and from 1 to 5 times of the sampling interval of the original series, using a λ^s of 10 samples). This result is motivated by inspecting the function responsible to convert the thresholds (expressed in percentage) into the absolute series-related values, since the majority of load spikes related to few data points becomes smoothed on a longest series, and vice versa. This trend is also visible looking at the results for Gain and Quality. When using small λ^s and/or small $t_m^{\bar{s}}$ and $t_M^{\bar{s}}$, the Gain is relatively low, while the Quality is high. By increasing λ^s, the Quality decreases while the Gain increases. This tendency is visible in all results, but especially for network and disk resources. Moreover, the results show that a λ^s of at least 10 samples is enough to obtain high performance (AVG Gain 75.5% Quality 93.75%). This result helps to reduce the cost of the algorithm during the training phase, in which large training sets seem not required.

We perform some experiments in order to demonstrate the robustness of the Adaptive Sampling algorithm with respect to different settings of the sampling interval t^s, the threshold parameters Δ^s and the training sets. We added and subtracted not more than 20% from the value of the best t_m^s, t_M^s, Δ_c^s, Δ_f^s previously calculated. We also use different training set of the same λ^s length into each series. We obtain that the averages of Gain and Quality differ of at most 15% from the best values in the 96.51% of cases, from 15% to 25% in the 3.47% of cases, and we have less than 0.02% of outliers.

TABLE II: Performance evaluation summary

λ^s	Fmeasure (%)	NRMSE (%)	Quality (%)	Gain (%)	Eval (%)
5	95,20	87,11	91,16	56,86	74,01
10	95,57	88,60	92,09	75,13	83,61
25	91,31	83,67	87,49	73,70	80,60
50	91,00	84,26	87,63	78,90	83,27
100	89,99	83,48	86,74	81,09	83,92
150	89,48	81,42	85,45	80,53	82,99
200	84,48	78,53	81,51	79,03	80,27

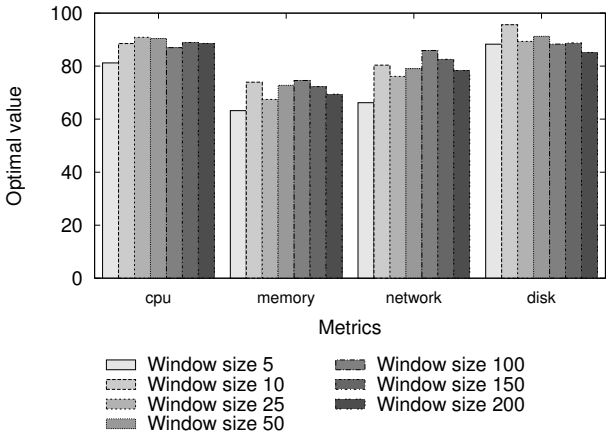
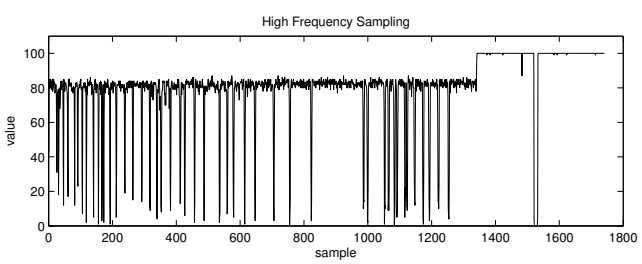
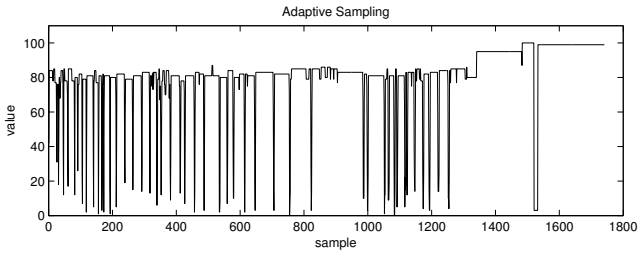
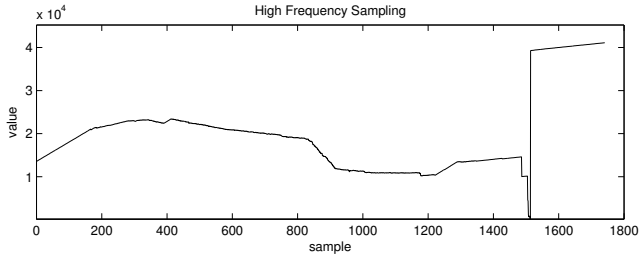


