A CLUSTER TRACKING ALGORITHM FOR DISTRIBUTED DATA ANALYTICS

BY RAUL S LASLUISA

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Manish Parashar

and approved by

New Brunswick, New Jersey

May, 2012

© 2012

Raul S Lasluisa

ALL RIGHTS RESERVED

ABSTRACT OF THE THESIS

A Cluster Tracking Algorithm for Distributed Data Analytics

by Raul S Lasluisa

Thesis Director: Manish Parashar

Large-scale data analytics has enabled society to model, and inspect their data to the point where useful information can be extracted, conclusions can be drawn and decision making can be enhanced. The breadth of data being analyzed today has enabled us to make proactive decision in processes we otherwise could not. At the same time the data being analyzed is both becoming larger and more distributed, making it more complex to aggregate the data to a central location and process in a timely manner in order to make decisions. This can be attributed to the scale of current distributed computational infrastructures used to solve complex problems, while generating an increasing amount of data. This data is being created not only from applications solving problems but also from the systems running the applications as well. Creating a situation where centralized data analytics benefits decline as appose to decentralized approaches.

Data analytics algorithms must therefore meet several new requirements in order to continue to process data in a timely manner. One approach to process distributed data is to use algorithms that themselves can run in a distributed manner. Using such algorithms benefit a variety of situations where there is a desire to reduce the cost of transporting and subsequently storing data. Examples can be seen in autonomic computing, where the goal is to manage large system with minimal intervention by

administrators and scientific visualization where visualization techniques are performed using a secondary system.

In this work we show that combining online (and distributed) data clustering, and cluster tracking can be effectively used to detect meaningful changes in data patterns occurring in the multiple streams. In doing so, we provide an alternative to a centralized approach where data must be centralized before any analytics may be executed. Specifically, we propose an cluster tracking algorithm which takes advantage of a decentralized clustering algorithm in order to detect changes in data to then take proactive decisions. We demonstrate its accuracy and effectiveness in three different case: 1) VM provisioning 2) scheduling of Hadoop resources, and 3) object tracking in scientific applications.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

The ability to do data analytics has allowed society to model, and inspect their data to the point where useful information can be extracted, conclusions can be drawn and decision making can be enhanced. At the same time the data being analyzed is both becoming larger and more distributed making it more complex to aggregate the data to a central location and process in a timely manner. For both autonomic computing and scientific applications this poses a problem since traditional data analytics rely on data aggregation as a prerequisite to analysis. The major expense of traditional aggregation methods come in the form of additional resources required to move and store data.

Although centralizing data is convenient for several reasons, such as ease of programming, and accuracy, it incurs large cost in terms of resource overhead to aggregate data. In some instance secondary computing resources are need to store and subsequently analysis the aggregated data. In order to eliminate this cost, a decentralize approach can be used to eliminate the aggregation step when doing data analytics. Decentralized approaches reduce the cost of transferring data by doing analytics at the origin of the data depreciating the need to aggregate. In addition, decentralized approaches can use large scale computational resources making them more attractive then centralized approaches in certain situations.

Today there are situations where the cost to centralize data create a need for secondary computing resources to do data analysis. This increases the total cost of hardware used and likelihood of increasing idle resources within a data center. Autonomic computing systems and scientific applications generally centralize data that needs to be processed, both these system will face increasing issues in order to aggregate data. The

issues will arise due to their systems becoming larger and more distributed as well as the amount of data generated increases. In order for autonomic systems and scientific applications to continue to advance, we argue that distributed algorithms are needed to handle this new large scale and distributed environment.

This work presents a distributed cluster tracking algorithm that builds upon a distributed clustering algorithm present in [14]. By using this distributed clustering algorithm additional computing resources can be used and provides a fault tolerant and robust framework for data analytics. With the incorporation of our feature based cluster tracking algorithm the frame can be reused to track clusters over time. By using the approach presented in this paper no additional algorithms are needed to track information across time.

1.2 Problem Description

Distributed systems which are used for high-performance computing (HPC) generate a large amount of data in a distributed manner. This data is generated not only from the applications running on these HPC systems but also from monitored components. This data must be analyzed in order to further understand or act upon the information received. Typically data analysis algorithms aggregate data creating a single stream of information.

Algorithms which require data to be seen as a single stream of information do not take advantage of parallelism at a task level within the algorithm. By not identifying these parallelisms, data is required to be gathered where it otherwise would need to be, restricting computation to local resources. An example of a data analytics algorithm that can be executed in a distributed manner is data clustering. This is not say that executing such an algorithm in a distributed manner is without disadvantages, such as programming concurrent process, but in situations where data is naturally distributed and generated in large quantities, distributed algorithms do provide an acceptable alternative to centralized approaches. Distributed data analytic algorithms in

HPC environments must also overcome their disadvantage of storage. The limited storage available on HPC nodes, when compared to their centralized counterparts, create issues when doing historical trend and pattern recognition.

For distributed data analytics algorithms to be just as effective as centralized solutions, the will need to be able to identify trends and patterns while not needing to store a large amount of data. One approach that may be used to detect trends and patterns within data is include cluster tracking to traditional data clustering techniques. Once again in a centralized approach a large amount of data can be stored in order to make the historical relationships between clusters. When furthering examining particular cluster tracking techniques it can be shown that they too can be executed in a distributed manner just like clustering algorithm.

1.3 Problem Statement

Data analytics can be divided into two different categories offline and online. The former is used when information can not be processed as it is created or when large historical trends are the objective. The expense incurred from offline approaches come from the resources used in order to move the information and store it. The latter must incur the expense of dividing the problem into parallel components to take advantage of distributed resources. This method additional can not retain large historical information and must therefore find a means of compressing information in order to use it for longer trend detection.

1.4 Research Overview

The work presented in this paper will develop and evaluate a cluster tracking algorithm that can process multidimensional information using distributed resources. Data clustering and cluster tracking have been used in many areas, the ones presented in this paper are autonomic computing and scientific applications. In both these areas information is created periodically allowing for the opportunity to identify changes and trends in information. This is the area which this work focus on by providing an algorithm

that detects changes by tracking clusters in information.

The approach used to track clusters in this work is based on the observation that clustering algorithms themselves can be used to track cluster over time. This observation was made based on how clustering works and what information can be used to determine if a cluster is related to another in a different point in time. As will be shown by using cluster features and analyzing them through a clustering algorithm the path of any given cluster can be identified across time. In addition this work will leverage a decentralized clustering algorithm to provide a robust and fault tolerant framework to do cluster tracking.

Finally we demonstrate the usefulness of our clustering tracking algorithm in both autonomic computing and scientific applications by providing three different use cases. In each of these use cases the accuracy of our cluster tracking algorithm is tested and used to optimize a metric in that specific use case.

1.4.1 Contributions

The contributions of this work are:

- Present a cluster tracking algorithm which using a decentralized clustering algorithm.
- An evaluation on the accuracy of the cluster paths detected from our algorithm and the predictive capability of algorithm once these paths have been identified.
- Three use cases highlighting not only the uses for our clustering algorithm but the need to create more algorithms like it.

1.5 Impact of Research

In order to perform high performance computing distributed infrastructures will continue to be used and grow in scale, along with their associated data. For this reason it is important to develop algorithms, such as the one presented in this work, in order to process information without incurring the expense of additional resources in order to

do offline analysis. This is not only necessary because the cost incur grow as the scale of the infrastructure grows but also because it enables online data analytics. With the advances in areas of data stream mining and in situ data analytics distribute algorithms will be a focal point. We expect this work to be used to advance research in both data stream mining and in situ data analytics because it is a distributed algorithm. We also expect our work to advance both autonomic computing and scientific computing alike since online analysis bring real-time decision making that much closer.

1.6 Thesis Overview

This document is organized as followed. Chapter 2 will present the related work of our research, focusing on clustering algorithms, their uses, designs, and use cases. Chapter 3 will cover the clustering algorithm, DOC, which we use and how its implementation enable distribute data analytics. Chapter 4 presents our cluster tracking algorithm and how it uses DOC in order to provide an accurate trajectory of cluster movement. Chapter 5 illustrates the usefulness of our algorithm by presenting three different use case and results from experiments taking them into account. Chapter 6 provides our thoughts and future work with our propose algorithm.

Chapter 2

Background and Related Work

The goal of our research is to provide cluster tracking algorithm that can be used in environments where data is naturally distributed and data can be processed where it is created. This provides a foundation where an aggregate view of a system can then be taken based on pre-analysis of local information. In order to appreciate the impact of this research this section has been divided into five sections describing the different areas which this work impacts. This section will first serve to provide a background for the type of algorithm which we have created, and the areas that would benefit from our algorithm.

2.1 Data Stream Mining

The advances of computing systems and their applications have allowed the capture of different measurements of information in various areas. The rate of creation of this captured information is steadily, and in some cases drastically, increasing, causing a situation where data can be see as a continuous stream. These streams of data are simply called data streams, based on their nature of continuous their output of information. These data streams have created a area of research which looks to analysis this information as fast as the data is being created. A through review of the are is provide by Gaber et al. [11].

2.2 Clustering

Guha et al. [12, 13] provides a study on K-median clustering in order to model data streams. A through explanation on the upper and lower bound of the algorithm is provide when a single pass is used on the data. The study thereby determines the total

runtime and memory usage required in order for this algorithm to process a given data stream. This work serve as a base to understand the inherent problems faced by data analytics algorithms in order to data in a continuous manner.

Charikar et al. [5] povide a means that enables k-means to cluster data streams more efficiently. In this work a means to parallelize the k-means algorithm is given so that portions of historical information are stored to then be used for incoming data. The method implode is a divide and conquer method which has a portion of the information processed offline and another as the data is created.

Aggarwal et al. [2] look to address the problem of the poor quality of clusters when data evolves considerably over time. The approach take is to do both an online and offline analysis of the information. The online component periodically stores detailed summary statistics of the data being processed. The offline component is utilized to provide an understanding of the overall clusters in the data stream. Using these two methods together provide a framework where cluster are of a better quality than those taken at any one point in time.

O'Callaghan et al. [23] provide a new clustering algorithms that is proven to theoretically provide better performance in certain situations. Their approach is split into 2 different algorithms STREAM and LOCALSEARCH, where the STREAM algorithm determines the amount of data to be processed and LOCALSEARCH is applied. LOCALSEARCH is an iterative aggregating the previous cluster centers from the previous iteration in order to converge to a solution.

