Empirical Evaluation and Architecture Design for Big Monitoring Data Analysis

Samneet Singh

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science () at

Concordia University

Montréal, Québec, Canada

August 2016

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Samneet Singh**

Entitled: **Empirical Evaluation and Architecture Design for Big Monitoring Data Analysis**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science ()

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

______________________________ Chair
Dr. R. Raut

______________________________ External Examiner
Dr. Roch H. Glitho

______________________________ Examiner
Dr. Wahab Hamou-Lhadj

______________________________ Supervisor
Dr. Yan Liu

Approved by ______________________________
W. E. Lynch, Chair
Department of Electrical and Computer Engineering

____________ 2016

____
Amir Asif, Dean
Faculty of Engineering and Computer Science

Abstract

Empirical Evaluation and Architecture Design for Big Monitoring Data Analysis

Samneet Singh

Today, monitoring data analysis is an underrated but important process in tech industries. Almost, every industry gathers and analyzes monitoring data to improve offered services or to predict critical issues in advance. However, the monitoring data constitutes V's of big data (i.e. Volume, Variety, Velocity, Value, and Veracity). Exploration of big monitoring data possess several issues and challenges. Firstly, a wide range of monitoring data analysis tools are available and these tools offer a variety of features (i.e. functional and non-functional) that affect the analysis process. However, these features come with their own setbacks. Therefore, selection of a suitable monitoring data tools is challenging and difficult to decide. Secondly, the big monitoring data analysis process contains two main operations of querying and processing a large amount of data. Since the volume of monitoring data is big, these operations require a scalable and reliable architecture to extract, aggregate and analyze data in an arbitrary range of granularity. Ultimately, the results of analysis form the knowledge of the system and should be shared and communicated. The contribution of this research study is two-fold. Firstly, we propose a generic performance evaluation methodology. The method uses the Design of Experiment (DoE) evaluation method for the assessment of tools, workflows and techniques. The evaluation results generated from this methodology provide a base for selection. Secondly, we designed and implemented a big monitoring data analysis architecture to provide advanced analytics such as workload forecasting and pattern matching. The architecture offers these services in an available and scalable environment. We implement our design using distributed tools such as Apache Solr, Apache Hadoop and Apache Spark. We also assessed the performance aspects (i.e. Latency and Fault-tolerance) of the architecture design using the proposed evaluation method.

Acknowledgments

I would like to thank all the people who contributed in some way to the work described in this thesis. First and foremost, I would like to express my sincere gratitude to my advisor Prof. Yan Liu for the continuous support for my Master's study and research, for her patience, motivation, enthusiasm, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Masters. I would also like to thank the various members of the SAC group with whom I had the opportunity to work: Xing Wu and Mehran Khan.

My sincere thanks also go to Mr. Wayne Ding for offering me the internship opportunity in their groups and leading me working on diverse, exciting projects. His role as a mentor helped me develop the ideas presented in this work. I would also like to thank Prof. Zheng Li for his guidance and support.

Also, I thank my friends who supported me and motivated me throughout my Masters: Tairman Singh, Gursimrat Bawa, Angad Bedi, Debal Saha, Mayank Chaudhary, Srishti Kumar and Zahraa Chorghay. Finally, I would like to thank my parent: Mr. and Mrs. Singh for their support throughout my studies. I couldn't have done it without them. At last I would like to dedicate this thesis work to my grandparents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

According to HACE theorem (70; 76),

"*Big Data starts with large-volume, Heterogeneous, Autonomous sources with distributed and decentralized control, and seeks to explore Complex and Evolving relationships among data*"

Within the last few years, an estimated 2.5 quintillion bytes of data are generated globally through a variety of sources. The popular social networking website Facebook, which has more than 950 million users, generates more than 500 terabytes of new data every day. A debate between American President Barak Obama and Governor Mitt Romney triggered more than 10 million tweets in 2012 within 2 hours. Such online discussions provide new means to sense the public interests, and ultimately, they generate a huge amount of data or "Big Data" (76). A prediction in an IDC report states that the global data volume will grow by a factor of 300 i.e. 130 exabytes to 40,000 exabytes, representing a double growth every two years until 2020 (33). Data generation has never been this powerful and enormous since the advent of information technology in the early 19th century (67).

Analysis of Big Data yields novel insights beneficial to businesses and organisations (10). It provides several use cases to enable big data exploration, improved 360 degree view of the customer, security system surveillance, operation analysis, and data warehouse moderation (71; IBM). Our work focuses on operation analysis, and specifically, on big monitoring data analysis. Several major companies such as Ericsson, Amazon, Microsoft, etc., provide a broad range of services, such as LTE mobile network and IaaS cloud services, which gather and process monitoring data to assess whether the deployed systems or offered services satisfy customer needs. For instance, a telecom vendor who manufactures base stations, gathers some sort of metric data to monitor the services provided by these base stations. Furthermore, monitoring data allows companies to make crucial decisions related to improvisation and prevent critical issues prior to their detection.

Big Data can be structured, unstructured or semi-structured (22), and processing it via traditional analysis tools and sequential analytics is inefficient and problematic. Therefore, new programming paradigms, like MapReduce, have quickly gained popularity. MapReduce is a parallel programming model for processing large amounts of data. It is widely used for data mining, bioinformatics, web indexing and machine learning (77). A well-known implementation of MapReduce programming model is Apache Hadoop, used for batch-processing large data-intensive applications (17). Similarly, several other distributed frameworks such as Apache Spark (52) or Apache Pig (27) for stream processing have recently been introduced.

1.1 Problem Statement

Several challenges are associated with analyzing big monitoring data as it exhibits the five VFLs of Big Data: Volume, Velocity, Variety, Value, and Veracity (51). A wide range of monitoring data analysis tools are available, offering a variety of functional and non-functional features that affect the analysis process. For instance, data mining tools such as RapidMiner or KNIME provide a variety of advanced analytic methods for forecasting, clustering, and visualization. The available features and the corresponding setbacks of these monitoring data tools need to be carefully considered in order to select the best tool. Hence, a methodology is required to assess these tools based on user requirements.