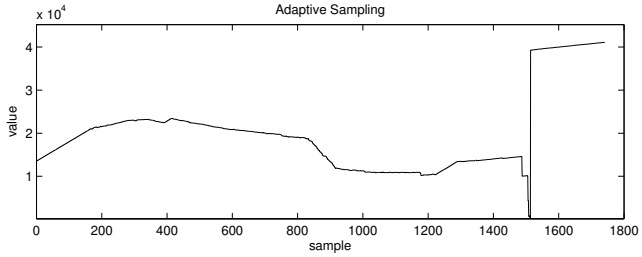
Fig. 3: Best values for $Eval$ as a function of the window size used for training



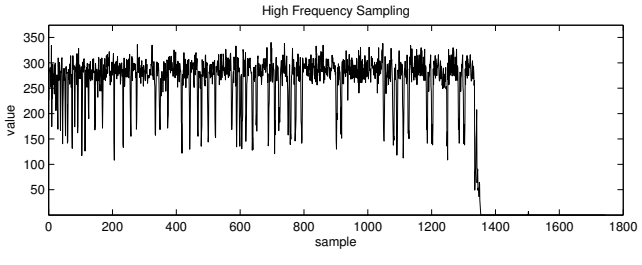


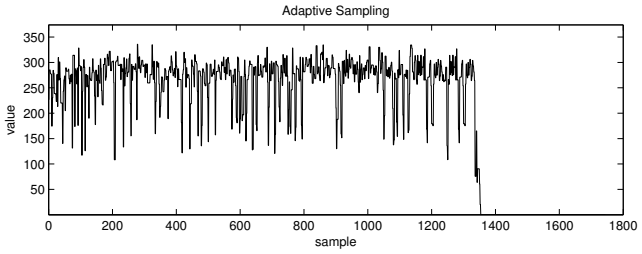
(a) CPU (idle)



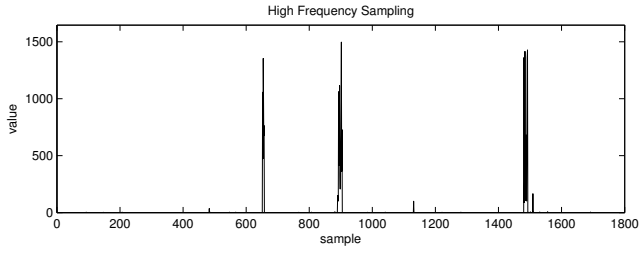


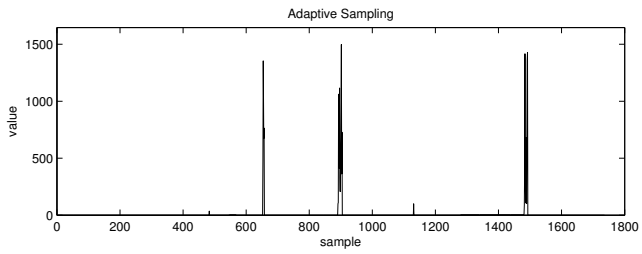
(b) Memory (buff)





(c) Network (txkBs)





(d) Disk (block in)

Fig. 4: Examples of high-sampled and reduced series using the adaptive algorithm for each type of resource

C. Impact of type of series on performance

We evaluate the performance of the Adaptive Sampling algorithm as a function of different types of series. We present some significative examples for each group of resources in Figures 4a, 4b, 4c and 4d. The four figures shows at the top the series sampled at the highest frequency, while at bottom the series are sampled with the Adaptive Sampling algorithm. Table III reports the values of Gain and Quality related to each class of resource.

