Clustering Analysis for the Management of Self-monitoring Device Networks

Andres Quiroz and Manish Parashar
CAIP Center, Rutgers University
The Applied Software Systems Laboratory
Piscataway, NJ, USA
{aquirozh, parashar}@caip.rutgers.edu

Nathan Gnanasambandam and Naveen Sharma
Xerox Research
Webster, NY, USA
first.lastname@xerox.com

Abstract— **The increasing computing and communication capabilities of multi-function devices (MFDs) have enabled networks of such devices to provide value-added services. This has placed stringent QoS requirements on the operations of these device networks. This paper investigates how the computational capabilities of the devices in the network can be harnessed to achieve self-monitoring and QoS management. Specifically, the paper investigates the application of clustering analysis for detecting anomalies and trends in events generated during device operation, and presents a novel decentralized cluster and anomaly detection algorithm. The paper also describes how the algorithm can be implemented within a device overlay network, and demonstrates its performance and utility using simulated as well as real workloads.**

I. Introduction

With rapid increases in the computing and communication capabilities of multi-function devices (MFDs) in an office environment, networks of such devices are emerging as providers of value-added services, such as the automation of business processes and temporary distributed storage. To support grid-wide services on these device networks, quality-of-service (QoS) metrics such as performance, and reliability need to be maintained at high levels. However, the scale and dynamics of the envisioned networks create significant management challenges, and require autonomic monitoring to ensure their robust operation. Fortunately, devices' additional computing resources can be harnessed and exploited to achieve self-monitoring and management behaviors.

A self-monitoring system is able to observe and analyze system state and behavior, to discover anomalies or violations, and to notify autonomic or human administrators in a timely manner so that appropriate management actions can be effectively applied. Ideally, rather than detecting an event after it occurs (e.g., failures or malicious behavior), a self-monitoring system should focus on the detection of indicators or patterns that precede and point to the event (e.g., possible failures). Typically, realizing such proactive self-monitoring behavior involves data collection from the target system and offline analysis of this data to produce models that can then be used for runtime self-management [1], [2], [3], [4]. The derivation of these models often involves sophisticated processing and data mining techniques. As a result, they can be impractical for dynamic, online monitoring. In contrast, this paper shows how a relatively simple clustering technique can be applied for the online, in-network analysis of monitored data to detect and produce notifications about system trends and anomalies.

In particular, the motivation for this work stems from the management complexity of a large fleet of MFDs that can provide physical services, such as printing and scanning, as well as communication and software services, such as fax, e-mail, and file storage and transformation. Monitoring and online data analysis in such networks is important for characterizing usage patterns and load distribution, and answering questions about, for example, the relative bias of users towards devices due to their location, performance, and the services offered. Such understanding would help track the relative importance of devices and services, delineate possible over/under or malicious utilization of the network, and manage user QoS. Furthermore, implementing the analysis technique in a decentralized and in-network fashion (using network resources and minimal extraneous information) ensures computational tractability and acceptable response times. Eventually, knowledge about user-device interactions in the network derived through data analysis can enable on-the-fly optimization of task scheduling and service provisioning, and the isolation of malicious or anomalous users and system nodes.

Working toward achieving the goals outlined above, the main contribution of this paper (described in Section III) is the formulation and validation of a decentralized data analysis algorithm that applies density-based clustering techniques [5], [6], [7]. Density-based clustering is especially advantageous when it is necessary to identify anomalies (points that should not belong to any cluster) and clusters of arbitrary size and shape. In this research, it is assumed that the device network is organized as a structured overlay network, such as Chord [8], and that devices in the system can represent their behavior and operational status using known attributes and periodically report this information in the form of events. The event attributes are used to construct a multidimensional coordinate space, which is then used to measure the similarity of events. Devices that behave in a similar fashion can then be identified as clusters of status events, while devices with abnormal behavior will produce isolated events. The clustering algorithm requires minimal computation at processing nodes, which makes it suitable for online execution.

Although the algorithm design does not imply the use of a particular overlay and messaging infrastructure, the paper also describes an application infrastructure to support its implementation. The algorithm relies on dynamically assigning events to processing nodes, while preserving the locality of these events. The application infrastructure, described in Section IV, provides message routing over a structured overlay [9] based on a locality preserving mapping [10] that fulfills this requirement, and that is efficient in terms of messaging overhead [9], [11]. An analysis of the performance of the clustering algorithm based on the use of this infrastructure appears in Section V.

Section VI then presents a two-fold evaluation of our approach. First, the effectiveness of the clustering algorithm is evaluated by measuring error rates in anomaly detection using a simulated dataset. Second, the utility of the online application of clustering in a real world scenario is demonstrated by means of a case study using real data from a device network. Here, the goal is to delineate anomalies in a multidimensional information space, which could then be utilized by policies for managing the performance of the network.

Before going into the details of the research outlined above, the next section discusses related work, both in self-monitoring, and in the clustering techniques that serve as the basis of this work. Section VII finally concludes the paper with deatils of current and future work.

II. Related Work

In general, methods for self-monitoring are based on characterizing normal system behavior and then detecting deviations from this normal behavior. A number of different approaches [1], [2], [3], [4] have been proposed and developed, most of them specialized to different application contexts. These include system-specific approaches that exploit the particular characteristics of the data acquired during monitoring and depend on offline training to develop the respective models used for runtime monitoring. Our monitoring mechanism uses data collected online to recognize and report anomalies, and, unlike the approaches described above, data analysis is carried out in-network by nodes in a decentralized manner.