2.3 Cluster Tracking

Abrantes and Marques [1] studied the stochastic information within a data cluster over time in order to track the movement of cluster in the context of computer vision. With their proposed technique motion models can be created for clusters discovered in a data set. Their approach also proposes a noise model which can increase the robustness of the motion model with respect to outliers effecting the quality of the motions detected. They show its effectiveness by evaluating car traffic sequences and ultrasounds.

Abrantes and Marques also explored the idea of tracking clusters based on specific internal features of a cluster and demonstrated how cluster features can be used to identify distinct clusters at different time frames. We use this concept outside the area from which this method was created, computer vision. This therefore dramatically changes the reasoning behind the use of the method and must be adapted to fit our needs in the area of autonomic computing. As a result the comparison to this work is more of how computer vision ideas must be altered to be used in a different area.

2.4 Autonomic Computing

The goal of autonomic computing is quite complex. In the context of this work it is creating a system which is self manageable and self configurable with minimal intervention by administrators. What is usually understated is that autonomic computing requires effective examination and interpretation of system data based on knowledge of the system. In order to meet the requirement of effective examination and interpretations of system information data analytic algorithms used must be able to represent collected system data in a form such that decision can be done autonomously.

One such data analytic algorithm which has proven to be useful in interpreting system information is data clustering. The usefulness of data clustering comes from its conceptual approach of examining how similar data points are to one another by analyzing the distance between them and neighboring points. This concept of data clustering has been shown to be a powerful tool because of its effectiveness in solving the problem of not only examining system data but also the interpretation of the data. In conjunction with the concept of cluster tracking, data clustering becomes even more appealing for as an autonomic computing algorithm.

Wang et al. [33] advances the case for what they call monalytics which enables continuous, real-time monitoring along with on-line analysis of the data captured by the monitoring system. Wang presents a flexible architecture for monalytics by using software overlays called Distributed Computation Graphs (DCGs) to implement analytics functions. Using two real world use cases of internet services, simulated using

RUBiS, and MapReduce applications reduce time to insight (TTI) which are the total delay between an event of interest happens and when it's detected. Their reductions in such metrics encourage more research in the area which can do in-situ analytics, which we look advance as well.

Quiroz et al. [26] proposed a novel policy definition framework that enables autonomic policy adaptation and proactive policy application, based on the online analysis through decentralized clustering mechanisms and characterization of system operation and feedback events. This paper extends some of these ideas with a tracking approach to create predictive models.

Domingos et al. [8] explored the management of web server cache using very fast machine learning techniques. In their approach webpage requests from various departments in the University of Washington were categorized to create an optimized web cache by identifying which department could benefit from sharing a single web cache rather than an individual. Domingos et al. showed that by processing fractions of a data stream, in one hour chunks, their clustering algorithm could converge within an allowable error rate, compared to processing the complete data. They demonstrated that by looking at subsets of data a centralized technique is still effective, even when processing a high data volumes. However, there is an assumption that the stream of data is able to get to a centralized location to then be processed in contrast to our approach that maintains its distributed nature to minimize the impact of transferring data, distribute the computation overhead.

Xu et al. [34] used data clustering techniques and fuzzy logic to provide resource savings for datacenters. They illustrate the usefulness of categorizing job request and also show how these categories can change based on clustering results and adapt the fuzzy logic accordingly. The use of these techniques allows for applications to be profiled and be given a configuration which supports their SLA as well as use the least amount of resources possible. Our technique differs in how we identify changes in the category of jobs over time, identify underlying relationships in the data, and take into account the trajectory of the clusters themselves, to allow us model how job categories evolve over time to identify instances that adaptive fuzzy logic alone would not be able to see.

Filali et al. [10] made a case for categorizing jobs before provisioning resources to maximize both QoS and profitability. In their approach the concept of global and local observers are used to not only allocate resources per user request, maximizing requests handled, but also to adjust the resources to improve QoS over time as resources are released. Their work is a traditional example in which the assumption where users know what resources they need ahead of time is used. As a consequence this method would not be able to handle the profiling of the application and a different method would be needed to effectively model an application to fit in this scheme. Our technique can be used to relate how users are requesting resources as related to the amount they use, in turn creating even more savings for both parties in terms of minimizing over provisioning. Due to our clustering method and tracking we can identify different states of an application and adapt as needed.

Urgaonkar et al. [32] discuss the idea that different application tiers need different amounts of resources even when handling a single request, one request may trigger multiple operation on a database for example. Their approach effectively demonstrates that it is not effective to simply increase resources across a multi-tier application as it leads to over provisioning of resources due to not effectively identifying the current bottleneck to support incoming request to use the application. There is an assumption made in which the user knows ahead head of time what is the best configuration for each server in each tier making this method ineffective in modeling application for the best use of resources.

Machida et al. [21] examined the idea of having VMs in different states depending on how resources are being used in a datacenter. The paper argues that it is possible to reduce wait times and support SLAs more effectively by placing VMs in different states in which they use less resource and provide saving to a datacenter. The paper also shows that using a stock program for predictions was good enough for establishing the correct state for a VM. Our method differs in that we look to adjust the configuration of the VM base on incoming request.

In [16], Kalman filters are used to provision CPU resources in virtualized servers. This paper shows how effective Kalman filters are at adapting to changes in CPU

utilization, rapidly allowing for self-configuration of the system over time and reducing over provisioning. The paper shows that when looking at one-dimensional data, such as CPU utilization, there are highly effective methods which can provision much better than algorithms with multi-dimensional data such as clustering.

Kundu et al. [18] showed how effective artificial neural networks are at modeling applications when looking at multidimensional datasets. Their approach effectively demonstrates the prediction accuracy and timeliness of these algorithms in the realm of resource provisioning based on applications modeling. However their approach needs data to first be centralized in order to do its processing.

2.5 Scientific Visualization

There is an increasing performance gap between computing and I/O, making the cost of moving large volume of data to/from disks more expensive. This has motivated computation scientists to employ the in-situ data processing approach to perform analysis, visualization [35], [20], indexing building [17], compression [19] etc in order to meet today's requirements for scientific applications. The core concept of these approaches it to move analytic operations to the location wherethe simulation is running. However, existing approaches [4], [9] tightly integrate analysis or visualization libraries into simulation code making them specific to applications and are not generic enough to create a framework of any scientific application.

Many techniques have been developed by computer vision community to extract, identify and track features. Silver et al. [29] presented a semi-automatic volume tracking algorithm to improve visualization of 3D time-varying Computational Fluid Dynamics datasets. Chen et al. [6] developed a parallel algorithm to analyze and visualize in realtime the evolving features extracted from time-varying simulation datasets. Our technique represents features as clustered data points in a multi-dimensional information space, and identifies and tracks the data clusters of interest in a distributed and timely manner using DOC.

Chapter 3

Infrastructure To Support Decentralized Cluster Tracking

This section will provide the a complete explanation to the distributed clustering algorithm used to enable cluster tracking[1] and background information on clustering itself.

3.1 Clusters

A popular data mining technique is data clustering, which establishes the concept of similarities or relationships between data points. Depending on the clustering method used the definition of similarity differ, but the concept of similarity remains. This proves to be quite powerful when taken from the point of view of categorization of monitoring data.

Categorization of monitoring data creates the opportunity for proactive management by expressing the data at a higher level of abstraction. This higher level of abstraction, clusters, allows for a simpler transition to higher levels of abstractions such as policies, which are at the core of any autonomous management system for a datacenter. For example, Quiroz et al. [26] proposed the ability to link clusters to policies by linking multiple information spaces and Xu et al. [34] look at adjusting their fuzzy logic as the data which is clustered changes

In addition, clusters and their movements can be used to define profiles. A profile can be used to indicate patterns of behavior or characteristics of one or more entities and can be used as a basis of prediction of said behavior or characteristics.

[1]The sections explaining in detail the DOC algorithm are a paraphrase of [14] and contains excerpts.

3.2 Decentralized Online Clustering

Decentralized online clustering (DOC) was create to provide online and decentralized data analysis, using the collective computing resources in distributed systems. DOC seeks to be an online algorithm, thereby allowing short-term system behavior to be captured, as oppose to an offline approach which can not capture short-term behavior. DOC by being a decentralized approach improves fault-tolerance, responsiveness, and reduces hardware cost to store and move data. DOC has also been tested to for its accuracy for identifying clusters within a given dataset and shown to as accurate to its offline counterparts such as K-means.

3.2.1 Infrastructure Support for Data Distribution

For DOC to communicate with all nodes in the system that have data which need to be clustered it use Meteor [15]. Meteor is a content-based communication platform for peer-to-peer systems based on a rendezvous messaging model. The Meteor framework is composed of the Meteor service itself and a content-based routing infrastructure built on a structured peer-to-peer overlay. The Meteor communication service is a kind of DHT based on associative rendezvous, a paradigm for content-based decoupled interactions.

In addition to Meteor, DOC uses squid [28] enabling access to dynamically map between multidimensional space and dynamically use sets nodes. Squid is made using a distributed system whos nodes have been indexed using Chord as a overlay network. Squid main contribution is to reduce indexing scheme and provide a routing mechanism that maps descriptors in a multidimensional space to nodes used to create the overlay network. Squid dynamically divides the multidimensional space among the distributed nodes, so that disjoint continuous subspaces of the multidimensional space are assigned to each participating node. Squid guarantees that all existing nodes to which a given descriptor corresponds can be reached with bounded costs in terms of the number of messages and the number of nodes involved.

3.2.2 Algorithm Description

The DOC algorithm was originally used with individually monitored components in a data center to determine the operational status. Events that are monitored create a data points where each one is represented as a single point in a multidimensional space. Each dimension in this space, referred to as an information space, corresponds to one attribute of the event being monitored. The location of a point represent the exact values of the attributes describing the event.

Once events have been collected along with the values of the attributes describing them clustering can be performed. The DOC approach is to divide the information space into regions and detect 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. Clusters are recognized within a region if the region has a relatively larger point count than this expected value. Conversely, if the point count is smaller than expected, then these points are potential outliers. However, clusters may cross region boundaries, and this must be taken into account when verifying potential outliers. The process described effectively determines the similarity between data points by the density of points within regions of the information space.