TABLE III: Results of Gain and Quality (ref. Fig. 4)

	CPU	**Memory**	**Network**	**Disk**
Gain (%)	*85,83*	*51,44*	*66,38*	*97,75*
Quality (%)	*99,82*	*99,85*	*98,53*	*100,00*

The results show that the value of Quality tends to be very high in any instance: series with high variability such as CPU and network (Figures 4a and 4c); stable series such as memory and disk (Figures 4b and 4d). On the other hand, the Gain value is high (98%) in correspondence of stable series (e.g., disk), while decreases (51%) for series presenting a recursive linear trend (e.g., memory). In variable series, the Gain grows according to decreasing number of load peaks: it is 66% in highly variable network samples, but it is 86% for the CPU. By using the adaptive algorithm, the Gain is nearly always more than the 50% while its average is about the 75%. Since the average Quality is always more than 81%, these tests demonstrate that the proposed adaptive algorithm is suitable for any time series related to systems or processes performance.

One of the problems is related to the computational com-

plexity of the $Eval$ computation. For this reason, we adopted also an adaptive binary search and compared its results with those obtained by the complete search. We cannot use the basic binary search because it does not guarantee that it can find all peaks because of the peculiarity of most considered series. The experiments shows that an adaptive binary search achieves the same results in the 93.08% of cases, similar results ($\pm 5\% Eval$) in the 6.51% of cases, and just 0.40% of outliers. By using this technique, it is possible to reduce the computational cost of the training phase to log_2 while maintaining a precision level higher than 99.2% in more than 99.6% of cases with respect to the naive search.

D. Performance comparison

We finally compare the results obtained using the proposed adaptive algorithms against those obtained by the state-of-the-art algorithms based on delta-encoding and static thresholds (e.g. [7]), and that considering a static sampling frequency.

TABLE IV: Algorithm based on static frequency sampling

t	Fmeasure (%)	NRMSE (%)	Quality (%)	Gain (%)	Eval (%)
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

From the experiments over all the 140K series, we note that the static sampling frequency algorithm has a Gain equal to $1 - 1/t$ (e.g., 50% t=2, 80% t=5), but it is affected by a low quality of data: Fmeasure ranges from about 72% when the sampling interval t is very low (t=2) to 33% when t=20 (Tab. IV). Moreover the Quality is quite low if considering the related frequency of sampling (e.g. 80,76% with t=2). The most important result is that it never achieves an $Eval$ higher than 76%. The sampling interval t of the algorithm based on delta-encoding and static thresholds is fixed, hence this value must be chosen a priori. Table V reports just the Gain related to the sending phase, because the results for the collection phase are identical to those shown for the static sampling algorithm.

We finally report the best results obtained by the proposed adaptive algorithm in Tab. VI, using the parameters calculated in Section IV-B.

The algorithm based on delta-encoding and static thresholds has an average $Eval$ of about 70,65%. Its best performance is $Eval$=77,61% (by using t=5 and $\Delta^r_{c\%}$=10). By comparing these values with the results obtained with our Adaptive Sampling algorithm ($Eval$=83,92%), we note that the proposed algorithm increases the performance of more than 6% with respect to the best value of the algorithm based on delta-encoding and static thresholds, and more than 13% with respect to its average. We emphasize also that the average $Eval$ of the Adaptive Sampling algorithm is better than 10% with respect to the average $Eval$ of the static algorithm. We should consider that the parameters that made the Adaptive Sampling algorithm static are very rarely chosen (less than 3% of times), since the inequality between t_m and t_M allows to achieve a better trade-off between Gain and Quality. The $Eval$ value of about 83% is robust for any choice of the training set size larger than 10 samples and lower than 200 samples.

