Identifying Similar Past Events in a Continuous Monitoring System

YongChul Kwon
University of Washington
Seattle, WA
yongchul@cs.washington.edu

Guiping Xu*
Huazhong Univ. of Sci. & Tech.
Wuhan, China
gpxu@mail.hust.edu.cn

Magdalena Balazinska
University of Washington
Seattle, WA
magda@cs.washington.edu

ABSTRACT

Previous work on data stream management systems has enabled sophisticated tools for continuous, low latency stream processing. These tools are useful in many application domains including computer system monitoring and network monitoring. In most monitoring applications, however, when an abnormal event occurs, an administrator must manually examine the current state and the history of the system to understand what happened and diagnose the problem.

To facilitate this process, we propose a new technique that automatically compares events on streams and identifies past events similar to newly detected events. With our technique, an administrator is shown not only an alert but also past alerts that resemble the current situation. At the heart of our technique is a new similarity measure geared specifically toward the continuous monitoring domain. In this domain, interesting events typically correspond to abnormal situations. Two events are thus most alike when the same monitored objects record similar abnormal values. We show that existing similarity measures do not work well in this environment and we develop a new measure, the *Context Distance Measure* (CDM), geared specifically toward the monitoring domain. Through experiments with a real dataset from the PlanetLab overlay network, we show that CDM outperforms existing techniques by producing more accurate rankings of similar past events.

1. INTRODUCTION

Exploiting and managing large systems (*e.g.*, computer networks, overlay networks, clusters of computers, grids) requires the ability to monitor these systems continuously [20, 33, 38]. Continuous monitoring enables an administrator to assess the health of a system, be notified when abnormal conditions occur, and diagnose problems. System monitoring, however, raises important data management challenges.

In monitoring applications, a set of data sources continuously produces information about the monitored system: *e.g.*, network monitors produce information about network traffic [17]; computer system monitors periodically relay information about the state of servers [33]. This information is processed continuously in near real-time by either a general-purpose [1, 2, 13, 21, 34] or a domain-specific [29, 33, 40] engine that typically provides real-time data visualization [40] and event detection (*e.g.*, intrusions, failures, overload, and other anomalies) functionality [1, 2, 13, 21, 29, 34, 40].

When a serious event is detected, the administrator must usually diagnose and fix (or otherwise handle) the problem, which can be difficult. Some domain specific tools support this task by performing automatic anomaly identification and classification [17, 29, 30]. However, when an event cannot automatically be categorized, an administrator must manually view [33], replay [17], or otherwise query [35] *historical* data to understand the cause of the problem. Because historical data is large, this process can be slow and difficult [25].

To improve this manual event investigation process, we propose to extend a general-purpose data stream management system (such as [1, 2, 13, 21, 34]) with the capability to automatically identify and extract for each newly detected event, similar events that occurred in the past. In the absence of a domain specific anomaly identification and classification tool, our general-purpose mechanism can help an administrator understand a new event by examining similar past events. For example, when a server suddenly becomes overloaded, our approach can automatically identify earlier overload events, where the same set of processes were hogging the same resources (cpu, memory, or network bandwidth). If a recent patch causes an application to experience severe memory leaks, our approach will help the administrator quickly identify this problem.

Automatic identification and extraction of similar past events raises two important challenges. First, the attributes of an event are often insufficient to fully describe the event. Frequently, a fraction of what constitutes the state of the world at the moment when the event occurred is also relevant to the event. In the above example, the set of processes running on the server are relevant to the overload event. We thus need a mechanism for the administrator to define such *context* of interest for an event. Second, we need to define an appropriate measure for computing event similarity. The measure should compare not only the events themselves but also their contexts. It should rank events in a manner that places the most useful events for a system administrator on top of the list. In previous work, we defined the notion of an event and its context [8]. In this paper, we focus on the latter problem of comparing these events and contexts.

In particular, we study the applicability of existing similarity measures including the Jaccard coefficient, the Earth Mover's Distance (EMD) [39], and the Match-And-Compare

*This work was done while the author was visiting the University of Washington partially sponsored by China Scholarship Council.

distance (MAC) [28] to the problem of comparing events on streams in a manner that supports monitoring applications. We show that these traditional measures often do not produce the desired results, and we introduce a new measure, the *Context Distance Measure* (CDM), that does. CDM builds on traditional distance measures, but it combines and extends them in a manner that meets the needs of monitoring applications. CDM is based on the observation that administrators are typically interested in entities that exhibit abnormal values. CDM thus favors past events that contain the same entities with similar abnormal values.

Through experiments with over 3,000 real events that occurred on the PlanetLab [37] overlay network (which spans hundreds of machines across the globe), we show that, for the monitoring domain, CDM outperforms existing distance measures ranking past events in a manner that more closely matches the needs of a system administrator. We also show that user-defined event contexts are effective at constraining the search for similar past events to only those events relevant to the user's interest: with our approach, two users interested in the same event but from different perspectives are often shown completely disjoint sets of past events.

The rest of this paper is organized as follows. In Section 2, we present our model and problem statement. In Section 3, we describe and discuss existing similarity measures and present CDM. We evaluate the performance of traditional distance measures and our approach on the PlanetLab workload in Section 4. We present related work in Section 5, and conclude in Section 6.

2. EVENTS AND CONTEXT

In this section, we review and refine the notion of an *event* and *event context* as introduced in our previous work [8]. We also define the problem of comparing events and their contexts.

2.1 Event

In our model, an event is a tuple in a stream that takes the form: (`eid, timestamp,` a_1`, ...,` a_n), where `eid` is an attribute that uniquely identifies the event, `timestamp` is the time when the event occurred, and a_1, ..., a_n are the other attributes of the event. Our definition of an event is thus broad. Almost any tuple in any stream can be considered an event as long as it takes the form specified above. Typically, events are output tuples produced by continuous queries (we call these queries *event-queries* [8]).

Our system does not know nor care that tuples in a stream correspond to events until the stream is connected to our special *Similarity Recall* operator (described below). At that point, the user specifies which attributes of the stream correspond to the event identifier and timestamp.

Figure 1 shows examples of "server overload events". Each tuple in the table corresponds to one event. Each event includes a unique identifier (`eid`), the identifier of the overloaded server (`sid`), and the time when the overload occurred (`timestamp`).

2.2 Event Context

The attributes of an event describe its main properties and usually suffice for a human observer to uniquely identify the event. For example, for a server overload event, the server identifier (`sid`) and `timestamp` suffice for the user to distinguish between different server overload events.

eid	timestamp	sid
1	3:20pm	13
2	3:25pm	23
3	4:05pm	123
4	5:53pm	13
5	7:29pm	3

Figure 1: Example of server overload events. `eid` is the event identifier, `timestamp` is the time when the event occurred. `sid` identifies the overloaded server.

ASPECT 1: Overall resource utilization

eid	timestamp	cpu	memory	network
1	3:20pm	0.98	0.96	0.4

ASPECT 2: Per-process resource utilization

eid	timestamp	pname	cpu	memory	network
1	3:20pm	P1	0.03	0.01	0.07
1	3:20pm	P2	0.25	0.20	0.30
1	3:20pm	P3	0.65	0.70	0.03
1	3:20pm	P4	0.05	0.05	0.00

Figure 2: Sample context for a server overload event with `eid` = 1. The context has two aspects. Each aspect is a relation.

To understand and diagnose an event, however, a user typically needs to see a significant amount of additional information about the state of the system at the moment when the event occurred. We call this state the *context of the event*. For example, when a server overload occurs, a user may be interested in seeing the exact resource utilization on the overloaded server along with the set of running processes and their respective resource utilization.

Because each overload event is associated with *a set* of processes running on the overloaded server, this information cannot be captured by extending the list of attributes of the event itself. Instead, the event is obtained with additional continuous queries that join the event stream with other streams and relations. We call these queries *context-queries* to distinguish them from event-queries [8]. For each event, the output of each context-query is a relation that forms *one aspect* of the event context. Figure 2 shows an example of a context for a server overload event. This context includes the two aspects that we described above: the overall resource utilization and the list of processes running on the overloaded server.

More precisely, an aspect is a relation containing tuples of the form: (`eid, timestamp,` c_1`, ...,` c_n), where `eid` and `timestamp` are the identifier and timestamp of the event, and $c_1, \ldots, c_n$ are the attributes of the tuples in this aspect. We keep both the event identifier and timestamp to ensure that all tuples in streams have a timestamp.

2.3 Query Model

Our system is based on the Borealis [2] distributed stream processing engine. As such, users express queries with boxes-and-arrow diagrams where boxes correspond to stream processing operators and arrows denote streams. The output of event and context queries is connected to a new operator that we call *Similarity Recall* [8]. For each event and its context, the Similarity Recall operator queries the historical log to extract a set of top-k most similar past events. In this

paper, however, the exact query model is inconsequential. We focus only on the similarity computation performed by the Similarity Recall operator.

2.4 Problem Statement

In this paper, we address the problem of designing, implementing, and evaluating a new measure, CDM, that enables a Similarity Recall operator to compute the distance between two events and their contexts. Similarity Recall uses this measure to compare and rank past events.

CDM is designed to operate on complex objects: it takes as input two *sets of relations*: *i.e.*, the contexts of the two events being compared. It outputs a positive real number which reflects the distance between the contexts. More specifically:

$$\text{CDM} : S \times S \rightarrow \mathcal{R}^+$$

where S is a set of relations.

Our goal is for CDM to reflect a notion of event distance that is useful for monitoring applications. In particular, our goal is for CDM to have the following three properties:

1. **Entity similarity**. If the aspects of two different event contexts contain the same entities (*i.e.*, tuples with the same keys), the contexts are similar and the distance between these contexts should be small.

2. **Value similarity**. If the aspects of two different event contexts contain entities with similar attribute-values, the contexts are similar and the distance between these contexts should be small.

3. **Prioritizing entities with abnormal values**. When comparing event contexts, entities with abnormal values should be prioritized over other entities.

   Indeed, in monitoring applications, when an event occurs, if an aspect of the context of that event contains entities with abnormal attribute values, these entities are most interesting to the administrator. For example, in Figure 2, processes `P3` and `P2` from `Aspect 2` use a large fraction of all system resources and are thus more interesting than processes `P1` and `P4`.