The approach used lends itself to a decentralized implementation because each region can be assigned to a particular processing node. Nodes can be used to analyze the points within their region and communicate with adjacent nodes in order to deal with boundary conditions.

Data points correspond to status or interaction events of distributed system components. The status of a distributed component 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 components may be completely or partially distinct.

Chapter 4

Cluster Tracking

This chapter describes our cluster tracking algorithm in detail and implementation using decentralized online clustering (DOC) algorithm, whose infrastructure is detailed in Chapter 3. This cluster tracking algorithm is created based on two observations, similarities between two or more clusters can be determined by using a clustering algorithm and distributed clustering algorithm can provide online data analysis. The first observation is deduced by the fact that clustering algorithms can categorize data points and a cluster can be described by metadata that can represent a data point. The second is based on our study of current data analysis techniques in Chapter 2 and with our experience with DOC in Chapter 3. The experiments will show that our cluster tracking algorithm is accurate in its tracking of clusters across time[1].

4.1 Algorithm Description

This algorithm is intended to be use on system where data is naturally distributed and have computational capabilities, chapter 5 will go more in depth of such systems by presenting three different use cases. The type of data that this algorithm can process is multidimensional making it effective in tracking data clusters identifed by the generated data in the aforementioned systems. The clusters identified are considered to be categories, object or states, depending on application, and must be tracked in order to determine how the data is changing over time. This is different, as will be shown, than simpling analyzing data at any one point and describe the system using only the most recently processed information. Our approach seek to use recent and

[1]Note that the terms time frame, frame, analysis windows are used interchangeably throughout this paper.

historical information so that there is a better understanding of how these identified clusters moving across time. Thereby allowing the opportunity to determine the why the event is happening and what can be done knowing this information.

4.1.1 Cluster Features

Once a cluster has been identified within a dataset a number of derived metrics and/or metadata can be extracted in order to describe the cluster. These metrics and/or metadata are called cluster features, which describe the data points which form the cluster (e.g location, density, etc.). When cluster features are used properly new methods of interpreting the data can be used as discussed in the following subsections.

Uses in cluster tracking

As will be explained throughout this chapter, cluster features can be used to uniquely identify a cluster. By providing means to uniquely identify cluster it is now possible to compare clusters at different points in time. In addition a path of cluster can be determine along with its trajectory providing the addition benefit of predictive capabilities.

Uses in cluster interpretation

Cluster features can not only be used for cluster identification the can also be used to describe an underlining phenomenon of the data points that have clustered. For example, in the case of datacenter management the features themselves (e.g., cluster's density) can drive decisions (e.g., if a cluster's density is low this can be interpreted as the data points being less related than high density cluster). This makes the collection of cluster feature not only useful from a tracking perspective but also from a decision making perspective.

4.1.2 Clustering Method and Cluster Feature Extraction

As was discrided in chapter 3 DOC is a clustering algorithm which can cluster data points in multidimensional space. Cluster detection in DOC is based on evaluating the

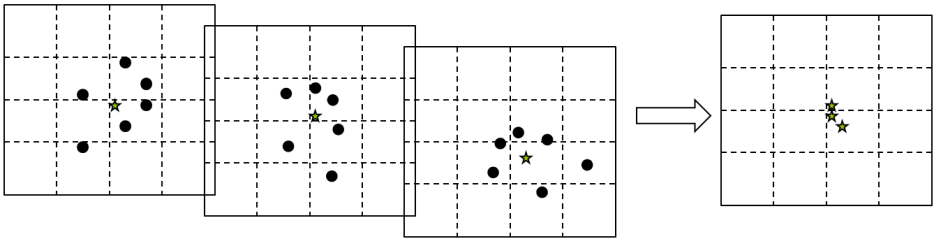

Figure 4.1: Example of recursive clustering using clusters' centriods

relative density of points within the information space. During the process of cluster discovery DOC itself is identifying features of the clusters. These collected cluster features can therefore be leveraged from DOC in order to do cluster tracking without having to create an external or additional code to describe the data clusters. Through experimentation we have found that density, centroid location, and bounding box to be useful cluster feature for track clusters in our datasets and are already determined within DOC.

4.1.3 Cluster Identification using recursive clustering

Differences in features from cluster to cluster allow for comparisons to be made between these clusters. These comparisons can then lead to identification of a cluster over time and provide the path that it has taken. Clustering collected features of processed data provides such a platform for comparison. Simply stated, the key to cluster identification is to determine a similarity threshold between a set of cluster features. If two or more sets of cluster features are sufficiently similar from one point in time to the next, then they can be considered to correspond to the same cluster. As an example Figure 4.1 is provided. In this figure there clusters at different points in time have their centroids collected and subsequently run through DOC. As can be seen the three centroids cluster with one another and conclude that based on this information the three clusters are in fact the same cluster at different times.

Note, however, that if enough feature for clusters a not used it will not be possible to distinguish them. Figure 4.2 is provided to illustrate what can occur if enough features are not used to distinguish clusters. In the first panel two different data set at different times are provided along with the directional arrow of how cluster moved from one time

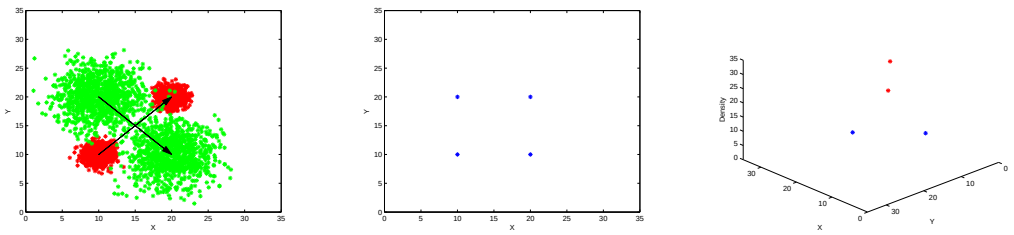

Figure 4.2: Comparison of tracking effectiveness by amount of features used to describe a cluster

period to the next. The second panel presents the four identified centroids, as can be seen the four centroids will most likely cluster together since their spacing[2] is so even among them and relatively close. In the next panel we include a new feature, cluster density, which serve to further differentiate the clusters and two clear groups can be seen, one marked in blue the other in red.

By using these two examples i can be seen that the ability to group data points within a set of points based on similarity, without necessarily having to determine a threshold beforehand, is precisely what clustering algorithms provide. In an information space where each dimension corresponds to a cluster feature, each set of cluster features representing a single cluster would be a single data point. Thus, in the cluster feature space, there will be one point per cluster per point in time. Across a time interval, the points corresponding to the feature sets of a single cluster are expected to cluster in turn since they are sufficiently similar. Therefore, by applying the clustering algorithm to the sets of cluster features over time, the identity of individual clusters in the original information space can be characterized by a single cluster in the cluster feature space. We call this technique recursive clustering since there is an initial clustering step performed on the original data points in the application information space and a recursive clustering step performed on the features in the cluster feature space.

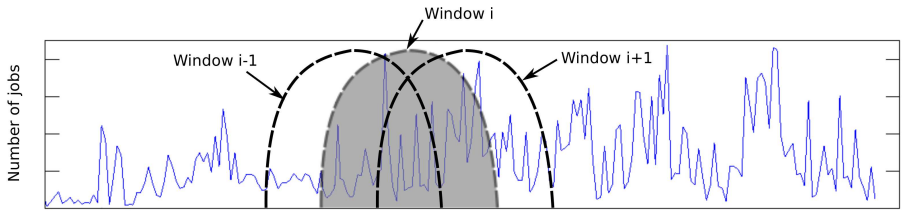

Figure 4.3: Sliding window technique example

4.1.4 Sliding Window

The cluster identification technique described in the previous section assumes that changes in cluster features of an individual cluster are gradual maintaining a certain level of similarity from one point in time to the next. Therefore, we must ensure that this is the case in the sets of points that are clustered in consecutive points in time. To this effect, we use a sliding window, which we refer to as an analysis window. The analysis window is used to capture incoming information from the data stream that will be clustered, save a portion of information while discarding the oldest, and allow new information to replace the discarded information. The analysis window can be described as a large time window that is comprised of several smaller time intervals.

Figure 4.3 shows an example of how the sliding window technique works. The figure shows the number of job arrivals over time of a trace from the Grid Observatory, from one our use cases, and 3 consecutive analysis windows. As can be seen in the figure, window_i contains new job requests from new time intervals, as well as some remaining job requests from intervals contained in the previous window (window_{i-1}), but dropped the oldest time intervals that were contained in that window. The same rule is applied for subsequent windows.

Depending on the size of the time intervals, on the size of the step of the analysis window, and on the size of the analysis window itself, we can ensure that the changes in clusters are gradual from one analysis window to the next. Since the size as well as the step for each window can be changed, the rate of change between analysis windows can always be controlled.

[2]Note the spacing can be closer but for better visibility these clusters are purposely placed relatively far apart.

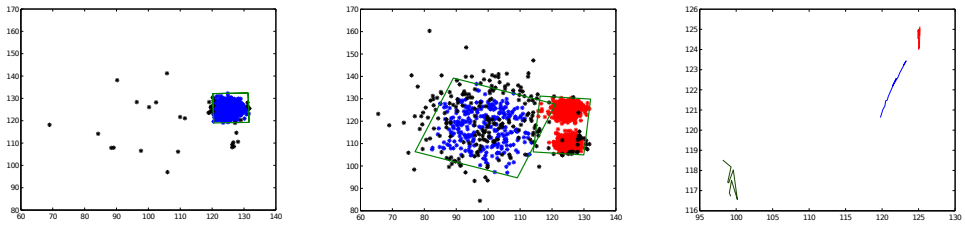

Figure 4.4: Initial cluster data set (left), final cluster data set (middle) and identified paths(right)

Despite the gradual changes afforded by the sliding window technique, eventually the dynamics of the underlying data set will cause large changes in cluster features when compared across many analysis windows, as well as new clusters to appear, and other clusters to disappear. All of these cases reflect changes in system state caused by both gradual and transitory events and conditions. Our feature tracking approach keeps track of the trajectories defined by feature sets across analysis windows, and is thus able to identify all of these changes.