Cluster recognition and analysis is a well-known problem in the field of data mining. There are several different types of clustering algorithms (see [5]), each suited to different types of applications. Because of the nature and requirements of self monitoring, this work deals with exclusive, unsupervised, partitional clustering. This type of clustering produces a single (as opposed to a hierarchical) partition of a set of points into clusters, such that each point belongs to at most one cluster, and which does not depend on an a-priori classification of points (category labels).

Although there is no single definition for a cluster, in general clusters are identified based on some definition of similarity, such that the elements in a cluster are similar to each other, whereas elements in different clusters are not similar to each other [5]. Two common ways to define similarity are based on distance and density. Among distance-based algorithms, probably the most widely employed is k-means clustering. Traditional k-means is an iterative process that calculates the coordinates of k centroids (points with minimum average distance to the points in their cluster) for a number of data points. Both k and an initial estimate for the centroids must be given as input, and the algorithm successively associates data points to their nearest centroid and recalculates centroids until they do not change significantly from one iteration to the next.

Distributed algorithms for k-means have been devised [12], [13]. These procedures partition data points uniformly among processing nodes, each of which runs the k-means procedure on their subset of points. Nodes then repeatedly exchange centroids with neighbors and rerun the algorithm until a consensus is reached. The rate of convergence, which determines the number of iterations and thus the execution time of the analysis, depends on several factors, including the number of nodes and the actual distribution of the data. If the number of iterations required is large, then the process will be slow, considering online analysis.

The algorithm presented here uses a definition of similarity that is based on density. This definition has some advantanges with respect to distance-based definitions, especially when it is necessary to identify anomalies (points that do not belong to any cluster) and clusters of arbitrary size and shape. This is because distance-based algorithms usually produce a complete partition of the data into convex sets. In this case, a point can only be identified as an anomaly by using an application-dependent maximum distance (points that have a distance to other points greater than this maximum distance will be excluded from a cluster). Furthermore, these algorithms cannot identify clusters of arbitrary (non-convex) shape [6].

A basic algorithm for density-based clustering is presented in [5], and the same principle is improved upon and used in a data-mining tool for large databases called DBSCAN [6]. These algortihms identify clusters based on comparing the point densities (number of points per unit space) around the points in the data set. An application of DBSCAN for a P2P system is presented in [14]. This is similar to the present work in that it divides data among processing nodes locally, so that nodes process points in subregions of the data space. However, there every node runs the DBSCAN algorithm locally, which is more expensive computationally than the approach presented here. Our approach is based mostly on the CLIQUE method, as presented in [7], and is, to our knowledge, the only distributed application of this kind of clustering algorithm. Furthermore, the present work does not require the use of the parameters that are required for clustering in the CLIQUE method.

The work in [15] shows an application of clustering in a distributed system, namely a large multi-agent system. In this type of system, agents need to form groups with other agents of similar characteristics. Agents are assumed to be connected initially to an arbitrary small number of other agents, and clustering is done by successively creating links with better

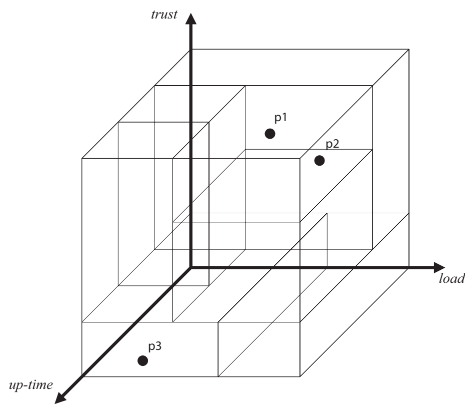

Fig. 1. Example of an information space with three attributes. Points $p1$ and $p2$ can constitute a cluster because of their relative distance, while isolated point $p3$ can be considered as an anomaly. The space is divided into regions, within which clustering analysis is performed individually.

matches provided by the views of agents' neighbors and dropping links between poor matches. The mechanism is very suitable for a multi-agent environment in which there is no pre-existing organization between agents. In contrast, in this work, we assume an existing system or virtual organization in which peers are connected in a well-known, structured (albeit dynamic) way, and exploit this organization to transfer data between nodes.

III. Decentralized Clustering Analysis

A. Conceptual Description

This paper presents a decentralized and online mechanism for the proactive self-monitoring of device networks. As mentioned in the introduction, this research targets networked systems in which individual nodes can monitor their operational status and represent it using sets of globally known attributes. Further, nodes periodically publish this status as semantic events that contain a sequence of attribute-value pairs.

Each one of these events is represented as a *point* in a multidimensional space, as shown in Figure 1. Each dimension in this space, referred to as an *information space*, corresponds to one of the event attributes, and the location of a point within the space is determined by the values for each of the attributes within the particular event. It is assumed that the range of values of each attribute is an ordered set. For each set, a distance function can be defined in order to measure the similarity between elements (i.e. similarity is inversely proportional to distance). This definition is straightforward for quantitative attributes, and can be applied to non-quantitative attributes as well with an appropriate encoding.

The notion of similarity based on distance in each dimension extends to the multidimensional information space, for which a distance function can also be defined in terms of the unidimensional distances. Conceptually, a cluster is a set of points for which mutual distances are relatively smaller than the distances to other points in the space [5]. However, the approach for cluster detection described in this paper is not

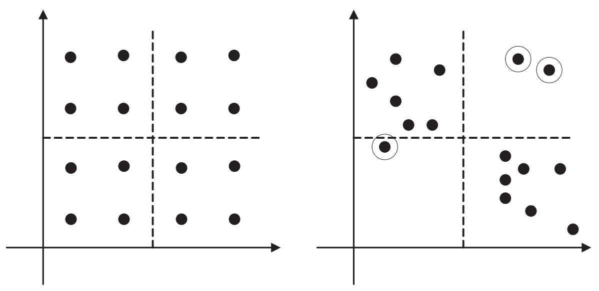

Fig. 2. (a) Uniform distribution of data points in the information space. (b) Point clustering and anomalies among regions; the circled points are potential anomalies.

based primarily on evaluating distances between points, but rather on evaluating the relative *density* of points within the information space. In this case, point similarity is directly proportional to point density.