A monitoring data analysis process contains the two main tasks: querying, and processing a large amount of data. For instance, the Google trace consists of the production workloads running on Google clusters collected over 6 hours and 15 minutes (19; 54; 62; 63; 75). This dataset contains over 3 million observations (or data points), each of which consists of six features: 1) Timestamp, in seconds; 2) JobID, as the unique job identifier; 3) TaskID, as the unique task identifier; 4) JobType; 5) Normalized Task Cores, as the normalized number of CPU cores used; and 6) Normalized Task Memory, as the normalized value of the average memory consumed by the task. In such a trace, to analyze the workload, data needs to be retrieved for particular attributes from an arbitrary range of granularity (such as in a minute or hourly aggregation). However, since the trace data is continuously generated at a large volume and requires scalable storage space, a scalable and reliable monitoring data analysis architecture design is necessary. The available monitoring systems rely upon simple aggregation methods for analysis but a system architecture is needed to support advanced analytics such as workload forecasting or pattern matching in an available and scalable execution environment.

1.2 Objective

The objective of the research study is two-fold. Firstly, we explore and devise a performance evaluation method to support the selection process of analytics tools for Big Data exploration. Secondly, we design and implement a monitoring data analysis architecture, which aims to provide a solution to the following monitoring challenges: 1) Limited monitoring solutions; 2) Availability and Scalability of the system, and 3) An analysis method to accommodate Big Data. We propose a monitoring infrastructure, *CloudAnalyzer*, which supports advanced data analytics like forecasting and pattern matching of workload data traces, provides monitoring services in an available and scalable environment, and extends distributed frameworks to enable storage and analysis of Big Data.

1.3 Contribution

The contribution of this research study is

I. A generic performance evaluation methodology for the assessment of tools, workflows or techniques to support selection process, and

II. A big monitoring data analysis architecture to provide the following features:

- Advanced analytics such as workload forecasting and pattern matching
- Integration with cloud platforms and distributed frameworks (such as Apache Hadoop or Spark)
- Quality attributes such as availability and scalability

We have utilized distributed frameworks such as Apache Hadoop, Apache Solr, and Apache Spark to design and implement big monitoring data analysis architecture. We have also assessed the performance of the architecture design using the proposed evaluation method.

1.4 Thesis Structure

This thesis is organized as follows.

Chapter 1: briefly discussed the concepts and tools used in the dissertation. It provides a rationale for the data analysis architecture design described in following chapters.

Chapter 2: discusses several evaluation approaches for efficient assessment of system architecture, and describes several monitoring architectures proposed over the past few years.

Chapter 3: explains the Design of Experiment (*DoE*) evaluation method and its application on two distinct case studies. It paves the path to designing monitoring data analysis architecture.

Chapter 4: proposes monitoring data analysis architecture (*i.e. CloudAnalyzer*) and discusses its design and implementation. It also describes the deployment process and provides insight using multiple case studies.

Chapter 5: evaluates the designed monitoring data analysis architecture using DoE methodology

Chapter 6: discusses the potential future work and summarizes the material presented in this thesis

1.5 Publications

The research presented in this dissertation have been accepted at the following conferences:

I. Singh, S., & Liu, Y. (2016). A cloud service architecture for analyzing big monitoring data. Tsinghua Science and Technology, 21(01), 55-70.

II. Khan, M. N., Liu, Y., Alipour, H., & Singh, S. (2015, September). Modeling the Autoscaling Operations in Cloud with Time Series Data. In Reliable Distributed Systems Workshop (SRDSW), 2015 IEEE 34th Symposium on (pp. 7-12). IEEE.

III. Singh, S., & Liu, Y., Khan, M. N. (2016). Exploring Cloud Monitoring Data Using Search Cluster and Semantic MediaWiki. 2015 2nd IEEE Conference on Cloud and Big Data Computing.

Chapter 2

Background

2.1 Introduction

In Chapter 1, we briefly discussed the problems associated with available monitoring data analysis systems. Several monitoring systems like Nagios, Ganglia, and MonALISA rely upon simple aggregation methods that cannot predict the problems or critical events in advance, nor can they anticipate future load balancing situations. Advanced analytics methodologies for monitoring data, such as workload prediction or pattern matching, currently have limited implementation. In this chapter, we introduce some of the advanced analysis methods that can be integrated with Cloud infrastructure for effective and efficient monitoring analysis.

2.2 Advanced Data Analysis

Big data analysis is a process of retrieving useful information from rapidly growing data. This analytical power makes it a useful methodology for industries attempting to gain an edge against their competition. It can be used for forecasting, anomaly detection, and classification clustering. This section introduces the preliminaries and discusses some analysis methods used in our research, based on the type of analysis being performed.

2.2.1 Forecasting Analysis

One of the popular statistical forecasting analysis approaches is exponential smoothing. The analysis method predicts the future values based on historic and current experience. It is widely used to process univariate time series data. The three type of exponential smoothing are 1) single (or simple) exponential smoothing, 2) double exponential smoothing, and 3) triple or Holt-Winters triple exponential smoothing.

1. *Single or Simple Exponential Smoothing* is the basic smoothing approach, which assumes the original data have a stable mean. It is widely used to analyze data for a very short range such a month, with no trends and seasonality factors (34; 38). It is formulated using the following equation 1

   $$S_t = \alpha * X_t + (1 - \alpha) * X_{t-1} \tag{1}$$

   In the above formulae, S_t and X_t are smoothed values and observed values (*for t = 1, 2, 3...T*), respectively. Where α is the smoothing factor with value ranging between $0 \leq \alpha \leq 1$. The weighted average of each smoothed value S_t depends upon the previous observations. It decreases exponentially based on the value of smoothing constant (α). According to equation 1, If the value of α is equal to 1, then the previous observations are completely ignored i.e. $S_t = X_t$. If the value of α is 0, then the smoothed value entirely depends upon the previous observation i.e. $S_t = X_{t-1}$. The other in-between values will produce intermediate results.