In the next generic example, it will be shown how useful this technique can be in order to show the traceability of clusters over time when taking multiple features into account. In Figure 4.4 it can be seen that there is a data set that contains a single cluster on the left panel and three clusters on the middle panel. However, the clustering algorithm has only identified two clusters, merging the two rightmost clusters together as one. Since our algorithm uses a sliding window the changes between the initial and final results are processed in such a manner that cluster features will change gradually. The resulting effect can be seen in the left panel of Figure 4.4, where there are three paths identified. Since cluster feature change gradually so will the results from the clustering algorithm, this will allow the clustering algorithm consistently provide accurate representations of clusters more often than not.

4.1.5 Infrastructure

Since our tracking mechanism uses a clustering algorithm to determine the similarities between discovered clusters, it is useful to reuse the DOC algorithm for our infrastructure for cluster tracking. As described in Chapter 3 DOC can cluster multidimensional

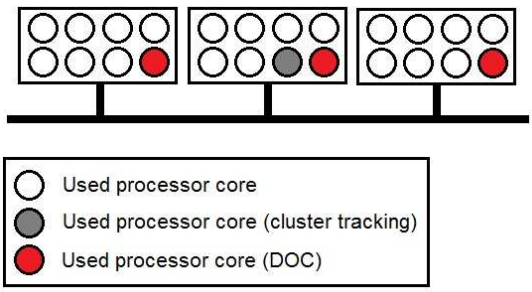

Figure 4.5: Infrastructure for cluster tracking

data which is precisely what is needed when comparing clusters across multiple features. In addition since DOC can read distributed data and can identify clusters in a distributed manner, it is not hard to implement a mechanism that once clusters have been discovered to place the collected feature information on the node where the cluster was discovered. This again is so that data need not be aggregated even in the case of the cluster features themselves. By doing we are consistent with the argument that even cluster tracking can be done in a decentralized manner. Figure 4.5 is provide to demonstrate how the cluster tracking algorithm sits on a master node and them requests all nodes using DOC to begin clustering.

4.1.6 Implementation

Our algorithm use only two components, DOC and external application we have created to identify the path taken by a cluster. The addition component will be explained in the following section but can be summarized as the component which relates clustered features back to the original data cluster which they represent.

4.1.7 Psuedo Code

Algorithm 1 shows the proposed feature tracking algorithm. The following is a step by step explanation to our cluster tracking algorithm. Algorithm 1 takes a dataset as its input and returns a list of cluster paths which were identified. The dataset is partitioned as analysis windows created using the sliding window technique. The cluster paths are given as cluster centroids at each window in which it exists.

Algorithm 1 uses five variables in order to contain all the information needed in order to identify clusters, their features, and their paths. The *clusters* variable has a data structure which can contain the data points of each cluster once the data has been analyzed. The *features* variable has a more complex data structure since its objective is to provide a means to find a analysis window, and a cluster id for any given feature. We found that using a hash table was a good data structure to best describe this variable since its has a key and value data structure. *featureClusters* is just like the *clusters* variable but used to store the clusters of feature for the dataset. *windowAndClusterID* is the variable which contains a list of cluster ids and the analysis window to which they belong to. *paths* is the output of the algorithm.

To begin, each analysis window is clustered using the function *analysisdata* in Algorithm 1. The next step is to extract the features of each individual cluster that has been identified for each analysis window using the *extractFeatures* function. This function extracts any relevant features which are important for the given application e.g. density. These features are then stored into the hash table *features* by means of *insert* function where *features_tmp* is the key and the cluster id along with the analysis window id is value. Note that the combination of features which describe a cluster must be unique inside its analysis window since without this condition it becomes quite complex to relate features back to the cluster which they describe. The next step is to analysis the cluster features which are being held in the hash table *features*. Once again this is done by running *analysisdata* function on *features*. The resulting clusters of features are stored in *clusteredFeatures*. The next step is to link the individual features for each cluster of features back to a cluster id and an analysis window id. This is done by looking at each cluster in *clusteredFeatures* one cluster at a time and subsequently looking at each feature one at a time. At this level each feature is used to search for the cluster id and analysis window id to which this feature belongs to using the *search* function. This information is stored for each cluster of features in *windowAndClusterID*. The final step is to sort cluster ids by analysis window ids to find the path the cluster took through time. This final process is done using the *sortFeatures* and the result is stored in *paths* for each cluster of features.

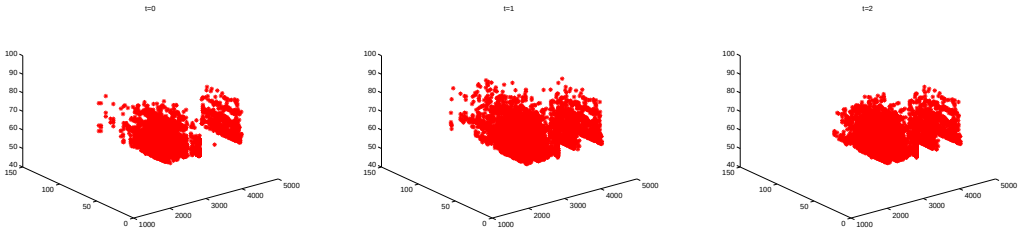

Figure 4.6: Data cluster at different times

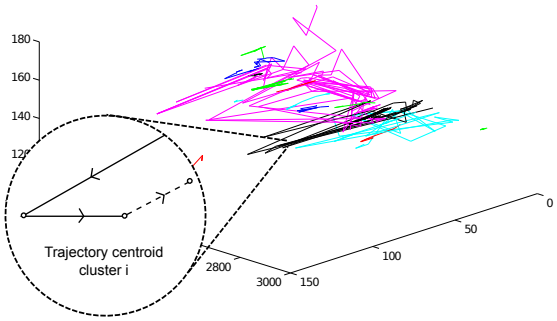

Figure 4.7: Example of cluster trajectories

In order to better illustrate our algorithm we provide Figures 4.6 and 4.7 that have been generated using our experimental data. Figure 4.6 highlights one of the clusters found from the data at three different times and show observable movement as well as clear similarities at each time step. These observable similarities are encompassed in the features of the cluster and make it possible to compare and conclude if a cluster indeed is the same at different time steps. Figure 4.7 illustrates the paths which have been identified for a given dataset as explained in Algorithm 1. A zoom-in is provided to show how the paths gathered from our algorithm can provide a clear directional path.

4.1.8 Work Flow

The following describes the work flow for cluster tracking using our algorithm. In addition Figure 4.8 to provide better detail.

Step 1. A data stream is read, chronologically an analysis window moves along the data and creating snapshots which will simply be called windows. This is done so that the algorithm can analyze individual windows while at the same

Algorithm 1 Cluster tracking algorithm

Input:
k: number of dimensions of the data space
$\{(d_{1^1}, \ldots, d_{1^k}), \ldots, (d_{n^1}, \ldots, d_{n^k})\}$: dataSet
$\{w_1, \ldots, w_m\}$: analysis windows | $w_1 \cup \ldots \cup\ w_m = dataSet$
Output:
paths[ ]: path for each feature extracted

$paths \leftarrow \emptyset$
$clusters \leftarrow \emptyset$ /* Array of set of clusters */
$featureClusters \leftarrow \emptyset$ /* Array of set of clusters */
$features \leftarrow \underline{new}\ Hashtable < feature, window\ set - cluster\ id >$
for each $w_i \in$ analysis windows **do**
  $clusters[] = \underline{analysisdata}\ (w_i)$
  for each $cluster_j \in clusters$ **do**
    $features\_tmp = \underline{extractFeatures}\ (cluster_j)$
    $features.\underline{insert}\ (< features\_tmp, w_i - cluster_j >)$
  end for
end for
$featureClusters[] = \underline{analysisdata}\ (features)$
for each $cluster_p \in featureClusters$ **do**
  $windowAndClusterID.\underline{clear}()$
  for each $feature_q \in cluster_p$ **do**
    $windowAndClusterID.\underline{add}(features.\underline{search}\ (feature_q))$
  end for
  $paths[p] = \underline{sortFeatures}\ (cluster_p, windwAndClusterID)$
end for
return *paths*

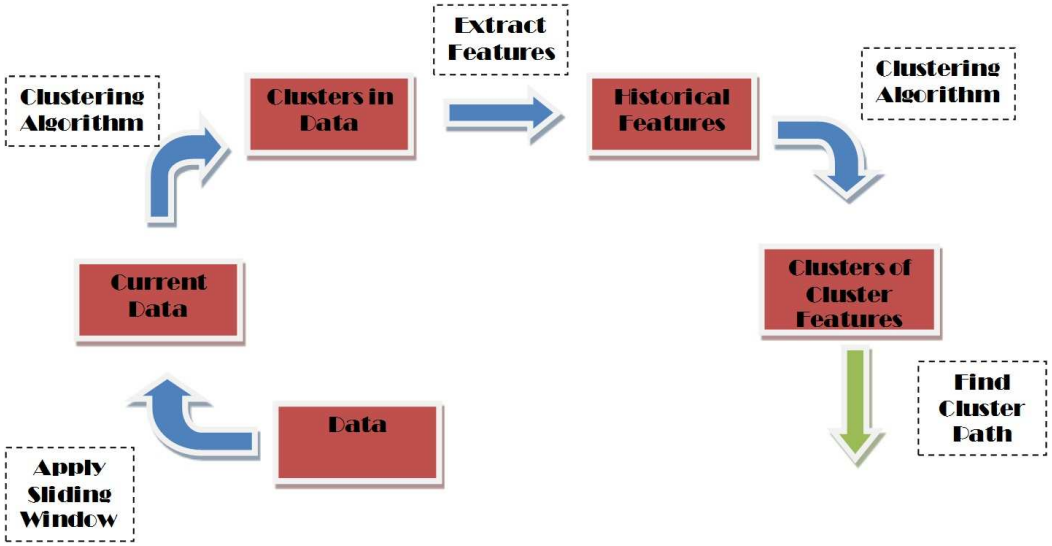

Figure 4.8: Work flow for data cluster tracking

time handle constantly updating information in the form of receiving new data points. The union of all windows are the original data stream but each window is not mutually exclusive as shown in Figure 4.3.

Step 2. Each window is clustered to identify the relationships between data points within that specific window. At this point clusters, which should be gradually changing, have now been discovered and post processing can begin.

Step 3. For every window each individual cluster inside has its features extracted. As mentioned previously and shown enough features must be selected so that clusters can be be unique within their window. By having this unique property, data structures such as hash tables can be used to retrieve data quickly.