The approach used for the evaluation of point density, and thus for the detection of clusters and anomalies, is sketched in Figure 2. The idea is to divide the information space into *regions* and to observe the number of points within each region. If the total number of points in the information space is known, then a baseline density for a uniform distribution of points can be calculated and used to estimate an expected number of points per region (Figure 2a). Clusters are recognized within a region if the region has a relatively larger point count than this expected value, as in Figure 2b. Conversely, if the point count is smaller than expected, then these points can potentially be anomalies. However, clusters may cross region boundaries, as shown in the lower left quadrant of Figure 2b, and this must be taken into account when considering potential anomalies.

The approach described above lends itself to a decentralized implementation because each region can be assigned to a particular *processing node*. Nodes can then analyze the points within their region and communicate with nodes responsible for adjoining regions in order to deal with boundary conditions. We assume that the information space is predefined and globally known; however, the assignment of regions to nodes need not be static, and can thus be adapted to the arrival and departure of processing nodes in the network, as long as a mechanism exists to map data points to the node responsible for the region in which the point is located (we describe an implementation of this mechanism in Section IV). Note the distinction between processing nodes and data points in the space, which correspond to device status events. The status of a device may be mapped to any node, depending on the region of the space where the point lies. In fact, the physical sets of machines corresponding to nodes and devices may be completely or partially distinct. Henceforth we use node to refer to a processing node and point to refer to an event (set of attribute values). The following section presents a formal description of the decentralized clustering algorithm that is based on the concepts described in this section.

```
SETUP ()
Join overlay
neighbors <- {}
For each node join or departure
  determine region R
  neighbors <-{node |node.region is neighbor of R}
END
```

Fig. 3. Algorithm for the setup phase at each node

```
ANALYSIS (size(D), T)
1  exp_count = size(R) * (size(D) / size(S))
2  clusters <- {}
3  anomalies <- {}
4  listen for period T
5  points <- {p | p received during T}
6  count = size(points)
7  If count > exp_count
8    w_cluster = width(points)*(2-density(points))
9    add (centroid(points), w_cluster) to clusters
10 Else
11   anomalies <- points
12   For each neighbor
13     If neighbor.clusters != {}
14       For each (centroid, w_cluster)
>>       in neighbor.clusters
15         For each a in anomalies st.
>>         dist(a, centroid) <= w_cluster
16           remove a from anomalies
END
```

Fig. 4. Algorithm for the data collection/analysis phase at each node

B. Decentralized Clustering Algorithm

The algorithm is based on a finite, discretized and normalized information space $S \subset \mathbb{Z}^{+^a}$, where a is the number of dimensions (attributes). The space is discrete to facilitate the application of the mapping described in Section IV, and it is normalized to enable the definition of a single unit of distance within the space. To convert an arbitrary space into the required form, each application must define, for each dimension i, a distance d_i that will correspond to a unit distance in the normalized space. The unit distance is a general property of each attribute, as determined by the application, and is the minimum difference in attribute values that should be distinguishable (i.e. a difference of less than d_i original units is considered negligible). The range of each dimension will thus be the interval $[0, x_i]$, where $x_i = (max_i - min_i)/d_i$.

The following definitions correspond to the concepts introduced in the previous section and to the main algorithm elements and operations:

- *Contiguous points*: Two points are contiguous in S iif if they differ by at most one unit in at most one dimension.
- *Region*: A region is a set $R \subseteq S$, such that any two points in R can be connected by a sequence of contiguous points in R. Each node is responsible for a unique region, denoted by node.region.
- *Neighbor*: Two regions R_1 and R_2 are neighbors iif $\exists p_1 \in R_1$ and $p_2 \in R_2$ st. p_1 and p_2 are contiguous points. Two nodes n_1 and n_2 are neighbors if n_1.region is neighbor of n_2.region.
- *Spatial density* (density(R, X)): The spatial density of a region R with respect to a point set $X \subseteq R$ is the ratio size(X)/size(R), where size() denotes the cardinality of a set. If the region is omitted for a set Y, density(Y) is the spatial density with respect to Y of the smallest region R_Y such that $Y \subseteq R_Y$.
- *Cluster*: Let D be the data set being analyzed, then a set of points C is a cluster with respect to a region R, if density$(R, C) >$ density(S, D).
- *Centroid of a set* (centroid(X)): Point with minimum average distance to each of the points in X.
- *Width of a set* (width(X)): The maximum distance of any point in X to centroid(X).

The algorithm is divided into two phases, 1) the setup phase and 2) the data collection and analysis phase. Figure 3 shows the generic (overlay independent) algorithm for the setup phase. Basically, upon joining, nodes determine their region and their neighbors based on neighboring regions in the information space, which they must repeat every time region assignment changes when nodes join or leave the network (this depends on the actual network overlay and region assignment strategy, which is described in Section IV).

Figure 4 shows the algorithm for the data collection and analysis phase. It shows how a node receives points from the network (those that map to the node's region) during a given period T and performs the clustering analysis for that period. The analysis consists of comparing the number of points received with an expected number of points (line 7). Because the expected number of points is calculated (line 1) based on the size of the region relative to the point density of the information space, given a particular data set size, this comparison actually applies the definition of a cluster given above.

