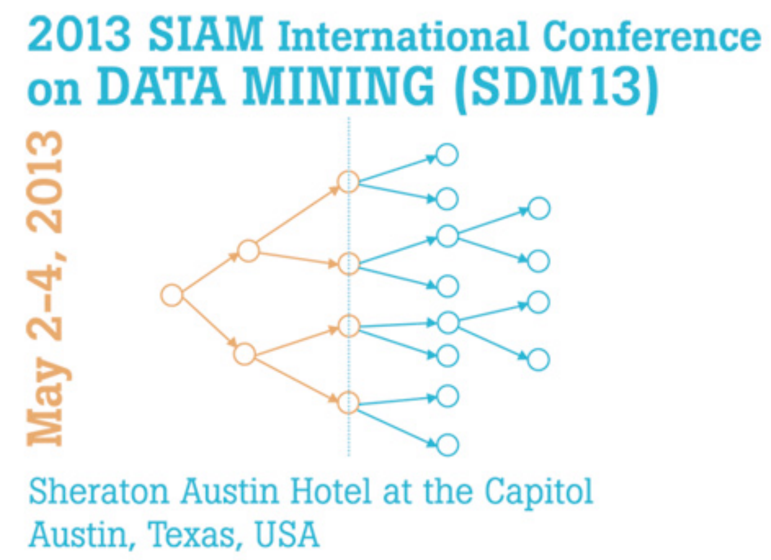
SDM4service

WORKSHOP NOTES
Second Workshop on Data Mining for Service and Maintenance

Sheraton Austin Hotel at the Capitol
Austin, TX
May 4, 2013
URL: http://www.siam.org/meetings/sdm13/

Organizers:
Nikunj C. Oza (NASA Ames Research Center)
Dragos Margineantu (Boeing Research and Technology)
Shipeng Yu (Siemens Healthcare)
Matan Ninio (IBM Research Haifa)


2013 SIAM International Conference
on DATA MINING (SDM13)
May 2–4, 2013
Sheraton Austin Hotel at the Capitol
Austin, Texas, USA

Background

The maintenance and servicing of system components, machines, or instruments are a crucial part of business for many industries for systems ranging from photocopiers to power plants and from trucks and locomotives to airplanes. It is a common industry standard to provide system service as part of the contractual agreement, and the quality of service is increasingly becoming the key differentiator between competing products. Appropriate and timely maintenance can increase the safety and reliability of the system, detect failing components to reduce collateral damage to other healthy subsystems, and ultimately increase revenue and improve customer satisfaction.

Instead of standard practices such as fixed maintenance schedules (by usage time) or performing reactive diagnostics maintenance (i.e., fixing the problem after it happens), increasingly industries are moving toward a so-called proactive service mode (which is often called condition monitoring or preventive maintenance, among other names). This is also enabled by the sensor data collected by most of the systems mentioned above. Hence, these systems can be monitored and serviced more efficiently by employing intelligent service data analytics solutions that lead to a proactive (act before the problem has occurred) timely maintenance schedule. Several such simple solutions have been already deployed but, based on the increased amounts of service data generated by all these systems, better and improved solutions are possible. The service data includes sensor data, system error messages, event log files, service & maintenance data, and other types data on machine/system status. The key questions here are (1) how to analyze this data effectively and efficiently in order to provide necessary service to the systems, while minimizing service cost and (2) how to employ this data for online decision support. Most of the data is time-series or event sequence data but these systems and processes also generate unstructured or semi-structured data such as text. Therefore the data analytics processes are challenging, as they require dealing with noise in data, unaligned multivariate data, missing and compressed data, and a multitude of data types. The complexity and dynamics of the systems that generate the data (e.g., aircraft engines) also lead to system-to-system variations and a large number of failure modes (i.e., thousands of categories of classes).

Because of its unique characteristics mentioned above, service data analytics is a domain in which data mining and knowledge discovery techniques just started to be applied and it requires the attention of both researchers and practitioners. Novel innovative methods in time series analysis and prediction, relational learning, multi-view learning, transfer learning or incomplete data analysis can have a profound effect.

We believe that this new domain can benefit from increased attention from our SIAM data mining community. There are various other communities that deal with systems health management such as the Prognostics and Health Management society and the DX Workshop community, but they tend to deal more with individual components or systems, and therefore have more of an engineering perspective that is reflected in their emphasis on model-based or physics-based approaches, even though they are increasingly investigating more data-driven approaches.

Objectives

The purpose of this multi-disciplinary workshop is to bring together researchers and industry experts in the fields of data mining, machine learning, text analysis and signal processing who share an interest in problems and applications of system service and maintenance. We believe that this is an important application domain for the data mining community. There has been significant interest displayed recently in this area, as evidenced by the previous workshop and the recent book on "Machine Learning and Knowledge Discovery for Engineering Systems Health Management" (Chapman and Hall). For this workshop, the goal is to discuss recent progress, to frame and further clarify the relevant research questions, and to bridge the gap between data mining research and industry needs on certain concrete problems. We will also discuss open research problems and challenges in this area, which will help the community to identify future directions.

The workshop will provide a platform for exchange of ideas, identification of important and challenging applications, and discovery of possible synergies. The difference between service and maintenance will be highlighted in multiple industries. It is our hope that this will spur vigorous discussions and encourage collaboration between the various disciplines, potentially resulting in collaborative projects. We will particularly emphasize the mathematical and engineering aspects of service data analysis.

We will address many of these topics through both invited and contributed talks and a combination of position papers (describing research ideas or new challenges) and full papers (describing more mature research and practical results). The workshop program will consist of presentations by invited speakers of both industry and academia, and by the authors of the abstracts and papers submitted to the workshop. In addition, there will be a slot for a panel discussion to identify important problems, applications, and synergies between the two disciplines. Topics of interest include but are not limited to:

Time series and data stream classification for prognostics
Feature extraction from time series, event data, and text
Semi-automated trend analysis
Prognostics modeling
Survival analysis
Regression and ranking from time series
De-noising and handling missing data in service data
Rule based systems for service
Combining multiple data sources for prediction
Classification with imperfect class labels (e.g., noisy service notifications)
Performance measures for preventive maintenance
Knowledge representation for service
Risk-sensitive data mining based decision support for prognostics tasks
Empirical preventive maintenance data sets and comparison
Testing and validation of data mining algorithms for service
Cost-sensitive data mining for service and maintenance
Other service applications

Table of Contents

Invited Talk: Estimating Engine Wear using High Volume Flight Data

James Schimert, The Boeing Company

One goal of a vehicle health management program for aircraft is to monitor engine health, in particular, engine wear. Wear affects engine performance. For example, to get the same thrust output, the engine requires more fuel as the engine wears, and so the engine's exhaust gas temperature (EGT) increases. However, environmental, flight, and other engine parameters also affect EGT. These other factors can affect EGT much more than wear for a given data point.

This talk discusses how data collected during operation over a system's life can be input to statistical learning models to estimate wear (and other changes). A research effort at The Boeing Company has developed data driven approaches to model gas turbine EGT as a function of other recorded parameters. Our investigation involved gas turbines, but these techniques could also be used for other systems that slowly change/degrade over time.

We start by fitting an empirical regression model on a training set collected at a fixed point of wear, and then apply it independently at time points over the life of an engine. The model accounts for the effect of other variables, and predicts what the EGT would have been at the fixed point of wear. The residuals estimate wear. However, wear typically occurs slowly and smoothly. By further relating wear predictions over time using a dynamic linear model, the combined model estimates wear with dramatically reduced variability over a baseline method.

Reduced variability makes engine wear more evident, and allow operators to make maintenance decisions more precisely. We also use the model as the basis for a Bayesian approach to monitor for sudden changes and reject outliers, and adapt the model after these events. This talk will also discuss practical considerations when dealing with high volume data collected during flight.

Biography: James Schimert received his Ph.D. in statistics from University of Washington, Seattle, Washington in 1992. He is an Advanced Computing Technologist in Boeing Research and Technology. His research activities include data mining and machine learning applied to vehicle health management applications for C-17, KC-135, 737, Airplane Health Management, and 787. Previously, he worked in the Research group at Insightful Corp., where he served as consultant and as principal investigator on NIH and NSF SBIR funded research efforts to develop commercial software for advanced statistical computing.

Monitoring Threshold Functions over Distributed Data Streams with Clustering

Jacob Kogan[*] Yaakov Malinovsky[†]

Abstract

Monitoring data streams in a distributed system has attracted considerable interest in recent years. The task of feature selection (e.g., by monitoring the information gain of various features) requires a very high communication overhead when addressed using straightforward centralized algorithms. Motivated by recent contributions based on geometric ideas we present an approach based on convex analysis techniques and clustering. We suggest to cluster together streams to minimize communication burden. Our preliminary results detect instances where communication between nodes is required, and we compare the approach and the results obtained to those recently reported in the literature.

Keyword list: data streams, convex analysis, distributed system, clustering

1 Introduction

In many emerging applications one needs to process a continuous stream of data in real time. Sensor networks [5], network monitoring [1], and real–time analysis of financial data [13], [14] are examples of such applications. Monitoring queries is a particular class of queries in the context of data streams. Previous work in this area deals with monitoring simple aggregates [1], or term frequency occurrence in a set of distributed streams [6]. The current contribution is motivated by results recently reported in [10], [11] where a more general type of monitoring query is described as follows:

Let $\mathbf{S} = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$ be a set of data streams collected at n nodes $\mathbf{N} = \{\mathbf{n}_1, \ldots, \mathbf{n}_n\}$. Let $\mathbf{v}_1(t), \ldots, \mathbf{v}_n(t)$ be d dimensional real time varying vectors derived from the streams. For a function $f : \mathbf{R}^d \to \mathbf{R}$ we would like to confirm the inequality

$$f\left(\frac{\mathbf{v}_1(t) + \ldots + \mathbf{v}_n(t)}{n}\right) > 0 \qquad (1.1)$$

while minimizing communication between the nodes. Often the threshold might be a constant r other than 0. In what follows, for notational convenience, we shall always consider the inequality $f > 0$, and when one is interested in monitoring the inequality $f > r$ we will modify the threshold function and consider $g = f - r$, so that the inequality $g > 0$ yields $f > r$ (see (2.6)).

In the sequel we denote the subset of $\mathbf{R}^d$ where f is positive by $\mathbf{Z}_+(f)$ and state (1.1) as

$$\mathbf{v}(t) = \frac{\mathbf{v}_1(t) + \ldots + \mathbf{v}_n(t)}{n} \in \mathbf{Z}_+(f). \qquad (1.2)$$

As a simple illustration consider the case of three scalar functions $v_1(t)$, $v_2(t)$ and $v_3(t)$, and the identity function f (i.e. $f(x) = x$).We would like to guarantee the inequality

$$v(t) = \frac{v_1(t) + v_2(t) + v_3(t)}{3} > 0$$

while keeping the nodes silent as much as possible. A possible strategy is to verify the initial inequality $v(t_0) = \frac{v_1(t_0) + v_2(t_0) + v_3(t_0)}{3} > 0$ and to keep the nodes silent while

$$|v_i(t) - v_i(t_0)| < \delta = v(t_0), \ t \geq t_0, \ i = 1, 2, 3.$$

The first time t_1 when one of the functions, say $v_1(t)$, crosses the boundary of the local constraint, i.e. $|v_1(t_1) - v_1(t_0)| \geq \delta$ the nodes communicate, the mean $v(t_1)$ is computed, the local constraint δ is updated and made available to the nodes. The nodes are kept silent as long as the inequalities

$$|v_i(t) - v_i(t_1)| < \delta, \ t \geq t_1, \ i = 1, 2, 3$$

hold. This type of monitoring was suggested in [3] for a variety of vector norms.

We note, that if, for example, the local constraint is violated at $\mathbf{n}_1$, i.e. $|v_1(t_1) - v_1(t_0)| \geq \delta$, and

$$v_1(t_1) - v_1(t_0) = -[v_2(t_1) - v_2(t_0)],$$

while $|v_3(t_1) - v_3(t_0)| < \delta$ then $|v(t_0) - v(t_1)| < \delta$, and $f(v(t_1)) > 0$. Separate monitoring of the two node cluster $\{\mathbf{n}_1, \mathbf{n}_2\}$ would require communication involving two nodes only, and could produce communication saving.

The numerical experiments conducted in [3] show that:

[*]Dept. of Math & Stat, UMBC, Baltimore, MD 21250, kogan@umbc.edu

[†]Dept. of Math & Stat, UMBC, Baltimore, MD 21250, yaakovm@umbc.edu

1. The number of time instances the mean violates (1.1) is a small fraction ($< 1\%$) of the number of time instances when the local constraint is violated at the nodes. The lion's share of communications is required because of a single node violation of the local constraint δ.

2. The smallest number of communications is required when one uses l_1 norm.

The main contribution of this work in progress is twofold:

1. We suggest to cluster nodes in order to reduce communication required, apply a specific clustering strategy, and report communication reduction achieved.

2. We apply the same clustering strategy with l_1, l_2, and l_∞ norms and report the results obtained.

The paper is organized as follows. In Section 2 we present a relevant Text Mining application. Section 3 briefly reviews results reported in [10]. The approach developed in [3] is contained in Section 4. Clustering for monitoring data streams is introduced in Section 5. Detailed description of message counting is given in Section 6. Numerical experiments are reported in Section 7. Section 8 concludes the paper.

In the next section we provide a Text Mining related example that leads to a non linear threshold function f.

2 Text Mining application

Let $\mathbf{T}$ be a finite text collection (for example a collection of mail or news items). We denote the size of the set $\mathbf{T}$ by $|\mathbf{T}|$. We will be concerned with two subsets of $\mathbf{T}$:

1. $\mathbf{R}$–the set of "relevant" texts (text not labeled as spam),

2. $\mathbf{F}$–the set of texts that contain a "feature" (word or term for example).

We denote complements of the sets by $\overline{\mathbf{R}}$, $\overline{\mathbf{F}}$ respectively (i.e. $\mathbf{R} \cup \overline{\mathbf{R}} = \mathbf{F} \cup \overline{\mathbf{F}} = \mathbf{T}$), and consider the relative size of the four sets $\mathbf{F} \cap \overline{\mathbf{R}}$, $\mathbf{F} \cap \mathbf{R}$, $\overline{\mathbf{F}} \cap \overline{\mathbf{R}}$, and $\overline{\mathbf{F}} \cap \mathbf{R}$ as follows:

$$\begin{aligned} x_{11}(\mathbf{T}) &= \frac{|\mathbf{F} \cap \overline{\mathbf{R}}|}{|\mathbf{T}|}, \quad x_{12}(\mathbf{T}) = \frac{|\mathbf{F} \cap \mathbf{R}|}{|\mathbf{T}|}, \\ x_{21}(\mathbf{T}) &= \frac{|\overline{\mathbf{F}} \cap \overline{\mathbf{R}}|}{|\mathbf{T}|}, \quad x_{22}(\mathbf{T}) = \frac{|\overline{\mathbf{F}} \cap \mathbf{R}|}{|\mathbf{T}|}. \end{aligned} \tag{2.3}$$

Note that

$$0 \le x_{ij} \le 1, \text{ and } x_{11} + x_{12} + x_{21} + x_{22} = 1.$$

The function f is defined on the simplex (i.e. $x_{ij} \ge 0$, $\sum x_{ij} = 1$), and given by

$$\sum_{i,j} x_{ij} \log \left(\frac{x_{ij}}{(x_{i1} + x_{i2})(x_{1j} + x_{2j})} \right), \tag{2.4}$$

where $\log x = \log_2 x$ throughout the paper. We next relate empirical version of Information Gain (2.4) and the Information Gain (see e.g. [2]).