Step 4. The cluster features must now be analyzed via clustering to identify how closely related to one another. As explained clustering different features simultaneously is essentially identifying intersections, with thresholds, for each features taken into account.

Step 5. The resulting clusters of cluster features can now be used to recursively identify from where the specific cluster features came from, meaning from what window they were identified. Once each cluster of cluster features has been processed a trajectory for the underlying cluster, which are represented by the clusters features, can begin to be identified.

Step 6. The final step is to relate each cluster feature back to its underlying cluster. Once this has been completed a path is formed by using the centroids of each cluster.

Algorithm 1 shows the algorithm of the approach described in the previous steps for a given input data set. Algorithm 1 has one input, *dataSet*, which is comprised of many data points. The input is also interpreted as many windows where each window is comprised of many data points. Windows can overlap with their data point and the union of all windows is the original data set. The size of the window, either by time

elapsed or total data points, is determined ahead of time as well as how many windows must exist, before running Algorithm 1 as described in (Step 1).

Accuracy

We define *tracking accuracy* to mean that by using the clusters which have an identifiable over a specific dataset a certain percentage of all data points in that dataset will be encompassed. In other words, if the clusters are truly being tracked correctly most of the points should lay within the movement of these clusters.

Chapter 5

Applications for Distributed Cluster Tracking

5.1 VM Provisioning

In previous work [25, 27], autonomic mechanisms for VM provisioning to improve resource utilization were presented. This approach focused on reducing the over-provisioning that occurs because of the difference between the virtual resources allocated to VM instances and those contained in individual job requests. In particular, clustering was used to efficiently characterize dynamic, rather than generic (such as Amazon's VM types[1]), classes of resource requirements to be used for proactive VM provisioning. Other existing techniques addressed efficient and on-demand resource provisioning in response to dynamic workload changes but using different approaches [22, 31, 30, 36].

In [25, 27], the flow of arriving jobs was also divided into analysis windows. However, these windows did not contain overlapping intervals as in our current approach. During each window, an instance of the clustering algorithm was run with the jobs that arrived during that window, producing a number of clusters or VM classes. At the same time, each job was assigned to an available VM class as it arrived. The provisioning was done based on the most recent analysis results from the previous analysis window. For the first window (ramp up time), VMs had to be created reactively for incoming jobs, but the clustering results were used for subsequent analysis windows. According to [24], the time required to create batches of VM in a cloud infrastructure does not differ significantly from the time for creating a single VM instance. Thus, the VMs for each class could effectively be provisioned within the given time window. The evaluation

[1] Amazon EC2, http://aws.amazon.com/ec2

of the approach showed that using the dynamic VM classes significantly reduced over-provisioning when compared to the use of statically provisioned VM classes.

In this work, we extend this idea using cluster tracking and cluster feature tracking. In contrast to the previous work, the clustering results from a previous analysis window are no longer used directly to obtain the VM classes for provisioning during each current window; instead, the feature trajectories found by the new approach are used to project the previous results into each current window. We expect that these predictions of VM classes will be more accurate, since they take into account the change history of the state of requests, and thus result in a further reduction in over-provisioning cost. We also take into account under-provisioning costs, which are due to outliers and may cause different problems [3].

5.1.1 Evaluation

In this work, we have used traces from the Grid Observatory[2], which collects, publishes, and analyzes data on the behavior of the EGEE Grid[3]. This trace meets our needs because it currently produces one of the most complex public grid traces from a large-scale distributed grid infrastructure, with multiple geographically distributed entry points and high arrival rates. The frequency of job request arrivals in terms of number of requests per unit of time is much higher in contrast to other large Grids such as Grid5000.

In previous work [25, 27], autonomic mechanisms for VM provisioning to improve resource utilization were presented. This approach focused on reducing the over-provisioning that occurs because of the difference between the virtual resources allocated to VM instances and those contained in individual job requests. In particular, clustering was used to efficiently characterize dynamic, rather than generic (such as Amazon's VM types[4]), classes of resource requirements to be used for proactive VM provisioning. Other existing techniques addressed efficient and on-demand resource

[2]Grid Observatory, http://www.grid-observatory.org/

[3]Enabling Grid for E-sciencE, http://www.eu-egee.org/

[4]Amazon EC2, http://aws.amazon.com/ec2

provisioning in response to dynamic workload changes but using different approaches [22, 31, 30, 36].

In [25, 27], the flow of arriving jobs was also divided into analysis windows. However, these windows did not contain overlapping intervals as in our current approach. During each window, an instance of the clustering algorithm was run with the jobs that arrived during that window, producing a number of clusters or VM classes. At the same time, each job was assigned to an available VM class as it arrived. The provisioning was done based on the most recent analysis results from the previous analysis window. For the first window (ramp up time), VMs had to be created reactively for incoming jobs, but the clustering results were used for subsequent analysis windows. According to [24], the time required to create batches of VM in a cloud infrastructure does not differ significantly from the time for creating a single VM instance. Thus, the VMs for each class could effectively be provisioned within the given time window. The evaluation of the approach showed that using the dynamic VM classes significantly reduced over-provisioning when compared to the use of statically provisioned VM classes.

In this work, we extend this idea using cluster tracking and cluster feature tracking. In contrast to the previous work, the clustering results from a previous analysis window are no longer used directly to obtain the VM classes for provisioning during each current window; instead, the feature trajectories found by the new approach are used to project the previous results into each current window. We expect that these predictions of VM classes will be more accurate, since they take into account the change history of the state of requests, and thus result in a further reduction in over-provisioning cost. We also take into account under-provisioning costs, which are due to outliers and may cause different problems [3].

5.1.2 Experimental setup

The first set of experiments uses k-means as our clustering algorithm in order to have a completely controlled baseline for under and over-provisioning, since this eliminates the possibility of having more clusters in a final result lead to lower over-provisioning. k-means was fix to provide 4 clusters, our analysis window is fixed at 3,000 job requests

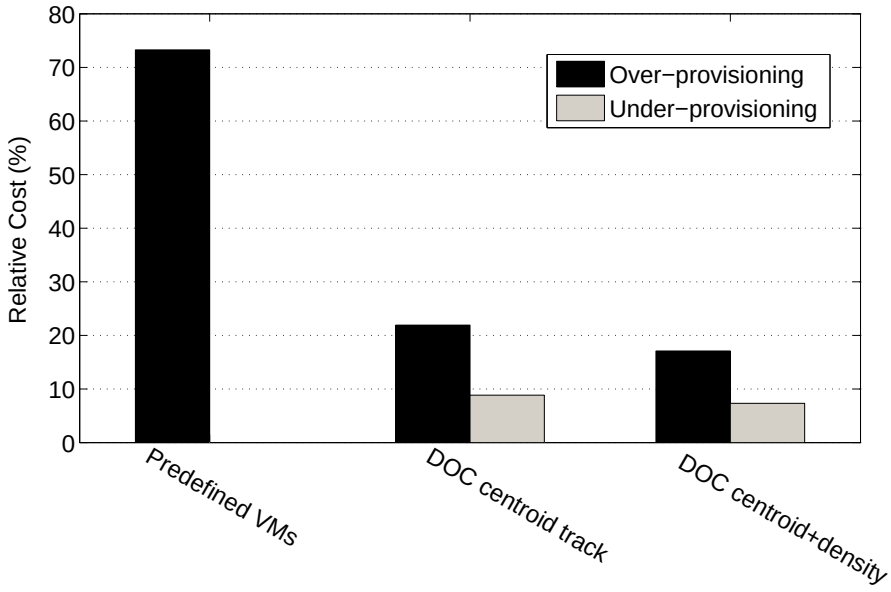

Figure 5.1: Relative over- and under-provisioning cost

and consecutive windows differ by 300 job requests and 30 and 50 windows are taken are used to make the prediction. The baseline considered is the over-provisioning cost for each window when the information is clustered into 4 clusters with no outliers. The other comparison is the over-provisioning cost using predefined VMs which are based on looking at all the information. Then taking the maximum and middle value for each resource and creating predefined VMs with a combination this information. The information used to determine the parameters for the VM are requested CPUs, and requested memory. In Figure5.1 the results in over- and under-provisioning can be seen in terms or percents for the following schemes:

- *Predefined VMs*: provisioning based on fixed types of VM classes.
- *DOC centroid track*: proactive (predictive) provisioning based on cluster tracking using DOC and considering only the centroid as a feature.
- *DOC centroid+density track*: proactive (predictive) provisioning based on cluster tracking using DOC and considering centroid and density features.

From Figure 5.1 it is clearly shown that using predefined VM parameter has the highest over-provisioning, as to be expected. The over-provisioning for clustering each

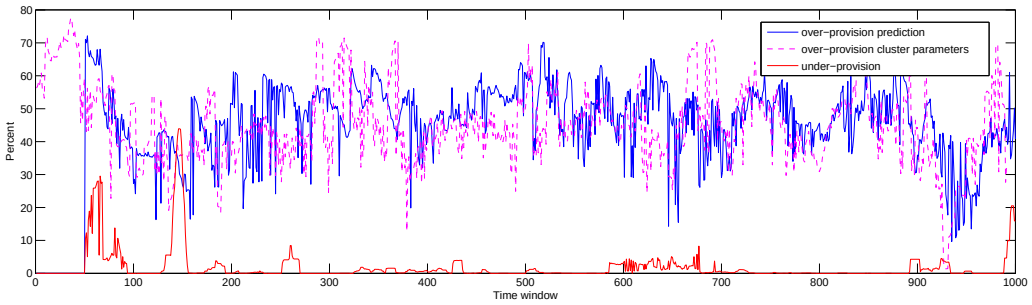

Figure 5.2: Under and over-provisioning comparison between feature tracking and clustering

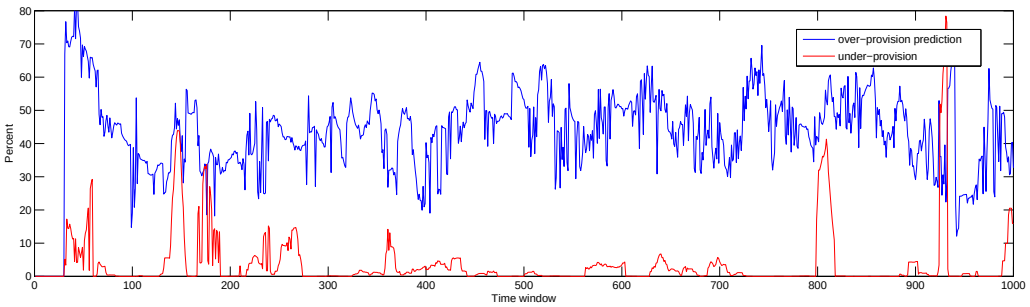

Figure 5.3: Under and over-provisioning using 30 analysis windows

individual window of data points was around 45%. When tracking the clusters over the span of 50 windows the over-provisioning was around 46% and the incurred under-provisioning was around 2%. The feature used to track the cluster was the bounding box of the cluster, meaning the maximum value for all the dimensions in the cluster, CPUs and memory.