If the point count is higher than the expected number of points, then these points form a cluster in the region under consideration. At this point, we assume that there is a single cluster per region. In Section III-C, we discuss the case where we consider the possibility of multiple clusters per region. When it finds a cluster, the node records its centroid and extended width ($w_{cluster}$) (lines 8-9). The extended width is calculated to find parts of clusters that may extend across boundaries by increasing the actual cluster width. Because a very dense cluster is less likely to include a point that is farther away from it than a cluster with lower density, the increase in a cluster's width is made with respect to its density. In the algorithm (line 8), the width of the cluster is increased by a percentage denoted by $1 - \text{density}(points)$. Notice that this definition ensures that the extended width is never more than twice the original width.

Conversely, when the point count at a particular node is not as large as expected, the points received in the corresponding region are potential anomalies. Before they are labeled as such, however, the boundary conditions must be checked. To do this, the local node queries its neighbors for clusters detected in their regions (line 13). If any of the potential anomalies are within the extended width of any of these clusters, then they are not anomalies and are therefore removed from the set of anomalies of the local node (lines 15-16). Because this work focuses on anomaly detection rather than on cluster detection, our current algorithm does not send the point to the remote node in order to update the cluster.

C. Applying the Algorithm

This section discusses the parameters and conditions required for the application of the decentralized clustering algorithm. First, the algorithm for the data collection/analysis phase (Figure 4) depends on knowledge of the size of the data set to be analyzed (i.e. the total number of data points for which the analysis is applied), which in turn depends on the number of devices generating status events, on the rate at which these events are generated, and on the collection period (T). We assume that the event generation rate is known and fixed, so only the number of devices and the collection period need to be determined. In a dynamic network, the former is not trivial, but there are existing methods for obtaining node counts in dynamic networks, even in the face of high churn [16]. Setting T is more application dependent, and will determine if the results of clustering apply only to a comparison of data between devices, or also to a temporal analysis of device behavior. Because we do not assume special synchronization between devices, however, the generation periods between devices will be staggered. Therefore, it is necessary to always extend T by one generation period, which will guarantee that each node receives the number of points for the intended number of periods.

The size of the regions assigned to nodes (where each region $R \subseteq S$) plays an important role in cluster detection. If the regions assigned to nodes are large with respect to cluster size, then there are two possibilities. One is that a cluster might not be detected (as an extreme case, any subset of D is not a cluster, as defined, with respect to S). The other is that there may be multiple independent clusters in a single region. In the first case, the algorithm can be modified to do a second cluster test before it assigns points as potential anomalies (between lines 10 and 11). This test is done with respect with the region of minimum size that contains all of the points in the analysis region (i.e. for the set of received points, the algorithm obtains density($points$)).

In the case of multiple clusters in a region, one option is to run different clustering algorithms, such as *k*-means or DBSCAN, since this would be a local computation. However, this requires knowing or estimating the necessary parameters for these algorithms and is more expensive computationally. Another option is to further subdivide the region into multiple regions by increasing the number of processing nodes, which basically increases the resolution of the analysis in the given region. Each one of the new nodes can be hosted at the same or some other physical device. Presently, a node that receives a large proportion (more than half) of the total number of data points simply creates one new node within its region for each original node in the network. Future work may determine more effective approaches. This option also favors load balancing because the new nodes can be spread out among the physical nodes. Load balancing is an important consideration for our clustering algorithm because, by design, heavily clustered data will be sent to a reduced number of nodes.

Because we focus on anomaly detection, our algorithm does not merge clusters when they spread out between neighboring nodes. Such spreading is likely to happen when the size of the analysis regions is of the same or smaller order than the size of clusters. However, this can be accomplished using the extended cluster widths in a way that is similar to associating border points with neighboring clusters. If these clusters are not merged, the multiple cluster centroids generated for a single cluster are basically an in-network aggregation of the event data in that cluster. Because the types of clusters that can be found by this method can be of different shapes and sizes, producing multiple centroids per cluster reduces the complexity of the data to be used for higher-level analysis, while preserving its shape and possibly protecting data privacy in systems where this is an issue.

IV. Messaging Substrate

The content-based messaging substrate described in this section supports the exchange of nodes' status updates and enables the partitioning of the information space into regions described in Section III by implementing the dynamic point to node mapping. The messaging substrate is responsible for getting the information used by the clustering analysis to the distributed processing nodes in a scalable fashion. The substrate essentially consists of a content-based Distributed Hashtable (DHT) that uses a particular locality preserving mapping. Content-based DHTs provide an interface that allows network nodes to be addressed by attribute-value pairs. This means that data points themselves can be used as addresses to send messages to particular nodes. The decentralized clustering algorithm requires that the mapping used by the DHT's addressing scheme preserve the spatial locality of the regions corresponding to different nodes (i.e. that all the points that map to a given node form a region as defined in Section III-B).

The required locality preserving mapping in this case is obtained using a Space-Filling Curve (SFC) [10], which is implemented by the Squid content-based routing engine [9]. SFCs are a family of computationally-efficient locality-preserving recursive mappings from *n*-dimensional space to one-dimensional space. An example of this mapping using the Peano-Hilbert SFC is shown in Figure 5.

The one-dimensional index space that results from this mapping is partitioned across the dynamic set of nodes, forming a ring overlay (the Chord [8] overlay is used in our implementation), where a node is responsible for a sequential range of data points in this index space (between its predecessor and itself), as shown in the figure. The important property of the SFC is that every sequence of points on the SFC curve map to a region of contiguous points in the multidimensional space [17]. Thus, this mapping meets the requirements of the decentralized clustering algorithm.