2. *Double Exponential Smoothing* is a smoothing approach to process data with levels and trends. It adds two additional components in the Simple Exponential Smoothing equation (1) to represent level and trends. A level is a smoothed estimate of the value of the data at the end of each period, whereas, a trend is a smoothed estimate of average growth at the end of each period (38). The equations associated with double exponential smoothing are provided in 6 and 5,

   $$S_t = \alpha y_t + (1 - \alpha) * (S_{t-1} + b_{t-1}) \tag{2}$$

$$b_t = \gamma * (S_t - S_{t-1}) + (1-\gamma) * b_{t-1} \tag{3}$$

Double exponential smoothing introduces a trend equation (*see equation 5*) and an additional smoothing constant γ. The value of both the smoothing factors (i.e. α and γ) lie between 0 and 1.

3. *Triple or Holt-Winters Triple Exponential Smoothing (HWTES)* is a data smoothing approach to process data with both trends and seasonality. A third equation is added to handle seasonality in the time series data. Holt-Winters proposed two models based on the type of seasonality, which are the Additive Seasonality Model and the Multiplicative Seasonality Model. The equations associated with HWTES are given below:

$$S_t = \frac{\alpha y_t}{I_{t-L}} + (1-\alpha) * (S_{t-1} + b_{t-1}) \tag{4}$$

$$b_t = \gamma * (S_t - S_{t-1}) + (1-\gamma) * b_{t-1} \tag{5}$$

$$I_t = \beta \frac{y_t}{S_t} + (1-\beta) * (I_{t-L}) \tag{6}$$

   The equations are collectively called *Holt-Winters Triple Exponential Smoothing* where α, β and γ are the smoothing constant. The values of these constants are set to obtained minimum Mean Square Error (MSE) for best forecasting results.

Another forecasting analysis approach used to smoothen data is Kernel density estimation (KDE). It is a non-parametric approach to estimate probability density function of a random variable. Let us consider a random variable X. The KDE of a sample $X_t = \{X_1, X_2, X_3 ...\}$ is given by,

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\Big(\frac{x - x_i}{h}\Big), \tag{7}$$

where K_h is the non-negative kernel function (30; Wikipedia). A Kernel is a special type of probability distribution function with the following properties: 1) non-negative; 2) real value; 3) even number; and 4) its definite integral over its support set must equal to 1 (68). It is applicable on both univariate and multivariate datasets.

2.2.2 Clustering Analysis

Within a decade, the size of the cloud has increased from a few hundred cluster nodes to thousands. Monitoring such a huge cluster results in accumulation of a large volume of data, which can have hidden useful information e.g. workload patterns. Data clustering is a powerful analysis approach to retrieve such hidden patterns inside the dataset. For instance, this information can be used for scheduling and load balancing the cluster, as given in (3; 49). There is a broad range of clustering algorithms available such as KMeans, DBSCAN, Fuzzy C-Means, BIRCH, and so on. Selecting these algorithms depends on various factors, which are beyond the scope of our study, but we used the K-Mean clustering algorithm for our study.

K-Means is a popular and simple approach for data partitioning. It provides an iterative approach to divide a dataset $X = \{x_1, x_2, ...x_n\}$ with n objects or data points and d attributes into k mutually exclusive clusters $C = \{c_1, c_2, ...c_k\}$ (13). Each center point, defined as the centroid of the cluster, characterizes the cluster. A KMeans clustering example is shown in figure 2.1. The

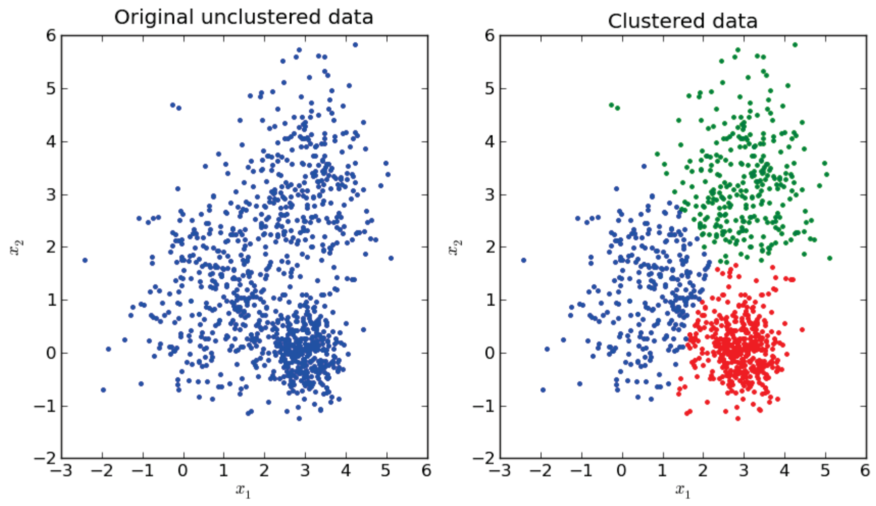


Figure 2.1: K Means Clustering Example

objects are distributed based on their distance to the centre point. Different distance metrics such as Euclidean distance and Mahalanobis distance can be used to calculate the distance between the object and centroid. Each object belongs to the cluster with shortest distance to its centroid (28).

2.3 General Tools

2.3.1 Apache Solr

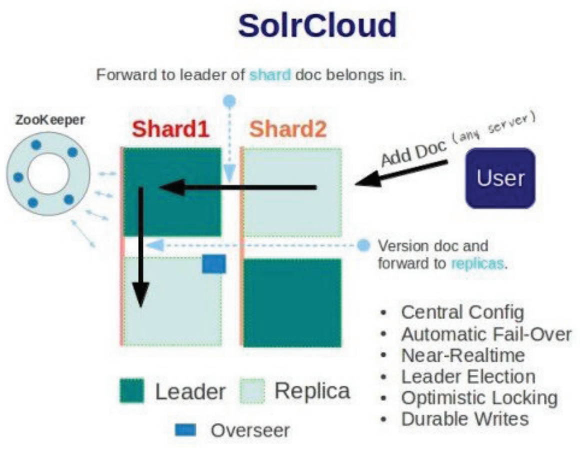
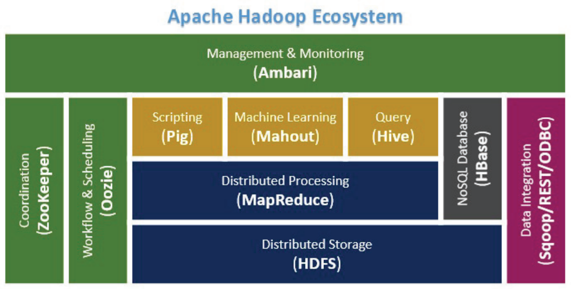