In Figure 5.2 a direct comparison is made window by window with clustering data in the window and prediction using cluster tracking over a span of 1000 windows. As can be seen the 2 over-provisioning results follow each other closely indicating that the prediction itself is accurate to the point of providing similar over-provisioning cost. This leads to the conclusion that doing proactive management of the datacenter in this manner, in term of creating VM ahead of the upcoming request, does not lead larger over-provisioning. What is also shown is the under-provisioning cost incurred by predicting upcoming job request. It is clearly shown that for the under-provisioning cost are low, 2%, the majority of the time. We believe the spikes in under-provisioning can

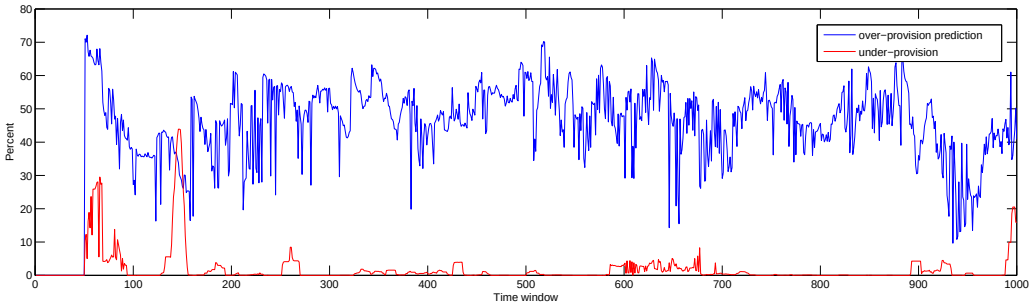

Figure 5.4: Under and over-provisioning using 50 analysis windows

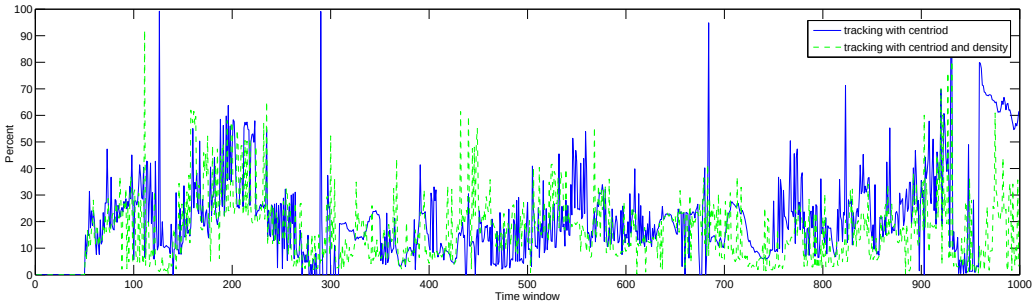

Figure 5.5: Under and over-provisioning comparison between number of features used to identify a cluster

happen for various reasons, two are: 1) outlier jobs which request more resources that normal 2) a new trend towards using more resources is emerging. In the case of 1) there is no clear defined way as to how this can be mitigated with cluster tracking. In the case of 2) more aggressive approaches for cluster tracking predictions can be made, for example the velocity at which a cluster is moving along the information space. Another possibility is to simply decrease the variance in the under- and over-provisioning while simply accepting the spike cost of under-provisioning.

For Figures 5.3 and 5.4 the afore mentioned effects of decreasing variance in under-provisioning is shown. The experiment consist of changing the required number of windows to be looked at before making a prediction. The finds were that it may be possible to reduce the variance within the over and under-provisioning for the prediction of the upcoming job request by simply increasing or decreases the number of data which are processed before making the predictions. As mentioned earlier a window can be based on time elapsed or number of data points, so there is a possibility of creating a

mechanism which can trigger the need to view more or less windows, directly effects the time before a prediction can be made. For instance, a predefined variance threshold is set in which if the variance is above the threshold more windows must be used before making a prediction and if below less windows can be used before making a prediction, a soft threshold can be used to not constantly change the number of windows. The result of this is a system which can adjust itself to mitigate the negative effects of under-provisioning by decreasing the spikes by simply waiting longer to make a prediction if needed and speed up if it can confidently make a prediction. As seen in the in figures this is clearly shown, in Figure 5.3, 30 windows must be used before making a prediction and in Figure 5.4, 50 windows must be used before making a prediction. There are less spikes in under-provisioning in Figure 5.4 than in Figure 5.3 thereby supporting the assertion that waiting to make a prediction leads to lower variance in under-provisioning.

These results however would rely on a centralized clustering algorithm like k-means which does not have the robustness and allow for distribution of the overhead like DOC, which can manage large-scale data efficiently and in a timely manner. For this reason we ran separate experiments running DOC. In these experiments we used DOC with the same parameters as k-means in terms of changing points in each window, 300, total points, 3,000, and number of windows taken into account before prediction, 50. As stated before the amount and quality of clusters directly effect the amount of over-provisioning cost. This is important to state because without this knowledge results may be misinterpreted such as Figure 5.1, in previous work this was shown [27]. For this reason the following results compare the benefits, in terms of reduction in over-provisioning, when adding extra features to describe a cluster's trajectory.

In the experiments involving DOC we looked to add on what was already learned in the original k-means experiments. In these experiments we compare over-provisioning when using 1 feature to track a clusters and when using 2 features to track a clusters. In Figure 5.5 a comparison of the over-provisioning is shown. As can be seen by only looking at the graph the two curves are very similar. From Figure 5.1 it is clear that there is a reduction in over-provisioning when including a new cluster feature. What

can be concluded from this is that better cluster identification is made when using more features, which is expected and explained as to why this occurs in Section **??**. The curves are similar but as seen from these results adding a new features decreased the average over-provisioning.

5.2 Hadoop Resource Scheduling

Consider an analytics application running as a MapReduce job on a hadoop cluster. At each step of the analytical application data about the tasks being performed can be collected. Since this collected data is in nature distributed among the nodes, the processing of management actions can be done in-situ. This in-situ processing follows naturally from the parallel and staged computation of analytics applications which is also being performed on the node. In a setup such as this the ability to do data clustering in an online manner emphasized. As the analytics application completes its tasks the monitoring information can be clustered using DOC. Note that distribution is an advantage, but not necessarily a requirement of this approach. As clusters form and are identified from the monitoring data at each node, the cluster features information can be used for the subsequent tracking process of our algorithm. In conjunction with categorical information such as node ID, application, and other configuration parameters, particular management actions can be executed or created with this information. For example, a cluster that is made up only of points from a particular application or run can be used as a prediction for subsequent runs of that application. On the other hand, a cluster made up of points from multiple nodes can be used as an indication of load balancing among the nodes and thus as a cue for scheduling.

The evaluation presented in this paper is a proof-of-concept example of the above scenario, meant specifically to demonstrate the predictive power of clustering for the runtime of an analytics application. Since our experiments only considered one analytics applications using Hadoop (LDA) we do not provide a load balancing scenario.

5.2.1 Evaluation

In order to evaluate this use case we collected monitoring data while running Mahout's LDA analytics application under different configurations when processing Wikipeida data dumps. Specifically we tested using all the below configurations in all combinations:

- 10 different wikipedia data dumps, each represent a different wiki site.
- Have LDA identify 4 or 5 topics.
- Allow LDA to iterate on the file 1 to 4 times.
- Configure Hadoop to use 1 to 5 worker nodes.

Monitoring data while these experiments where running was taken every 30 seconds to provide and abundant amount of data points as well as illustrate how different steps use more resources than others. The monitoring data from LDA runs is used to create a real world situation in which if interesting information is extracted from LDA the file is then further processed a finer granularity in terms of iterations.

For these experiments we found that clustering 10,000 data points and sliding our analysis window along 1,000 data points, meaning removing 1,000 points and inserting 1,000 was effective for tracking clusterings. In addition we used 50 analysis windows at a time to perform our tracking algorithm, in total this amounted to running the algorithm 150 times. The accuracy of our algorithm is determined by percentage of data points that are captured by the trajectories of the clusters which have been identified using our tracking algorithm. This serves as baseline to confirm that the algorithm is identifying meaningful clusters and their movements over time. In addition we provide the error of our prediction of the upcoming centroids of the currently identified clusters. This metric serves to determine the effectiveness of the prediction capabilities of our approach.

The experimental evaluation for scheduling Hadoop resources focus on the accuracy of the proposed techniques. Specifically, the *tracking accuracy* of our algorithm in this use case was determined to be 71.52% when averaging 46 of the 51 of the datasets used for cluster tracking. Figure 4.7 shows a sample of identified cluster paths for a dataset

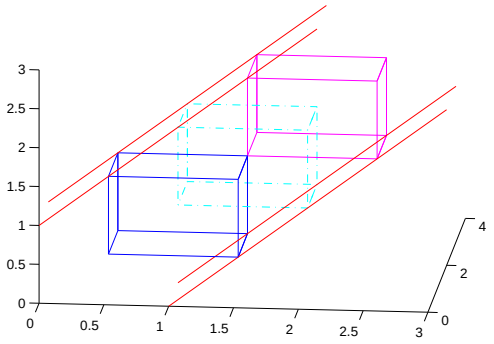

Figure 5.6: Cluster Path Cone

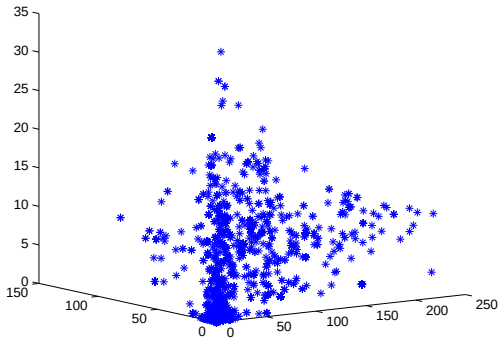

Figure 5.7: Absolute error of all clusters predictions in experiment in terms of distance

and figure 5.6 illustrates how we calculate accuracy. Figure 5.7 shows the accuracy of the predictions. Each data point represents the average prediction error of all clusters' predictions in terms of distance, which means that a prediction was made for all clusters that had a path identified for them. Using this identified path at each change of position is predicted based on the past movement, previous position to current position.