Overall design goal. Taken together, the above three properties define the requirement for CDM: for each newly detected event, CDM should identify as most similar those past events where the *same entities appear with similarly abnormal attribute-values.*

To the best of our knowledge, existing similarity and distance measures satisfy only properties 1 and 2. They have no notion of normal or abnormal attribute values. CDM is the only distance measure that satisfies all three properties and achieves our overall design goal.

Given this design goal, for the event shown in Figure 2, CDM should produce a ranked list of past events where the overall CPU and memory utilization were abnormally high and where:

1. processes `P3` and `P2` were hogging most of these resources.
2. process `P3` was using a large fraction of system resources.
3. process `P2` was using a large fraction of system resources.
4. processes `P3` and/or `P2` were responsible for using a significant fraction of resources other than CPU and memory.
5. a small set of processes other than `P3` and `P2` was using most system resources.
6. any other condition holds.

This example illustrates well the type of ranking that we wish to achieve. In this example, `P3` is prioritized over `P2` because, in the newly detected event (shown in Figure 2), the attribute values of `P3` are more abnormal.

3. CONTEXT DISTANCE MEASURE

Comparing the contexts of two events requires comparing two sets of relations. Because no existing metric is designed for comparing such sets of sets of multidimensional objects, we cannot apply an existing metric directly. Instead, we propose a new measure, CDM, that integrates, combines, and extends different metrics in a manner that satisfies all three of the above properties. In this section, we present CDM by building it bottom-up: we first describe distance functions for comparing individual entities within an aspect (Section 3.1), then aspects within a context (Section 3.2), and finally entire contexts (Section 3.3).

In the previous section, we defined the overall desired properties for CDM. In this section, we translate these globally desirable properties into specific properties for comparing two entities in an aspect or two aspects of a context. We present these properties as we describe the corresponding distance measures.

3.1 Comparing Entities

In the past years, there has been significant work in the area of ranking query results and top-k queries [10, 12, 14, 16, 22, 31]. As a result, a large number of distance functions have evolved for measuring how close two tuples are from each other. In this section, we discuss the applicability of these different functions to our problem.

For queries over continuous-valued real attributes, commonly used distance functions use vector p-norms [10], defined as: $||x||_p = \left(\sum_i |x_i|^p\right)^{\frac{1}{p}}$, $p \geq 1$. Given a p-norm $||.||$, the distance D between two tuples q and t is defined as $D_{||.||}(q,t) = ||q-t||$. Commonly used p-norms include sum ($||q-t||_1 = \sum_{i=1}^{n} |q_i - t_i|$) and Euclidean distance ($||q-t||_2 = \sqrt{\sum_{i=1}^{n}(q_i-t_i)^2}$).

To support arbitrary attribute types, some systems [11, 31] define separate scoring functions for individual attributes and measure the distance between tuples as a (possibly weighted) sum of attribute-value scores. For example, for a query requesting a list of cheap restaurants near a hotel, a scoring function may compute the weighted sum of the restaurant's price and distance from the given hotel. The weights enable the user to indicate a stronger preference for either cheaper or closer restaurants.

For categorical attributes, another commonly-used technique adapts the well-known Cosine Similarity metric from Information Retrieval [7] by treating each tuple as a small document and by computing the TF-IDF weights of all the attribute values in the tuple, where TF represents the frequency of a value in a tuple and IDF its inverse document frequency. Agrawal *et al.* propose an extended metric, called IDF Similarity [3], that works for both categorical and numerical data.

In our case, tuples contain a mix of numerical and categorical attributes. Therefore both IDF Similarity and p-norm are good candidates for capturing the distance between tuples. Because it would be difficult and cumbersome for a system administrator to define other scoring functions or assign weights to individual attributes, we do not investigate such techniques. We thus focus our study on the IDF Similarity metric and the Euclidean distance, a representative of the family of functions based on p-norms. When comparing tuples, their `eids` and `timestamps` are ignored as they correspond to tuple meta-data.

Because the domain of different attributes can be significantly different, we normalize all values before computing the distance between tuples: we divide all values by the maximum value measured for the given attribute.

Additionally, instead of distances, IDF Similarity produces similarity values. We sometimes need to convert these similarity values into distances to use them with similarity measures for entire aspects (as we discuss next). To perform this conversion, different functions are possible. We use a simple inverse hyperbolic secant function[1], which nicely spreads the values across the entire range. Tuples that are either very similar or very different get assigned values close to 0 and 1 respectively. The other tuples get spread over the entire range. We also found this conversion function to perform well in our experiments.

An important advantage of IDF Similarity over Euclidean distance is that abnormal values are given more weight, which matches well our goal of prioritizing entities with abnormal attribute values. On the other hand, IDF Similarity does poorly as soon as the numeric values of some attributes are even moderately distant: the similarity function quickly goes to zero failing to distinguish between nearby and very distant values. We experiment with both metrics in Section 4.2 where we show that, in our context, IDF Similarity is insufficient for capturing abnormality in an event context and does not outperform the simpler and computationally lighter Euclidean distance metric.

Even though Euclidean distance performs better than IDF Similarity, it does not have all our desired properties as a distance measure. For example, assume that process $P1$ is using 25% of CPU and considered as abnormal. Two past CPU usages of $P1$ are fetched: 1% and 70% each. In Euclidean space, clearly 1% usage is about 2 times closer to 25% than 70% usage. However, 70% usage is more interesting than 1% usage because it is an abnormal value and just more severe than 25%. One key observation is that all abnormal values are either below or above a certain threshold in many cases. Sometimes they are even extreme as illustrated in the example thus the difference in Euclidean space fails to capture the closeness in abnormality. Another observation is that abnormal values are infrequent than normal values. IDF Similarity also catches this but it fails to deal with the first observation. Thus, the distribution is dense in normal region and sparse in abnormal region. From the observations, we can say that given two values are far if many samples lie between them and close if there are few samples no matter how far they are apart in Euclidean space. This successfully captures the closeness in abnormality which Euclidean distance can't. From different perspective, this idea is exactly the same as the simplest row estimation technique of modern database for range query. To rest of the paper, we call this measure as the `Histogram distance` because the approximation can be easily computed by using histogram. Another advantage is that it is computationally cheaper than IDF Similarity while it is more suitable for our context. We further discuss the implementation issues of the Histogram distance in Section 3.2.2.2. We also explore the Histogram distance as well in Section 4.2 and show that it outperforms Euclidean distance.

[1] $f(x) = \ln \frac{1+\sqrt{1-x^2}}{x}$

3.2 Comparing Aspects

Comparing two aspects involves comparing two relations. In this section, we first present traditional distance measures for comparing sets of objects and discuss why applying these measures directly does not achieve our three desired properties. We then present our measure, ADM, which builds on existing techniques, but combines and uses them in a way that meets our goals.

Throughout the section, we use Figure 3 as running example. The Figure shows five aspects. Aspect `Q` is the aspect of the newly detected event from Figure 2. This aspect includes two processes `P2` and `P3` with abnormally high resource utilization levels; `P3` having more abnormal resource utilization levels than `P2`. Aspects `T1` and `T2` each contain one of these processes. Looking only at the attribute values, `T2` should be closer to `Q` than `T1`. Based on property 3, however, the distance between `Q` and `T1` should be smaller because `T1` contains the more abnormal of the two processes. Aspect `T3` is less similar than both `T1` and `T2` because the abnormal processes it includes are different from the ones in `Q`. Finally, aspect `T4` is the least similar of the four. Even though it contains two processes, `P1` and `P4`, that perfectly match those in `Q`, these processes use only little resources, their attribute values are within the normal range.

3.2.1 Traditional Distance Measures

Jaccard Similarity Coefficient

Many traditional (multi)set-similarity metrics assume that the elements of the (multi)set come from a flat domain. As such, they are only applicable when comparing set memberships. The commonly used Jaccard similarity coefficient is one such example. For two sets Q and T, their Jaccard coefficient is given by:

$$\text{JACC}(Q,T) = \frac{|Q \cap T|}{|Q \cup T|}$$

The Jaccard similarity coefficient thus only measures the amount of overlap in the two sets being compared. In our case, this information is insufficient, as it does not consider the attribute values of the tuples in the sets. For the aspects from Figure 3, this metric identifies aspects `T4` followed by `T3` as the two most similar of the four past aspects. This is exactly the reverse of what we would like.

Hausdorff Distance

The Hausdorff distance [27] is a set-comparison metric that takes the distance between individual points into account. Informally, given the minimum distance between each element in the first set and some element in the second set, the Hausdorff distance between the sets is the maximum of these minimum distances. Because it is asymmetric, it is common to take maximum of the two distances from each set and makes it symmetric.

ASPECT Q: Aspect of the newly detected event

eid	timestamp	pname	cpu	memory	network
4653	3:20pm	P1	0.03	0.01	0.07
4653	3:20pm	**P2**	**0.25**	**0.20**	**0.30**
4653	3:20pm	**P3**	**0.65**	**0.70**	0.03
4653	3:20pm	P4	0.05	0.05	0.00

ASPECT T1: Aspect of past event 1

eid	timestamp	pname	cpu	memory	network
1001	6:24am	**P3**	**0.70**	**0.75**	0.08
1001	6:24am	P5	0.05	0.05	0.00

ASPECT T2: Aspect of past event 2

eid	timestamp	pname	cpu	memory	network
2357	8:00am	**P2**	**0.27**	**0.22**	**0.32**
2357	8:00am	P5	0.05	0.05	0.00

ASPECT T3: Aspect of past event 3

eid	timestamp	pname	cpu	memory	network
2653	10:52am	P1	0.04	0.02	0.06
2653	10:52am	P4	0.05	0.05	0.03
2653	10:52am	P6	**0.25**	**0.20**	**0.30**
2653	10:52am	P7	**0.65**	**0.70**	0.03

ASPECT T4: Aspect of past event 4

eid	timestamp	pname	cpu	memory	network
3988	1:03pm	P1	0.03	0.01	0.07
3988	1:03pm	P4	0.05	0.05	0.00

Figure 3: Examples of five aspects. Based on our desired properties, given the query aspect `Q`, aspect `T1` should be ranked first, followed by `T2`, `T3`, and finally `T4`.