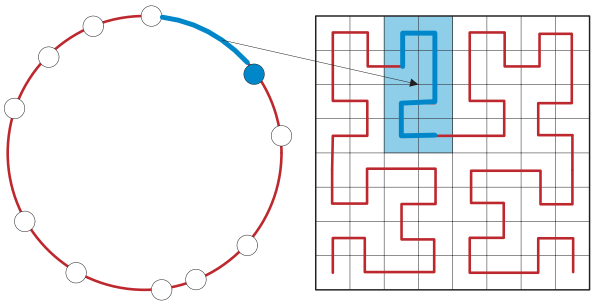

Fig. 5. The highlighted node is responsible for the highlighted region, which is mapped onto a multidimensional space using an SFC.

When a node joins the Chord overlay, it is assigned a specific ID in the Chord index space using the Chord join protocol. Since this node will be responsible for the span of the index space between its ID and the ID of its predecessor, the corresponding region of the information space can be computed using a reverse SFC mapping. The size of the region is simply the difference between the node's ID and the ID of its predecessor, which is discovered during the join procedure. To route a particular message (data point) to a node, the SFC mapping is applied to the values contained in the message to obtain a unique Chord index. This index is then used to route the message using Chord's routing protocol.

V. Complexity Analysis

Because most of the complexity of the algorithm depends on messaging and the mapping between nodes and regions in the information space, the complexity analysis must be done with respect to the particular overlay on which the algorithm is implemented. In this case, we use the Chord overlay and SFC mapping described in Section IV. For this analysis, n is the number of nodes in the network, as well as the size of the data set, a is the number of dimensions (attributes) in the information space, and b is the number of bits used to encode values along each dimension.

In the setup phase, joining Chord requires $O(\log^2 n)$ number of messages, for each node, and a time that is proportional to this number. The computational complexity of the setup phase depends mostly on step 5 in Figure 3. For a node to guarantee finding all of its neighbors, it must calculate the reverse SFC mapping for the index points between its predecessor and itself, and map their contiguous points to see if they correspond to another region. For a large information space, this approach can be time consuming, since the average time required would be $O\left(\frac{a2^b}{n}\right)$. Because node regions are expected to be nearly convex, however, a less costly approach is to find the nodes that contain the points that first intersect with lines extending along every dimension from some point within the local region.

Now, the data collection and analysis phase is considered. The complexity of this algorithm in terms of the number of messages is as follows. Each monitoring event sent over the network is a singular point in the information space, which requires $O(\log n)$ routing hops in Chord. This translates to $O(n \log n)$ messages for the n nodes. When possible anomalies are detected, at most $2a$ neighbors are queried to rule out boundary conditions. Thus, the worst case complexity of querying for neighboring clusters (if possible anomalies existed at all nodes) is $O(an \log n)$ in total. In practice, of course, the number of regions containing anomalies is expected to be a small fraction of the number of nodes. Temporally, the complexity due to messaging is $O(\log n)$ given that nodes exchange data points in parallel. In terms of processing, each event requires a single SFC lookup, which takes $O(ab^2)$ time.

VI. Experimental Evaluation

We evaluate the decentralized clustering algorithm and self-monitoring infrastructure in two stages. In the first stage (Section VI-A), we use a simulated dataset whose characteristics are known a-priori and check if the algorithm performs as it should. In this case, we report the effectiveness of the anomaly detection algorithm which is characterized by type I and type II error rates (false positives and negatives). In the second stage (Section VI-B), we collect real-world data from a functioning device network and perform anomaly detection using the decentralized clustering algorithm. Here, we report clusters and anomalies found with respect to device usage and job patterns. Through these experiments, we hope to illustrate the effectiveness and utility of our procedure. Evaluations of the content-based routing and messaging substrate using both simulations and real deployments on clusters, wide-area testbeds and sensor networks have been reported in [9], [11], confirming $O(\log n)$ routing hops for messages and scalable latencies for message processing and delivery.

In the result figures presented below we use the following conventions: dots represent data points, crosses represent cluster centroids, and circled dots or circles identify anomalies, as detected by the algorithm.

A. *Effectiveness of Anomaly Detection*

We measure the effectiveness of the algorithm in terms of its error rate, both in terms of false positives (points labeled as anomalies that are actually part of a cluster), and of false negatives (non-cluster points that are not labeled as anomalies). Because there is no benchmark for the accuracy of anomaly detection in the real data used for the case study below, artificially generated data is employed for this evaluation. This data consists of 2D points that are made to cluster with different cluster shapes and densities around three different points, with a small percentage of random noise (see Figure 6). This way, we are able to definitively verify the classification of each point and calculate the error rate.

Because the definition of a cluster depends on the analysis region sizes, which in turn depend on the number of processing nodes, the first experiment consists of running the clustering algorithm with different numbers of processing nodes n. The total number of points in all runs is kept constant. Figure 7 shows that the false positive rate increases and the false

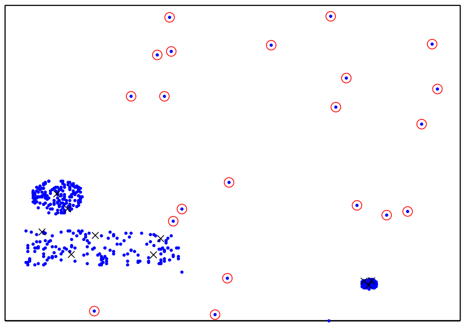

Fig. 6. Analysis results for simulated data

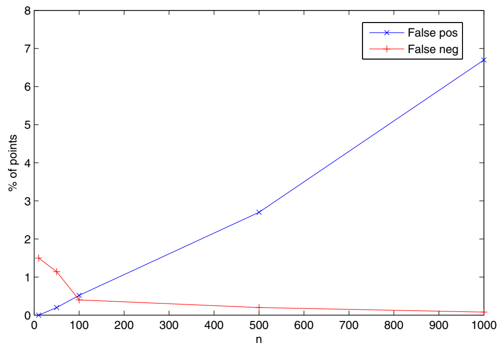

Fig. 7. Error rates for varying n.