5.3 Scientific Visualization

Scientific simulations running at extreme scale on leadership class systems are generating unprecedented amount of data. To enable scientific discovery, the high amount of simulation data has to be analyzed and understood by domain scientists. However, the

increasing gap between computation and disk IO speeds makes traditional data analytics pipelines based on post-processing cost prohibitive and often infeasible [7]. Storing entire datasets from from the large scale systems running the simulation to storage servers is becoming increasingly expensive in terms of the time required as well as the energy costs associated with data movement. Moreover, the efficiency and scalability of subsequent analysis operations on the data are also hindered by the cost of disks-based data I/O. This trend of big simulation data is resulting significant challenges limiting the ability of scientists to translate this data into insights, and as a result, the impact of the simulations themselves. Clearly, this required rethinking the analytics pipelines to incorporate new approaches to data analytics that are cost-effective and scalable. In-situ data processing approaches have recently emerged as a promising approach. These techniques can effectively reduce data movement and data IO overheads by placing analysis operations at the simulation machines closer to where the data is being produced.

We investigate this alternate approach that aims to bring the analytics closer to the data using the in-situ execution of data analysis operations. Specifically we explore in-situ cluster-based feature tracking in distributed scientific datasets. In order to extract insightful from the large datasets produced by simulations over thousands of time steps, scientists often need to follow data objects of interest (i.e., features) across the different time steps, such as tracking storm formation and movement in climate modeling simulation, or identify burning regions in combustion simulations. As a result, feature extraction and tracking is an important technique for analyzing and visualizing scientific datasets. However, most feature extraction and tracking techniques operate offline by post-processing data dumped from the simulation runs. Being able to perform such feature-based analytics online, i.e., concurrent with a simulation run, and in-situ can significantly increase the utility of these techniques and the productivity of the simulations, and also lead a better utilization of expensive high-end resources.

5.3.1 Evaluation

The prototype implementation of our framework was evaluated on the Lonestar linux cluster at Texas Advanced Computing Center (TACC). The Lonestar has 1,888 compute nodes, and each compute node contains two hex-core Intel Xeon processors, 24GB of memory and a QDR InfiniBand switch fabric that interconnects the nodes through a fat-tree topology. The system also supports a 1PB Lustre parallel file system.

Our evaluation presented in this section consists of two parts. The first part evaluates the end-to-end data transfer performance of our in-situ data analysis framework, and also compared it with the traditional disk IO approach. The second part evaluates the effectiveness and accuracy of our DOC-based feature tracking algorithm, using time-varying dataset generated by simulation of coherent turbulent vortex structures.

5.3.2 Performance of Data Transfer

This section evaluates the end-to-end data transfer performance, and more specifically the time used to transfer data from simulation processes to DOC workers, for both our in-situ memory-to-memory and the disk IO approaches. In this case, we use a testing MPI program as the parallel data producing simulation, which runs on a set of m processor cores. The parallel DOC workers runs on a separate set of n processor cores where the ratio of m:n is 10. In our in-situ data analysis approach, each DOC worker runs on a processor core co-located with 10 simulation cores of the same compute node, and retrieves data generated by the 10 intra-node simulation processes through CoDS *get_local()* interface. In the disk IO approach, simulation processes dump data to disk with the one file per process method using binary POSIX IO operations. Data files are then read by parallel DOC workers. For this evaluation, the number of simulation processes m is varied from 50 to 800, and the size of data produced per simulation process at each timestep is varied from 1MB to 64MB. The testing program is configured to run for 100 timesteps at each data output size.

Figure 5.8 and 5.9 compares the performance of the two evaluated end-to-end data transfer approaches. As shown in Figure 5.8, our in-situ memory-to-memory method is

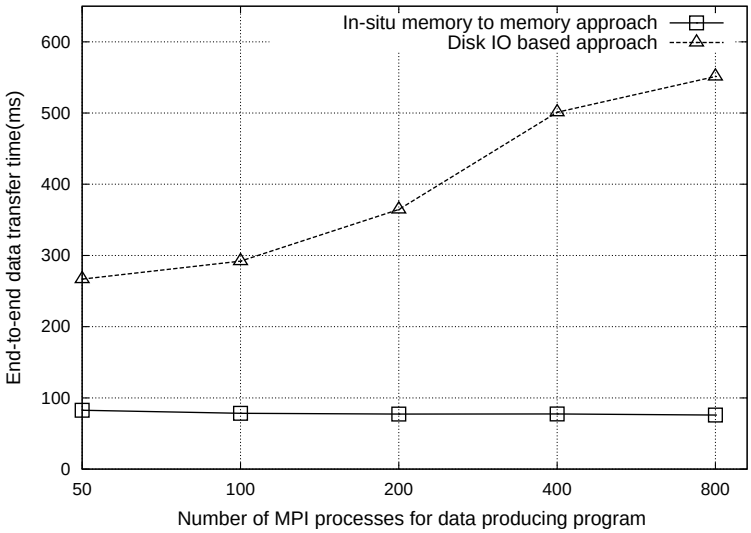

Figure 5.8: End-to-end data transfer time in millisecond. The size of data produced per simulation process at each timestep is 16MB.

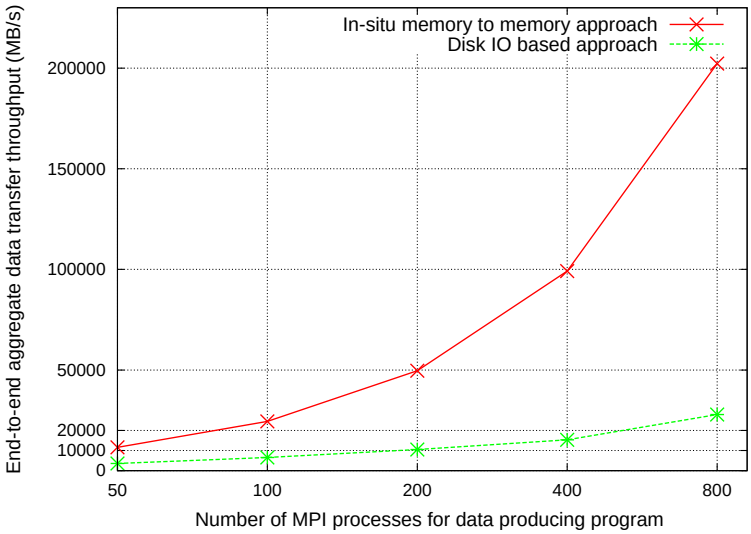

Figure 5.9: Aggregate data transfer throughput. The size of data produced per simulation process at each timestep is 16MB.

much faster than the disk IO approach, with average speedup of transfer performance as about 5. Also, the in-situ memory-to-memory method is scalable, and shows no performance degradation when the number of MPI processes in data producing program increases from 50 to 800. The reason accounts for this significant performance gain is that all data movement is intra-node, and performed through the on-node fast IO path - shared memory. But for the disk IO approach, both data producer and DCO worker processes have to use the off-node slow path - disk. Figure 5.9 illustrates the performance gain from another dimension - aggregate data transfer throughput. The fast intra-node shared memory approach enables much higher aggregate bandwidth for the data movement between simulation and DOC.

5.3.3 Effectiveness of the DOC-based Feature Tracking

This section evaluates the effectiveness and accuracy of our proposed feature tracking algorithm, using time-varying 3D dataset generated by simulation of coherent turbulent vortex structures with 128^3 resolution (vorticity magnitude) and 100 time steps. In this case, the feature of interest is defined as thresholded connected voxel regions. The tracking information from our algorithm is used to determine how the feature evolves, e.g. size, location, density, over the time steps.

For these experiments the we define *tracking accuracy* as the ratio of vortex points encompassed by the tracked objects and total number of vortex points in the observed times steps. Objects are not associated with tracked object represent vortex points which will not be encompassed and lower the tracking accuracy. To test the tracking accuracy 50 frames of our scientific data were used to identify the paths of the objects within these 50 frames. The tracking accuracy average across 47 tests was 92.28%, meaning only 7.72% of all vortex points were not associated to any trackable object in our experiments.

In Figure 5.10 we present an example of a object, as seen by DOC, at 3 different times. As can be seen this object is moving from left to right . We include figure which shows this object as seen by VIST, visualization software for scientific data.

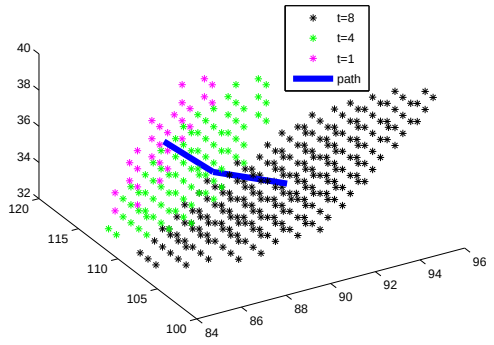

Figure 5.10: Identified cluster path in scientific data

Chapter 6

Conclusion

6.1 Thesis Summary

In this work we have presented a decentralized cluster tracking algorithm, that seeks to continue to reduce the overall resource expense when compared to its centralized counterpart. It was shown that the need to centralize data before processing, in our case the process was to tracking clusters, has been reduced, creating the opportunity to reduce resource cost which are incurred by typical data analytic algorithms. We presented a comprehensive overview of our algorithm by providing a justification, the distributed framework, tracking mechanism, an evaluation and real-world use cases. Making our approach more compelling at addressing the extremely challenging problem of processing data from large-scale distributed systems in order to characterize and predict operational states to manage the systems efficiently. In addition this work explored the in-situ execution of feature-based objects tracking of time-varying scientific simulation data providing a platform for new methods of real-time interaction with simulations.