Let Y and X be random variable with know distributions

$$P(Y = y_i),\ i = 1, \ldots, n, \text{ and } P(X = x_j),\ j = 1, \ldots, m.$$

Conditional Entropy $H(Y|X)$ and Information Gain $IG(Y|X)$ are given by

$$\begin{gathered} H(Y|X) = \sum_{j=1}^{m} P(X = x_j) H(Y|X = x_j) \\ \text{and} \\ IG(Y|X) = H(Y) - H(Y|X), \end{gathered} \tag{2.5}$$

where $H(Y)$ is the entropy of Y, and $H(Y|X = x)$ is the entropy of Y conditional on $X = x$. Information Gain is symmetric and, due to convexity of $g(x) = -\log x$, Information Gain is non negative

$$\begin{aligned} & IG(Y|X) \\ = \ & \sum_{i,j} P(Y = y_i, X = x_j) g \left(\frac{P(Y = y_i)P(X = x_j)}{P(Y = y_i, X = x_j)} \right) \\ \ge \ & g \left(\sum_{i,j} P(Y = y_i, X = x_j) \frac{P(Y = y_i)P(X = x_j)}{P(Y = y_i, X = x_j)} \right) \\ = \ & g \left(\sum_{i,j} P(Y = y_i)P(X = x_j) \right) = -\log 1 = 0. \end{aligned}$$

It is easy to see that (2.4) provides information gain for the "feature."

As an example we consider n agents installed on n different servers, and a stream of texts arriving at the servers. Let $\mathbf{T}_h = \{\mathbf{t}_{h1}, \ldots, \mathbf{t}_{hw}\}$ be the last w texts received at the h^{th} server, with $\mathbf{T} = \bigcup_{h=1}^{n} \mathbf{T}_h$. Note that

$$x_{ij}(\mathbf{T}) = \sum_{h=1}^{n} \frac{|\mathbf{T}_h|}{|\mathbf{T}|} x_{ij}(\mathbf{T}_h),$$

i.e., entries of the global contingency table $\{x_{ij}(\mathbf{T})\}$ are the weighted average of the local contingency tables $\{x_{ij}(\mathbf{T}_h)\}$, $h = 1, \ldots, n$.

To check that the given "feature" is sufficiently informative with respect to the target relevance label r one may want to monitor the inequality

$$(2.6) \quad f\left(x_{11}(\mathbf{T}), x_{12}(\mathbf{T}), x_{21}(\mathbf{T}), x_{22}(\mathbf{T})\right) - r > 0$$

while minimizing communication between the servers.

In the next section we briefly recall a geometric approach to monitor values of $f(\mathbf{v})$ suggested in [10].

3 Monitoring the convex hull

Instead of monitoring the mean $\mathbf{v}(t)$ we would like to monitor the location of a convex set containing $\mathbf{v}(t)$. The later task will be split into n tasks, each one performed independently by one of the n nodes.

Consider the vectors $\mathbf{u}_j(t) = \mathbf{v}(t_i) + [\mathbf{v}_j(t) - \mathbf{v}_j(t_i)]$, $j = 1, \ldots, n$, $t \geq t_i$. We would like to monitor the values of f on the convex hull conv $\{\mathbf{u}_1(t), \ldots, \mathbf{u}_n(t)\}$ instead of the value of f at the average (1.1). This strategy leads to sufficient conditions for (1.1), and may be conservative.

The monitoring techniques for values of f on conv $\{\mathbf{u}_1(t), \ldots, \mathbf{u}_n(t)\}$ without communication between the nodes are based on two observations:

1. *Convexity property.* The mean $\mathbf{v}(t)$ is given by $\dfrac{\mathbf{v}_1(t) + \ldots + \mathbf{v}_n(t)}{n} = \dfrac{\mathbf{u}_1(t) + \ldots + \mathbf{u}_n(t)}{n}$, i.e., the mean $\mathbf{v}(t)$ is in the convex hull of $\{\mathbf{u}_1(t), \ldots, \mathbf{u}_n(t)\}$ and $\mathbf{u}_j(t)$ is available to node j without much communication with other nodes.

2. If $B_2(\mathbf{x}, \mathbf{y})$ is an l_2 ball of radius $\dfrac{1}{2}\|\mathbf{x}-\mathbf{y}\|_2$ centered at $\dfrac{\mathbf{x}+\mathbf{y}}{2}$, then

$$(3.7) \quad \text{conv}\{\mathbf{v}, \mathbf{u}_1, \ldots, \mathbf{u}_n\} \subseteq \bigcup_{j=1}^{m} B_2(\mathbf{v}, \mathbf{u}_j)$$

(see Figure 1). Since each ball

$$(3.8) \quad B_2(\mathbf{v}(t_i), \mathbf{u}_j(t)), \ t \geq t_i, \ j = 1, \ldots, n$$

can be monitored by node j with no communication with other nodes (3.7) allows to split monitoring of conv $\{\mathbf{v}(t_i), \mathbf{u}_1(t), \ldots, \mathbf{u}_n(t)\}$, $t \geq t_i$ into n independent tasks executed by the the n nodes separately and without communication. While the inclusion (3.7) holds when B_2 is substituted by B_p with $p \geq 2$ the inclusion fails when $p < 2$ (see [3], for experimental results obtained with different norms see Section 7).

An alternative strategy that works for many different norms was suggested recently in [3]. The approach is briefly explained in the next section.

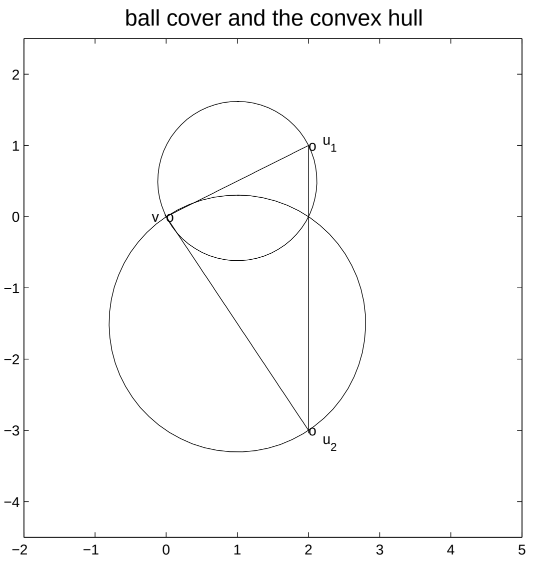


Figure 1: ball cover

4 Monitoring threshold functions through l_p balls

In this section we state the monitoring problem as the following optimization problem [9]:

Problem 4.1. *For a function $K : \mathbf{R}^{d+nd} \to \mathbf{R}$ concave with respect to the first d variables $\lambda_1, \ldots, \lambda_d$ and convex with respect to the last nd variables $x_1, \ldots, x_{nd}$ solve*

$$(4.9) \quad \inf_{\mathbf{x}} \sup_{\boldsymbol{\lambda}} K(\boldsymbol{\lambda}, \mathbf{x}).$$

A solution with appropriately selected $K(\boldsymbol{\lambda}, \mathbf{x})$ is provided in [3] and summarized below for the sake of completeness.

The connection between Problem 4.1, and the monitoring problem is explained next. Let B be a $d \times nd$ matrix made of n blocks, where each block is the $d \times d$ identity matrix multiplied by $\dfrac{1}{n}$, so that for a set of n vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ in $\mathbf{R}^d$ one has

$$(4.10) \quad B\mathbf{w} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{v}_i, \text{ where } \mathbf{w}^T = \left(\mathbf{v}_1^T, \ldots, \mathbf{v}_n^T\right)$$

Assume that $B\mathbf{w} \in \mathbf{Z}_+(f)$, that is $f(B\mathbf{w}) > 0$ (see Figure 2). We are looking for a vector $\mathbf{x}$ "nearest" to $\mathbf{w}$ so that $f(B\mathbf{x}) = 0$, i.e. $B\mathbf{x} = \mathbf{z}$ for some $\mathbf{z} \in \mathbf{Z}(f)$ (where $\mathbf{Z}(f)$ is the zero set of f, i.e. $\mathbf{Z}(f) = \{\mathbf{z} : f(\mathbf{z}) = 0\}$). We now fix $\mathbf{z} \in \mathbf{Z}(f)$ and denote the distance from $\mathbf{w}$ to the set $\{\mathbf{x} : B\mathbf{x} = \mathbf{z}\}$ by $r(\mathbf{z})$. Note that for each $\mathbf{y}$ inside the ball of radius $r(\mathbf{z})$ centered at $\mathbf{w}$ one has $B\mathbf{y} \neq \mathbf{z}$. If $\mathbf{y}$ belongs to a ball of radius $r = \inf_{\mathbf{z} \in \mathbf{Z}(f)} r(\mathbf{z})$

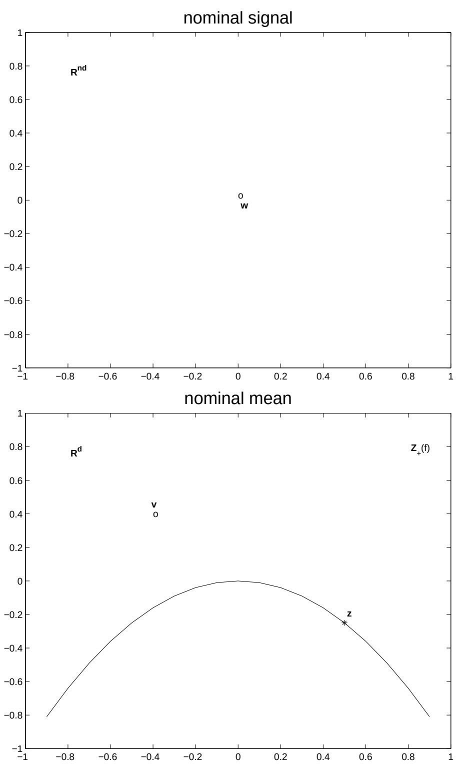


Figure 2: $\mathbf{w} \to B\mathbf{w} = \mathbf{v}$

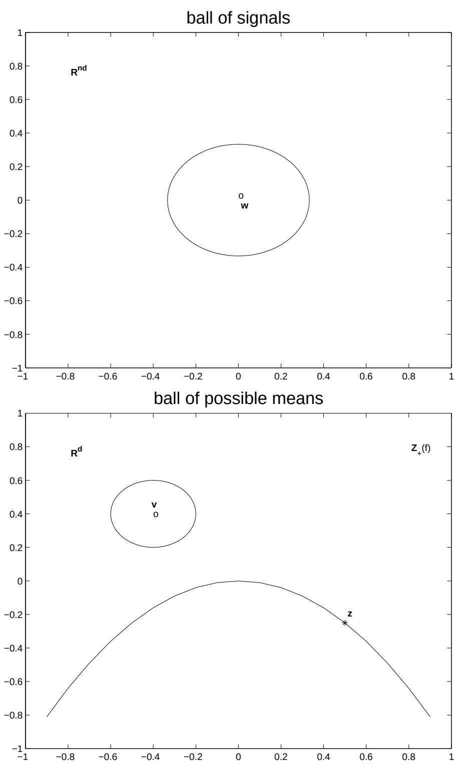


Figure 3: The nearest "bad" vector problem for a fixed vector $\mathbf{z} \in \mathbf{Z}(f)$

centered at $\mathbf{w}$, then the inequality $f(B\mathbf{y}) > 0$ holds true. Let $F(\mathbf{x})$ be a "norm" on $\mathbf{R}^{nd}$ (see Table 1). The nearest "bad" vector problem described above is the following.

PROBLEM 4.2. *For $\mathbf{z} \in \mathbf{Z}(f)$ identify*

$$r(\mathbf{z}) = \inf_{\mathbf{x}} F(\mathbf{x} - \mathbf{w}) \text{ subject to } B\mathbf{x} = \mathbf{z}. \quad (4.11)$$

(See Figure 3). We note that (4.11) is equivalent to $\inf_{\mathbf{x}} \left[\sup_{\boldsymbol{\lambda}} \left\{ F(\mathbf{x} - \mathbf{w}) - \boldsymbol{\lambda}^T (B\mathbf{x} - \mathbf{z}) \right\} \right]$, hence when $K(\boldsymbol{\lambda}, \mathbf{x}) = F(\mathbf{x} - \mathbf{w}) - \boldsymbol{\lambda}^T (B\mathbf{x} - \mathbf{z})$ we deal with Problem 4.1. The expression for $r(\mathbf{z})$ is provided in Table 1 (for solution see e.g. [3]). In the algorithm described below the norm is denoted just by $\|\cdot\|$.

ALGORITHM 4.1. Threshold monitoring algorithm.

```
1. Set $i = 0$.
2. Until end of stream.
3.      Set $\mathbf{v}_j = \mathbf{v}_j(t_i)$, $j = 1, \dots, n$ (i.e. the nodes
        remember "initial" values for the vectors).
4.      Set $\delta = \inf_{\mathbf{z} \in \mathbf{Z}(f)} ||\mathbf{z} - B\mathbf{w}(t_i)||$ (for definition of
        $\mathbf{w}$ see (4.10)).
5.      Set $i = i + 1$.
6.      If $\|\mathbf{v}_j - \mathbf{v}_j(t_i)\| < \delta$ for each $j = 1, \dots, n$
            go to step 5
        else
            go to step 3
```

Algorithm 4.1 implements a sufficient condition that guarantees (1.1). As any sufficient condition it can be conservative. In fact when the distance is provided by the l_2 norm this sufficient condition is more conservative than the one provided by "ball monitoring" (3.8) suggested in [10]. On the other hand only a scalar δ should be communicated to each node, the value of the updated mean $\mathbf{v}(t_i)$ should not be transmitted (hence communication savings are possible), and there is no need to compute the distance from the center of each ball $B_2(\mathbf{v}(t_i), \mathbf{u}_j(t))$, $j = 1, \dots, n$, $t > t_i$ to the zero set $\mathbf{Z}(f)$ (for detailed comparison of results we refer the reader to [3]).

While a detailed message counting procedure will be provided in Section 6, Table 2 presents a result of an experiment reported in [3] for a distributed system with 10 nodes. The first row of the table shows the number of nodes simultaneously violating local constraint, the second one reports the number of time instances over the life time of the system when

$F(\mathbf{x})$	$r(\mathbf{z})$
$\max_i\{\lVert\mathbf{x}_i\rVert_1\}$	$\lvert\lvert\mathbf{z}-B\mathbf{w}\rvert\rvert_1$
$\max_i\{\lVert\mathbf{x}_i\rVert_2\}$	$\lvert\lvert\mathbf{z}-B\mathbf{w}\rvert\rvert_2$
$\lVert\mathbf{x}\rVert_\infty=\max_i\{\lVert\mathbf{x}_i\rVert_\infty\}$	$\lvert\lvert\mathbf{z}-B\mathbf{w}\rvert\rvert_\infty$

Table 1: norm–ball radius correspondence for three different norms and fixed $\mathbf{w}\in\mathbf{R}^{nd}$

1	2	3	4	5	6	7	8	9	10
3034	620	162	70	38	26	34	17	5	0

Table 2: number of local constraint violations simultaneously by k nodes and l_2 norm

the violation of local constraint happened exactly at k nodes. The table reports total of 4006 time instances when the local constraint is violated. In 3034 time instances the violation was caused by exactly one node, in 620 time instances the violation was caused by exactly two nodes, and at no time instance violation was caused by all 10 nodes. Results presented in Table 2 immediately suggest to cluster nodes to further reduce communication required to monitor the mean. We describe the node clustering approach in the next section.