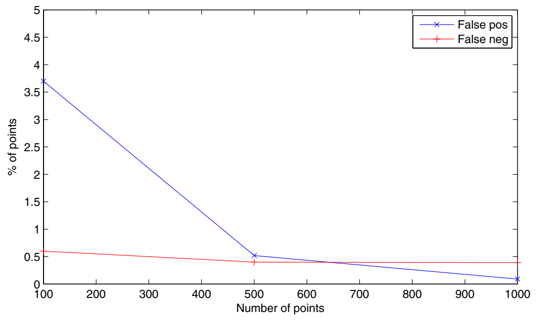

Fig. 8. Error rates for varying data set sizes.

negative rate decreases with the number of nodes. The optimal total error rate (adding both rates) is found for 100 processing nodes. Notice that the false positive rate rises sharply with the number of nodes, which is attributed to the regions becoming too small, so that it is unlikely that there are enough points in any region to form a cluster.

We also wanted to see how the accuracy was affected by the ratio of the number of points in the data set and the number of nodes, because the number of expected points at each node depends directly on this ratio. For this experiment, we used 100 processing nodes (the optimal number from the previous experiment). Figure 8 shows how the false positive rate indeed improves when the ratio is greater because it becomes less likely that a cluster of points will have less than the expected number of points. The false negative rate is not significantly affected by the changing ratio.

B. *Device Network Monitoring*

The following experiments correspond to the analysis of real data collected from the operation of a device network consisting of 54 devices. The network actually serves an office environment that produces several hundred thousand pages a month. The data consists of descriptions of jobs submitted to the devices during a seven week period. In all, there were over 10,000 jobs of varying sizes (if the jobs were represented in terms of pages there would be several hundred thousand) in the dataset. During the seven week period, there were about 200 users interacting with these devices, which could be spread out over a small geographical area within a large organization spanning several buildings. The job descriptions identify the user submitting the job, the device receiving the job, and the job size, and include the date and time of the submission. The experiments correspond to two usage scenarios and are designed to show the utility of our clustering analysis.

1) Scenario 1: Monitoring device usage levels: In this scenario, the devices periodically monitor and report their usage level in order to compare and detect normal usage patterns as well as abnormally utilized devices. To this end, devices aggregate the job submissions for the given time period and produce a status update consisting of three dimensions: average job size, number of jobs received, and number of different users that submitted jobs to the device. Each one of these attributes is a different indicator of usage that can be interpreted and controlled in different ways.

In order to define the information space for this scenario, we must first define the unit distances d_i for each attribute. Recall that the unit distance is the minimum distance at which two attribute values are considered to be different, and that it is strictly application dependent. In this case, we select $d_{jobSize}$ equal to 10KB because there were relatively fewer jobs below this value in our dataset. Since we wanted to analyze the patterns relating to individual users and jobs that were served/received by each device, $d_{numJobs}$ and $d_{numUsers}$ are both set to 1 - the smallest quantum for those dimensions.

Assuming no previous knowledge of event distribution in this space, the network is initially set up with 54 processing nodes (one for each device). One set of experiments was conducted setting the analysis period to one day, i.e. clustering was done on the data generated for each day, while for another the analysis was done for each week. In our dataset, the granularity of the weekly analysis provided better clarity, and is thus used for illustration. Figure 9 shows the results for a single week, in which events cluster near the origin, with a single anomaly for a very large average job size. However, due to the large size of the information space (given the relatively

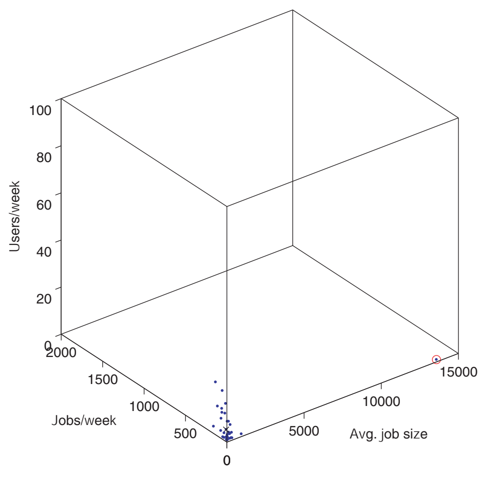

Fig. 9. Initial analysis for one week period using uniformly distributed regions for the set of processing nodes. The figure illustrates the size of the information space compared to the region where points cluster, which makes the assignment of additional nodes in that region necessary.

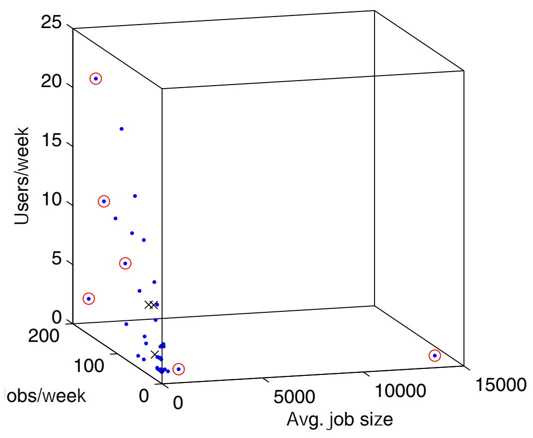

Fig. 10. Analysis of one week period after increasing the number of processing nodes near the origin

large range of the job size attribute) and the number of nodes used, this is an example of analysis regions being too large. In this case, nearly all data points ended up in the region corresponding to a single node.

To improve the resolution of the analysis near the origin, new processing nodes were created following the approach described in Section III-C. Figure 10 shows the new results of the clustering analysis for the same week as above. The scale has been modified to better visualize the new anomalies that were identified. Figure 11 summarizes the results for the seven weeks of the experiment, showing only cluster centroids and anomalies.