TABLE V: Delta encoding and static threshold algorithms

t	$\Delta^r_{c\%}$	Fmeasure (%)	NRMSE (%)	Quality (%)	Gain (%)	Eval (%)
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

TABLE VI: Adaptive algorithm

Fmeasure (%)	NRMSE (%)	Quality (%)	Gain (%)	Eval (%)
89,99	83,48	86,74	81,09	83,92

V. Related Work

To reduce the overhead of data acquisition in large system monitoring, we can distinguish lossy and lossless techniques that are mainly oriented to time series data compression. Classical lossless techniques cannot achieve a compaction rate as high as methods which are customized to the nature of time series, but they can be applied to data independently [10].

Lossy techniques can be further distinguished between offline and online schemes. Offline techniques need to obtain the whole series, while the online techniques process the data points on the fly. Other techniques, such as adaptive dimensionality [6], differ from the proposed approach because they aim to reduce the dimensionality of series.

The field of offline lossy techniques is rich of proposals, such as Fast/Discrete Fourier [23] and Wavelet [24] Transform, Piecewise Aggregate Approximation [25], Singular Value Decomposition [26], Symbolization [27] and Histograms [28]. Some recent studies achieve probabilistic or deterministic error bounds on individual data points [29], but these algorithms work by assuming the knowledge of the entire time series [10],

hence they are unsuitable to real-time monitoring contexts considered in this paper. Other approaches based on linear segmentation (e.g., PLA [9], sliding window [30]) 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 point of a segment and use the next data point to evaluate the approximation error. The three main variants of this algorithm improve the quality for specific data sets, but they are not robust if we consider arbitrary datasets [9], [10].

The state-of-the-art in the field of online lossy techniques can be distinguished between delta-encoding techniques based on static thresholds [7], [8], and architecture-related aggregation techniques [5]. These 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 Gain and Quality that represents the main novelty of the proposed approach. An accurate evaluation of the improvements of an adaptive approach with respect to the state of the art is carried out in the previous section.

VI. Conclusion

We propose a real-time adaptive algorithm for scalable data monitoring that is able to adapt dynamically 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 you have to support gathering and analysis operations of huge numbers of data streams coming from large and heterogeneous resource monitors. A large set of experiments performed on real and synthetic traces show that the proposed adaptive algorithm reduces the resource utilization and the communication overhead without penalizing the quality of data with respect to the state-of-the-art real-time algorithms.

References

[1] A. Keller and H. Ludwig, "The wsla framework: Specifying and monitoring service level agreements for web services," *Journal of Network and Systems Management*, vol. 11, pp. 57–81, 2003.

[2] C. Wang, K. Schwan, V. Talwar, G. Eisenhauer, L. Hu, and M. Wolf, "A flexible architecture integrating monitoring and analytics for managing large-scale data centers," in *Proc. of the 8th ACM international conference on Autonomic computing*, New York, USA, 2011, pp. 141–150.

[3] S. Meng and L. Liu, "Enhanced monitoring-as-a-service for effective cloud management," *Computers, IEEE Transactions on*, 2012.

[4] A. Rabkin and R. Katz, "Chukwa: a system for reliable large-scale log collection," in *Proceedings of the 24th international conference on Large installation system administration*, ser. LISA'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–15.

[5] S. Böhm, C. Engelmann, and S. L. Scott, "Aggregation of real-time system monitoring data for analyzing large-scale parallel and distributed computing environments," in *Proceedings of the* 12^{th} *IEEE International Conference on High Performance Computing and Communications (HPCC) 2010*, Melbourne, Australia, Sep. 2010, pp. 72–78.

[6] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani, "Locally adaptive dimensionality reduction for indexing large time series databases," *ACM Trans. Database Syst.*, vol. 27, no. 2, pp. 188–228, Jun. 2002.