5 Node clustering

A standard clustering problem is often described as "…finding and describing cohesive or homogeneous chunks in data, the clusters" (see e.g. [8]). The monitoring data streams problem requires to assign to the same cluster i nodes $\mathbf{N}_i=\{\mathbf{n}_{i_1},\dots,\mathbf{n}_{i_k}\}$ so that the total change within cluster

$$\left\|\sum_{l=1}^{k}[\mathbf{v}_{i_l}(t)-\mathbf{v}_{i_l}(t_j)]\right\| \quad \text{for } t>t_j$$

is minimized, i.e., nodes with **different** variations $\mathbf{v}_{i_l}(t)-\mathbf{v}_{i_l}(t_j)$ that cancel out each other as much as possible are assigned to the same cluster. Hence, unlike classical clustering procedures, one needs to combine "dissimilar" nodes together.

Since "dissimilarity" of vectors in each cluster should be maximized (and the "similarity" should be minimized) a direct application of center–based clustering algorithms (such as, for example, $k-$means) is not possible [12]. In what follows we briefly outline a new clustering strategy that may lead to partitions with clusters combining "dissimilar" nodes together. The motivation presented next is based on simple convexity arguments.

To simplify the exposition we first consider a two cluster $\{\pi_1,\pi_2\}$ partition problem for a given set of n vectors $\pi=\{\mathbf{v}_1,\dots,\mathbf{v}_n\}\subset\mathbf{R}^d$. We are seeking a partition $\Pi=\{\pi_1,\pi_2\}$ so that

$$\pi_1\bigcup\pi_2=\pi,\ \pi_1\bigcap\pi_2=\emptyset$$

and the partition Π quality $Q(\Pi)$ given by

$$Q(\Pi)=\max\left\{\left\|\frac{1}{|\pi_1|}\sum_{\mathbf{v}\in\pi_1}\mathbf{v}\right\|,\left\|\frac{1}{|\pi_2|}\sum_{\mathbf{v}\in\pi_2}\mathbf{v}\right\|\right\}$$

is minimized. Due to convexity of any norm one has

$$\begin{aligned}\left\|\frac{1}{|\pi|}\sum_{\mathbf{v}\in\pi}\mathbf{v}\right\| &= \left\|\frac{|\pi_1|}{|\pi|}\frac{1}{|\pi_1|}\sum_{\mathbf{v}\in\pi_1}\mathbf{v}+\frac{|\pi_2|}{|\pi|}\frac{1}{|\pi_2|}\sum_{\mathbf{v}\in\pi_2}\mathbf{v}\right\|\\ &\le \frac{|\pi_1|}{|\pi|}\left\|\frac{1}{|\pi_1|}\sum_{\mathbf{v}\in\pi_1}\mathbf{v}\right\|+\frac{|\pi_2|}{|\pi|}\left\|\frac{1}{|\pi_2|}\sum_{\mathbf{v}\in\pi_2}\mathbf{v}\right\|\\ &\le \frac{|\pi_1|}{|\pi|}Q(\Pi)+\frac{|\pi_2|}{|\pi|}Q(\Pi)=Q(\Pi).\end{aligned}$$

This inequality shows that the norm of the mean is a lower bound for $Q(\Pi)$. We next show how to build an optimal partition for a special particular case of the data set.

Assume that $n=2m$, and the vector set π consists of two identical copies of m vectors, i.e.

$$\pi=\{\mathbf{v}_1,\dots,\mathbf{v}_m,\mathbf{v}_1,\dots,\mathbf{v}_m\}.$$

If $\pi_1^o=\pi_2^o=\{\mathbf{v}_1,\dots,\mathbf{v}_m\}$, then $|\pi_1^o|=|\pi_2^o|=\frac{1}{2}|\pi|$, and

$$\max\left\{\frac{1}{|\pi_1^o|}\left\|\sum_{\mathbf{v}\in\pi_1^o}\mathbf{v}\right\|,\frac{1}{|\pi_2^o|}\left\|\sum_{\mathbf{v}\in\pi_2^o}\mathbf{v}\right\|\right\}=\left\|\frac{1}{|\pi|}\sum_{\mathbf{v}\in\pi}\mathbf{v}\right\|,$$

i.e., $\{\pi_1^o,\pi_2^o\}$ is an optimal partition.

This observation motivates the following two cluster $\Pi=\{\pi_1,\pi_2\}$ partition strategy:

1. Apply any clustering algorithm to the dataset π to generate clusters of size 2.
2. Select one vector from each cluster generated and assign selected vectors to cluster π_1.
3. Assign remaining vectors to cluster π_2.

Generalization of this strategy to a k cluster partition is straightforward.

While the full description of the algorithm is beyond the scope of this paper and will be provided elsewhere in Section 7 we apply a different clustering strategy attempting to "balance out" vectors assigned to the same cluster, and by doing so to minimize the norm of the clusters' average. The numerical results obtained with this simple clustering scheme are provided in Section 7. The results demonstrate improvement over those reported in [3], and motivate applications of clustering tools to the monitoring threshold functions. The grouping strategy we use in the paper is described next.

6 Node clustering and message count

At the first stage of the monitoring (steps 1-5) each vector as a singleton cluster. After the first violation of a local constraint a prespecified number of "shortest" and "longest" vectors form cluster π_1. All other vectors form singletons. The suggested monitoring is the following:

Algorithm 6.1. For a user specified integers s and l $(s + l < n)$

```
1. Set $i = 0$.
2. Compute $\mathbf{v}(t_i) = \frac{1}{n}\sum_{j=1}^{n} \mathbf{v}_j(t_i)$.
3. Compute the local constraint $\delta = \inf_{\mathbf{z}\in\mathbf{Z}(f)} \|\mathbf{v}(t_i)-\mathbf{z}\|$.
4. Set $\mathbf{v}_j = \mathbf{v}_j(t_i)$, $j = 1, \ldots, n$.
5. do
       set $i = i + 1$
   while $(\max_j ||\mathbf{v}_j(t_i) - \mathbf{v}_j|| < \delta)$ or (end of stream)
6. if (end of stream) then stop.
7. Select $s$ nodes with smallest norm difference $||\mathbf{v}_j(t_i)-\mathbf{v}_j||$, and $l$ nodes with largest norm difference $||\mathbf{v}_j(t_i)-\mathbf{v}_j||$. This set of $s+l$ nodes forms the first cluster $\pi_1$. Each other node forms a singleton cluster.
8. If at least one singleton cluster violates the local constraint $\delta$, then update the mean, recompute $\delta$, goto 6.
9. If at least one of the nodes in $\pi_1$ violates the local constraint $\delta$, then provide the cluster coordinator with $\frac{1}{|\pi_1|}\left\|\sum_{\mathbf{n}_j\in\pi_1}[\mathbf{v}_j(t_i)-\mathbf{v}_j]\right\|$.
10. If the cluster coordinator violates the local constraint, i.e. $\frac{1}{|\pi_1|}\left\|\sum_{\mathbf{n}_j\in\pi_1}[\mathbf{v}_j(t_i)-\mathbf{v}_j]\right\| \geq \delta$, then update the mean, recompute $\delta$, goto 6.
```

We next turn to message counting procedures. We shall assume that each node is equipped a "response time" slot, i.e., for example, if n nodes are supposed to respond within 1 second, then the first node "response time" slot is the first $\frac{1}{n}$ of the second, the second node "response time" slot is the second $\frac{1}{n}$ of the second, and so on. This arrangements allows to collect responses coming from singletons first, and later deal with reporting coming from cluster π_1. We shall assume that transmission of a double precision real number amounts to broadcasting one message. The message computation presented below is based on the assumption that all nodes are updated by a new text simultaneously, and one node is designated as "root" coordinating the computation.

First consider counting messages that should be broadcasted per one iteration of Algorithm 4.1 (and first 5 steps of Algorithm 6.1). We denote the set of nodes complying with the constraint by $\mathbf{N}^C$, and the set of nodes violating the constraint by $\mathbf{N}^V$ (so that $\mathbf{N} = \mathbf{N}^C \bigcup \mathbf{N}^V$, and $n = |\mathbf{N}^C| + |\mathbf{N}^V|$). Assuming $|\mathbf{N}^V| > 0$ one has the following:

1. $|\mathbf{N}^V|$ nodes violators transmit their scalar ID and new coordinates to the root ($(d + 1) \times |\mathbf{N}^V|$ messages).
2. the root sends scalar requests for new coordinates to the complying $\mathbf{N}^C$ nodes ($|\mathbf{N}^C|$ messages).
3. the $|\mathbf{N}^C|$ complying nodes transmit new coordinates to the root ($d \times |\mathbf{N}^C|$ messages).
4. root updates itself, computes new distance δ to the surface, and sends δ to each node (n messages).

This leads to total of

(6.12) $(d + 2)n$ messages per mean update.

Next we count messages that should be broadcated when newly arriving vectors cause violations of local constraint by nodes in π_1 of size n_1. We assume that one node is designated as cluster coordinator. At each iteration we first check the local constraints at the singletons (i.e. nodes that do not belong to π_1). If one of those constraints is violated, then:

1. the violating node reports to the root its ID and coordinates ($d + 1$ messages).
2. The root broadcasts a request for updated coordinates to all other $n - 1$ nodes ($n - 1$ messages).
3. The $n - 1$ nodes respond to the root with the ID and the coordinates ($(d + 1) \times (n - 1)$ messages).

4. The root recomputes the local constraint δ and sends it to each node (n messages).

5. The root forms a new cluster π_1 and notifies each node whether the node is a singleton, or a cluster element (n messages).

This brings the total to

$$(d+4)n-1 \text{ messages.} \tag{6.13}$$

If no singleton violates the local constraint, then we proceed to check the local constraint violations at the nodes that belong to π_1. If a node $\mathbf{n}_j \in \pi_1$ violates the local constraint, but the coordinator does not, then

1. the node $\mathbf{n}_j$ reports to the coordinator its ID and the change in coordinates $\mathbf{v}_j(t) - \mathbf{v}_j(t_i)$ ($d+1$ messages).

2. The coordinator broadcasts a request for coordinate changes to its other n_1-1 nodes (n_1-1 messages).

3. The n_1-1 nodes respond to the coordinator with the coordinate changes ($(d+1)\times(n_1-1)$ messages).

This brings the total to

$$(d+2)n_1-1 \text{ messages.} \tag{6.14}$$

If as a result of the local constraint violation by a node in π_1 the cluster coordinator also violate the local constraint, then additional messaging is needed as follows:

1. the coordinator reports the violation to the root (1 message).

2. The root broadcasts a request for updated coordinates to all n nodes (n messages).

3. The nodes respond to the root with the ID and the coordinates ($(d+1)\times n$ messages).

4. The root recomputes the local constraint δ and sends it to each node (n messages).

5. The root forms a new cluster π_1 and notifies each node whether the node is a singleton, or a cluster element (n messages).

This brings the total of number of messages required as a result of cluster violation to

$$(d+2)n_1 + (d+4)n \text{ messages.} \tag{6.15}$$

In the next section we apply Algorithm 4.1 and Algorithm 6.1 to a real life data and report number of required mean computations and messages exchanged.

7 Experimental Results

The data streams analyzed in this section are generated from the Reuters Corpus RCV1–V2. The data is available from `http://leon.bottou.org/projects/sgd` and consists of 781, 265 tokenized documents with did (document ID) ranging from 2651 to 810596.

The methodology described below attempts to follow that presented in [10] and [3]. We simulate n streams by arranging the feature vectors in ascending order with respect to did, and selecting feature vectors for the stream in the round robin fashion.

In the Reuters Corpus RCV1–V2 each document is labeled as belonging to one or more categories. We label a vector as "relevant" if it belongs to the "CORPORATE/INDUSTRIAL" ("CCAT") category, and "spam" otherwise. Following [10] we focus on three features: "bosnia," "ipo," and "febru." Each experiment was performed with 10 nodes, where each node holds a sliding window containing the last 6700 documents it received.

First we use 67, 000 documents to generate initial sliding windows. The remaining 714, 265 documents are used to generate datastreams, hence the selected feature information gain is computed 714, 265 times. Based on all the documents contained in the sliding window at each one of the 714, 266 time instances we compute and graph 714, 266 information gain values for the feature "bosnia" (see Figure 4). For the experiments described below the threshold value r is predefined, and the goal is to monitor the inequality $f(\mathbf{v}) - r > 0$ while minimizing communication between the nodes.

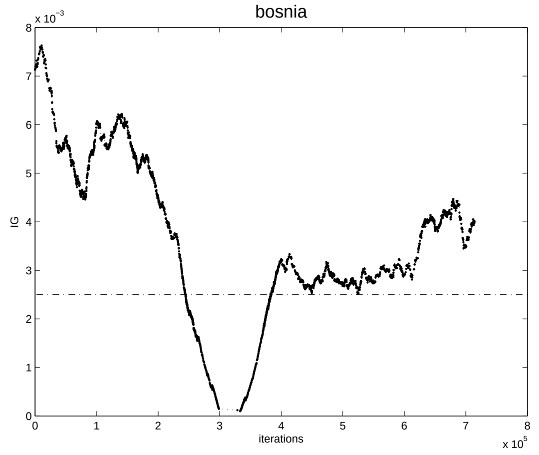


Figure 4: information gain values for the feature "bosnia"

Next we assume that new texts arrive simultaneously at each node. The numerical experiment reported

Norm	Mean Comps	Messages
l_2	4006	240360

Table 3: number of mean computations, and messages computed with l_2 norm for feature "bosnia" with threshold $r = 0.0025$ (simultaneous text arrival, Algorithm 4.1 results)

Norm	s	l	Mean	Coordinator	Messages
l_2	1	1	1820	4740	196212
l_2	2	1	1763	6577	251700
l_2	1	2	1744	6014	240714
l_2	2	2	1703	7317	303790

Table 4: number of mean and cluster coordinator computations, messages required for monitoring involving clustering with l_2 norm for feature "bosnia" with threshold $r = 0.0025$ (simultaneous text arrival, Algorithm 6.1 results)

in [3] with the feature "bosnia," the l_2 norm, and the threshold $r = 0.0025$ (reported in [10] as the threshold for feature "bosnia" incurring the highest communication cost), shows overall 4006 computations of the mean vector. We apply (6.12) to compute the number of messages required and summarize the result in Table 3. Next we apply Algorithm 6.1 with various choices of s and l and report the results in Table 4. We note that the number of mean updates reported in Table 4 is reduced by more than 50% as compared to that reported in Table 3. However as number of mean updates goes down, the number of times the coordinator has to be checked is growing. Coordinator's violations of the local constraint are very expensive. Indeed in case of $s = l = 1$ (first line in Table 4) the coordinator was checked at 4740 instances. In 4575 of these instances the coordinator did not violate the local constraint, and the required number of messages came to $[6 \times 2 - 1] \times 4575 = 50325$ (see (6.14)). In additional 165 instances the coordinator did violate the local constraint, this triggered the mean update and resulted in $[6 \times 2 + 8 \times 10] \times 165 = 15180$ messages (see (6.15)).