Our algorithm can be quickly summarized in 4 steps. Cluster data using a distributed clustering algorithm, eliminating the need to centralize the data being clustered. Store metadata of clusters, reducing storage requirements for historical information to a fraction of the original data cluster, thereby uniquely identifying clusters across time. Cluster the collected metadata in order to determine how clusters at different times compare to on another, following the assumption that if cluster have similar features, their features will cluster. Finally use the clustered metadata to determine a the clusters' locations across time. In certain situations we proved that an additional step of trajectory projection is straightforward and added for prediction perposes.

In order to create our infrastructure we leveraged DOC providing a distributed framework for clustering information. Providing a robust, fault tolerant and scalable framework for clustering data in distributed systems found in HPC. In addition by using a cluster tracking algorithm which uses a "recursive clustering" mechanism the same DOC framework can be reused to do cluster tracking. This in effect eliminates the need to have data centralized not only to categorizing information but also to track patterns in data across making this a general purpose data analytics algorithm to be used across different fields.

In our work we presented three different cases where our cluster tracking algorithm proved to be effective. In our LDA experiments the accuracy of our algorithm was tested in both its ability to track clusters and make a prediction to where these clusters were moving towards. By showing the ability to do both tasks effectively, the scheduling of Hadoop tasks is feasible. For our VM provisioning use case shows that tracking clusters of job requests can lead to a reduction in over-provisioning, which in turn has a traded-off of a small amount request going under-provisioned. In our evaluation we were able to prove the accuracy of our algorithm along many metrics specific to the presented use cases. In our scientific object tracking the proposed use case was conducted on Lonestar cluster at TACC, which include evaluation of the end-to-end data transfer performance and the effectiveness and accuracy of our cluster tracking algorithm. These use cases have served to prove the feasibility of doing such in-situ data analytics and serve as a building block for more algorithms like it.

The key contributions of this paper are: 1) an approach, based on feature extraction and recursive clustering, for identifying and tracking clusters (and thus patterns in information) across time; 2) an algorithm for applying the approach to streams of system data, using a distributed clustering algorithm and sliding window mechanism; 3) the demonstration of the feasibility of the approach in real world use cases.

Specifically for the latter contribution, we have shown that tracking clusters in different types of information in a distributed manner can enhance decision making in different situations. For VM provisioning better understanding job requests via cluster tracking can lead to a reduction in over-provisioning, with the traded-off of increasing a

controllable amount of under-provisioning. In our LDA experiments effectively tracking clusters enables us to correctly predict execution times of the process, thereby provide information to a task scheduler in order to determine where the next task should be executed in the data center. Finally our object tracking in scientific applications has provide a new means of visualizing data without the need of secondary computing infrastructure.

6.2 Observations and Future Work

There are many possible directions for future work in the area of in-situ data analytics as well as for our proposed algorithm. For the tracking algorithm itself it will be interesting to include new metrics such as velocity and acceleration of features along tracked trajectories in order to fine tune the predictive capabilities of the approach. The inclusion of new features will also affect other metrics such as the variance in the accuracy of predictions as well as spikes. Preliminary results suggest that variance of the accuracy is affected by the number of analysis windows used to determine cluster paths and will be important to see how the different parameter affect the results. In addition to the accuracy of the tracking of clusters new use cases will be interesting to explore. One of the most interesting of these is the possible use of our approach in other areas such as federated cloud data exchange, in which a cloud provides information to another and can predetermine if the data should be sent or not based on prior requests. Another possible direction for future work is in-situ data analytics in scientific computing so that real-time visualization can be done on a specific object that could be identified through clustering. We will also explore the use of on-node SSD storage device to support energy-efficient in-situ staging of large data sets which could not simply stored in current on-node memory. In addition to those scenarios we will also look at meaningful scenarios in which the tracking of data clusters could be useful to develop autonomic management techniques, such as adaptive policy management or scheduling/placement policies to reduce hotspots, reduce over-provisioning even further and possibly reduce energy costs. Finally, a large scale demo of many of our current use cases will be important in order to demonstrate the cost savings in a large scale

environment.

References

[1] A. J. Abrantes and J. S. Marques. A method for dynamic clustering of data. In *British Machine Vision Conference*, 1998.

[2] C. C. Aggarwal, T. J. Watson, R. Ctr, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *In VLDB*, pages 81–92, 2003.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53:50–58, April 2010.

[4] J.-M. F. Brad Whitlock and J. S. Meredith. Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In *Proc. of 11th Eurographics Symposium on Parallel Graphics and Visualization (EGPGV'11)*, April 2011.

[5] M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 30–39, 2003.

[6] J. Chen, D. Silver, and M. Parashar. Real-time feature extraction and tracking in a computational steering environment. In *Proc. of Advanced Simulations Technologies Conference (ASTC'03)*, 2003.

[7] H. Childs. Architectural Challenges and Solutions for Petascale Postprocessing. *Journal of Physics: Conference Series*, 78(1):012012, 2007.

[8] P. Domingos and G. Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In *18th International Conference on Machine Learning*, pages 106–113, 2001.

[9] N. Fabian, K. Moreland, D. Thompson, A. Bauer, P. Marion, B. Gevecik, M. Rasquin, and K. Jansen. The paraview coprocessing library: A scalable, general purpose in situ visualization library. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 89 –96, oct. 2011.

[10] A. Filali, A. S. Hafid, and M. Gendreau. Adaptive resources provisioning for grid applications and services. In *IEEE International Conference on Communications (ICC'08)*, pages 186–191, 2008.

[11] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: a review. *SIGMOD Rec.*, 34:18–26, June 2005.

[12] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE Trans. on Knowl. and Data Eng.*, 15(3):515–528, Mar. 2003.

[13] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. pages 359–366, 2000.

[14] A. Q. Hernandez. *Decentralized Online Clustering for Supporting Autonomic Management of Distributed Systems.* PhD in Electrical and computer engineering, Rutgers, The State University of New Jersey, 2010.

[15] N. Jiang, C. Schmidt, V. Matossian, and M. Parashar. Enabling applications in sensor-based pervasive environments. In *In Proc. of BaseNets 2004, SanJose,CA*, pages 871–883, 2004.

[16] E. Kalyvianaki, T. Charalambous, and S. Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *6th international conference on Autonomic computing*, ICAC '09, pages 117–126, 2009.

[17] J. Kim, H. Abbasi, L. Chacon, C. Docan, S. Klasky, Q. Liu, N. Podhorszki, A. Shoshani, and K. Wu. Parallel in situ indexing for data-intensive computing. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 65 –72, oct. 2011.

[18] S. Kundu, R. Rangaswami, K. Dutta, and M. Zhao. Application performance modeling in a virtualized environment. In *16th International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–10, 2010.

[19] S. Lakshminarasimhan, J. Jenkins, I. Arkatkar, Z. Gong, H. Kolla, S.-H. Ku, S. Ethier, J. Chen, C. Chang, S. Klasky, R. Latham, R. Ross, and N. Samatova. Isabela-qa: Query-driven analytics with isabela-compressed extreme-scale scientific data. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1 –11, nov. 2011.

[20] K.-L. Ma. In Situ Visualization at Extreme Scale: Challenges and Opportunities. *IEEE Computer Graphics and Applications*, 29(6):14–19, 2009.

[21] F. Machida, M. Kawato, and Y. Maeno. Just-in-time server provisioning using virtual machine standby and request prediction. In *5th International Conference on Autonomic Computing*, ICAC '08, pages 163–171, 2008.

[22] D. A. Menasce and M. N. Bennani. Autonomic virtualized environments. In *Intl. Conf. on Autonomic and Autonomous Systems*, page 28, 2006.

[23] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering, 2001.

[24] S. Osternmann, A. Iosup, N. Yigibasi, R. Prodan, T. Fahringer, and D. Epema. An early performance analysis of cloud computing services for scientific computing. Technical report, Delft University of Technology, December 2008.

[25] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma. Towards autonomic workload provisioning for enterprise grids and clouds. In *IEEE/ACM Intl. Conf. on Grid Computing*, pages 50–57, 2009.

[26] A. Quiroz, M. Parashar, N. Gnanasambandam, and N. Sharma. Autonomic policy adaptation using decentralized online clustering. In *Proceedings of the 7th international conference on Autonomic computing*, pages 151–160, 2010.

[27] I. Rodero, J. Jaramillo, A. Quiroz, M. Parashar, F. Guim, and S. Poole. Energy-efficient application-aware online provisioning for virtualized clouds and data centers. In *International Conference on Green Computing*, GREENCOMP '10, pages 31–45, 2010.

[28] C. Schmidt and M. Parashar. Flexible information discovery in decentralized distributed systems. In *in Proceedings of the 12th High Performance Distributed Computing (HPDC*, pages 226–235, 2003.

[29] D. Silver and X. Wang. Tracking and visualizing turbulent 3d features. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):129–141, Apr. 1997.

[30] Y. Song, Y. Sun, H. Wang, and X. Song. An adaptive resource flowing scheme amongst vms in a vm-based utility computing. In *IEEE Intl. Conf. on Computer and Information Technology*, pages 1053–1058, 2007.

[31] A. A. Soror, U. F. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath. Automatic virtual machine configuration for database workloads. In *ACM SIGMOD Intl. Conf. on Management of data*, pages 953–966, 2008.

[32] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. Agile dynamic provisioning of multi-tier internet applications. *ACM Trans. Auton. Adapt. Syst.*, 3:1–39, March 2008.

[33] C. Wang, K. Schwan, V. Talwar, G. Eisenhauer, L. Hu, and M. Wolf. A flexible architecture integrating monitoring and analytics for managing large-scale data centers. In *Proceedings of the 8th ACM international conference on Autonomic computing*, ICAC '11, pages 141–150, 2011.

[34] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. On the use of fuzzy modeling in virtualized data center management. In *4th International Conference on Autonomic Computing*, pages 25–25, 2007.

[35] H. Yu, C. Wang, R. Grout, J. Chen, and K.-L. Ma. In Situ Visualization for Large-Scale Combustion Simulations. *IEEE Computer Graphics and Applications*, 30(3):45–57, 2010.

[36] J. Zhang, J. Kim, M. Yousif, R. Carpenter, and R. J. Figueiredo. System-level performance phase characterization for on-demand resource provisioning. In *IEEE International Conference on Cluster Computing*, CLUSTER '07, pages 434–439, 2007.