Figure 2.2: SolrCloud

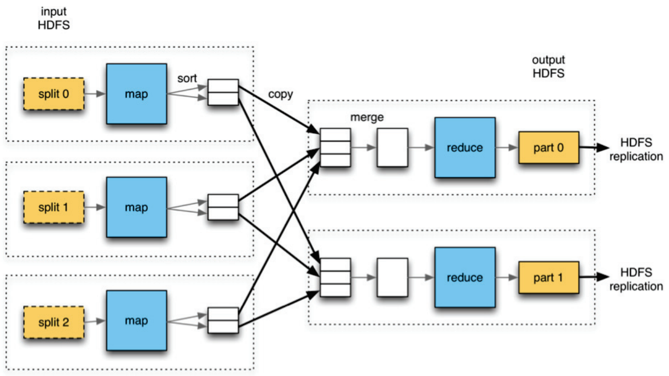
Apache Solr is an open source full text-based search engine. It provides features of faceted search, hit highlighting and real-time indexing. The core of Solr consists of Apache Lucene's search library that allows distributed search and index replication. Solr also provides a Representational State Transfer (REST), like the Application Program Interface (API), which enables language binding with Solr. Based on Solr 4.0, SolrCloud further provides the functionalities of running multiple Solr nodes in a cluster to devise a scalable and highly available cloud computing environment (44).

2.3.2 Apache Hadoop

Apache Hadoop is a framework designed to provide distributed storage and analysis of large volumes of data. Its architecture design includes Hadoop Commons, Hadoop Yarn, Hadoop Distributed File System (HDFS) and Hadoop MapReduce(29). Hadoop Commons are Java libraries and utilities required by other Hadoop modules. These libraries provide filesystem and OS level abstractions and contain the necessary Java files and scripts required to start Hadoop. Hadoop Yarn is a scheduling module and is responsible for cluster resource management. It facilitates additional services such as HBase or Apache Pig to run on the top of Hadoop framework. HDFS adopts a master/slave architecture, whereby the master node consisting of a single NameNode is responsible for managing


Apache Hadoop Ecosystem
Management & Monitoring (Ambari)
Coordination (ZooKeeper)
Workflow & Scheduling (Oozie)
Scripting (Pig)
Machine Learning (Mahout)
Query (Hive)
Distributed Processing (MapReduce)
Distributed Storage (HDFS)
NoSQL Database (HBase)
Data Integration (Sqoop/REST/ODBC)

(a) Apache Hadoop Ecosystem



(b) MapReduce Paradigm

Figure 2.3: Apache Hadoop Framework

the file system metadata, and the slave nodes consisting of DataNodes store the actual data(32; 66).

MapReduce is a programming paradigm designed to process large volumes of data in a distributed environment. Its programming model consists of two stages, the Map Stage and the Reduce Stage. The Map Stage is responsible for converting the input dataset into key/value pairs (*see figure* 2.3b). The input dataset is the file or directory, usually stored in HDFS. The reduce stage is the combination of two steps, shuffle and reduce. In shuffle step, the worker nodes redistribute data based on the output keys from mapper stage, such that all data belonging to one key is located at the same worker node. Finally, the reduce step processes the shuffles key and value dataset in parallel.

2.3.3 Apache Spark

Apache Spark is a distributed computing framework designed to analyze a large amount of data, especially Big Data. It extends the map-reduce paradigm for complex computation. In contrast to Apache Hadoop, it can compute using the main memory for faster computation. It requires a resource manager and a distributed storage system for data processing. Currently, it supports three resource managers, which are Spark Standalone, Apache Yarn, and Apache Mesos. It runs on various distributed storage platforms, such as HDFS, Amazon S3, Cassandra, etc. Apache Spark also provides a unified stack, i.e., integrated components such as Spark SQL, Spark Streaming, Machine learning library (Mlib), and GraphX, for better data processing. For instance, Mlib provides the K-Means clustering algorithm efficiently to divide a large number of data points into k specified sets or clusters.

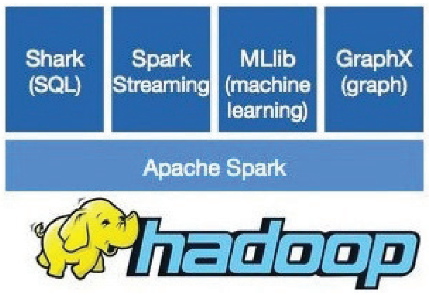


Figure 2.4: Apache Spark

2.3.4 MediaWiki

MediaWiki is an open-source application originally developed to power Wikipedia. MediaWiki helps to search, organize, tag, browse, evaluate, and share information as the wiki's content (8). Semantic MediaWiki (SMW) is an open-source extension developed for MediaWiki. It adds semantic annotations that allow a wiki to function as a collaborative database. SMW provides a flexible means for a developer to describe the trace data that he or she intends to analyze. SMW annotations define the fields and attributes of the trace data, and their relationship helps to form a structured query (43). Instead of storing the whole set of raw trace data into SMW's content database, we propose a cloud architecture that stores the trace data in a search cluster using SolrCloud. SMW page issues the queries, and the results are stored back to SMW's content database for display.

Chapter 3

Related Work

3.1 Introduction

In this chapter, we discuss related studies related to our research approach. We explored and comprehend several software application performance evaluation methods. The evaluation approach design in our research studies is inspired by these evaluation approaches. Similar, we also discussed several monitoring and analysis systems. This systems provides a rational to design big monitoring data analysis system.