When $s = l = 2$, the corresponding numbers are 6817 and 500, and the number of required messages is $[6 \times 4 - 1] \times 6817 = 156791$, and $[6 \times 4 + 8 \times 10] \times 500 = 52000$. We repeat this experiment with l_∞, and l_1 norms. The results previously reported in [3] and presented in Table 5 show that the smallest number of the mean updates is required for the l_1 norm. Application of clustering messages required,

Norm	Mean Comps	Messages
l_∞	3801	228060
l_1	3053	183180

Table 5: number of mean computations, and messages computed for feature "bosnia" with threshold $r = 0.0025$ (simultaneous text arrival, Algorithm 4.1 results)

Norm	s	l	Mean	Coordinator	Messages
l_∞	1	1	1738	5044	193032
l_1	1	1	1420	5586	173818

Table 6: number of mean and cluster coordinator computations, messages required for monitoring involving clustering with l_2 norm for feature "bosnia" with threshold $r = 0.0025$ (simultaneous text arrival, Algorithm 6.1 results)

and results are reported in Table 6. We than repeat these experiments with the other two features, and compare the results with those reported in [3]. Table 7 displays results of the experiments with the feature "ipo" obtained without clustering. Table 8 shows results for the same feature obtained with clustering. Consistent with [3] the results show that l_2 is probably not the most convenient norm to be used if the number of messages required is to be minimized. It appears that computation performed with l_1 norm requires smallest number of messages for selected threshold value. The two last tables reports algorithms' performance for the feature "febru."

8 Conclusion

Monitoring streams over distributed systems is an important and challenging problem with a wide range of applications. In this note we propose application of clustering tools to monitoring threshold functions. A specific monitoring algorithm involving clustering is applied to a real life data, and results reported show improve-

Norm	Mean Comps	Messages
l_2	21109	1266540
l_∞	19598	1175880
l_1	15331	919860

Table 7: number of mean computations, and messages computed for feature "ipo" with threshold $r = 0.0025$ (simultaneous text arrival, Algorithm 4.1 results)

Norm	s	l	Mean	Coordinator	Messages
l_2	1	1	6183	4090	534929
l_∞	1	1	5649	4468	496753
l_1	1	1	4946	4252	438593

Table 8: number of mean and cluster coordinator computations, messages required for monitoring involving clustering for feature "ipo" with threshold $r = 0.0025$ (simultaneous text arrival, Algorithm 6.1 results)

Norm	Mean Comps	Messages
l_2	3140	188400
l_∞	3044	182640
l_1	2591	155460

Table 9: number of mean computations, and messages computed for feature "febru" with threshold $r = 0.0025$ (simultaneous text arrival, Algorithm 4.1 results)

ment over those reported in the literature. Introduction of feedback mechanism that controls local constraints (see e.g. [7]) may further improve the results. For example, while the parameters s and l in Algorithm 6.1 are kept constant an update based on the changes across different time periods may decrease the communications. Research in this direction is already underway.

As was already noted in the literature "There does not exist a best method, that is, one which is superior to all other methods, for solving all other problems in a given class of problems" (see [4] for enlightening discussion and short proofs). Clustering complies with this statement, and is not a universal remedy. It is clear that when, for example, vectors to be clustered are identical (or almost identical) the proposed monitoring with clustering will not be helpful. Identification of distributed systems for which clustering techniques may assist monitoring tasks are future research goals we plan to pursue.

Norm	s	l	Mean	Coordinator	Messages
l_2	1	1	1024	5738	144130
l_∞	1	1	1008	6498	151226
l_1	1	1	918	5400	132014

Table 10: number of mean and cluster coordinator computations, messages required for monitoring involving clustering for feature "febru" with threshold $r = 0.0025$ (simultaneous text arrival, Algorithm 6.1 results)

References

[1] M Dilman and D. Raz. Efficient reactive monitoring. In *INFOCOM '01*, pages 1012–1019. Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communication Societies, 2001.

[2] R.M. Gray. *Entropy and Information Theory.* Springer–Verlag, New York, 1990.

[3] J. Kogan. Feature selection over distributed data streams through convex optimization. In *SDM12*, pages 475–484. Proceedings of the 2012 SIAM International Conference on Data Mining, 2012.

[4] G. Leitmann. One one approach to the control of uncertain systems. *Transactions of ASME, Journal of Dynamic Systems, Measurement and Control*, 50(5):373–380, 1993.

[5] S. Madden and M.J. Franklin. An architecture for queries over streaming sensor data. In *ICDE 02*, page 555, Washington, DC, USA, 2002. IEEE Computer Society.

[6] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *ICDE 05*, pages 767–778, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[7] O. Mayr. *The Origins of Feedback Control.* MIT Press, 1970.

[8] B. Mirkin. *Clustering for Data Mining: A Data Recovery Approach.* Chapman & Hall/CRC, Boca Raton, 2005.

[9] R. T. Rockafellar. *Convex Analysis.* Princeton University Press, Princeton, NJ, 1970.

[10] I. Sharfman, A. Schuster, and D. Keren. A geometric approach to monitoring threshold functions over distributed data streams. *ACM Transactions on Database Systems*, 32(4):23:1–23:29, 2007.

[11] I. Sharfman, A. Schuster, and D. Keren. A Geometric Approach to Monitoring Threshold Functions over Distributed Data Streams. In M. May and L. Saitta, editors, *Ubiquitous Knowledge Discovery*, pages 163–186. Springer–Verlag, 2010.

[12] M. Teboulle. A unified continuous optimization framework for center-based clustering methods. *Journal of Machine Learning Research*, 8:65–102, 2007.

[13] B.-K. Yi, Sidiropoulos N., Johnson T., H.V. Jagadish, C. Faloutsos, and Biliris A. Online datamining for co–evolving time sequences. In *ICDE 00*, page 13, Washington, DC, USA, 2000. IEEE Computer Society.

[14] Y. Zhu and D. Shasha. Statestream: Statistical monitoring of thousands of data streamsin real time. In *VLDB*, pages 358–369, 2002.

Context-Aware Time Series Anomaly Detection for Complex Systems

Manish Gupta[1], Abhishek B. Sharma[2], Haifeng Chen[2], Guofei Jiang[2]
[1]UIUC, [2]NEC Labs, America

Abstract

Systems with several components interacting to accomplish challenging tasks are ubiquitous; examples include large server clusters providing "cloud computing", manufacturing plants, automobiles, etc. Our relentless efforts to improve the capabilities of these systems inevitably increase their complexity as we add more components or introduce more dependencies between existing ones. To tackle this surge in distributed system complexity, system operators collect continuous monitoring data from various sources including hardware and software-based sensors. A large amount of this data is either in the form of time-series or contained in logs, e.g., operators' activity, system event, and error logs, etc.

In this paper, we propose a framework for mining *system operational intelligence* from massive amount of monitoring data that combines the time series data with information extracted from text-based logs. Our work is aimed at systems where logs capture the *context* of a system's operations and the time-series data record the *state* of different components. This category includes a wide variety of systems including IT systems (compute clouds, web services' infrastructure, enterprise computing infrastructure, etc.) and complex physical systems such as manufacturing plants. We instantiate our framework for Hadoop. Our preliminary results using both real and synthetic datasets show that the proposed context-aware approach is more effective for detecting anomalies compared to a time series only approach.

1 Introduction

Complex distributed systems with multiple components interacting to accomplish challenging tasks are drivers of our economic growth. Examples of such systems include large server clusters providing cloud computing services, critical infrastructure such as power plants and power grids, manufacturing plants, automobiles, etc. Our relentless efforts to improve the capabilities of these systems inevitably increase their complexity as we add more components or introduce more dependencies between existing ones to provide new functionality. The complex nature of such systems can lead to failures like the Amazon Outage [1]. Such unanticipated faults and unknown anomalies are a major source of service disruption in complex systems. System operators try to reduce such incidents through better visibility achieved by collecting continuous monitoring data from multiple vantage points. The ultimate goal is to transition from the current practice of reactive maintenance to a more pro-active approach where operators can predict system behavior and address possible service disruptions.

A significant barrier to making this leap from improved visibility to pro-active system management that goes beyond the current simple rules based solutions is the heterogeneity of monitoring data. Monitoring data can be in the form of time series and structured or semi-structured text logs (operators' activity, system events, and error logs), and unstructured text. Mining of log and time series data for fault and anomaly detection is an area of active research. However, previous work analyzes logs and time series data *separately* (Section 8) which has several shortcomings when detecting anomalies in distributed systems.

Anomaly detection using only logs. Fault detection using only logs suffers from two major shortcomings. (1) *Incomplete coverage*: It is challenging to *a priori* determine the right granularity for logging, and often programmers/designers do not want to log several relevant details fearing that the log sizes and the associated time overheads may be large. (2) *Lack of root-cause visibility*: Logs contain information at conceptual/application level and hence often do not contain sufficient information to identify the cause of faults at the component level.

Anomaly detection using only time series. System monitoring using just the time series data suffers from the *lack of a global semantic view*. For example, CPU utilization measurements of a server supply information that is context insensitive. In the absence of context, any significant change in the metrics might seem quite abnormal, even to a human, though some of them might be due to normal system events. Such scenarios can lead to a lot of false positives. Another drawback is the *proliferation of alarms*. A method that just uses time series data can raise multiple alarms (one for a fault in each time series) even when many of these faults are actually related to the same underlying problem. But the lack of a global view prevents identification of any commonality between these faults.

Our hypothesis is that performing data mining *jointly* on logs and time series data can address these shortcomings. We refer to it as *context-aware time series anomaly detection*. The following example provides an illustration.

Importance of Context. Figure 1 shows the variation in

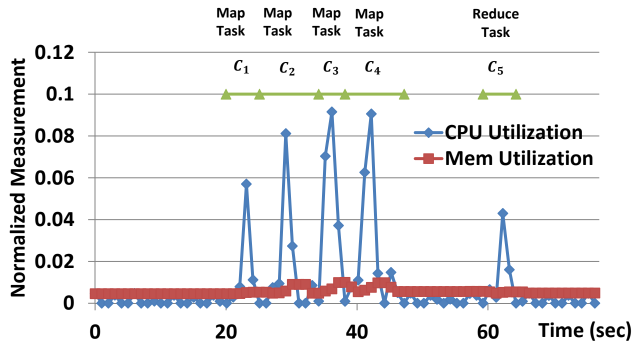


Figure 1: Logs and OS Metrics are Complementary

the normalized operating system (OS) performance metrics such as CPU and memory utilization for a server executing MapReduce [3][1] tasks. The measurements have been normalized for clear display. The contexts $C_1, \ldots, C_5$ (represented by green lines) denote the server's state. We assume that the server's state changes whenever a task starts or stops. Suppose we define an anomaly as unusual values in the time series data for CPU and memory utilization. If we detect anomalies based only on time series data on CPU and memory utilization, we will report one whenever a new task starts or stops. However, when we combine the information on the server's states (extracted from logs and characterized by variables like number of running tasks) along with the time series data, then we can explain the observed CPU and memory utilization behavior and not consider them as anomalous. For example, if we know that a new CPU resource intensive task has started on the server, we expect the CPU utilization to memory utilization ratio to go high, and so such an increase will not be flagged as an anomaly.

Summary of our Contributions

(1) We introduce the notion of context-aware anomaly detection in distributed systems by integrating the information from system logs and time series measurement data. (2) We propose a two-stage clustering methodology to extract context and metric patterns using a PCA-based method and a modified K-Means algorithm (see Figure 2). Anomalies are then detected based on these context and metric patterns. (3) We instantiate our framework for Hadoop, and show that our approach can detect interesting and meaningful anomalies from real Hadoop runs and can accurately discover injected anomalies from synthetic datasets.

The rest of this paper is organized as follows. We provide an overview of our work and highlight the main challenges that it addresses in Section 2. In Section 3, we provide background on Hadoop and define the context-aware time series anomaly detection problem. Section 4 describes the context and metric patterns, and the methodology used to extract them. In Section 5, we present our anomaly detection approach, and discuss various interesting aspects in

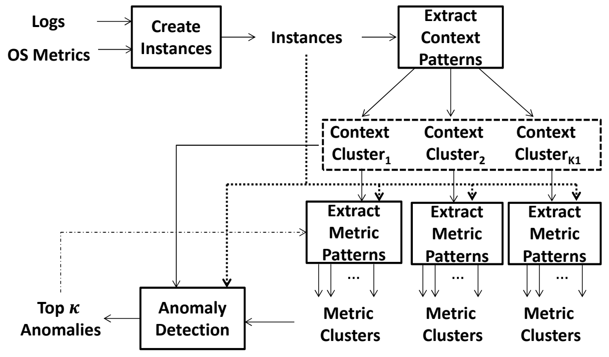


Figure 2: Anomaly Detection Methodology

Section 6. We present experimental results in Section 7 and the related work in Section 8. We conclude with a summary of our work in Section 9.

2 Overview: Challenges and the Proposed Solution

Challenges: There are three main challenges in combining log and time series for context-aware anomaly detection: (1) How do we *represent* the structured or semi-structured log data and the multivariate time series data together? (2) How do we *model* the combined data? (3) What is our *definition of an anomaly* and how do we *detect them*? Figure 2 shows the main steps of our proposed methodology to address these three challenges. We instantiate this framework for Hadoop.

Combining system logs and time series. We assume that system logs capture the current operational context, and define *context variables* and extract their time-varying values from the logs. This process exploits the fact that log data is often structured or semi-structured. We also define system events that signal a context change and assume that they are recorded in system logs. (In general, we may not have enough domain knowledge about a system to define context changing events. However, there exist tools such as Sisyphus [18] and PADS [5] that can help with this task.)

We combine logs and time series data through our definition of an *instance*. An instance spans the interval between two consecutive context changing events on a component. For example, we will define an instance for each of $C_1, \ldots, C_5$ in Figure 1. Each instance consists of context variables and multivariate time series data collected during the interval associated with it. For $C_1, \ldots, C_5$ in Figure 1, the context variables can be the number and type (map or reduce) of running tasks, and the time series data consists of the CPU and memory utilization measurements. Each component in a distributed system can have its own set of instances, i.e., for a large cluster of servers, we can extract instances at the granularity of individual servers.

Iterative clustering for anomaly detection. It is reasonable to assume that a system's components will display similar behavior under similar context. Hence, given the context values for an instance, we define an instance to be anomalous

[1]We briefly describe the MapReduce framework in Section 3

if the time series data associated with it contains unusual or inconsistent values. To detect anomalies as per this definition, we first cluster instances based on their context variables. We refer to these clusters as *context clusters*. Then we partition the instances belonging to a context cluster based on their time series data, and refer to these clusters as *metric patterns*. We formally define the similarity measures for context variables and time series data in Section 4. We want to point out that our approach of identifying similar context across instances is valid for two different scenarios: when similar contexts appear over time on the same component or when different components provide similar functionality. The latter case happens when a functionality is replicated across multiple components, e.g., when popular web sites are hosted across multiple servers in order to support large demand. Hence, we refer to the similarity in context variables across instances as *peer similarity*, and the similarity in time series data for similar context as *temporal similarity*. We assign a score to each instance based on the distance of its multivariate time series from the closest metric pattern, and flag instances with high score as anomalous (we define this *anomaly score* in Section 5).