Interpretation: Cluster centroids give an average normal device usage, i.e. the usage levels exhibited by most devices. This is useful for load balancing because load balancing techniques

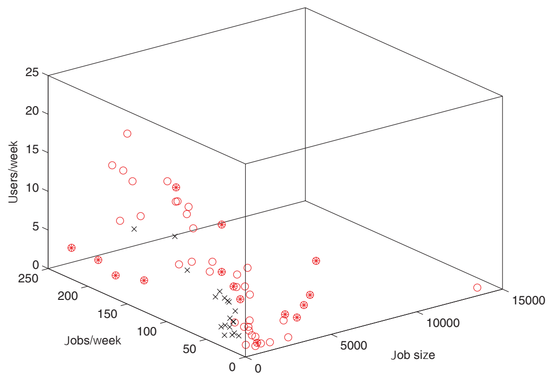

Fig. 11. Summary of results for weekly periods (device usage). The filled circles represent the anomalies from consistently anomalous devices.

can be applied to devices that have an anomalous utilization in some time period, to drive them toward the normal utilization level. Note that without comparing utilization across devices, the values for "normal" utilization cannot be determined.

Additionally, anomalies can be observed over several time periods to detect devices that have consistently abnormal usage. During the seven week period, three devices were consistently present within the anomaly reports for the system. Their anomalous status updates are highlighted in Figure 11. It can be inferred that the high usage of these devices is due to their importance within the network and/or preference of users toward them (e.g. because of high quality of job execution, strategic location in the system, kinds of services provided, etc.). This information can subsequently be used to allocate greater resources for those particular devices, or to improve other devices given their configuration/characteristics.

In this environment, we not only want to monitor device usage, but also user behavior, for example to know which users submit jobs more frequently than others, or perhaps jobs with a greater average size than those of others. This type of knowledge can be exploited to enforce local policies on device usage, to identify potentially abusive or misbehaving users, and/or to provision sufficient resources to meet the expected user demand. This can be accomplished by creating an information space much like that for the different devices, but from the users' point of view; i.e. the dimensions of the space are: average job size submitted by the user during the monitoring period, number of jobs submitted, and number of different devices to which submissions were made. In this case, status updates are not generated within the device network, but from the users' workstations, where this information can be aggregated [1]. These workstations can also make part of the node overlay, along with the device network, lending their processing power for the monitoring and analysis. Figure 12

[1]This assumes that a user always submits jobs from a single location. However, even if this is not the case, a user account can be used to keep track and aggregate users' requests from different locations.

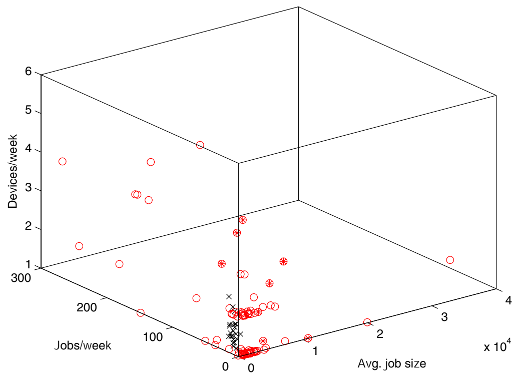

Fig. 12. Summary of results for weekly periods (user behavior). The filled circles represent the anomalies from consistently anomalous users.

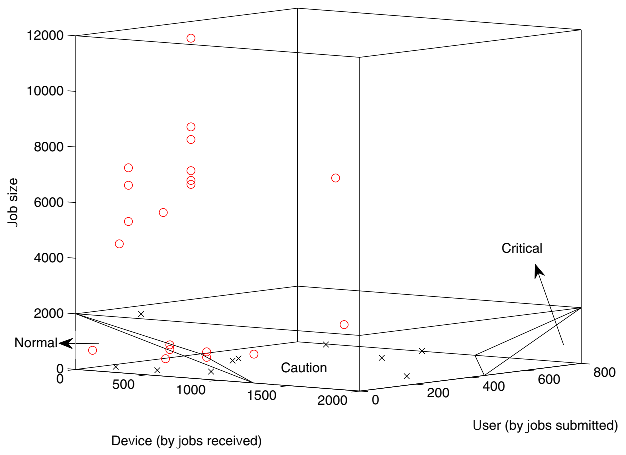

Fig. 13. Result of clustering of job requests, which are interpreted with respect to a division of the information space by the separation planes shown.

is analogous to Figure 11 in the case of users.

2) Scenario 2: Monitoring job patterns: While in the previous scenario clustering analysis is used to separately analyze devices and users, clustering can also be used to relate particular devices to users and observe patterns in these relations. This is accomplished by clustering job requests directly in an information space corresponding to their attributes: receiving device, submitting user, and job size. However, because device and user IDs do not have an ordering that can be used for clustering along those dimensions in the information space, we order devices and users by the number of jobs that they receive/submit within the analysis period. We call this the frequency of the device/user, and plot a job corresponding to a device/user pair at the intersection of their respective frequencies. This is effectively a representation of their popularity or influence within the network, and can also be obtained more meaningfully by the results of the previous scenario.

Interpretation: A cluster in this information space will represent a significant number of jobs of certain size being submitted by a user or users of similar frequency to a device or devices of similar frequency. Anomalies will correspond either to abnormal job sizes or to an abnormal matching of users and devices. Figure 13 shows the cluster centroids and anomalies found after applying the clustering analysis to the entire data set.