3.2 Performance Evaluation Methods

Selection of suitable software applications to carry out specific tasks has become challenging, due to the rapid development and availability of software. A number of performance evaluation techniques or methods have been developed to support this selection process. Performance evaluation is a method to determine the strengths and weaknesses of the underlying architecture or design pattern. Some evaluation methods developed are discussed in (65), where they are categorized into early and late evaluation.

The early evaluation methods are software evaluation methods that can assess the software application based on its specification and description. They are employed to analyze software quality attributes such as reliability, performance, scalability and availability. Most of these evaluation methods are scenario based. Scenario based evaluation methods identify scenarios in close interaction with the stakeholders and systematically investigate the software architecture based on these scenarios. Some of the examples are Software Architecture Analysis Method (SAAM), Architecture Trade-off Analysis Method (ATAM), Family – Architecture Analysis Method (FAAM), and Domain Specific Software Architecture Comparison Model (DoSAM) (9; 36; 40; 41).

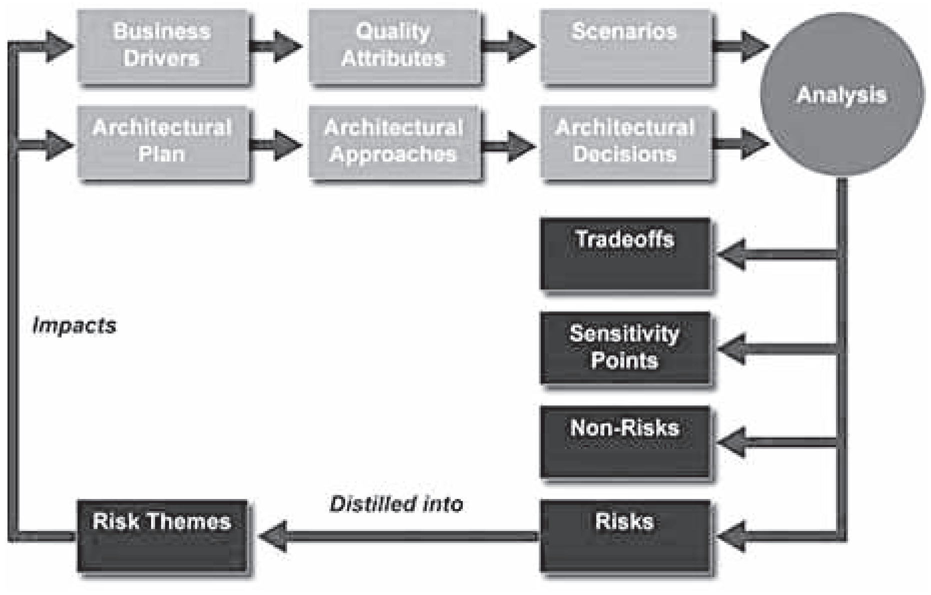


Figure 3.1: A Conceptual Flow of ATAM

SAAM was first introduced in 1993 as a scenario based early evaluation method (20). The main advantage is this evaluation method is its adaptive design (39) that allows modification of SAAM's rationale design to develop evaluation methods for specific requirements. The method includes the six steps or activities of 1) scenario development, 2) System Architecture (SA) description, 3) Scenario classification and prioritization, 4) individual scenario evaluation, 5) scenario interaction, and 6) overall evaluation (6). ATAM is another evaluation method for assessing quality attributes such as modifiability, portability, extensibility, and integrability. The assessment method has nine steps as shown in figure 3.1. DoSAM is another scenario-based evaluation method designed to assess software quality attributes like performance, scalability, and availability (9).

The late evaluation methods are employed where the software application is prone to changes. An approach is introduced in (69) to evaluate software application which undergoes modification during the implementation process. It avoids system degeneration by actively and systematically detecting and correcting deviations from the planned design as soon as possible, based on explicit and implicit architectural guidelines. It has the following seven steps which are as follows: 1) Select the perspective for evaluation; 2) Define and select guidelines, and establish metrics to be used in the evaluation; 3) Analyze the planned architectural design in order to define architectural design goals; 4) Analyze the source code in order to reverse-engineer the actual architectural design; 5) Compare the actual design to the planned design in order to identify deviations; 6) Formulate change recommendations in order to align actual and planned designs; and 7) Verify that the design goal violations have been corrected by repeating steps 4 through 6. There are other late evaluation methods discussed in (26; 48; 57; 64).

Most of these evaluation methods are intended for the evaluation of a single architecture at a particular point in time. In a case of comparing tools, they mainly focused on comparing their results against a certain analysis job, for example, the accuracy of classification (12; 72). Inspired by the property “Velocity” of Big Data analytics, we are more concerned with the performance of different tools. More importantly, to our knowledge, there lacks a systematic study of evaluating data mining tools that are driven by requirements derived from a data analysis context. Thus, our study can also be viewed as an experience report on evaluating data mining tools by following a relatively rigorous methodology and applying principles of Design of Experiment (DOE) techniques.

DOE emerged traditionally for agriculture, chemical, and process industries. Considering its natural relationship with experimental activities, suitable DOE techniques have also been employed in experimental computer science. When it comes to the software engineering field, the main interest of applying DOE seems to be in software testing from the developer’s perspective (21; 45; 58). Our study essentially extends the applicability of DOE to software comparison from the end user’s perspective.

3.3 Distributed Monitoring Data Analysis Architecture

The survey papers (2; 73) of cloud monitoring abstract the process of monitoring process in three main steps: 1) collection of relevant state, 2) analysis of the aggregated state. and 3)decision making as a result of the analysis. The requirements of monitoring systems on scalability and fault-tolerance have an inherent propensity from cloud computing. A scalable and fault tolerant cloud monitoring system is free from a single point of failure and bottlenecks, and adapts to elasticity. Since the monitoring data can be useful long after it has been collected, it is continuously accumulated, so failures must not interrupt monitoring. The existing monitoring data analysis services are mostly available as part of IaaS or PaaS cloud services. In general, these monitoring services face many challenges as discussed in detail in (2), and researchers and several cloud service providers have proposed monitoring infrastructures to encounter them. As briefly described in Chapter 1, our work focuses on the following problems associated with monitoring data analysis systems: 1) Need of advanced monitoring solutions, 2) Extensibility, 3) Scalability, and 4) Availability. In the following section, we discuss some monitoring data analysis architectures designed to address these issues and to offer efficient monitoring of IT infrastructures.