Instantiating our framework for Hadoop. Our approach is aimed at distributed systems with a large number of components that often have functionality replicated across components to support heavy workload. It also needs data on system events and errors logged for each component, and time series measurements from each component. One widely used system that meets these two criteria, and is also easy to deploy is Hadoop, the de facto industry standard for building large, parallel data processing infrastructure. Hence, we instantiate our framework for Hadoop, i.e., we use Hadoop system logs to demonstrate how to extract context variables and instances from real world data, and also collect time series measurements on resource usage (CPU, memory, disk, and network utilization) from servers used to deploy Hadoop. Another advantage of using Hadoop is that we can inject anomalies while the system is running.

3 Problem Definition for Hadoop

In this section, we briefly introduce Hadoop and define our problem.

3.1 MapReduce and Hadoop The MapReduce programming model facilitates processing of a massive dataset by partitioning it into smaller chunks that are distributed across individual machines in the cluster. Data processing is done in two phases – Map and Reduce. During the Map phase the same (Map) function is applied to all the chunks belonging to a data set in parallel at individual machines. The output of the Map phase (spread across the cluster) is fed into the Reduce phase. The Reduce phase consists of multiple Reduce tasks that are run in parallel, and produces the final output. Thus, a MapReduce job consists of multiple Map and Reduce tasks.

Hadoop is an open-source implementation of the MapReduce framework. In a typical Hadoop cluster, the master and each slave run on separate servers. The master (slave) node runs the *NameNode* (*DataNode*) and *JobTracker* (*TaskTracker*) to manage the distributed file system and job execution, respectively. Each slave node is configured with a fixed number of Map and Reduce slots and each running task takes up one slot.

Hadoop collects several logs at the master as well as slave nodes [9]. The TaskTracker and DataNode processes record task level events, errors and resource usage statistics on each slave node. These task level information are aggregated as job level events and statistics in the JobTracker and NameNode logs at the master. The master node also records control decisions, e.g., which task is scheduled on which slave node, as well as information on any errors or failures of slave nodes. Together, these logs capture the application level semantics, i.e. the context, of an operational Hadoop cluster. From each server in a Hadoop cluster, we can also collect time series data on OS performance metrics such as CPU and memory utilization.

3.2 Context and Metric Variables for MapReduce

Consider a Hadoop cluster with M machines. We parse the Hadoop logs to identify the start and end times of the Map and Reduce tasks run on each machine. This gives us a list of *events* and the time at which they occur for each machine. Based on this information, we define the set of instances for each machine where an instance spans the duration between two consecutive events. A new instance is thus created whenever a Map or Reduce starts or ends on the machine. We discard instances corresponding to time intervals during which no task (Map or Reduce) is running. We compute instances for each machine and then aggregate them into a set represented as $\mathcal{I} = \{I_1, I_2, \ldots, I_L\}$.

Instance Each instance is identified by a machine and its time duration. It consists of two components (1) Context values and (2) Metric time series. Thus, we represent an instance I_r as (C_r, M_r) where C_r represents the vector of context variables and M_r represents the vector of metric variables.

Context Variables Context variables characterize the Hadoop tasks running on a machine corresponding to the time duration of an instance. We capture the number and nature of Hadoop tasks using the #Maps, #Reduces and task counters. Task counters are of 5 main types: (1) Map counters: Map Input/Output Bytes/Records (2) Reduce counters: Reduce Input/Output Records, Reduce Shuffle Bytes (3) Combine counters: Combine Input/Output Records (4) Machine counters: CPU Milliseconds, Physical Memory Bytes, Spilled Records (5) HDFS (Hadoop Distributed File System) counters: HDFS Bytes Read/Written. Note that these counters are all task specific and are available in the job history logs after the job gets finished.

For a particular instance (identified by a machine and a duration), there might be many tasks running concurrently. The context values for that instance are then computed as the average of the values across all these tasks.

Time series Metric Variables For every machine we can record a variety of metrics that can help us understand the performance of the machine. We use system level tools to get the values of the following metrics for each machine every second. (1) CPU Utilization (2) Memory Utilization (3) Disk Read Operations (4) Disk Write Operations (5) eth0 RX/TX Bytes. Other system specific measures like eth0 RX/TX Errors, eth0 RX/TX Drops, Memory Buffers, Mem SwapFree, Used File Descriptors, sda Disk Read/Write Sectors, sda Disk pending IO Operations, Interrupts, OS Context Switch, OS Running Processes, OS Blocked Processes could also be exploited for the purpose. However, we focus on showing the benefits of combining the context variables with a simple set of metric variables to obtain better results compared to using just the metric variables without any context information.

3.3 Context-Aware Time Series Anomaly Detection Problem Given the instances represented in terms of context and metric values, the anomaly detection problem can be defined as follows.

Input: Instances $I_1 = (C_1, M_1), \ldots, I_L = (C_L, M_L)$.

Output: Top κ instances with highest anomaly scores.

We discuss extraction of patterns in Section 4 and define outlier scores and sum up the entire anomaly detection algorithm in Section 5.

4 Pattern Extraction

In this section, we discuss how to extract patterns from the instances and their context supplied by the job history logs as well as from the OS metrics time series. Since the Hadoop logs capture a global semantic view of the system, we propose to first cluster the instances using the context variables to extract context patterns. For each context cluster, we find metric patterns by clustering the corresponding OS metrics time series.

4.1 Extraction of Context Patterns Context patterns are modeled as K_1 Gaussians in the multi-dimensional space defined by the context variables and can be discovered using K-Means. However, we need to pre-process the data before applying K-Means clustering. Some context variables can have large values while some are quite small. This can cause the K-Means clusters to be biased towards the attributes with large values. Hence, we normalize all the attributes such that they have zero mean and unit variance. We also find that some context variables are highly correlated with each other and can artificially bias the clustering. E.g., for our workload, the following features were highly correlated: bytes written, file bytes written and HDFS bytes written. We calculate the correlation between each pair of variables and retain only one variable from a group of highly correlated variables.

A context pattern represents the logical state of a machine (running many Map jobs, moving many HDFS blocks, etc.) for the time duration corresponding to the instance. Such states capture the diversity of MapReduce computation including the type and intensity of workload, number of Map and Reduce tasks in a job, number of memory-heavy versus CPU-heavy tasks, etc.

4.2 Extraction of Metric Patterns Context patterns can cluster the dataset with respect to the logical view of the system. To further characterize the dataset within each context, we cluster the metric variables corresponding to all the instances belonging to the same context cluster. Metric patterns are defined with respect to a particular state (context) of a machine. Essentially, metric patterns denote the usual patterns observed in the OS metrics when the machine is in a logical state described by its context variables. Note that the OS metrics associated with an instance are represented as a multi-variate time series. The number of components of this time series is equal to the number of metrics and the length of the time series corresponds to the time duration of the instance. Thus the problem of clustering metric variables is equivalent to clustering of a multi-variate time series dataset.

Clustering multivariate time series is a challenging task. For the proposed problem, a technique is needed such that it can cluster multi-variate time series that (1) may not be of the same size, i.e., may not be defined over the same length of time period, and (2) may not be defined over the same exact time interval (asynchronous). Clustering of general objects needs a definition of a measure of similarity between a pair of objects. We want a measure for *similarity* between two multivariate time series that stresses both on the similarity of interactions between two (or more) univariate time series from a multivariate time series set as well as on the similarity of interactions (across time) within a univariate time series. To capture such semantics and to avoid the problem of uneven lengths of the multi-variate time series, we first define the similarity between two multivariate time series in the space of their principal components.

Similarity Between Two Multi-Variate Time Series Consider two multi-variate time series corresponding to the metrics part M_a and M_b of the instances I_a and I_b. Let the time series be defined over time intervals of length l_1 and l_2 respectively. Thus the multi-variate time series can be represented using matrices ${M_a}^{l_1 \times c}$ and ${M_b}^{l_2 \times c}$ respectively where c is the number of metric variables. For both the matrices, we can compute the top k principal components (that can capture $\geq 95\%$ variance) to obtain smaller subspaces $P^{c \times k}$ and $Q^{c \times k}$ respectively. Similarity between the two multi-variate time series can be expressed in terms of the similarity between their principal components as follows.

$$sim(M_a, M_b) = \frac{trace(P'QQ'P)}{k} \tag{4.1}$$

Geometrically, this means that the similarity between the two multivariate time series is the average sum of squares of the cosines of the angles between each principal component in P and Q. Thus, we can rewrite Eq. 4.1 as follows.

$$sim(M_a, M_b) = \frac{1}{k}\sum_{i=1}^{k}\sum_{j=1}^{k} cos^2\theta_{ij} \tag{4.2}$$

where θ_{ij} is the angle between the i^{th} and the j^{th} principal components in P and Q respectively.

Further, Eq. 4.2 can be modified using a weighted average, with the amount of variance explained by each principal component as weights, as follows.

$$sim(M_a, M_b) = \frac{\sum_{i=1}^{k}\sum_{j=1}^{k} \lambda_{Pi}\lambda_{Qj} cos^2\theta_{ij}}{\sum_{i=1}^{k} \lambda_{Pi}\lambda_{Qi}} \tag{4.3}$$

where λ_{Pi} is the eigen value corresponding to the i^{th} principal component (eigen vector) in P. Note that the similarity value lies between 0 and 1.

Modified K-Means to obtain Metric Patterns Algorithm 1 shows the modified K-Means algorithm for clustering of multi-variate time series datasets using the PCA-based similarity measure given in Eq. 4.2. The metric patterns (clusters) are first initialized randomly (Step 1). Then, the aggregate dataset representative of a cluster is created by concatenating matrices of all constituent datasets that currently belong to the cluster (Step 5). Next, each time series is reassigned to the cluster corresponding to the most similar aggregate dataset (Step 18) where similarity is defined in the PCA space using Eq. 4.3. The modified K-Means algorithm finally returns the metric patterns for the input instances.

5 Anomaly Detection

Given an instance from a machine, the proposed method first assigns it to the nearest context pattern and then assign it to a metric pattern within that context cluster. Thus, the method first estimates the context in which the machine was executing the set of Maps and Reduces, and then discovers the closest matching OS metrics behavior for that context. For an instance, if the right context cluster can be detected and if the instance is far away from any of the existing metric patterns for that cluster, then that instance can be labeled as an anomaly. Thus, an instance is anomalous if its metrics exhibit an unexpected behavior given its context.

The anomaly score of an instance should therefore depend on (1) how confidently the method can detect its context cluster, and (2) how far away the instance is from its nearest metrics cluster. If the distance between an instance and its

Algorithm 1 Modified K-Means for Metric Pattern Discovery

```
Input: (1) Multi-variate time series $M_1, M_2, \ldots, M_N$ representing the metrics
    part of $N$ instances $I_1, I_2, \ldots, I_N$ respectively belonging to the context
    cluster $CP$, (2) Number of metrics clusters $K_2$.
Output: Metric clusters $MP_1, MP_2, \ldots, MP_{K_2}$.
 1: Randomly assign each dataset $M_1, M_2, \ldots, M_N$ to one of the
    $MP_1, MP_2, \ldots, MP_{K_2}$ clusters.
 2: while $MP_1, MP_2, \ldots, MP_{K_2}$ change do
 3:     for Each cluster $MP_i$ do
 4:         Let $\overline{M_1}, \overline{M_2}, \ldots, \overline{M_{|MP_i|}}$ be the elements of $MP_i$.
 5:         Aggregate dataset $D_i = [\overline{M_1}', \overline{M_2}', \ldots, \overline{M_{|MP_i|}}']'$.
 6:     end for
 7:     $MP_1 \leftarrow \phi, MP_2 \leftarrow \phi, \ldots, MP_{K_2} \leftarrow \phi$.
 8:     for Each multi-variate time series $M_i$ do
 9:         $max \leftarrow 0$.
10:         $nearestCluster \leftarrow 1$.
11:         for Each aggregate dataset $D_j$ do
12:             $currSim \leftarrow sim(M_i, D_j)$.
13:             if $currSim > max$ then
14:                 $max \leftarrow currSim$.
15:                 $nearestCluster \leftarrow j$.
16:             end if
17:         end for
18:         Assign $M_i$ to cluster $MP_{nearestCluster}$.
19:     end for
20: end while
```

nearest context cluster centroid is large, then the appropriate context of that instance cannot be detected. Hence, the normal metric patterns associated with this instance cannot be determined and so it cannot be marked as an anomaly.

Anomaly score of an instance $I = (C, M)$ is thus calculated as follows.

$$score(I) = 1 - sim(M, MP_M) \tag{5.4}$$

where MP_M is the nearest metrics cluster for the instance I. Top κ instances with highest anomaly scores can be marked as anomalies.

Anomaly Post-processing As a post-processing step, the instances which cannot be clustered into any context cluster properly, are removed from the set of anomalies. Specifically, if $dist(C, CP_C) \geq median(CP_C)$, the instance is removed from the set of anomalies where CP_C is the nearest context cluster for the instance I and $median(CP_C)$ is the median of the distances of any instance within cluster CP_C from cluster centroid CP_C.

Online Analysis Big companies like Yahoo! and Microsoft process similar jobs on a daily basis. Our system can build the unsupervised model of context patterns and metric patterns based on a training workload and then apply the model for new test instances. New test instances arriving every day can be cross-checked with the patterns learned using logs of past jobs. For the online case, rather than finding top κ anomalies, we can consider an instance as an anomaly if its distance from the nearest cluster centroid is greater than that of any instance in the training set. If the number of such discovered anomalies increases beyond κ for a batch of test instances, due to drift in the model, the model can be retrained on the recent data.

Our system can classify an instance as normal or anomalous as soon as the task corresponding to the instance finishes on the machine. Even for MapReduce jobs running for several hours, the individual map and reduce tasks are quite short (on the order of tens of seconds to a few minutes) and so the analysis is timely enough to be useful [24].

Iterative Computation The set of instances in the input dataset contains anomalies which can affect the metrics pattern discovery. Thus, once the anomalies have been discovered, the clustering of metrics time series can be performed again, ignoring the anomaly instances. This motivates an iterative approach where clustering of metrics time series and anomaly detection are performed within each iteration. The iterations terminate when the set of top κ anomalies detected in the last two consecutive iterations remains the same.

Summary We summarize our approach in Algorithm 2. The algorithm consists of two stages: extracting patterns and performing anomaly detection based on these patterns. Pattern extraction in turn consists of two phases: extraction of context patterns (Steps 1 and 2) followed by extraction of metric patterns (Steps 5 to 7). On termination, the algorithm returns the anomaly score for each instance.

Algorithm 2 Context-Aware Time Series Anomaly Detection

```
Input: Instances $I_1 = (C_1, M_1), I_2 = (C_2, M_2), \dots, I_L = (C_L, M_L)$.
Output: Anomaly scores for each instance, $score$.
 1: Normalize the attributes in $C_1, C_2, \dots, C_L$.
 2: $\{CP_1, CP_2, \dots, CP_{K_1}\} \leftarrow K-Means(C_1, C_2, \dots, C_L)$.
 3: Anomaly Set $A \leftarrow \phi$
 4: while $A$ does not change do
 5:     for $i = 1$ to $K_1$ do
 6:         Compute metric patterns for instances $\in (CP_i - A)$ using Algorithm 1.
 7:     end for
 8:     for $i = 1$ to $L$ do
 9:         Compute anomaly score for instance $I_i$ (Eq. 5.4).
10:     end for
11:     $A \leftarrow$ Set of instances with top $\kappa$ anomaly scores.
12: end while
```

6 Discussions

In this section, we will discuss various aspects of our algorithm and the general framework.