This metric thus captures our second desired property (value similarity). Because entity identifiers are included in the distance computation it also somewhat captures our first property (entity similarity). This metric, however, does not capture our third desired property. In our example, `T3` has the smallest Hausdorff distance from `Q`, just because tuples have very similar values. However, entities in `T3` containing abnormal values are different from those in `Q` and that property is not captured. Additionally, with the Hausdorff distance, a single outlier may significantly impact the distance between two aspects.

Cartesian Product

A naïve approach for comparing two relations in a manner that captures the distance between all tuples is to simply compute the sum of the distances between all pairs of tuples and normalize by the product of the relation cardinalities. We call this naïve metric, the *Cartesian product* distance (CART). For two relations Q and T:

$$\text{CART}(Q,T) = \frac{\sum_{q\in Q}\sum_{t\in T}\text{dist}(q,t)}{|Q|\cdot|T|}$$

As we show in Section 4.2, this metric, however, is ineffective at capturing similarity between aspects as soon as the points in the aspect all not all clustered together.

Match And Compare Distance

The Match And Compare (or MAC) distance [28] measures the similarity between two sets of objects more accurately than the above techniques. MAC has been introduced to quantify the error in approximate answers produced by a database and is thus designed specifically for comparing relations. Instead of computing the distance between all pairs of tuples in the two relations, MAC first matches tuples between the two sets. MAC then computes the distance between the sets by taking into account both the distance between individually matched tuples and the cardinality of the different matches.

More specifically, given two multisets $Q = \{q_1,\ldots,q_n\}$ and $T = \{t_1,\ldots,t_m\}$ that represent the two relations, MAC first matches elements of the two sets by computing a minimum cost edge cover of the bipartite graph formed by the relations. Given this match, the distance between the relations is then given by the following expression, where C is the set of edges in the minimum cost edge cover, and d_1 and d_2 are the number of edges incident to q_i and t_j respectively.

$$\text{MAC}^{l,m}(Q,T) = \sum_{(q_i,t_i)\in C}\text{mult}^{l}(q_i,t_j) * dist[q]^{m}(q_i,t_j)$$

where

$$\text{mult}(q_i,t_j) = \max(1,\max(d_1,d_2)-1)$$

$$\text{dist}[q](q_i,t_j) = \begin{cases}\text{dist}(q_i,t_j), & \text{if} \quad \max(d_1,d_2)=1\\ \text{dist}(q_i,t_j)+q, & \text{otherwise}\end{cases}$$

l, m, and q are configurable parameters of the MAC distance that adjust the penalty when multiple tuples match with the same nearby tuple.

Because MAC is much more accurate than a naïve metric such as Cartesian Product, it is a good candidate for our aspect distance measure. The only limitation of MAC is that it does not satisfy property 3: it does not capture how normal or abnormal different attribute values are. In our example, MAC ranks `T3` as the closest aspect to `Q`. It also ranks `T4` above `T2` and `T1`. Finally, MAC has three configurable parameters, thus requiring careful tuning.

Earth Mover's Distance

The Earth Mover's Distance (EMD) [39] is a metric originally introduced for content-based image retrieval and successfully used in multimedia databases [6].

EMD takes as input two sets(or signatures) $Q = \{(q_1,w_{q_1}),\ \ldots,\ (q_m,w_{q_m})\}$ and $T = \{(t_1,w_{t_1}),\ \ldots,\ (t_n,w_{t_n})\}$ and a matrix $D = [d_{ij}]$, where each d_{ij} is the distance between q_i and t_j. EMD computes the distance between Q and T as the minimum amount of work to be done to fill the "holes" represented by points in T with the "mounds of earth" represented by the points in Q. EMD achieves this goal by solving the following linear programming problem. Find a flow $F = [f_{ij}]$, with f_{ij} the flow between q_i and t_j that minimizes:

$$\text{WORK}(Q,T,F) = \sum_{i=1}^{m}\sum_{j=1}^{n} d_{ij}f_{if}$$

subject to ensuring that all flows are positive and they do not move more earth or try to fill more holes than is available. EMD is then defined as the work normalized by the total flow.

$$\text{EMD}(Q,T) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} d_{ij}f_{if}}{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{if}}$$

In our case, the q_i's and t_j's are tuples in the relations,

all weights w are equal to one, and the distances d_{ij} are the distances between individual tuples with the two aspects. Setting weights to one means that we equally treat all tuples in each aspect and EMD will find the minimum cost pairs between two aspects.

Similar to MAC, EMD is another good candidate for our aspect distance measure. EMD has the same limitation as MAC: it does not satisfy property 3. In our example, EMD ranks `T4` highest and `T3` second. In contrast to MAC, EMD itself has no configurable parameters, making it easier to use.

In Section 4.2, we experimentally compare the performance of the above functions in our application domain and find that MAC and its variants outperform the other schemes because it takes into account the cost of all mappings from querying aspect to the other aspect.

3.2.2 Aspect Distance Measure (ADM)

We now present our new measure, called Aspect Distance Measure (ADM) for comparing aspects. In general, ADM uses three techniques: ValueFilter, AttributeWeight, and TupleWeight. In addition, we present the KeyFilter, an extension of ADM. KeyFilter further improves the quality of results where the identity of monitored object is crucial. Because we find that, in our application domain, Normalized MAC(NMAC, see Section 3.2.2.3) outperforms the other measures (see Section 4.2), ADM uses NMAC. In this section, we first describe the filtering technique to illustrate overall structure of ADM. Then, we describe AttributeWeight and TupleWeight and conclude by describing NMAC, how we pick MAC parameters and normalize the distance using TupleWeights.

3.2.2.1 ValueFilter and KeyFilter.

To achieve property 3, ADM penalizes aspects that do not contain entities with abnormal attribute values (we call this the *ValueFilter* technique). To enforce this penalty, instead of using a scalar distance value, ADM represents the distance as a 3-dimensional vector.

Given two aspects `Q` and `T`, where `Q` is the aspect corresponding to the newly detected event, ADM computes their distance as follows:

$$Q' = \text{ValueFilter}(Q, X)$$
$$T' = \text{ValueFilter}(T, X)$$
$$\text{ADM}(Q, T) =$$
$$(\text{NMAC}(Q', T'), \text{NMAC}(Q', T), \text{NMAC}(Q, T))$$

where function `ValueFilter(R,X)`, extracts all tuples from relation `R` having at least one abnormal attribute value as defined by the vector `X`. More formally, the `ValueFilter` function is defined as follows:

$$\text{ValueFilter}(R, X) = \{r \in R \mid \exists i \; r_i > x_i \text{ for } x_i \in X\}$$

The abnormality vector `X` is set to select only unusual values. The choice of `X` is based on a histogram of the historical data[2]. The administrator only sets the percentile of data that should be considered abnormal. This threshold is the only tunable parameter in our technique, but tuning this parameter should be natural to an administrator accustomed to monitoring the system.

Note that ADM is asymmetric because it is based on an asymmetric distance, MAC: ADM(R_1,R_2) $\neq$ ADM(R_2,R_1).

For performance reasons, instead of computing all elements in the vector, ADM computes only the left-most possible element and sets all other elements to ∞. We consider that it is possible to compute MAC(R_1,R_2) only when both relations contain at least one tuple.

The comparison semantics of the ADM distance are then as follows:

$$\text{ADM}(Q, T_1) < \text{ADM}(Q, T_2) \iff$$
$$\exists i, \; \forall j \; \text{ADM}(Q, T_1)[i] < \text{ADM}(Q, T_2)[i]$$
$$\wedge (j < i \implies \text{ADM}(Q, T_1)[j] = \text{ADM}(Q, T_2)[j])$$

Thus, two ADM vectors are compared in lexicographic order.

In some environment, like system or network monitoring, the identity of abnormal entity is much more crucial than just similarity in values as elaborated in property 3. For example, if a process is constantly using lots of resources, only past events which the same process is hogging resources are interesting. To satisfy this requirement, ADM can be extended by a technique called `KeyFilter`. `KeyFilter(`R_1`,`R_2`)` is a function which extracts all tuples from relation R_2 that have the same key as some tuple in R_1. The extended ADM with `KeyFilter`, KADM, computes the distance between two aspects `Q` and `T` as follows:

$$Q' = \text{ValueFilter}(Q, X)$$
$$T' = \text{ValueFilter}(T, X)$$
$$\text{KADM}(Q, T)[0] = \text{NMAC}(Q', \text{KeyFilter}(Q', T'))$$
$$\text{KADM}(Q, T)[i] = \text{ADM}(Q, T)[i-1] \text{ where } 1 \leq i \leq 3$$

Thus, KADM returns a 4-dimensional vector. The difference is that it has one extra dimension which represents the distance between abnormal tuples of common key values. The lower three dimensions are identical to ADM. The comparison semantics of distance vector is intact. In Section 4.2.4, we show that the extended ADM(`KADM`) outperforms in considered application domain but we emphasize that `KeyFilter` is just an extension of ADM and only expected to be used where the identity of abnormal entity is the most concern. To the rest of this section, we use KADM in the examples because the identity is important in discussed application domain.

In our example, only the third element of the ADM(`Q, T4`) vector has a value other than ∞ because `T4` contains no entities with abnormal attribute-values (*i.e.*, $T'' = \emptyset$ and $T' = \emptyset$). In contrast, the second element of ADM(`Q, T3`) has a value other than ∞ as `T3` contains entities with abnormal attribute values, but these entities are different from those in `Q` (*i.e.*, $T'' = \emptyset$ but $T' \neq \emptyset$). For ADM(`Q, T1`) and ADM(`Q, T2`), their first element is set to a value other than ∞ (since $T'' \neq \emptyset$). Additionally, in the distance computations for `T1`, `T2`, and `T3` the processes `P1`, `P4`, and `P5` are filtered because they only contain uninteresting values. These entities do not affect any of the distance computations, which is desirable because they should not affect the final ranking of these similar past events.