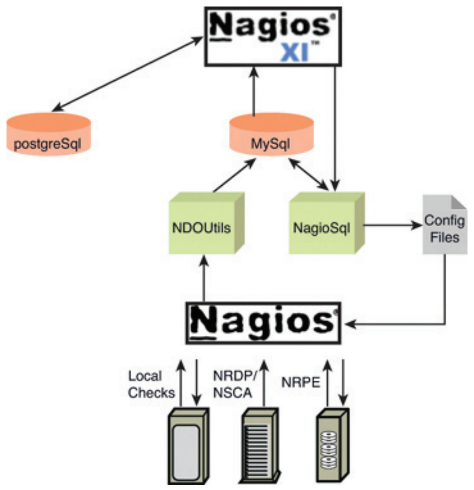


Figure 3.2: Nagios Network Model

Nagios is one of the most popular general-purpose monitoring infrastructures available in the market. It is an integral solution designed to achieve instant awareness of IT infrastructure problems (23) (2). It adopts *Client/Server* architecture, carefully designed to inherit scalability and flexibility (61). Nagios also supports different plugins for local resource monitoring, as shown in Figure 3.2. Its Executor Plugin (Nagios NRPE) allows remote resource monitoring. Its Service Check Adaptor Plugin (Nagios NSCA) supports asynchronous push measurements and events from monitored nodes. However, the disadvantages of the approach is that it lacks the capability to monitor rapidly changing and dynamic infrastructure (61) (55).

Another monitoring system architecture design to monitor large-scale IT infrastructures is Monitoring Agents using a Large Integrated Services Architecture (MonALISA) (59). It adopts the agent-based architecture in which agents are dynamically registered to the central server. This architecture design implicitly provides scalability of the system. Similar to Nagios, MonALISA also does?t support dynamic infrastructure monitoring.

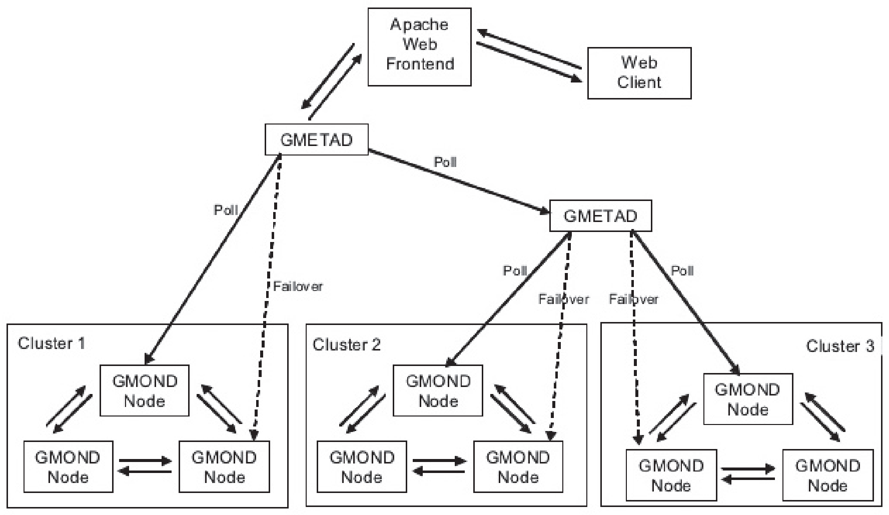


Figure 3.3: Ganglia Architecture

Ganglia (79) is another popular resource monitoring system. It is designed to monitor grids but was later adapted to cloud infrastructure (61). Ganglia adopts a hierarchical agent-based architecture consisting of two main components, *gmonds*, and *gmetad*, as shown in Figure 3.3. In this hierarchical architecture, grounds run on each monitoring nodes. They gather resource information and pass it on to *gmetad* upon request. *gmetads* are responsible for aggregation and visualization of the collected information. Also, it leverages XDR format for compact, portable data transportation and RRDtool for data storage and visualization (50). Ganglia lacks some relevant features such as the discovery function at the inter-cluster level.

DORGAS, introduced in (61), is a monitoring infrastructure built upon data-centric data distribution service (DDS) (60). It adopts the *Publish/Subscribe* distributed architecture design to effectively analyze multi-tenant cloud resources. The architecture is composed of two main components: 1) Node Monitoring Agent (NMA), which runs on local nodes, to gather their resource information; and 2) Node Superior Agent(NSA), which collects data remotely from nodes and forward it to the central system. This data-centric approach provisions fault-tolerance and scalability in the system. Another relevant monitoring system introduced in (55) is GMonE. It uses the typical *Publisher/Subscriber* paradigm, which is composed of four main components: 1) GMonEMon; 2) GMonE plugins; 3) GMonEDB; and 4) GMonEAccess. Its major advantage is that it provides monitoring services at all levels. Besides, GMonE plugins can be used to provide additional monitoring solutions.

Chaves et al. proposed an abstract general-purpose monitoring data analysis system, *Private cloud monitoring system architecture (PCMONS)* in (15), as a monitoring solution for private clouds. In contrast to previously discussed monitoring solutions, *PCMONS* utilizes a layered architecture style consisting of the following three layers: 1) Infrastructure layer, which offers basic functionalities, services, and contains software and hardware installations; 2) Integration layer, composed of heterogeneous resources, such as data collector/gatherer, configuration generator, etc.; and 3) View layer, which provides abstract functionalities and an interface for visualizing monitoring resources. This architecture style can integrate with traditional monitoring solutions like Nagios and enhance their capabilities.

Another interesting monitoring architecture, *Flexible automate cloud monitoring slices (FlexACMs)* has been discussed in (7). Its architecture is designed to overcome the issue of manual configuration of monitoring solutions. For instance, the monitoring solution includes aggregating CPU usage to detect critical events and abnormalities in the IT infrastructure. The design is composed of the following three main components: 1) Gatherers, to collect the resource information, such as CPU, Memory, Storage, etc. from the cloud and forward it to the framework core using REST API; 2) Framework core, which is responsible for detecting changes in collected resource information; and 3) Configurators, to receive resource information and provide required monitoring solutions. These monitoring applications, *PCMONS* and *FlexACMs*, are proposed to inherit extensibility.