Time Complexity Let L be #instances, i_1 be #K-Means iterations, i_2 be #iterations for Algorithm 1, i_3 be #iterations of while loop from Step 4 to 12, K_1 be #context clusters, K_2 be #metric clusters, D_1 be #context variables, and k be #principal components used for PCA-based similarity computation. Then the overall time complexity of Algorithm 2 is $O(Li_3(i_1K_1D_1 + i_2K_2k^3))$ which is linear in the number of instances L. We omit details for lack of space.

Selecting Number of Clusters (K_1 and K_2) K-Means needs the number of clusters as an input. We use K_1 and K_2 as the input for clustering of context patterns and metric patterns. We start with #clusters as 1 and keep increasing them by 1. For each iteration, we compute the within cluster sum of squares of distances. We pick K_1 as the value where the percentage change in the within cluster sum of squares is small (knee of the curve).

Richer Context Besides the #Maps, #Reduces, and the task counters, Hadoop configuration settings like input format, output format, types of compression, file descriptor limits, can also be used as context variables. OS logs such as syslogs on Linux or Event Tracing for Windows [4] can also provide context information. We plan to study these as part of our future work.

7 Experiments

Evaluation of anomaly detection algorithms is quite difficult due to lack of ground truth. We generate multiple synthetic datasets by injecting anomalies, and evaluate anomaly detection accuracy of the proposed algorithms on the generated data. We also conduct case studies by applying the method to real data sets.

7.1 Baselines

We compare the proposed algorithm *Context-Aware (CA)* with two baseline methods: *Single Iteration (SI)* and *No Context (NC)*.

Single Iteration (SI) As described in Algorithm 2, *CA* performs metric pattern discovery and anomaly detection iteratively until the set of top κ anomalies do not change. *SI* is a simpler version of *CA*, which performs only one iteration. Thus the pattern discovery phase in *SI* suffers from the presence of anomalies. This baseline will help us evaluate the importance of ignoring the anomalies when computing the metric patterns.

No Context (NC) *CA* performs pattern discovery in a context-sensitive manner. A particular metric pattern may be very popular in a particular context but may never be exhibited in another context. Thus, performing context-aware anomaly detection should be better than a context insensitive approach. We test this claim by comparing *CA* with *NC*. Note that this baseline is similar to [14] which performs fault detection by using K-Means based clustering in the metrics space only.

7.2 Synthetic Datasets

We generate our synthetic datasets as follows. The context part of our dataset comes from real Hadoop runs, while the metrics part is generated synthetically. Our Hadoop cluster consists of 6 nodes – 1 master and 5 slaves. We run the standard Hadoop examples like Pi Estimator, Word Count, Sorter, Grep-Search (tasks run in parallel) and gather N instances. For each of the N instances, we obtain the context values (#Maps, #Reduces, task counters) from the Hadoop logs. We cluster the context part of these instances into 3 clusters. Fig. 3 shows the context clusters with respect to the different context variables. The values of context variables have been normalized as discussed in Section 4. Note that Cluster 1 contains instances with large number of Map tasks and hence display high values for Map counters. Cluster 3 contains instances with large number of Reduce tasks and hence display high values for Reduce counters. Cluster 2 contains instances with a few Map and a few Reduce tasks.

For each context cluster, we synthetically generate the metrics time series as follows. We first fix a number of metric clusters and randomly assign the instances to one of the metric clusters. We also fix the number of metrics. Next, we randomly generate a template matrix for each of the metric clusters. This template matrix serves as the basic multi-variate time series for that metric cluster. For a metric cluster, we generate the multi-variate time series for each instance by adding 10% uniform noise to the each of the entries of the template matrix for the cluster. We randomly select the duration of the time series between 20 and 50. Thus, we have for each instance, context values obtained from real logs and metric time series generated synthetically by adding random perturbations to a template matrix for the metric cluster of the instance.

Anomalies are injected in the metric time series as follows. First we set an anomaly factor Ψ and choose a random set of instances, R with $\frac{N \times \Psi}{2}$ instances. For each instance, we swap the time series with that of another randomly chosen instance. This corresponds to the situation of observing metric values which usually do not appear with the given context but may appear in some other context. For some of the instances rather than adding swap-anomalies, we replace the metric time series with random new matrices. This corresponds to the situation of observing metric values which usually never appear in any of the contexts.

Results on Synthetic Datasets We generate a variety of synthetic datasets capturing different experimental settings. For each setting, we perform 20 experiments and report the average values. We vary the number of instances as 500, 1000, 2000 and 5000. We also study the accuracy with respect to variation in the number of metrics (5, 10, 20) and variation in the number of metric clusters (4, 6, 10). We also vary the percentage of injected anomalies as 2%, 5% and 10%. We terminate the external while loop after a maximum of 5 iterations ($i_3 = 5$). For each algorithm, we show the accuracy with respect to matches in the set of detected anomalies and the set of injected anomalies, in Tables 1 and 2 (best accuracy in bold). Each of the accuracy values is obtained by averaging the accuracy for that experimental setting across 20 runs. Average standard deviations are 3.34% for *CA*, 7.06 % for *SI* and 4.58% for *NC*. As the table shows, the two versions of the proposed algorithm outperform the baseline algorithm (*NC*) for all of the settings by a wide margin. On an average across all experimental settings, *CA* is 28% better than *NC*. For small N, we observe that *CA* performs significantly better than *SI*.

Running Time The experiments were run on a Linux machine with 4 Intel Xeon CPUs with 2.67GHz each. The code was implemented in Java. Fig. 4 shows the average execution time spent in the metric pattern discovery stage of the *CA* algorithm for different number of instances and metrics. Note that the algorithm is linear in the number of instances. As number of machines in the cluster increase, number of instances increases proportionately. Thus, our algorithm scales well with number of machines. These times are averaged across multiple runs of the algorithm across different settings for number of metric clusters and across multiple runs for each setting. We also observed that the time spent in anomaly detection (~188ms on an average) is a small fraction of the time spent in clustering metric time series.

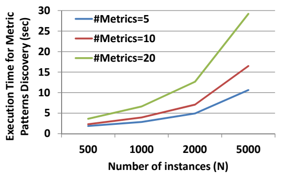


Figure 4: Average Running Time (sec) for Metrics Pattern Discovery Stage of *CA*

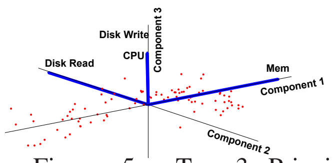


Figure 5: Top 3 Principal Component Representation for Metric Pattern 1 for Context Cluster 3

N	Ψ	$\#Metrics = 5$			$\#Metrics = 10$			$\#Metrics = 20$		
	(%)	CA	SI	NC	CA	SI	NC	CA	SI	NC
500	2	**0.8**	0.55	0.28	**0.741**	0.642	0.281	**0.86**	0.59	0.255
	5	**0.641**	0.641	0.302	**0.79**	0.627	0.332	**0.777**	0.69	0.345
	10	**0.661**	0.617	0.351	**0.742**	0.666	0.35	0.605	**0.633**	0.345
1000	2	**0.716**	0.603	0.255	**0.695**	0.683	0.314	**0.631**	0.608	0.272
	5	**0.706**	0.636	0.368	0.651	**0.654**	0.311	**0.691**	0.641	0.315
	10	0.655	**0.66**	0.359	0.637	**0.679**	0.345	0.659	**0.665**	0.322
2000	2	**0.64**	0.638	0.258	0.603	**0.643**	0.27	0.537	**0.583**	0.278
	5	0.578	**0.655**	0.289	**0.623**	0.623	0.319	0.586	**0.632**	0.321
	10	0.574	**0.6**	0.331	0.659	**0.715**	0.299	**0.626**	0.611	0.311
5000	2	0.664	**0.71**	0.361	0.666	**0.7**	0.362	0.636	**0.661**	0.362
	5	0.604	**0.62**	0.343	0.594	**0.626**	0.423	0.628	**0.669**	0.36
	10	0.661	**0.67**	0.414	0.654	**0.7**	0.41	**0.585**	0.555	0.418

Table 1: Synthetic Dataset Results (*CA*=The Proposed Algorithm, *SI*= Single Iteration, *NC*=No Context) for K_2=4

N	Ψ	$\#Metrics = 5$			$\#Metrics = 10$			$\#Metrics = 20$		
	(%)	CA	SI	NC	CA	SI	NC	CA	SI	NC
500	2	0.462	**0.482**	0.181	**0.71**	0.5	0.265	**0.682**	0.497	0.241
	5	**0.514**	0.505	0.279	**0.488**	0.465	0.3	**0.531**	0.461	0.298
	10	0.466	**0.485**	0.314	**0.527**	0.503	0.334	0.536	**0.549**	0.371
1000	2	0.406	**0.459**	0.223	**0.501**	0.493	0.276	**0.548**	0.548	0.225
	5	0.491	**0.543**	0.264	0.452	**0.516**	0.286	0.53	**0.541**	0.292
	10	0.509	**0.537**	0.327	0.535	**0.556**	0.331	0.476	**0.505**	0.316
2000	2	0.532	**0.564**	0.228	0.511	**0.565**	0.226	**0.576**	0.538	0.26
	5	**0.582**	0.579	0.296	0.503	**0.539**	0.286	0.453	**0.5**	0.237
	10	0.533	**0.55**	0.319	**0.508**	0.495	0.289	**0.499**	0.486	0.288
5000	2	0.506	**0.546**	0.26	0.531	**0.552**	0.352	0.532	**0.545**	0.309
	5	0.544	**0.585**	0.356	**0.543**	0.529	0.362	0.501	**0.522**	0.343
	10	0.557	**0.587**	0.41	0.496	**0.553**	0.398	0.517	**0.534**	0.374

Table 2: Synthetic Dataset Results (*CA*=The Proposed Algorithm, *SI*= Single Iteration, *NC*=No Context) for K_2=10

7.3 Real Datasets

We test the effectiveness of our approach for two different real dataset scenarios. These scenarios capture the usual faults that occur in Hadoop instances: CPU Hog and Disk Hog. Such faults have been reported on Hadoop users' mailing list or on the Hadoop bug database. See [20] for details.

Dataset Generation The workload comprises of the two standard Hadoop examples – RandomWriter and Sort. We configure the RandomWriter to write 1GB of random data to HDFS. We run RandomWriter with 16 Maps and each Map generates 64 MB of data. We run sort with 16 Maps and 16 Reduces. Logs and metrics are copied to a single machine

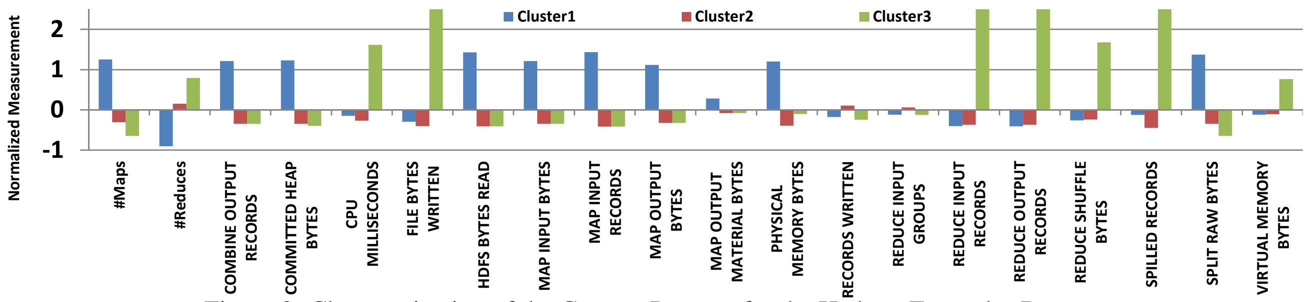


Figure 3: Characterization of the Context Patterns for the Hadoop Examples Dataset

where the processing is performed. Hadoop history logs and metric files are very small and hence it takes negligible time to process/copy them. Also, for our experiments, the cluster had only Hadoop processes running on the machines. However, in general for clusters with multiple applications running on the machines, one needs to extract metrics corresponding to the Hadoop processes only to avoid interference from other applications.

For lack of space, we do not show the characterization of the patterns obtained after clustering in the context space. But we present a few observations. Cluster 1 consists of a mix of Maps and Reduces, Cluster 2 has large number of Maps while Cluster 3 has large number of Reduces. Cluster 1 has a distinctly high number of HDFS bytes being written. Cluster 2 shows a large number of Map Output Records. Cluster 3 demonstrates a large activity in Reduce counters. Further, Fig. 5 shows the various metric variables in the space of the top three principal components for the first metric pattern in the third context cluster. Each metric variable is represented by a vector; direction and length of the vector indicates how each variable contributes to the three principal components in the plot. E.g., Memory and Disk read operations have strong influences on the first and second principal components respectively. The red dots are the instances belonging to the metric pattern as plotted in the top 3 principal component space.

Results on Real Datasets We run the workload 10 times for each of the CPU hog and the Disk hog experiments. For the first run in both the experiments, we inject an anomaly (CPU hog or disk hog) on one of the machines. CPU hog is simulated by running a infinite loop program while disk hog is simulated by writing multiple files to the disk in parallel. Overall, we have 134 and 121 instances for the disk hog and CPU hog datasets. There are 7 and 4 anomalous instances from the machine where the hog was injected for the disk hog and the CPU hog datasets respectively.

Fig. 6 shows a plot of the anomaly scores of various instances. The X axis shows the instance number and the Y axis shows the anomaly score. The steep decrease in the anomaly score both for the CPU Hog and the Disk Hog case clearly shows that our algorithm computes a discriminative score for the anomalies while giving distinctly lower anomaly scores to the normal instances.

As expected we observed that the instances on the machine where the hog is injected has an abnormal metrics behavior. E.g., in case of CPU hog, the CPU utilization of the machine is much higher than that expected for the context. Similarly, in case of disk hog, the number of disk write operations increases by a large amount. Out of the 7 anomalies for the disk hog, 4 of them are present in the top 5 and all 7 get detected in the top 10. In the case of CPU hog, 3 get detected in the top 5 and all 4 in the top 10. Thus, our methodology is quite effective in detecting anomalies, specifically in localizing the anomalies to the machine level and also with respect to the time duration.

For the CPU hog, we did some analysis of the false positives that were reported. One of the false positives appeared to have some reasonable amount of CPU utilization (about 37% which was higher compared to the other members of its metric cluster (average=5.08%, std dev=1.37%)). We believe that this was because of some other process running on the machine at that time. Another false positive showed high number of disk reads (71 compared to an average 28) and low disk write operations (13 compared to an average of 17) for the same metrics cluster.

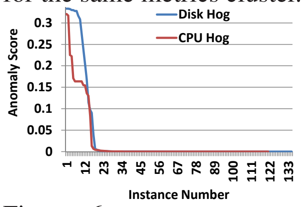


Figure 6: Anomaly Scores for CPU Hog and Disk Hog Datasets (Both the series have been sorted with respect to anomaly scores)

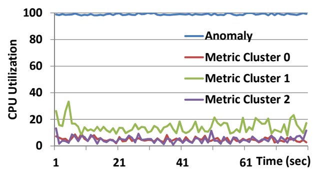


Figure 7: CPU Utilization of an Anomaly Versus Average CPU Utilization of Normal Instances for Metric Patterns within the same Context

We show one of the anomalies for the CPU Hog case in Fig. 7 for a period of 80 seconds. Note that the anomalous instance has a very high CPU Utilization. The other curves show the average CPU Utilization of the instances belonging to the metric patterns within the same context. The graph shows that in the particular context, the expected CPU utilization is quite low and shows a fluctuating trend, while the anomaly shows a very high CPU utilization which is almost flat. Hence, the proposed algorithm marks this instance as an anomaly.