3.2.2.2 AttributeWeight and TupleWeight.

In addition to the above ValueFilter and KeyFilter techniques, ADM also modifies MAC to take into account the

[2]Today, all major database management systems already maintain histograms over their data. Our approach simply exploits the existence of these histograms.

importance of different attributes in a tuple (we call this the *AttributeWeight* technique) and different tuples in an aspect (we call this the *TupleWeight* technique). Indeed, tuples that pass the ValueFilter may still have ordinary values for some of their attributes. To ensure that abnormal values are always prioritized over normal ones, ADM puts a heavier weight on the abnormal values when computing the distance between tuples.

For example, let's imagine a newly detected event with an aspect `S` similar to those shown in Figure 3, and two past events with their respective aspects `U1` and `U2`. Let's assume that `S` contains a process `P` with 60% CPU utilization and a 10Kbps network bandwidth utilization. `U1` and U_2 also contain `P`. However, in `U1`, `P` uses 30% CPU and 8Kbps network bandwidth but in `U2`, `P` uses 70% CPU and 30Kbps network bandwidth. Using our techniques so far, `U1` would be ranked as closer to `S`. However, we are more interested in `U2` because in this aspect the CPU utilization of `P` is more similar and the CPU utilization was more abnormal than the network bandwidth utilization in `S`. To capture this requirement, the distance for each attribute should not be treated equally but weighted according to the abnormality observed in the value of that attribute in the newly detected event. This is the role of the AttributeWeight.

We compute the abnormality for an attribute value as follows. Given, D_A, a database of all past aspects A, the abnormality of a value v for attribute c is defined as:

$$\text{Abnormality}_{D_{A.c}}(v) = \frac{|\{t \in D_A | t.c \leq v\}|}{|D_A|}$$

The abnormality of a value v is thus simply the fraction of tuples in past aspects that contain values below v. The abnormality of a value can easily be computed approximately using a histogram and standard catalog information. For better accuracy in both abnormality and Histogram distance, it is good to keep a small extra histogram which only covers the range above the abnormality threshold because the range is likely to be covered by only the rightmost(or leftmost) one or two buckets.

Dynamically weighing attribute distances based on the abnormality of their values helps capture the importance of certain attributes, but it does not help solve conflicts between tuples in an aspect. If a tuple contains multiple abnormal values, it is more important than a tuple with fewer abnormal values. Also, a tuple with multiple barely abnormal values is less important than a tuple with a single extremely unusual value. Certain tuples in the query aspect should thus be given more weight than others. To achieve this goal, we define a weight of tuple weight, called `TupleWeight`, equal to the average abnormality of the values in the tuple. The key may be included in the `TupleWeight` if the presence of an entity is interesting. Otherwise, it is excluded.

$$\text{TupleWeight}(q) = \frac{\sum_{i=1}^{|q|} \text{Abnormality}(q_i)}{|q|}$$

In our example, when using Euclidean distance with regular MAC, the distance between `Q` and `T2` is smaller than the distance between `Q` and `T1`. However, `T1` should be closer to `Q` because `P3` has significantly higher CPU and memory usage and has thus more abnormal values than `P2`. Thanks to TupleWeight, ADM correctly considers `T1` closer to `Q` because `P3` is given a higher TupleWeight.

3.2.2.3 Normalized MAC with TupleWeight.

AttributeWeight can be naturally embedded in computing distance between each attribute in tuple so we don't discuss how to apply it in ADM computation in detail. Original MAC distance has three parameters(l, m, q). Even though they make the measure flexible, they are just adding another complexity in our context. Thus, we end up with static parameters of $l = 0, m = 1, q = 0$ for ADM from following reasons. In ADM, we don't care about whether an element is matched multiple times in the other set because our top concern is only the best 1:1 matches of abnormal tuples. Thus, we set the multiplicity parameters l, q to 0. Along the same reasoning, m is set to 1.

Even with the chosen parameters, original MAC is still sensitive to the number of tuples in each set because it simply adds up all the distances among tuples in compared sets. Another problem is that the final distance is an arbitrary number which introduces another complication in CDM computation. On the other hand, EMD produces the average cost of flows between two sets. By doing so, EMD returns a normalized value between 0 and 1 if the weight and cost are normalized. By inspired EMD, we add normalization term to the MAC to address both the sensitivity of number of tuples and the distance in arbitrary range. In Section 4.2, we show that the normalized MAC with chosen parameters outperforms the regular MAC with different parameters. The final normalized MAC with TupleWeight is following:

$$\text{NMAC}(Q, T) = \frac{\sum_{(q,t) \in EC} dist(q, t) \cdot TupleWeight(q)}{\sum_{(q,t) \in EC} TupleWeight(q)}$$

where EC is a minimum cost edge cover of bipartite graph constructed over Q and T. To the rest of the paper, we refer this normalized MAC as `NMAC`.

In summary, ADM uses several techniques to (1) identify and isolate tuples that are significant for the comparison of two aspects and (2) put emphasize on those tuples and those attributes that contain more abnormal values in the newly detected event. The complexity of the resulting ADM measure is equal to the complexity of MAC. Additionally, thanks to the ValueFilter and KeyFilter techniques, ADM runs the modified version of MAC only on a small subset of all tuples in the original aspects.

3.3 Comparing Contexts

The Aspect Distance Measure, ADM, presented above is the core of our approach; it captures the key properties that we wish to achieve when comparing the contexts of two events. Event contexts, however, comprise more than one aspect. To compare event contexts, we thus need a way to combine the distance measures for individual aspects.

The Context Distance Measure (CDM) performs this task. Given two contexts $C = \{Q_1, Q_2, \ldots, Q_n\}$ and $D = \{T_1, \ldots, T_n\}$, where Q_i is an aspect of C and T_j is an aspect of D:

$$\text{CDM}(C, D) = \frac{\sum_{i=1}^{n} \text{ADM}(Q_i, T_i)}{n}$$

CDM is thus simply the average distance between the aspects of the two contexts. When computing this average, we

ignore all values set to ∞ as soon as one vector has a value other than ∞ for a given element.

The resulting distance is thus also a 3D vector(or 4D vector if it is extended by KeyFilter), which has the nice property that similarly to ADM, CDM naturally prioritizes event contexts comprising aspects with abnormal entities, especially if these same entities appear in the context of the newly detected event.

4. EVALUATION

In this section, we evaluate the performance of traditional measures and CDM for computing the distance between event contexts in monitoring applications.

4.1 Experimental Setup

We perform all our experiments on a real dataset from the computer-system monitoring domain. As example monitored system, we use the PlanetLab [37] overlay network. PlanetLab is a planetary-scale overlay network with over 700 machines at over 300 institutions around the globe. PlanetLab serves as a testbed for networking and distributed systems research.

PlanetLab is continuously monitored by two services called CoMon [18] and CoTop [19]. The CoMon service monitors each server in the overlay network and collects various runtime statistics such as average process queue length, network bandwidth, CPU utilization, and vmstat. The CoTop service monitors individual slices running on each server and collects their resource utilization. Each slice corresponds to one virtual machine that runs the processes belonging to one researcher. These virtual machines provide isolation between researchers. CoTop and CoMon poll all machines every 5 minutes. In our experiments, we use CoMon and CoTop data collected between October 2006 and December 2006.

From the CoMon log, we extracted 20,000 system overload events using the following selection predicates: (load1 $>$ load5) $\wedge$ (load5 > 50) $\wedge$ (load5 $> 2 \times$ load1D), where load1, load5, and load1D denote the average load over the last minute, 5 minutes, and 1 day respectively. The first predicate enables us to capture actual load spikes. The second predicate filters out inherently lightly loaded nodes. The number 50 is arbitrarily chosen based on sample queries from the CoMon web site. It is 10 times higher than the number provided in sample queries that select lightly loaded nodes. We find that only 0.8% of data has a value higher than this threshold. These two predicates, however, are insufficient to identify overload events because several PlanetLab nodes are very popular and are thus constantly overloaded. To eliminate these nodes, we add the last predicate.

4.2 ADM

In this section, we evaluate the performance of traditional measures and ADM for computing the distance between individual aspects.

There are many potential aspects that can be associated with system overload events. In this experiment, we use an aspect analogous to the one presented in the examples from Figures 2 and 3. This aspect shows the resource consumption of slices running on the overloaded server.

To extract these aspects, we take the CoTop log entry for the overloaded server at the time when the overload occurs. Because there are multiple concurrent active slices on a server at any time, a single CoTop log entry consists of multiple tuples, one tuple per slice. In our experiments, the resulting aspects contain between 1 and 53 tuples. Each tuple includes the unique slice identifier, and various resource utilization statistics. We select only the upstream and downstream network bandwidth utilization and the CPU utilization to avoid the curse of dimensionality [9] and speed up the evaluation process.

To better control and understand the data used in our experiments, we classify the 20,000 extracted aspects into ten different categories based on important aspect properties: *e.g.*, aspects where a single slice uses-up most system resources, aspects where multiple slices use moderate to high amounts of resources, etc. Because we operate on sample load information, some samples indicate overall system overload but fail to capture the actual overload activity. We ignore these events in our evaluation and focus on the remaining 3,000 events. Ignoring such events actually improves the performance of traditional similarity measures by filtering out past events that may appear similar but do not contain any abnormally behaving entities.

4.2.1 Performance of Traditional Measures

The goal of the first experiment is to evaluate the applicability of traditional set-similarity measures for ranking aspects in the order required by monitoring applications. We only evaluate the measures that take the distance between individual elements in the set into account (*i.e.*, Hausdorff distance, MAC, EMD, and the naïve Cartesian product). For the underlying distance function, we compare Euclidean distance, IDF Similarity (transformed into a distance through the hyperbolic secant function) and Histogram distance[3]. The goal of this experiment is only to identify and eliminate the measures that are clearly unsuitable. For the other measures, we refine our evaluation further in the following section.

We evaluate the measures using the following technique (based on [24]): we hand pick 20 triples of the form $(\mathtt{Q}, \mathtt{T}_1, \mathtt{T}_2)$. For each triple, we use our desired properties to determine whether $\mathtt{T}_1$ or $\mathtt{T}_2$ should be closer to $\mathtt{Q}$. We only pick clear-cut cases, similar to the examples in Figure 3. We then compute the distance between $\mathtt{Q}$ and $\mathtt{T}_1$ and $\mathtt{Q}$ and $\mathtt{T}_2$ using each measure. We count how many times the computed distances agree with our manual order.