Brinkmann, Andre, et al., have designed a monitoring system complementary to their EASI-CLOUD cloud service to tackle the challenge faced by monitoring systems concerning scalability. The architecture is divided into three components similar to the previously described *PCMONS* and *FlexACMs* monitoring systems, including *Data supplier*, *Data manager* and *Data storage and* *preprocessing*. One of the main advantages this architecture offers is its distributed management tree approach. As specified by the author in (78), there are no standard specification and protocols for cloud-based services, which makes it difficult for service providers to support all the protocols to comply user requirements. However, the distributed management tree comprising of *Data managers* makes the system flexible to cover protocol-specific parameters for data acquisition by implementing specific handlers (78). *Data supplier* component has the basic functionalities of resource gatherer, and EASI-CLOUD allows external tools for data collection. The third component, *Data storage, and preprocessing*, is responsible for aggregating the gathered data and acting as the intermediate stage between management and supplier. The author has described scalability at the component level. However, the solution does not support storing original resource information that could be utilized to get some hidden insight of the cluster.

A much closer approach to our research is Cloud Monitoring Engine (CME). It is a distributed monitoring architecture, introduced in (42), which ensures elasticity and scalability of the monitoring system. Its advantage is that it encompasses use of parallel computing for data aggregation. Also, it uses the MapReduce paradigm to normalize collected data. However, it doesn't provide any advanced analytics methods for processing.

Commercial cloud solutions often make use of their monitoring systems. Several cloud-specific monitoring infrastructures are also known for their efficient monitoring capabilities. Examples of these monitoring systems are CloudWatch, AzureWatch, and OpenNebula. CloudWatch is a high-level cloud monitoring solution developed by Amazon to monitor EC2 instances (4) (16) but it hides low-level services from consumers, such as resource information gathering, collection, and analysis. Also, the primary focus of CloudWatch is virtual platforms. AzureWatch is another commercial monitoring system to monitor Azure resources, such as instance, storage, websites and web applications (2).

Chapter 4

Empirical Evaluation using Design of Experiments Technique (DoE)

4.1 Introduction

In this paper, we experimentally explore tools and methodologies for running data analysis on telecom monitoring data. According to the clarifications in (18; 24), using experimentation to evaluate tools and methodologies can be regulated by experimental computer science. Considering that evaluation methodology underpins all innovation in experimental computer science" (11), we employ the recently available Domain Knowledge-driven Methodology (DoKnowMe) (47) for this study, and particularly utilize a set of principled techniques of Design and Analysis of Experiments (DOE) (5; 56).

We consider two case studies. Firstly, we evaluate the data mining tools on the basis of a set of requirements. Secondly, we assessed the analysis algorithms (i.e. Outlier detection techniques) identity a suitable algorithm for a particular workflow design. We conduct the evaluation on datasets collected from base stations in a trial mobile network with two data ming tools, RapidMiner and KNIME. The dataset time frames are 1 month, 6 months, 1 year and 2 years. Additionally, we design workflows on a KPI that provides the average number of connected users per cell on base stations. We used these workflows to conduct our studies to evaluate data mining tools and outlier detection algorithms. The observations from this evaluation provide insights of each data mining tool and the outlier detection techniques in the context of data analysis workflows. This documented design of experiment facilitates telecom software engineers to replicate this evaluation study, and can be expanded for other evaluations.

4.2 The Data Analysis Scenario and Workflow

Big data is the new reality in the telecom world. Over recent years, the mobile broadband traffic has had an explosive growth due to widespread adoption, advanced new networks, increasing penetration of smartphones, and millions of mobile applications. This growth will continue at a rapid pace as increasing deployment of Internet of Things, sharable/uploadable/findable content by mobile users, sensors, connected cars and so on. Mobile big data has proven useful for capacity and performance monitoring (e.g., during normal operation or under massive events), troubleshooting, realistic lab testing, simulation, new feature design, an architectural evolution of mobile network infrastructure products.

A telecom vendor is specialized in manufacturing of E-UTRAN Node B (a.k.a eNodeB) or base stations. eNodeB is the hardware that is connected to the mobile phone network that communicates directly with mobile handsets. To manage the quality of service delivered by eNodeBs, they are monitored and observed by metrics or key performance indicators (KPIs). KPI measurement is frequently used and is effective means that mobile operators and mobile network infrastructure vendors use to search systematically and identify the system/network bottlenecks, troubleshooting, dimensioning and anomalies. For example, mobile network operators monitor KPIs to ensure an optimal dimensioning and configuration for their network with efficient use of network resources and to provide robust QoS and consistently good user experience. For wireless infrastructure vendors, the data collected from the network contains valuable information that can reveal the current state and quality of both hardware and software.

This brings increasing challenges to processing and analyzing this vast amount of data with huge variety in a timely fashion. The monitoring data are time series data that consist of more than 200 columns with each representing one KPI. Each KPI is read every 15 minutes, leading to 96 readings per day per KPI. Mining these KPI data will result in technical insights of the effectiveness and efficiency of the service provided to the customers.

The existing conventional tools and methodologies of telecom services are not designed for and simply cannot handle at this scale. As a result, there is only a small percentage of monitoring data has been analyzed at present. To expend the data analytics scales, data mining workflow tools play an important role. A suitable tool should address the business requirements of 1) Support workflow management that scales up day to day productivity of data analytics; 2) Provide flexibility and modularity for rapid workflow modification; 3) Make use of popular languages and libraries in data science such as R and Python; 4) Provide user-friendly workflow front end; and 5) Allow scheduling workflow jobs and even back-fill missing data.