8 Related Work

Our work is related to previous work on anomaly detection for distributed systems using logs only and using performance metrics only.

Anomaly/Fault Detection using System Logs Recently, there has been work that uses system logs for anomaly detection ([11, 12, 16, 17]). Researchers have studied logs using Hidden Markov Models [23], state machines and control and data flow models (SALSA) [19] for fault identification. Besides system logs, message logs across components have been exploited in fault detection systems like Pip [15] and Pinpoint [2]. While log analysis-based systems exploit the event and context information from logs, they fail to exploit the rich information contained in the operating system performance metrics.

Anomaly/Fault Detection using OS Performance Metrics Tiresias performs anomaly detection by exploring individual performance metrics by putting thresholds on each metric separately [22]. Clustering on OS metrics has been used for fault detection (Ganesha [14]) and prediction ([6, 7], ALERT [21]). Time series models and probabilistic correlations among monitored metrics have also been exploited for the task [8, 10]. Recent works [13, 20] discuss about combining Hadoop logs and OS metrics for better fault detection. But our work is different from them in several ways: 1. Though they point out that OS metrics and Hadoop logs are complementary, they do not provide any principled methodology to tightly integrate the two pieces. 2. They provide a supervised approach while our approach is completely unsupervised and requires no prior labeling. 3. Their study is focused on jobs that take exceptionally longer, while we focus on out-of-context metric anomalies. Our method helps in finding time durations on any machine for which the metrics were inconsistent with respect to the context in which they were observed.

9 Conclusions

We motivated the need of combining system log information and OS performance metric observations for effective anomaly detection for MapReduce systems. The anomalies were detected based on the context patterns and metric patterns. The context patterns were derived by clustering in the space of context variables associated with the instances. Metric patterns were discovered for every context cluster using a PCA-based similarity measure for multi-variate time series and a modified K-Means algorithm. Metric pattern discovery and anomaly detection was performed iteratively to mutually improve the quality of results. Using synthetic datasets and real Hadoop runs, we showed the effectiveness of the proposed approach in finding interesting anomalies. The approach could be extended to study anomalies related to *change* in OS performance metrics with *transitions* in the context captured by the system logs.

References

[1] Amazon Outage. http://aws.amazon.com/message/65648/.

[2] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. A. Brewer. Pinpoint: Problem Determination in Large, Dynamic Internet Services. In *ICDSN*, pages 595–604, 2002.

[3] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *CACM*, 51(1):107–113, Jan 2008.

[4] Improving Debugging And Performance Tuning with ETW. http://msdn.microsoft.com/en-us/magazine/cc163437.aspx.

[5] K. Fisher, D. Walker, K. Q. Zhu, and P. White. From Dirt to Shovels: Fully Automatic Tool Generation from Ad hoc Data. In *POPL*, pages 421–434, 2008.

[6] X. Gu, S. Papadimitriou, P. S. Yu, and S.-P. Chang. Toward Predictive Failure Management for Distributed Stream Processing Systems. In *ICDCS*, pages 825–832, 2008.

[7] X. Gu and H. Wang. Online Anomaly Prediction for Robust Cluster Systems. In *ICDE*, pages 1000–1011, 2009.

[8] Z. Guo, G. Jiang, H. Chen, and K. Yoshihira. Tracking Probabilistic Correlation of Monitoring Data for Fault Detection in Complex Systems. In *ICDSN*, pages 259–268, 2006.

[9] Hadoop Logs. http://tinyurl.com/hadoop-logs.

[10] G. Jiang, H. Chen, and K. Yoshihira. Modeling and Tracking of Transaction Flow Dynamics for Fault Detection in Complex Systems. *TDSC*, 3:312–326, Oct 2006.

[11] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo. Failure Prediction in IBM BlueGene/L Event Logs. In *ICDM*, pages 583–588, 2007.

[12] J. F. Murray, G. F. Hughes, and K. Kreutz-Delgado. Machine Learning Methods for Predicting Failures in Hard Drives: A Multiple-Instance Application. *JMLR*, 6:783–816, Dec 2005.

[13] X. Pan, J. Tan, S. Kalvulya, R. Gandhi, and P. Narasimhan. Blind Men and the Elephant: Piecing Together Hadoop for Diagnosis. In *ISSRE*, 2009.

[14] X. Pan, J. Tan, S. Kavulya, R. Gandhi, and P. Narasimhan. Ganesha: BlackBox Diagnosis of MapReduce Systems. *SIGMETRICS Perform. Eval. Rev.*, 37(3):8–13, Jan 2010.

[15] P. Reynolds, C. Killian, J. L. Wiener, J. C. Mogul, M. A. Shah, and A. Vahdat. Pip: Detecting the Unexpected in Distributed Systems. In *NSDI*, pages 9–22, 2006.

[16] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical Event Prediction for Proactive Management in Large-scale Computer Clusters. In *KDD*, pages 426–435, 2003.

[17] B. Schroeder and G. A. Gibson. Disk Failures in Real World: What does MTTF of 1,000,000 Hours mean to you? In *FAST*, 2007.

[18] Sisyphus – A Log Data Mining Toolkit. http://www.cs.sandia.gov/~jrstear/sisyphus/.

[19] J. Tan, X. Pan, S. Kavulya, R. Gandhi, and P. Narasimhan. SALSA: Analyzing Logs as State Machines. In *WASL*, pages 6–13, 2008.

[20] J. Tan, X. Pan, E. Marinelli, S. Kavulya, R. Gandhi, and P. Narasimhan. Kahuna: Problem Diagnosis for Mapreduce-based Cloud Computing Environments. In *NOMS*, pages 112–119, 2010.

[21] Y. Tan, X. Gu, and H. Wang. Adaptive System Anomaly Prediction for Large-Scale Hosting Infrastructures. In *PODC*, pages 173–182, 2010.

[22] A. W. Williams, S. M. Pertet, and P. Narasimhan. Tiresias: Black-box Failure Prediction in Distributed Systems. *IPDPS*, pages 1–8, 2007.

[23] K. Yamanishi and Y. Maruyama. Dynamic Syslog Mining for Network Failure Monitoring. In *KDD*, pages 499–508, 2005.

[24] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In *EuroSys*, pages 265–278, 2010.

Early fault detection and diagnosis for improving process safety: application to a FCC refinery process

Carlos Agudelo
Instituto Colombiano del Petroleo – Ecopetrol S.A.
Bucaramanga, Colombia
carlos.agudelo@ecopetrol.com.co

Eduardo Quiles Cucarella
Universidad Politecnica de Valencia
Valencia, 46022, Spain
equiles@isa.upv.es

Francisco Morant Anglada
Universidad Politecnica de Valencia
Valencia, 46022, Spain
fmorant@isa.upv.es

Emilio García Moreno
Universidad Politecnica de Valencia
Valencia, 46022, Spain
emilio@isa.upv.es

***Abstract*— In this article it is demonstrated the integration of techniques for early fault detection and diagnosis in industrial processes, for the purpose of making recommendations to operations personnel and prevent safety process events. Early fault detection and diagnosis can assist operations staff in an industrial plant to take the best actions during the actual state of the process, preventing incipient faults to climb critical situations where there is risk of loss of life, damage environmental and economic losses. It´s shown a prototype in a Fluidized Catalytic Cracking unit.**

Keywords – Early fault detection, diagnosis, knowledge engineering, alarm systems, Fluid Catalytic Cracking.

I. Introduction

Process safety is about preventing catastrophic events of very low probability of occurrence but a very high impact on organizations (Hopkins, 2007). Names such as Chernobyl, Bhopal, or Piper Alpha are well known in industrial environments, for the tragedy which meant not only for the workers involved, but to the surrounding community, and ecological damage (Norman, 1986; Reason, 1987; Salge and Milling, 2006; Belke and Dietrich, 2001; Shrivastava, 1987; Drysdale and Sylvester-Evans, 1998; Paté-Cornell, 1993).

These names remind us that safety should always accompany technological advances. These accidents (with the loss of human life, environmental damage, economic losses, ethical and moral considerations) are incentives to focus our work on process safety and prevent accidents like these occur in the future.

Through the use of advanced software tools can assist the operations personnel of an industrial plant to prevent incipient faults to climb into more serious conditions. This article demonstrates the use of techniques for early fault detection and diagnosis in industrial processes, and how these techniques can generate recommendations to operations personnel.

II. SAFETY PROCESS AND AUTOMATIC PROCESS CONTROL

In (Bakolas and Saleh, 2011) is shown how it is necessary to increase the observability to prevent process safety events. A system is said to be observable if one can determine the behavior of the entire system from the measurement of its outputs, otherwise the current values of some of its states can not be determined from the output sensors, implying that its value remains unknown to the control system, and you can not meet the specifications of these control outputs.

In (Saleh and Cummings, 2011) it´s shown how to maintain effective control of hazards and establish a set of defenses to block accidents, it is important sensing the signs that a situation is growing into a dangerous situation.

The control system helps prevent critical situations through the override control strategies (Smith and Corripio, 1997), but you need a system that tells the operators that complex faults are beginning to occur in the process increasing the observability of the process, here is where fault detection and diagnosis appears on the stage.

III. THE ALARM SYSTEM IN THE PROCESS INDUSTRY

There are international standards and guides (ISA 18.2; EEMUA 191, etc.) to address the problem of flooding alarms (activation of too many alarms in the electronic control system). An alarm is a visual and audible indication that an abnormal situation is present in the process and operations personnel must implement immediate action, and whose implications are well known to all personnel involved. The alarms are linked to acceptable limits for process variables, set in the electronic control system to indicate low limit violations and high limit. There are alarms on the emergency shutdown system to alert operations personnel about an imminent emergency stop.

Alarms should address the attention of the operator to properly assess the actions that respond to the current process conditions, avoiding redundant information, reducing risk in people, environment and equipment (Acero et al, 2005a). Unfortunately it is common to confuse the operator with alarms instead of helping him, flooding the system with too many events, not only during emergencies but also during normal operation. Therefore it is mandatory to optimize the information on the process alarms, prioritized using a methodology for analyzing hazardous operations (Acero et al, 2005b; Acero et al, 2005c). To do this it´s been applied international standards (EEMUA Publication No. 191 "A guide to design, management and procurement of Alarms Systems") define criteria for the optimization of the alarms, and criteria for performance measurement and benchmarks during normal operation and for fault scenarios.

It´s been applied this methodology to several process units at refineries of Barrancabermeja and Cartagena (Colombia). This methodology has been divided in phases: Phase 1 has to do with the "bad actors" of alarms, alarms that are activated most of the time in the process. These "bad actors" are detected using statistical tools on historical reports of alarms in the electronic control system. With an interdisciplinary team (consisting of the electronics engineer responsible of the control system, the chemical engineer responsible for the process, expert operator and/or supervisor of the process, and facilitator of alarms), discusses why these alarms are activated. Sometimes it is due to problems of control system configuration (such as wrong adjustment of parameters), it is sometimes due to tuning control loops problems, it is sometimes for limit issues. Phase 2 is rationalizing alarms, performing the analysis to all alarms in the process, eliminating redundant alarms, and alarms that are not really alarms but maintenance activities. Phase 3 is intelligent alarm management, which is the application of advanced software tools for early fault detection and diagnosis.

Many techniques for fault detection and diagnosis have been developed and proven in industrial environments (Dvorak and Kuipers, 1991; Mannila and Ronkainen, 1998), showing their strengths and weaknesses. The integration of techniques for early fault detection and diagnosis has been studied to incorporate the best of each to detect abnormal situations in complex processes. Techniques for early fault detection and diagnosis help operations staff in making the best decisions, preventing incipient faults to climb into crises. Integrated techniques for early fault detection and diagnosis use the information available in industrial environments. A prototype tool in a Fluidized Catalytic Cracking Unit (FCC) has been developed.

IV. INTEGRATION OF TECHNIQUES

Loss of life and economic impact has been documented in the petrochemical industry due to improper handling of abnormal situations (Venkatasubramanian et al, 2003; Marsh Risk Consulting, 2001). The desired attributes for a fault detection and diagnosis system have been defined previously (Venkatasubramanian et al, 2003): Early detection and fault diagnosis, discrimination between different faults, robustness in the presence of noise and uncertainty, identification of new faults, identification of multiple faults; ease of explanation of performance and adaptability. Comparing different methods for fault detection and diagnosis, according to these criteria, none of the techniques used so far covers all the attributes of evaluation (Dash and Venkatasubramanian, 2003; Biswas et al, 2004). As a result of this comparison, a new approach is needed to use these techniques. It´s proposed a possible solution using an integrated architecture combining three methods, using artificial intelligence tools.

It´s been reviewed existing techniques for fault detection and diagnosis. Special emphasis has been drawn on those techniques that make use of available information in industrial processes: Alarms associated with the operational limits of the process variables, the experience of operations personnel in common faults and recorded in the form of rules, a simplified model of the process (step response) to predict the dynamic behavior during normal operations and fault scenarios. It´s been developed an extended fault dictionary and implemented a logical inference system to integrate symptoms (characteristic sign or indication of the existence of a fault) and results of each technique for fault detection and diagnosis.

A logical inference system has been used to incorporate the best of the techniques used, compiling expert knowledge in rules that have the form: IF antecedent is true THEN consequent is true, capturing the causal relationships in the process.

The inference process to validate the hypothesis of a fault takes place through backward chaining. The model used to detect disturbances (an early form in which faults are displayed in the process) is a unit step response model used in the multivariable control of the process unit to predict the behavior of the controlled variables based on calculated movements for the manipulated variables.

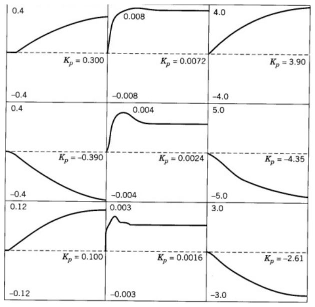


Fig. 1. Step model used to detect disturbances in the process.

Available information (from the alarms and other techniques) is incorporated in the extended fault dictionary (Agudelo et al, 2007). This information is: detected disturbances using the step model response of the process; similarity between the observed and previously identified alarm sequences during fault scenarios; symptoms observed during fault scenarios (base on operations personnel expertise in fault detection and diagnosis). The symptoms are the antecedents of the rules of the expert knowledge base, which replicate the inference mechanism used by operations personnel in detection and diagnosis of abnormal situations.

Alarms should be seen as tools for the fault detection and diagnosis. In the following figure, the sequences of alarms to major faults of the process are shown. These sequences were constructed from FMEA analysis (Fault Modes and Effects Analysis) made in the process (Suttinger et al, 2005).