The results, presented in Table 1, show that Cartesian product is clearly inappropriate as it agrees with our manual ranking only 25% of the time or less. The problem is that many aspects are relatively large (leading to large values in the denominator), but they contain mostly similar, ordinary values (causing the numerator to be small). This causes Cartesian product distances to all be close to zero.

The Hausdorff distance, MAC, and EMD agree with our manual ranking at least 50% of the time when using the Euclidean distance for comparing tuples. We thus investigate these three measures further below.

Using IDF Similarity instead of Euclidean distance does not improve the quality of the results as expected, except for EMD. One observed reason why IDF Similarity does not help is that, for numeric values, the similarity between two tuples drops too quickly as their attribute values become

[3]Per each numeric attribute in aspect, we constructed a histogram of 5 buckets over the values greater than the ValueFilter threshold.

		CART	Hausdorff	MAC	EMD
IDF similarity	no key factor	10%(2)	55%(11)	45%(9)	80%(16)
	with key factor	20%(4)	40%(8)	40%(8)	55%(11)
Euclidean distance	no key factor	25%(5)	60%(12)	50%(10)	70%(14)
	with key factor	30%(6)	70%(14)	60%(12)	55%(11)
Histogram distance	no key factor	40%(8)	55%(11)	60%(12)	75%(15)
	with key factor	45%(9)	65%(13)	70%(14)	75%(15)

Table 1: Initial evaluation of traditional measures (and simple variants) against a selected set of 20 test-cases. For each measure, we show the fraction of time (and number of test-cases) where the measure agrees with our manual ranking. For MAC, the table shows the best-case parameter setting ($l = 1, m = 1, q = 0$). The results show that Cartesian product and IDF Similarity are not well suited for our application domain.

different. To a human observer, the values are still similar, while the IDF Similarity is already almost zero. Because, in a monitoring environment, a significant fraction of data is numerical, the Euclidean distance is thus more appropriate as a measure of distance between tuples. We further investigated outstanding performance of EMD with IDF Similarity and found that it was just a coincidence. IDF Similarity between two values is weighted by the infrequency of querying value. For example, $(0, 0)$ is less similar than $(5, 0)$ because 5 is less frequent than 0 in our data set. Thus, if a tuple contains infrequent values, having close corresponding values would result in greater similarity. It is desirable property of IDF Similarity for tuples whose values are highly abnormal thus interesting. However, this property is also applied to the tuples of relatively high values but they are considered as normal. We observed that it is not the similarity between abnormal tuples but similarity between such normal tuples with relatively high values that led the correct results. In fact, the similarity between abnormal tuples were often reversed in our tests because of the sensitivity of IDF Similarity discussed before. Thus, even though the results were correct, they were not from our expected behavior but from such corresponding tuples by chance.

One hypothesis why the results for the Hausdorff and MAC measures are not better than 60% is that these measures do not put any special weight on tuples that correspond to the same entity. Of course, the key attribute is taken into account in the distance computation, but it is not given any special weight. We thus try a simple variant of these traditional distance measures. Instead of embedding difference of key attribute in distance, we multiply the distance of other attributes excluding the key attribute by a key factor of $(1.0 - e^{-distance})$. The results, also presented in Table 1, show that with the Euclidean distance, the key factor improves (or at least does not hurt) performance for all measures.

Lastly, we also showed the results of histogram distance in Table 1. Overall, it shows comparable performance to Euclidean distance and works well with all measures unlike IDF similarity and even with key factor.

In the next experiment, we investigate these different measures more deeply, but we only study the candidates that showed reasonable performance. We thus rule out the Cartesian product and the IDF similarity.

4.2.2 Benefits of ValueFilter

In this section and the next one, we demonstrate the performance of ADM by separately demonstrating the impact of its different components and an extension: ValueFilter, TupleWeight, AttributeWeight and KeyFilter(the latter three are always applied along with ValueFilter).

As discussed in Section 3.2.2, the first key idea of ADM is to filter out all entities in an aspect that do not have any abnormal attribute values (ValueFilter). Indeed, we noticed that the vast majority of the data in practice is normal and this normal data generates sheer noise in the distance computations. By filtering out the normal uninteresting data, we should get more accurate distances.

To demonstrate the effectiveness of the ValueFilter, we use the following approach. We first randomly pick 20 aspects from each one of our 10 categories. This gives us a list of 200 past aspects with different interesting properties. We then pick 2 query aspects (different from the 200 past aspects) from each one of our categories, for a total of 20 query aspects. For each query, we ensure that at least 6 past aspects come from the same server and are thus likely to be highly similar. For the ValueFilter, we set a reasonable threshold for all attributes, ensuring that 10% of all values pass the filter. We apply the filter to all aspects before measuring their distances.

For each query, we perform the following experiment. First, we use each measure (and each variant of each measure) to sort the 200 aspects in ascending order of distance to the query aspect. We then select the top-3 results from each list and union them together to produce a single list. All three authors separately manually rank the resulting list to create the "ideally" ranked list. The list is ranked in a manner that follows our desired properties. Finally, we compute the distance between the ideal ranking and the ranking produced by each measure. To compute this distance, we use the distance function proposed by Nie *et al.* [36]. In the rest of the paper, we use this as a standard metric to quantify the quality of each measure. With this metric, lower values indicate that a measure is better as it means that it produced a ranking closer to the ideal one.

We perform the above experiment on the traditional set-similarity measures based on Euclidean and Histogram distance (with and without the key factor) and also on the same measures but after filtering out all normal values (*i.e.*, with ValueFilter). Figure 4 shows the results. The bars show the overall average distance for all queries and all three manually ranked lists. MAC1 is regular MAC with parameters from the previous experiment and MAC2 is with $l = 0, m = 1, q = 0$ as illustrated in Section 3.2.2.3. NMAC is normalized MAC but the all TupleWeights are set to 1 so that all tuples are equally handled and this is assumed when NMAC is not used with TupleWeight. Clearly, for each measure, filtering out normal values significantly improves the quality of the final ranking. In this experiment, the standard deviations are between 0.10 and 0.28. These

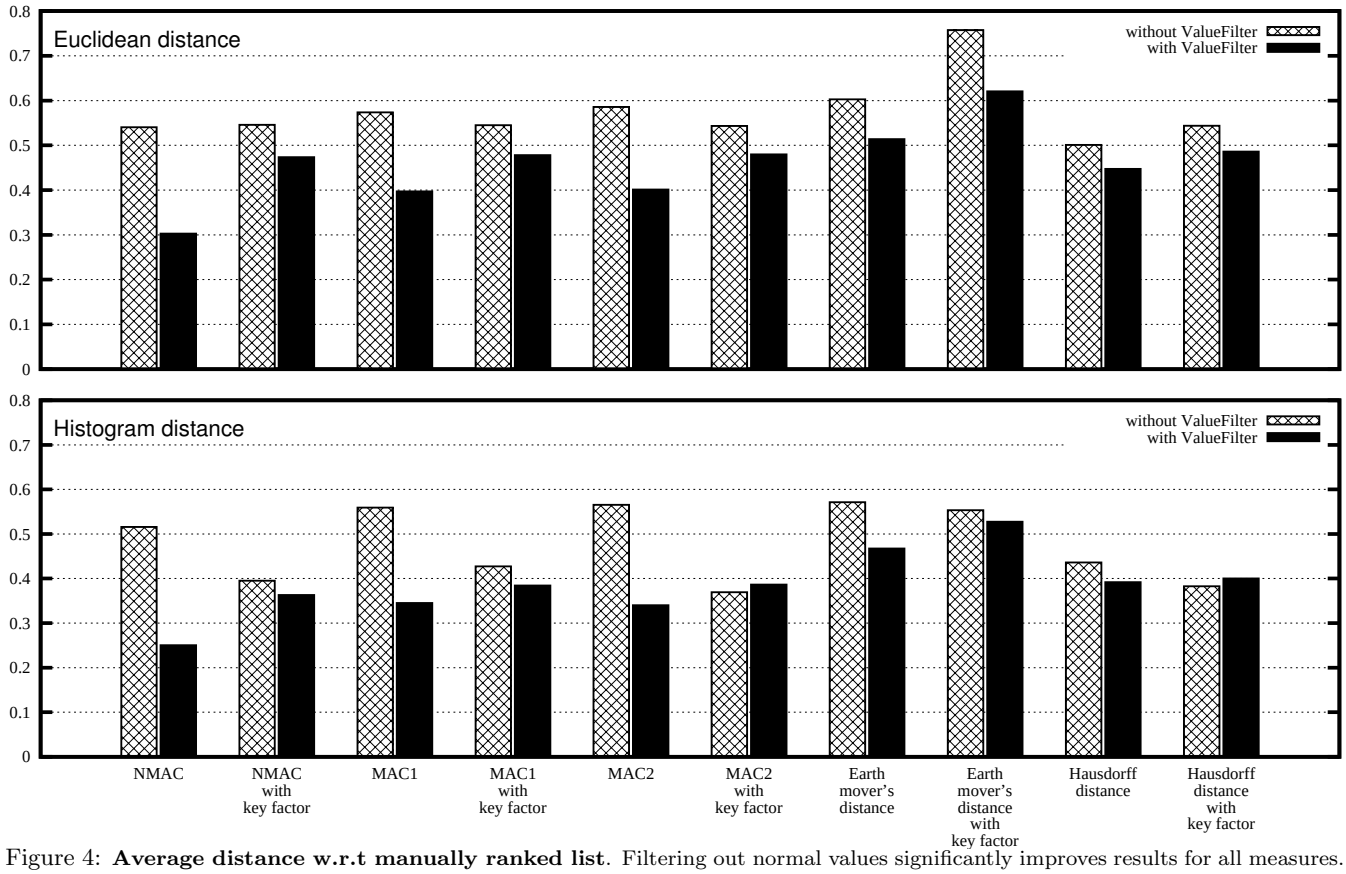

Figure 4: **Average distance w.r.t manually ranked list**. Filtering out normal values significantly improves results for all measures. Histogram distance outperforms Euclidean distance and key factor does not improve quality of the results when it is used with ValueFilter.

high variances indicate that none of the metrics precisely captures our desired priorities.