In this scenario, most clusters occur between relatively low frequency devices and users and most anomalies are due to large job sizes. However, we would like to be able to further classify clusters and anomalies with respect to their significance to system operation. For example, a high frequency user submitting large jobs to a high frequency device is a possibly misbehaving or malicious user. However, although clustering is useful for finding relative patterns in data that can only be found by comparison, this kind of classification requires assigning absolute values to notions like 'high' and 'large'. In general, *a-priori* knowledge or system policy, represented by the division of the information space shown in Figure 13, can be combined with the clustering analysis to accomplish this classification. The planes in the figure divide the space into two critical regions, for large jobs and combinations of high frequency users and devices, a normal region near the origin, and a caution region for intermediate values of the different attributes. Although this classification is very domain and organization specific, action policies may result from such analyses leading to the management of the overall QoS of the network.

C. Discussion

It is important to note that all of the above results were obtained without the need to specify data-dependent parameters, such as the number of clusters in the data (required for k-means) or minimum cluster densities (required for DBSCAN and CLIQUE). The information required by our approach is summarized in two parameters: the number of data points, from which a baseline point density is obtained, and the number of processing nodes. Both of these parameters can be calculated and adapted at runtime, as we have shown.

We do not claim that the accuracy of the algorithm is superior to that of previous algorithms, as in fact, it is a decentralized adaptation of density-based clustering techniques, focused mainly on anomaly detection. Based on the results in Section VI-A, however, it has been shown that the algorithm is accurate despite the tradeoffs inherent in decentralization, which limits the communication between nodes. Furthermore, the computation overhead at each node consists mainly of keeping track of and comparing point counts, and the messaging substrate used is also designed to minimize the overhead of event mapping and distribution.

Finally, we point out that in the same way that the messaging infrastructure supports the content-based distribution of events to processing nodes for the clustering analysis, it can also support the distribution of analysis results to network management components: these can subscribe to receive results only for certain regions of the information space, which are important according to management policies.

VII. Conclusions and Future Work

This paper has described the application of clustering analysis for detecting anomalies and trends in monitoring events from device networks. It presented a novel decentralized algorithm for cluster and anomaly detection and showed how it can be implemented for in-network execution on top of an overlay network. The performance of the algorithm was analyzed, and its utility demonstrated through a case study using real data collected from a device network.

This work is part of an ongoing effort to create tools for the integrated analysis of performance, security/trust and reliability for autonomic management at a system level. Current and future efforts are focused on an integrated proactive self-monitoring and autonomic management framework that is based on an in-network and online unified analysis of system events, exchanged in a decentralized fashion.

Acknowledgements

The research presented in this paper is supported in part by National Science Foundation via grants numbers CNS 0305495, CNS 0426354, IIS 0430826 and ANI 0335244, and by Department of Energy via the grant number DE-FG02-06ER54857.

References

[1] E. Kiciman and Y.-M. Wang, "Discovering correctness constraints for self-management of system configuration," in *Proceedings of the First International Conference on Autonomic Computing*, New York, NY, May 2004, pp. 28–35.

[2] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *Proceedings of the First International Conference on Autonomic Computing*, New York, NY, May 2004, pp. 36–37.

[3] G. Jiang, H. Chen, C. Ungureanu, and K. Yoshihira, "Multi-resolution abnormal trace detection using varied-length n-grams and automata," in *Proceedings of the 2nd International Conference on Autonomic Computing*, Seattle, WA, June 2005, pp. 111–122.

[4] G. Jiang, H. Chen, and K. Yoshihira, "Discovering likely invariants of distributed transaction systems for autonomic system management," *Cluster Computing*, vol. 9, no. 4, pp. 385–399, 2006.

[5] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, A. R. Series, Ed. Prentice Hall, 1988.

[6] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 1996.

[7] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic subspace clustering of high dimensional data for data mining applications," in *Proceedings of 1998 ACM-SIGMOD Int. Conf. Management of Data*, Seattle, Washington, June 1998, pp. 94–105.

[8] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of ACM SIGCOMM*, San Diego, CA, August 2001, pp. 149–160.

[9] C. Schmidt and M. Parashar, "Flexible information discovery in descentralized distributed systems," in *12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12'03)*, 2003.

[10] H. Sagan, *Space-Filling Curves*. Springer-Verlag, 1994.

[11] N. Jiang, C. Schmidt, V. Matossian, and M. Parashar, "Enabling applications in sensor-based pervasive environments," in *Basenets 2004*, San Jose, CA, October 2004.

[12] S. Bandyopadhyay, C. Gianella, U. Maulik, H. Kargupta, K. Liu, and S. Datta, "Clustering distributed data streams in peer-to-peer environments," *Information Sciences*, vol. 176, no. 14, pp. 1952–1985, July 2006.

[13] S. Datta, C. Giannella, and H. Kargupta, "K-means clustering over a large, dynamic network," in *Proceedings of the Sixth SIAM International Conference on Data Mining*, Bethesda, MD, April 2006.

[14] M. Li, W.-C. Lee, and A. Sivasubramaniam, "Pens: An algorithm for density-based clustering in peer-to-peer systems," in *Proceedings of the International Conference on Scalable Information Systems (INFOSCALE)*, June 2006.

[15] E. Ogston, B. Overeinder, M. van Steen, and F. Brazier, "A method for decentralized clustering in large multi-agent systems," in *Proceedings of the 2nd International Joint Conference on Autonomous Agent and Multi-Agent Systems*, 2003, pp. 798–796.

[16] O. Babaoglu, G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montressor, and T. Urnes, "Design patterns from biology for distributed computing," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 1, no. 1, pp. 26–66, September 2006.

[17] B. Moon, H. Jagadish, and C. Faloutsos, "Analysis of the clustering properties of the hilbert space-filling curve," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 1, pp. 124–141, Jan/Feb 2001.