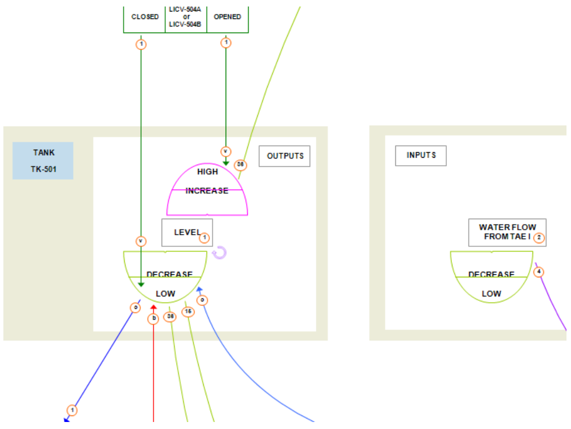


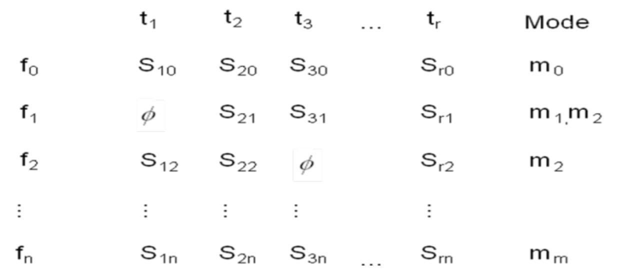
Fig. 2. Process main faults Alarm sequences.

Knowledge engineering methodologies have to build the knowledge base with experience of operations personnel (Russell and Norvig, 1996): Definition of terms to include in the model, variables influencing it; coding qualitative and quantitative dependence of variables; special cases consultation inference procedures to validate the experts' responses, and sensitivity analysis to establish the robustness in the presence of disturbances.

V. THE EXTENDED FAULT DICTIONARY

The integration of techniques for detecting and diagnosing faults must solve conflicts when each technique results do not match. Each technique does not detect all possible faults. Diagnostic information from different sources must be merged correctly (ordered timestamps for system events, updated results of each diagnostic technique, equipment performance monitoring to validate hypotheses, etc.).

Available information (alarms and other techniques) is incorporated in the extended fault dictionary (Agudelo et al, 2007).



	t_1	t_2	t_3	…	t_r	Mode
f_0	S_{10}	S_{20}	S_{30}		S_{r0}	m_0
f_1	ϕ	S_{21}	S_{31}		S_{r1}	m_1, m_2
f_2	S_{12}	S_{22}	ϕ		S_{r2}	m_2
⋮	⋮	⋮	⋮		⋮	
f_n	S_{1n}	S_{2n}	S_{3n}	…	S_{rn}	m_m

Fig. 3. Extended fault dictionary.

The software tool, implemented as a prototype, mimics the inference process carried out by operators in analyzing an abnormal situation, doing backward chaining of the rules of the knowledge base. These rules recognized the possible fault scenarios and are constructed from the extended fault dictionary. The rule ij is:

$$S_{ij} \wedge Mode(m) \Rightarrow f_j \tag{1}$$

Where the fault hypothesis (f_j) is on the consequent, and the validation symptoms (S_{ij}) are in the antecedent side along with the operation mode in which the fault hypothesis is valid. There are 5 modes used to represent the conditions of the process: Start mode, normal production, production defective, stop mode and emergency mode. The transition between modes is estimated by monitoring the operational conditions of the process, as shown in (Caceres and Ropero, 2006).

VI. PROTOTYPE

The intelligent software tool that performs fault detection and diagnosis has been built from the above concepts in a Fluidized Catalytic Cracking Unit, located at the refinery of Barrancabermeja (Colombia). The FCC process was chosen because there are economic incentives regarding proper fault detection and diagnosis in this process, besides the process safety incentives. The knowledge base required to build the application was compiled through interviews with operators, supervisors and engineers, experts in the process, using the appropriate methodology (Brulé, 1989). The software tool has been built using Builder C++[1]. A prototype tool to connect to the electronic control system and the database in real time from the refinery have been built. The connection between these two platforms is achieved through a SQL database Server[2].

Main faults detected by the software tool include (but are not limited to): Problems in the catalyst circulation; catalyst losses; coke formation / fouling; reverse flow; high temperature in the regenerator; afterburn; problems with the quality and quantity of products. The main reasons for these faults are: Operating conditions; mechanical problems; feed characteristics; and catalyst characteristics, as described in (Sadeghbeigi, 2000).

One rule for the fault "Catalyst circulation limitation and reversal flow" due to a mechanical failure, in the intelligent software tool has the following symptoms:

Decrease in PDIC27149

Decrease in ZI27100

Decrease in ZI27101

Increase in PIC27116

Decrease in ZI27103

Increase in ZI27104

Where PDIC27149 is the tag for the differential pressure between reactor and regenerator, which indicates that a low differential pressure between the reactor and regenerator has been caused due to clogging in the tap of the reactor-regenerator differential pressure transmitter. ZI27100 and ZI27101 are the position of the slide valves of the regenerator flue gas lines, which indicates that the control system wants to press the regenerator. PIC27116 is the pressure controller of the regenerator. ZI27103 and ZI27104 are the positions of the slide valves for the regenerated catalyst and spent catalyst lines between the reactor and the regenerator, respectively, and the control system decreases the position of the ZI27103 valve in order to decrease the reactor pressure, due to a decrease in temperature. The cause of the fault is "clogging in the pressure taps in the reactor pressure transmitter". If there is a clogging in the tap of the reactor differential pressure transmitter, the control system is going to close the slide valves to press the regenerator, increasing the differential pressure between the regenerator and the reactor, increasing the risk that the air blower decreases its flow. The recommendation to the operator is: To put the PDIC27149 in manual and to adjust its output in the value it had just before the disturbance. Once the software tool detects a fault, it warns the operator, and gives him/her the recommendations needed to prevent an incident to happen or to recover from the fault state in the best way.

At first, a direct connection to the control system (to download online the values of the measuring instruments of the process and to make the estimation of symptoms), was developed, but then it was discovered that this connection could overload the electronic control system, which is impermissible in any case. Then, the software was change to connect to the real time database from the refinery. The software architecture is shown in the figure below.

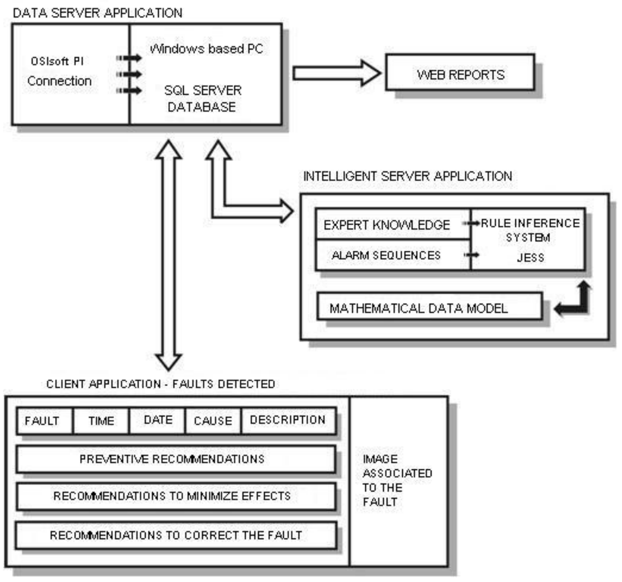


Fig. 4. Software tool architecture.

Connectivity tests were made using an Operator Training System (OTS) of the process, and the installation in the real process at the refinery has already been done. The prototype tool is in evaluation to measure the impact of the tool on-line in the FCC process and the software tool has detected and diagnosed several flaws in the process. A list of some faults detected by the software tool is provided:

Decreased catalyst / oil ratio.

Increased addition of fresh catalyst by pressure.

Increasing the reaction temperature.

[1]Builder is a trademark of Borland Corporation.

[2]SQL Server is a trademark of Microsoft Corporation.

Decreased end point of the gasoline, raising the temperature of the top of fractionator.

Increasing the peak temperature of the fractionator because an increase in the level of acid water (LIC27001) of D2705.

Reactor cyclones failure.

Regenerator cyclones failure.

Failure by low signal in the instrument flow transmitter pressure outlet anti-surge main air blower.

VII. CONCLUSIONS

This article shows the integration of three techniques for fault detection and diagnosis: Alarms using electronic control system, the faults knowledge base based on operations personnel expertise, and a simplified model to detect disturbances in the controlled variables of the process. It supports the use of advanced software tools to help early fault detection and diagnosis. It shows a prototype tool in an FCC process. It shows how early fault detection and diagnosis aid the process safety.

REFERENCES

Hopkins, Andrew. Thinking About Process Safety Indicators. Oil and Gas Industry Conference. Manchester (UK), November 2007.

Norman C. Chernobyl—errors and design flaws. Science 1986;233(4768): 1029–31.

Reason J. The Chernobyl errors. Bulletin of the British Psychological Society 1987;40:201–20–>206.

Salge M, Milling PM. Who is to blame, the operator or the designer? Two stages of human failure in the Chernobyl accident. System Dynamics Review 2006;22(2):89–112.

Belke JC, Dietrich DY. Chemical accident risks in US industry—a preliminary analysis of accident risk data from US hazardous chemical facilities. Paper presented at the 10th international symposium on loss prevention and safety promotion in the process industries, Stockholm, Sweden; 2001.

Shrivastava P. Bhopal: anatomy of a crisis. Cambridge, Mass: BallingerPub. Co.; 1987.

Drysdale DD, Sylvester-Evans R. The explosion and fire on the PiperAlpha platform, 6 July 1988. A case study. Philosophical Transactions of the Royal Society of London Series, a Mathematical Physical and Engineering Sciences 1998; 356(1748):2929–51.

Paté-Cornell ME. Learning from the Piper Alpha accident—a postmortem analysis of technical and organizational factors. Risk Analysis 1993;13(2): 215–32.

Bakolas, Efstathios; Saleh, Joseph. Augmenting defense-in-depth with the concepts of observability and diagnosability from Control Theory and Discrete Event Systems. Reliability Engineering and System Safety 96, (2011) 184–193

Saleh, Joseph; Cummings, Amy. Safety in the mining industry and the unfinished legacy of mining accidents: Safety levers and defense-in-depth for addressing mining hazards. Safety Science 49, (2011) 764–777.

Smith, Carlos; Corripio, Armando. Principles and Practice of Automatic Process Control. Ed. Wiley, 2nd edition. 1997.

Agudelo, Carlos. Integration of early fault detection and diagnosis techniques. Application to a Fluid Catalytic Cracking process. Document developed as requirement for the Advanced Studies Diploma. Universidad Politécnica de Valencia, 2010.

Acero, Carolina; Riascos, Federico; Agudelo, Carlos; Torres, Enrique. Gerenciamiento de Alarmas: Documento filosófico para el manejo de alarmas en la Gerencia Refinería de Cartagena. Instituto Colombiano del Petróleo, ECOPETROL. Piedecuesta (Colombia), May 2005.

Acero, Carolina; Riascos, Federico; Agudelo, Carlos; Torres, Enrique. Gerenciamiento de Alarmas en GRC: Diagnóstico preliminar Unidad de Ruptura Catalítica-Fase I. Instituto Colombiano del Petróleo, ECOPETROL. Piedecuesta (Colombia), September 2005.

Acero, Carolina; Riascos, Federico; Agudelo, Carlos; Torres, Enrique. Gerenciamiento de Alarmas en GRC: Informe de medición Post-Fase I Unidad de Ruptura Catalítica. Instituto Colombiano del Petróleo, ECOPETROL. Piedecuesta (Colombia), December 2005.

A. Ghariani, A. K. A. Toguyéni, E. Craye, A Functional Graph Approach for Alarm Filtering and Fault Recovery for Automated Production Systems, wodes, pp.289, Sixth International Workshop on Discrete Event Systems (WODES'02), 2002.

Dvorak, Daniel y Kuipers, Benjamin. Process monitoring and Diagnosis: A model-based approach. IEEE Expert 6(3): 67-74, june 1991.

Mannila, Heikki; Ronkainen, Pirjo. Similarity of Event Sequences (Extended Abstract). University of Helsinki, Department of Computer Science, 1998.

Venkatasubramanian, V. et al. A review of process fault detection and diagnosis. Part I: Quantitative model-based methods. Computers and Chemical Engineering 27 (2003), pp.293-311.

Marsh Risk Consulting, Large Property Damage Losses in the Hydrocarbon Industries, A Third Year Review. 2001.

Dash and Venkatasubramanian. Integrated Framework for Abnormal Event Management and Process Hazards Analysis. AIChe Journal, Enero 2003. Vol 49, No 1.

Biswas, Cordier, Lunze, Travé-Massuyès, Staroswiecki. Diagnosis of Complex Systems: Bridging the Methodologies of the FDI and DX Communities. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics, Vol. 34, No 5, Octubre 2004. pp.2159-2162.

Hamscher, W.; Console, L.; De Kleer, J. Readings in Model-based Diagnosis. Morgan Kaufmann Publishers Inc., California, 1992.

Agudelo, Carlos; Quiles, Eduardo; Morant, Francisco. Uso de Sistemas Expertos en el Diagnóstico de Fallos en Procesos Complejos. XIII Convención de Ingeniería Eléctrica. Univ. Central Maria Abreu de las Villas. Villa Clara (Cuba), June 2007.

Suttinger, Larry; et al. FMEA and the Installed SIS. ISA EXPO 2005 Technical Conference. Westinghouse Savannah River Company. 2005.

Russell, S and Norvig, P. Artificial Intelligence: A modern approach. Prentice Hall, 1st edition. 1996.

Caceres, L. and Ropero, A. Desarrollo de un modelo de simulación para el análisis de fallos en la unidad de ruptura catalítica de la refinería de Cartagena – ECOPETROL S.A. B.S. Tesis. Universidad Pontificia Bolivariana, Bucaramanga (Colombia), 2006.

Brulé, James F; Knowledge acquisition. New York McGraw-Hill 1989.

Sadeghbeigi, Reza. Fluid Catalytic Cracking Handbook. Gulf Professional Publishing Company, 2nd edition. Houston (USA), 2000.

Carlos Agudelo is an advanced process control specialist in the oil and gas industry, working for the Colombian Petroleum Institute (the R&D center of the Colombian Petroleum Company, Ecopetrol S.A.) for more than ten years. He studied his bachelor of Design and Electronic Automation Engineering at Universidad De La Salle in Bogota (Colombia), and his Master of Automation and Industrial Informatics from Universidad Politecnica de Valencia, at Valencia (Spain). He is currently finishing his PhD of Automation and Industrial Informatics at Universidad Politecnica de Valencia.

Francisco Morant received the B.S and the Ph. D. degree from the Universidad Politecnica de Valencia. He is a professor of Electrical Engineering at Universidad Politecnica de Valencia. His theoretical research interest include: adaptive control and the application of Artificial Intelligence Techniques to continuous process control. His theoretical work has been applied in process control industry.

Eduardo Quiles Cucarella was born in Valencia, Spain, in May 1967. He received the M.Sc in Electrical Engineering from Universidad Politecnica de Valencia in 1993, and the Ph.D. in the same University in 1998. In 1996 he joined the Systems and Control Engineering Department of the Universidad Politecnica de Valencia, where he is Associate Professor. His current research areas are Fault Detection and Diagnosis and System Reliability.

Emilio García Moreno is at the Department of Systems Engineering and Control of the Universidad Politecnica de Valencia (Spain) since 1989. He is Associate Professor of Electrical and Automation Engineering. His research interest covers discrete-event and hybrid systems control, failure diagnosis and predictive supervision and applications in Factory Automation and Process Control Systems. At the moment, he is the main researcher in a subproject of Eolia Project, focused on the predictive supervision, fault diagnosis and maintenance of offshore farms of Power Wind Generators.