In Figure 4, one can easily notice that the Histogram distance performs better than Euclidean distance because the latter often fails to catch the closeness in abnormality when one of the comparing value is extreme.

Key factor introduced in previous section improves the results without ValueFilter. The exceptions are EMD, Hausdorff and NMAC distance with Euclidean distance. One observed reason is that a tuple which contains abnormal values with non-matching key introduces more significant cost in distance computation than its non-key factor counterpart. Recall that EMD and NMAC, in our setting, the distance between two aspects are divided by the number of mappings. By the key factor, the distances of non-matching abnormal tuples get several order of magnitude higher value than matching tuples according to their distances. In the end, the presence of such tuple results in inversion in final ranking. The same argument holds for Hausdorff distance; the minimum distance is likely to be dominated by those non-matching abnormal tuples where current minimum distance is between matching tuples. Histogram distance is less susceptible to this problem because, at least, histogram distance can handle the distance between two abnormal values where Euclidean distance may end up with a huge distance.

Interestingly, the key factor impairs results when it combined with ValueFilter because now non-matching abnormal tuples which barely pass through the ValueFilter take significant portion in distance unless the comparing aspect have tuples close to them. (Recall that the key factor would reduce the distance at least by half). However, we can't assume that there always exists such a tuple. Without ValueFilter, such a tuple can be mapped to the closest normal tuple and only incurs small costs. The results show that simply incorporating tuple keys in distance computation suffices to ensure that similar aspects with similar entities are ranked higher. Because attribute-values are normalized to fall in the range [0,1], the impact of a matching key is significant: if the key matches for two tuples, the distance for the key attribute is 0; and it is 1 otherwise. Given that we are now comparing only small sets where all tuples are potentially interesting, adding the key factor biases the distance too much for aspects with the same entities, even when their values differ.

Now let's turn to the effectiveness of ValueFilter. ValueFilter performs better than vanilla measures in all combinations except MAC2 and Hausdorff distance based on Histogram distance with key factor. Because of key factor and Histogram distance, non-matching less abnormal tuple introduce far distance between two aspects if there are only highly abnormal tuples in comparing aspects. Without filtering, the impact of such mapping is attenuated by either normalized by large number of cheap mappings or having the closest normal tuples. However, in MAC2 and Hausdorff, unlike Normalized MAC or EMD, there is no such mechanism to reduce the impact of such non-interesting mapping.

Finally, the figure shows that NMAC based on Histogram distance with ValueFilter clearly outperforms the other measures. Thus, we further investigate techniques to improve NMAC in following sections.

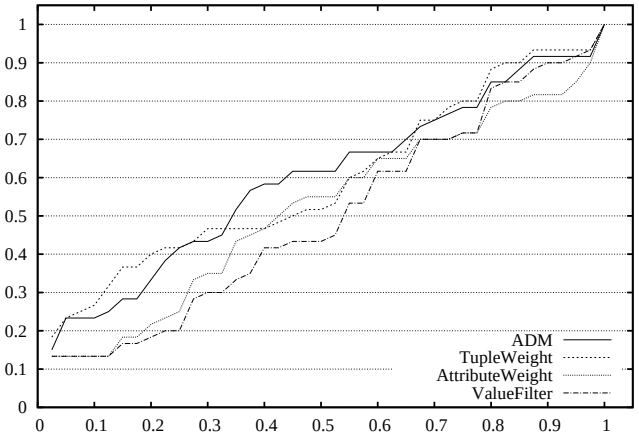

Figure 5: **CDF of distance w.r.t. manually ranked list**. TupleWeight greatly improves the results. ADM shows the best average performance.

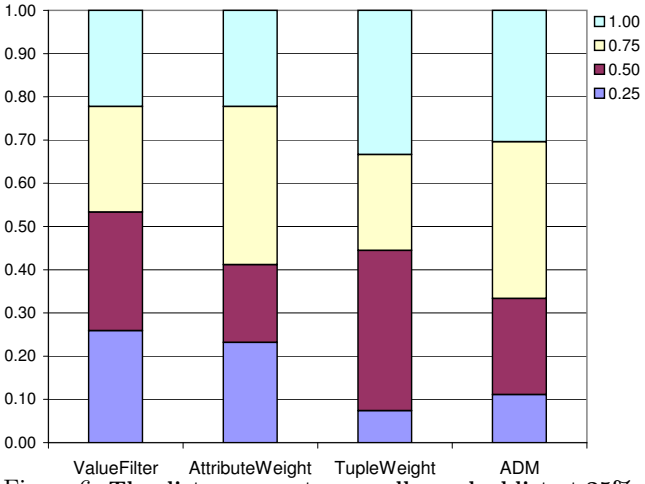

Figure 6: **The distance w.r.t manually ranked list at 25%, 50%, and 75% percentile**. Each weight technique improves the results compared to ADM with only ValueFilter.

To sum up, we evaluated and investigated the effectiveness of key factor as well as ValueFilter. The intuition behind key factor is interesting but the benefit of ValueFilter is too attractive. One may argue that our key factor setting is simply too strong but it is too cumbersome to tune it whenever you deploy a new query. Thus, we rule out key factor in following experiments. We also rule out Euclidean distance because histogram distance outperforms in every case. All experiment data is based on histogram distance.

4.2.3 *Benefits of AttributeWeight and TupleWeight*

In this section, we demonstrate the benefits of the other components of ADM. ADM puts more weight on both abnormal attribute-values(AttributeWeight) and tuples containing such attribute values (TupleWeight) when computing the distance between aspects.

To evaluate the performance of AttributeWeight, TupleWeight and ADM, we run the same experiments over the same aspects pool and query set in previous section with all of these variants based on both NMAC and its base measure MAC2. Again, all of the authors manually ranked the union of top-3 results from all variants and computed distance. In this and following sections, we analyze each

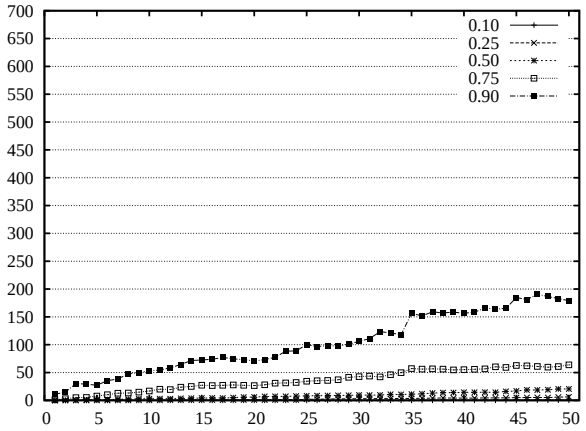

Figure 7: **Average rank displacement of top-k results produced by ADM with only ValueFilter w.r.t ADM**. For top-10 results, the result is differ by about two pages for more than 25% of all queries when 10 entries per page assumed.

variation in detail based on this result. As in previous section, Y-axis of all figures are the distance to the manually ranked "ideal" list. However, the numbers can't be directly compared with those in previous section because they are generated by different data set. The results analyzed in this section and Section 4.2.4 are based on NMAC. We show the difference between NMAC and its base measure MAC2 in Section 4.2.5.

Figure 5 shows the CDF of distance of list generated by each technique with respect to manually ranked "ideal" list. The distances to the manually ranked "ideal" lists at 25%, 50% and 75% percentile from each technique is shown in Figure 6. We set ValueFilter without no weights as a baseline. TupleWeight plays significant role in enhancing the quality because it directly participates in NMAC computation. One can easily notice that distribution is always on the left side of the baseline. AttributeWeight also helps in improving quality but sometimes ends up with completely different results where TupleWeight must be considered for correct ranking. By applying all three techniques together, ADM shows the best average behavior and the distribution is either close or better than TupleWeight.

To measure the improvement in performance more accurately and elaborate the difference in ADM and the baseline, NMAC with ValueFilter, we perform a larger-scale experiment. We choose 167 query aspects from our categories (we intended to select 20 from each category, but came up with 167 due to discrepancies in the sizes of the clusters). For each one of the 167 queries, we extract the top 50 most similar past aspects (which seems likely to be way above the maximum number of results an administrator will examine) out of the complete list of 3,000 past aspects.

Figure 7 shows the average rank displacement in the top-k results produced when using NMAC with ValueFilter compared with ADM. The y-axis shows the value of the standard displacement metric [24], where values closer to zero indicate that two ranks are more similar. Higher values indicate larger differences between ranked lists. The figure shows the displacement distance for different percentiles. Interestingly, for about half the queries, there is little difference between the two ranked lists. However, for the other 50% of the queries, the difference is significant, with the top 25% of queries resulting in widely different ranked lists.

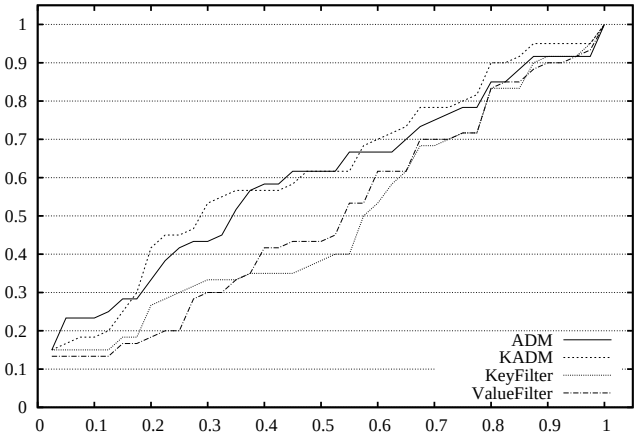

Figure 8: **CDF of distance w.r.t manually ranked list**. KeyFilter improves results where the identity of abnormal entity matters. When it is used with AttributeWeight and TupleWeight, KADM shows the better average performance than ADM.