[7] Y. Kamoshida and K. Taura, "Scalable data gathering for real-time monitoring systems on distributed computing," *Cluster Computing and the Grid, IEEE International Symposium on*, pp. 425–432, 2008.

[8] G. Cormode, R. Keralapura, and J. Ramimirtham, "Communication-efficient distributed monitoring of thresholded counts," in *ACM SIGMOD International Conference on Management of Data*, 2006.

[9] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "An online algorithm for segmenting time series," in *Proceedings IEEE International Conference on Data Mining (ICDM)*. IEEE, 2001, pp. 289–296.

[10] Z. Xu, R. Zhang, R. Kotagiri, and 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*. ACM, 2012, pp. 192–203.

[11] C. Canali and R. Lancellotti, "Automated clustering of virtual machines based on correlation of resource usage," *Journal of Communications Software and Systems (JCOMSS)*, vol. 8, no. 4, Jan. 2013.

[12] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817 – 840, 2004.

[13] M. Andreolini, M. Colajanni, and 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)*, September 2011, pp. 389 –394.

[14] M. Andreolini, M. Colajanni, and M. Pietri, "A scalable architecture for real-time monitoring of large information systems," in *IEEE Second Symposium on Network Cloud Computing and Applications*, ser. IEEE NCCA, December 2012, pp. 143–150.

[15] J. R. Fienup, "Invariant error metrics for image reconstruction," *Applied optics*, vol. 36, no. 32, pp. 8352–8357, 1997.

[16] A. G. Ramakrishnan and S. Saha, "Ecg coding by wavelet-based linear prediction," *Biomedical Engineering, IEEE Transactions on*, vol. 44, no. 12, pp. 1253–1261, 1997.

[17] C. J. van Rijsbergen, "Information retrieval," *Butterworths*, 1979.

[18] J. Han, M. Kamber, and J. Pei, *Data mining: concepts and techniques*. Morgan kaufmann, 2006.

[19] Amazon, "Elastic compute cloud (ec2)," http://aws.amazon.com/ec2.

[20] University of Utah, "Emulab - total network testbed," http://www.emulab.net, 2013.

[21] Transaction Processing Performance Council, "Tcp-w," http://www.tpc.org/tpcw, 2013.

[22] OW2 Consortium, "Rubis: Rice university bidding system," http://rubis.ow2.org, 2013.

[23] D. Rafiei and A. Mendelzon, "Efficient retrieval of similar time sequences using dft," *arXiv preprint cs/9809033*, 1998.

[24] F.-P. Chan, A.-C. Fu, and C. Yu, "Haar wavelets for efficient similarity search of time-series: with and without time warping," *Knowledge and Data Eng., IEEE Transactions*, vol. 15, no. 3, pp. 686–705, 2003.

[25] E. J. Keogh and M. J. Pazzani, "A simple dimensionality reduction technique for fast similarity search in large time series databases," in *Knowledge Discovery and Data Mining. Current Issues and New Applications*. Springer, 2000, pp. 122–133.

[26] K. Ravi Kanth, D. Agrawal, and A. Singh, "Dimensionality reduction for similarity searching in dynamic databases," in *ACM SIGMOD Record*, vol. 27, no. 2. ACM, 1998, pp. 166–176.

[27] J. Lin, E. Keogh, S. Lonardi, and 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*. ACM, 2003, pp. 2–11.

[28] B. Babcock, S. Babu, M. Datar, R. Motwani, and 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*. ACM, 2002, pp. 1–16.

[29] M. Garofalakis and A. Kumar, "Deterministic wavelet thresholding for maximum-error metrics," in *Proceedings of the 23th ACM SIGMOD symposium on Principles of database systems*, 2004, pp. 166–176.

[30] U. Appel and A. V. Brandt, "Adaptive sequential segmentation of piecewise stationary time series," *Information sciences*, vol. 29, no. 1, pp. 27–56, 1983.