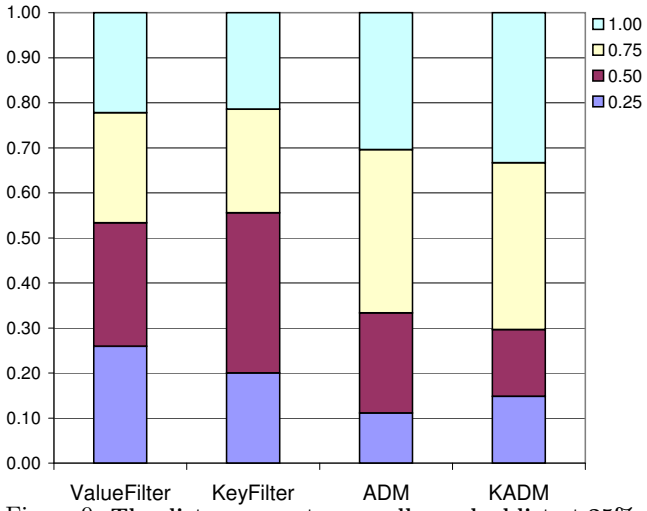

Figure 9: **The distance w.r.t manually ranked list at 25%, 50%, and 75% percentile**. KeyFilter alone doesn't improve the results compared to ADM with only ValueFilter. However, with AttributeWeight and TupleWeight technique, KADM achieves the best average behavior in our data set.

4.2.4 *Extended ADM with KeyFilter*

In this section, we evaluate the effectiveness of KeyFilter extension to ADM and compare with regular ADM. The extended ADM(KADM) strictly enforces that past aspects containing the same abnormal entities appear before aspects containing different abnormal entities, even if in the former case, the other attribute-values are less similar. Additionally, because the presence of such entities significantly increases the important of a past aspect, KADM ignores all other entities when computing the distance between such an aspect and the query aspect.

Figure 8 shows the CDF of distances of list produced by each technique with respect to manually ranked "ideal" list. The distances to the manually ranked "ideal" lists at 25%, 50% and 75% percentile from each technique is shown in Figure 9. The distance appeared in the figures are compatible to the numbers in previous section because both are based on the same data set.

KeyFilter alone does not significantly enhance the quality of results as both distributions of filters intertwine. However, with AttributeWeight and TupleWeight altogether, KADM outperforms ADM on average in our data set where identity of abnormal entity is important.

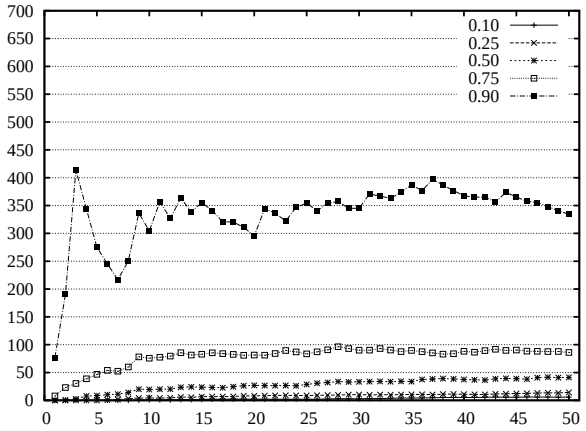

Figure 10: **Average rank displacement of top-k results produced by ADM w.r.t. KADM**. For more than 50% queries, the result is widely different.

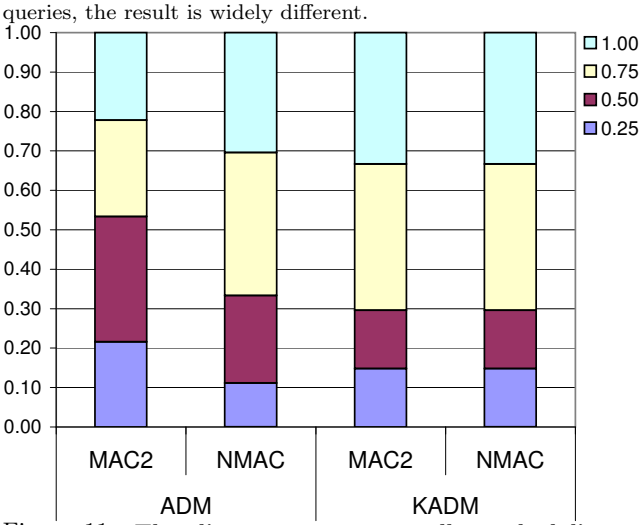

Figure 11: **The distance w.r.t manually ranked list at 25%, 50%, and 75% percentile**. Normalization improves results when it is used in ADM. However, it produces identical results to non-normalized MAC when it is used in KADM.

Figure 10 shows the average rank displacement of top-k results of ADM with respect to KADM. Because KADM strictly prioritizes past aspects which have identical abnormal entities to querying aspect, the results are radically different from the results of ADM in majority of queries.

4.2.5 *Effectiveness of normalization in MAC*

As a last evaluation for ADM, we further investigate the effect of normalization in MAC. We used NMAC and its base measure MAC2 to compute ADM and KADM . Figure 11 shows that the distance to the ideal list at 25%, 50% and 75% percentile. For ADM, NMAC clearly outperforms MAC2; the distance at each percentile is lower than MAC2. However, they produce identical results when they are used in KADM. The result is partly because of the MAC and partly because of KeyFilter. MAC always consider edge cover but the cardinality of edge cover is bounded by the number of abnormal tuples in querying aspects, N. Because KeyFilter filters out tuples from a past aspect based on the key values of abnormal tuples in querying aspect, the cardi-

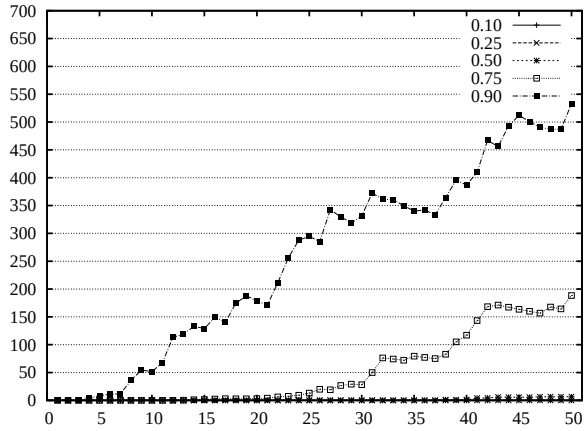

Figure 12: **Average rank displacement of top-k results produced by ADM with MAC2 w.r.t. ADM.** As soon as clear cases running out, the results quickly become different.

nality of filtered aspect is at most N. As long as the filtered past aspect is not empty, the edge cover between two aspects always exactly contains N edges. In other words, the normalized term is always constant, sum of TupleWeights of querying aspect, if the KeyFiltered aspect is not empty. In this case, it is identical to MAC2. Thus, the result shown in Figure 11 just tell us that there are sufficiently many past aspects which have at least one abnormal entity having the same key value.

In large scale, we observed that the average top-k displacement for KADM is 0 until k went beyond 100 which is less meaningful in practice. Instead, we show the displacement for ADM in Figure 12. There exist some clear cases where both measures produce identical ranking. However, as soon as they are running out, the two measures start to produce widely different ranking and the displacement quickly diverges.

4.3 CDM

In this section, we evaluate the impact of using contexts composed of different aspects when identifying similar past events. We show that for the same set of queries, if the user defines different contexts, the system returns a different set of top-k most similar past events.

In this experiment, for each one of our 3,000 events, we construct a second aspect using CoTop data. This second aspect includes the number of processes running in each slice and the overall memory utilization for that slice. A large number of processes increases the probability that the slice will use significant amounts of system resources. A high memory utilization by one slice may negatively affect other slices running on the same server.

We run 148 queries from the previous experiment over all 3,000 past events. The number of queries is lower than 167 because some of the queries do not have any abnormal values in their new aspect. For all queries, we compare the overlap in the past events returned when using either the first *or* the second aspect as the context of the event. Figure 13 shows the average, minimum, maximum overlap and standard deviation measured for all queries. Even when returning as many as 50 most similar past events, the average overlap is less than 4 events. The context of an event is thus effective at defining what constitutes the most relevant past events to the user.

Top-k	10	20	50
Average overlap	0.5512	1.2910	3.9918
Minimum overlap	0	0	0
Maximum overlap	10	19	35
Standard deviation	1.2316	2.3861	5.4086

Figure 13: Overlap in the top-k most similar past event returned for the same newly detected event, but for different contexts. Each contexts comprised a single different aspect. Event contexts significantly impact the set of most similar past events.

Top-k	10	20	50
Average overlap	5.8142	11.8412	29.7364
Minimum overlap	0	0	0
Maximum overlap	10	20	50
Standard deviation	2.9897	5.5898	14.7340

Figure 14: Overlap in the set of top-k most similar past event returned for the same newly detected event, but for different contexts. The contexts have two common aspects and one different aspect. The overlap is higher when event contexts are more similar, but even one different aspect frequently causes different past events to be returned.

To further study the impact of event contexts, we repeat the same experiment, but add two more aspects to our event contexts. One aspect contains the day of the week and the time of day for when the event occurred. The second aspect contains the domain name of the overloaded server. We add these same two new aspects to both contexts. Hence, this time, each context has three aspects, but two of these aspects are shared between the contexts. Figure 14 shows the results. The most noticeable result is the increased average overlap between the top-k results. Such an increased overlap is expected as the event contexts also overlap. Interestingly, in many cases, the one distinct aspect is enough to significantly change the past events that qualify as most similar.

In summary, using event contexts effectively discriminates between relevant and irrelevant past events from a user's perspective. CDM, based on ADM, produces past events following an order that meets the requirements of monitoring applications; and ADM clearly outperforms existing set-similarity measures when ranking past events.

5. RELATED WORK

There exists a significant body of work on high-performance domain specific monitoring engines [29, 33, 38, 40] and general purpose data stream management systems [1, 2, 13, 21, 34]. Both types of tools provide effective, near real-time event detection, but do not support the subsequent investigation of newly detected events. Moirae is a general purpose continuous monitoring system that strives to support all aspects of a monitoring application from event detection to root cause analysis. Moirae achieves this goal by integrating the processing of streaming data with that of stream data archives [8]. The ability to automatically identify similar past events is an important component of Moirae. In previous work, we proposed an architecture that incorporates such event matching functionality into the overall Moirae design [8]. In this paper, we investigated the

specific problem of devising measures to effectively compute event similarity using their context information.

Similarity queries (*e.g.*, similarity searching [23, 41], similarity joins [4], top-k queries [10, 12, 14, 16, 22, 31], keyword searching over relational databases [26, 32], and ranking of query results [3]), have been extensively investigated in the database research area. For these similarity queries, query results heavily depend on the similarity measures applied. Various similarity measures have been proposed. The important representatives include similarities based on item frequency and distance function of objects. The Cosine Similarity metric with TF-IDF [7] has been successfully used in areas such as document searching and ranking of database query results [3]. This measure is mostly appropriate to express the similarity of data objects with only categorical attributes. The Jaccard Similarity of two sets has been successfully used for similarity joins in Data Cleaning applications [5, 15]. Several distance functions such as Minkowsky Distance, Quadratic Distance and so on [43] are measures of image and multimedia similarity searching based on vector metric space model. These distance functions are suitable to process numerical attribute values. The above measures, however, can not directly be used to compare objects like database tuples with both categorical and numerical attributes. For the latter case, Agrawal *et al.*, [3] proposed an extended measure, the IDF Similarity, that works for both numerical and categorical attributes. For distance function of vector space model, Wilson and Martinez proposed Heterogeneous Euclidean-Overlap metric(HEOM) [42] which extends the numeric value distance functions to the heterogeneous attribute domain.

In our case, the similarity measure is more complex and it is hierarchical with four levels, i.e., attribute value distances, tuple distances, aspect distances, and similarity of entire event contexts. Techniques for computing the distance between sets of multidimensional objects are related to our problem of comparing aspects. The Hausdorff distance is an efficient metric for comparing two sets in image matching and pattern recognition [27]. The Earth Mover's distance(EMD) is also successfully applied as a metric to compare multidimensional distributions for content-based image retrieval in large multimedia databases [39]. To reduce the computational complexity of EMD, Assent *et al.*, proposed an index-supported multi-step algorithm [6]. The Match And Compare(MAC) distance [28] is another numerical measure to compute the distance between multisets for set-valued queries. For exploiting hierarchical domain structure, the Generalized Cosine-Similarity Measure(GCSM) [24] was introduced to compute set-similarity by using induced trees. In this paper, we explored the applicability of these measures to the problem of comparing event contexts. Our own measures, ADM and CDM, build on these existing techniques, but use, integrate, and extend them in a manner that better supports the requirements of monitoring applications.

6. CONCLUSION

Continuous monitoring plays a key role in facilitating the administration of large systems such as clusters of servers, computer networks, Grids, etc. In these applications, administrators need to receive timely alerts when problems or abnormal events occur, but they also need support for understanding and diagnosing these events. One approach for helping administrators is to show them, with each newly detected event, a small set of top-k most similar past events. Of course, event similarity depends on what the administrator defines as the interesting context of these events.

In this paper, we proposed two new measures for comparing event contexts in a monitoring system. Our first measure, ADM, computes the distance between two relations, such that relations containing the same entities with similarly abnormal attribute-values are ranked closest, followed by relations with different entities containing abnormal attribute values, followed by all other relations. This measure is particularly well suited for monitoring applications, where detecting and inspecting abnormal behaviors is a key component of understanding and diagnostic problems. Our second measure, CDM, builds on ADM and enables the comparison of contexts composed of multiple different aspects.

Using a real workload from the computer-system monitoring domain, we showed that ADM outperforms existing techniques and produces a ranked list of top-k most similar past events that matches more accurately the desired ranking in monitoring applications. We also showed that CDM is effective at exploiting context information to find those past events most relevant to a user: *i.e.*, specifying different context results in seeing drastically different lists of similar past events.

There are a few interesting future works. First, we can do similarity matching on key attribute ranther than considering only exact matches. For example, [24] proposed to use static semantic hierarchy in their set similarity measures. If we can't find any interesting past aspects for the current aspect, we can use this technique to fetch the past aspects where tuples with similar keys were showing similar abnormalities to current ones. This approach intuitively improves the results especially when the monitored object is new in the system thus no history has been accumulated yet. Second, we observed that several aspects in very close patterns appear in the results together. However, such results just wasting invaluable top k slots. Instead, like modern search engine, clustering aspects in common pattern and presents them as the representative aspect as well as the size of cluster would greatly improve the comprehensibility of results. Efficient query processing technique as well as index structure for our CDM is necessity to achieve near-realtime perfomance. Finally, evaluating applicability of CDM in various application domains would be interesting.

Overall, we view this work as an important component in building systems that support, in an integrated fashion, all aspects of monitoring applications from near-real time event detection to root cause problem analysis.

7. ACKNOWLEDGMENTS

We thank the CoMon team for their help with using the PlanetLab monitoring data. We thank James Lee and the database group at the University of Washington for helpful discussions. This material is based upon work supported by Cisco Systems Inc. through the Cisco University Research Program.

8. REFERENCES

[1] Abadi et. al. Aurora: A new model and architecture for data stream management. *VLDB Journal*, 12(2), Sept. 2003.

[2] Abadi et. al. The design of the Borealis stream processing engine. In *Proc. of the Second CIDR Conf.*, Jan. 2005.

[3] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. In *Proc. of the First CIDR Conf.*, Jan. 2003.

[4] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *Proc. of the 32nd VLDB Conf.*, Sept. 2006.

[5] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *Proc. of the 32nd VLDB Conf.*, Sept. 2006.

[6] I. Assent, A. Wenning, and T. Seidl. Approximation techniques for indexing the earth mover's distance in multimedia databases. In *Proc. of the 22nd ICDE Conf.*, Apr. 2006.

[7] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[8] M. Balazinska, Y. Kwon, N. Kuchta, and D. Lee. Moirae: History-enhanced monitoring. In *Proc. of the Third CIDR Conf.*, Jan. 2007.

[9] R. E. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton Universsity Press, 1961.

[10] N. Bruno, S. Chaudhuri, and L. Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM TODS*, 27(2), 2002.

[11] N. Bruno, L. Gravano, and A. Marian. Evaluating top-k queries over web-accessible databases. In *Proc. of the 18th International Conference on Data Engineering (ICDE)*, Feb. 2002.

[12] M. J. Carey and D. Kossmann. On saying "enough already!" in SQL. In *Proc. of the 1997 SIGMOD Conf.*, May 1997.

[13] Chandrasekaran et al. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *Proc. of the First CIDR Conf.*, Jan. 2003.

[14] K. Chang and S. Hwang. Minimal probing: supporting expensive predicates for top-k queries. In *Proc. of the 2002 SIGMOD Conf.*, June 2002.

[15] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *Proc. of the 22nd ICDE Conf.*, Apr. 2006.

[16] S. Chaudhuri and L. Gravano. Evaluating top-k selection queries. In *Proc. of the 25th VLDB Conf.*, Sept. 1999.

[17] Cisco IOS NetFlow. `http://www.cisco.com/go/netflow`.

[18] CoMon: A Monitoring Infrastructure for PlanetLab. `http://comon.cs.princeton.edu/`.

[19] CoTop: A Slice-Based Top for PlanetLab. `http://http://codeen.cs.princeton.edu/cotop/`.

[20] L. Cottrell. Network monitoring tools. Stanford Linear Accelerator Center. `http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html`.

[21] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *Proc. of the 2003 SIGMOD Conf.*, June 2003.

[22] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4), 2003.

[23] C. Faloustos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. of the 1994 SIGMOD Conf.*, May 1994.

[24] P. Ganesan, H. Garcia-Molina, and J. Widom. Exploiting hierarchical domain structure to compute similarity. *ACM Transactions on Information Systems*, 21(1), 2003.

[25] R. Geambasu, T. Bragin, J. Jung, and M. Balazinska. On-demand view materialization and indexing for network forensic analysis. In *Proc. of the NetDB'07 Workshop*, Apr. 2007.

[26] R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H. Garcia-Molina. Proximity search in databases. In *Proc. of the 24th VLDB Conf.*, Aug. 1998.

[27] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rocklidge. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993.

[28] Y. E. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query-answers. In *Proc. of the 25th VLDB Conf.*, Sept. 1999.

[29] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *Proc. of the ACM SIGCOMM 2004 Conf.*, Aug. 2004.

[30] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *Proc. of the ACM SIGCOMM 2005 Conf.*, Aug. 2005.

[31] C. Li, K. C.-C. Chang, I. F. Ilyas, and S. Song. RankSQL: query algebra and optimization for relational top-k queries. In *Proc. of the 2005 SIGMOD Conf.*, June 2005.

[32] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *Proc. of the 2006 SIGMOD Conf.*, June 2006.

[33] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30(7), July 2004.

[34] Motwani et. al. Query processing, approximation, and resource management in a data stream management system. In *Proc. of the First CIDR Conf.*, Jan. 2003.

[35] D. Narayanan, A. Donnelly, R. Mortier, and A. Rowstron. Delay aware querying with seaweed. In *Proc. of the 32nd VLDB Conf.*, Sept. 2006.

[36] Z. Nie, Y. Zhang, J.-R. Wen, and W.-Y. Ma. Object-level ranking: bringing order to web objects. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 567–574, New York, NY, USA, 2005. ACM Press.

[37] PlanetLab: An open platform for developing, deploying, and accessing planetary-scale services. `http://www.planet-lab.org/`.

[38] R. V. Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM TOCS*, 21(2), 2003.

[39] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover"s distance as a metric for image retrieval. Technical report, Stanford, CA, USA, 1998.

[40] S. T. Teoh, S. Ranjan, A. Nucci, and C.-N. Chuah. BGP Eye: a new visualization tool for real-time detection and analysis of BGP anomalies. In *Proc. of VizSEC '06*, 2006.

[41] K. Vu, K. A. Hua, H. Cheng, and S.-D. Lang. A non-linear dimensionality-reduction technique for fast similarity search in large databases. In *Proc. of the 2006 SIGMOD Conf.*, June 2006.

[42] D. R. Wilson and T. R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6(1):1–34, 1997.

[43] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search The Metric Space Approach*. Springer Science+Business Media, Inc., New York, NY, USA, 2006.