A concept workflow architecture is depicted in Figure 4.1. The workflow includes three phases 1) Extract, 2) Transform and 2) Load. These phases consist of multiple tasks to complete. The Extract phase consists of two tasks, namely Collect Data, and Merge Data. This phase retrieves the data from various sources and stores at the staging area 1. The staging area 1 is the file system to store data in raw form i.e. (flat files). The stored data files are merged to form a single data file for processing. The merged file is then pushed forward to the Transform phase. In the Transform phase, the Clean Data task removes inconsistency and bad records in the data file to prevent errors. The cleaned data is stored in staging area 2 in the form of flat files. The Process Data task of the Transform phase reads the data file from staging area 2 and commences analysis. A range of analysis methods can be used to process data. For example, outlier detection techniques, such as LOC are used for analysis. Finally, the analysis results and the cleaned data are moved to the final staging area (i.e. Staging Area 3). The staging area 3 is of data storage in relational or No-SQL databased. The Load phase is responsible for generating a report or visualizing the process data. The analysis results in the data storage are queried for different purposes. The workflow is composed and executed within a data mining workflow management framework to execute it in a periodic manner.

A variety of data mining workflow tools exit. Each has its own set of features. When processing the data analytics logic on a given infrastructure, these features become a set of factors that affect both the functional and non-functional attributes of the end data analytics product. The choice of a tool demands systematic evaluation to identify the relations among the factors, the combinations

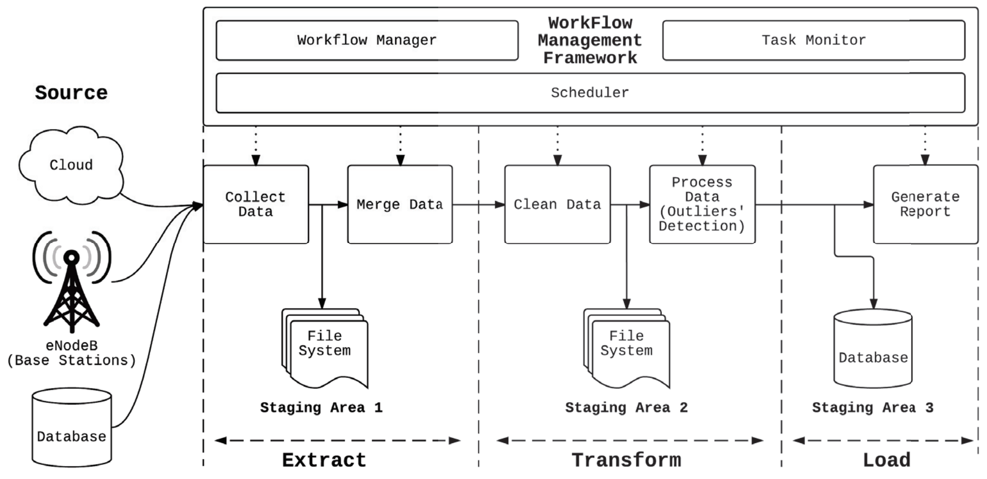


Figure 4.1: The Architecture of Data Analysis

and requirements of requirements of data analytics. Likewise, machine learning algorithms need to be selected according to the desired features of the data analytics product. The evaluation of data mining workflow tools and machine learning algorithms are at different granularities: the former includes the entire workflow runtime, while the latter is a composition component to a workflow. The question is how to develop a systematic evaluation method that covers both kinds of evaluation.

4.3 The Evaluation Method

Data mining tools are a special type of software that aims for facilitating data mining jobs (53). The comparison between software products is a typical evaluation practice that belongs to the field of experimental computer science (18). We adopt the principles of Design of Experiment (DoE) to guide evaluation implementation for selecting suitable data mining tools. Figure 4.2 outlines the steps of DoE in eight steps. The description of each step is as follows:

Step 1: Requirement Recognition. Identification of evaluation requirements is the first and foremost task in DoE evaluation method. These evaluation requirements are necessary to comprehend system related problems as well as the evaluation purpose. The experts with prior knowledge of associated problems brainstorm to write down a set of evaluation requirement. At first, the experts use natural language to describe evaluation requirements. They are then scrutinized thoroughly and mapped to requirements questions. This process determines the objective of the assessment process which is to address these requirement questions.

Step 2: Feature Identification. It standardizes the terms, concepts and their relationships within a system domain to determine a set of features. The features include both functional and non-functional features. This step utilizes the requirement questions to identify relevant features for evaluation.

Step 3: Metrics and Benchmark Listing and Selection. Firstly, experts investigate the relevant metrics and benchmark and prepare a list constituting them. Then, the most appropriate ones are selected based on the available resources in hand, estimating the overhead of potential experiments, and judging the evaluatorsFL s capabilities of operating different benchmarks. The selection process is a crucial task and plays an essential role in evaluation implementation. Once the metric and benchmarks are chosen, the selection of experimental factors begin.

Step 4: Experimental Factors Listing and Selection. The experimental factors are the parameters or variables that affect the performance features selected for evaluation. Similar to the previous step (*i.e. Metrics and benchmark listing and selection*), this step lists a set of potential candidate experimental factors. Later, the selection process considers the candidate factors with highest impact and relevance.

Step 5: Experiment Design. According to DoE, the next step after careful selection of the metrics, benchmarks, and experimental factors is to design experiments for evaluation as shown in Figure 4.2. In the beginning, simple experiments are designed based on the pilot trials. Later these experiments are modified for more complex designs using DOE techniques. The step outputs experimental instructions, experimental blueprints, and driving benchmarks. They are then used to implement these experiments.

Step 6: Experiment Implementation. means carrying out series of experiments. For instance, observing the behaviour of the system by gradually increasing the value of a selection factor and taking multiple reading for each value. The results obtained from the implementation step is moved forward for analysis.

Step 7: Experiment Analysis. In this step, evaluators comprehend the results and compare different systems on both functional and non-functional grounds.

Step 8: Conclusion and Documentation. Finally, the requirement questions are answered based on the analysis results and documented for future reference.

We apply this evaluation method based on DoE to two aspects of the architecture depicted in Figure 4.1. The first aspect is evaluating and comparing data mining tools that support all tasks of the workflow. The second aspect is evaluating the outlier detection algorithms in the task of Process Data.

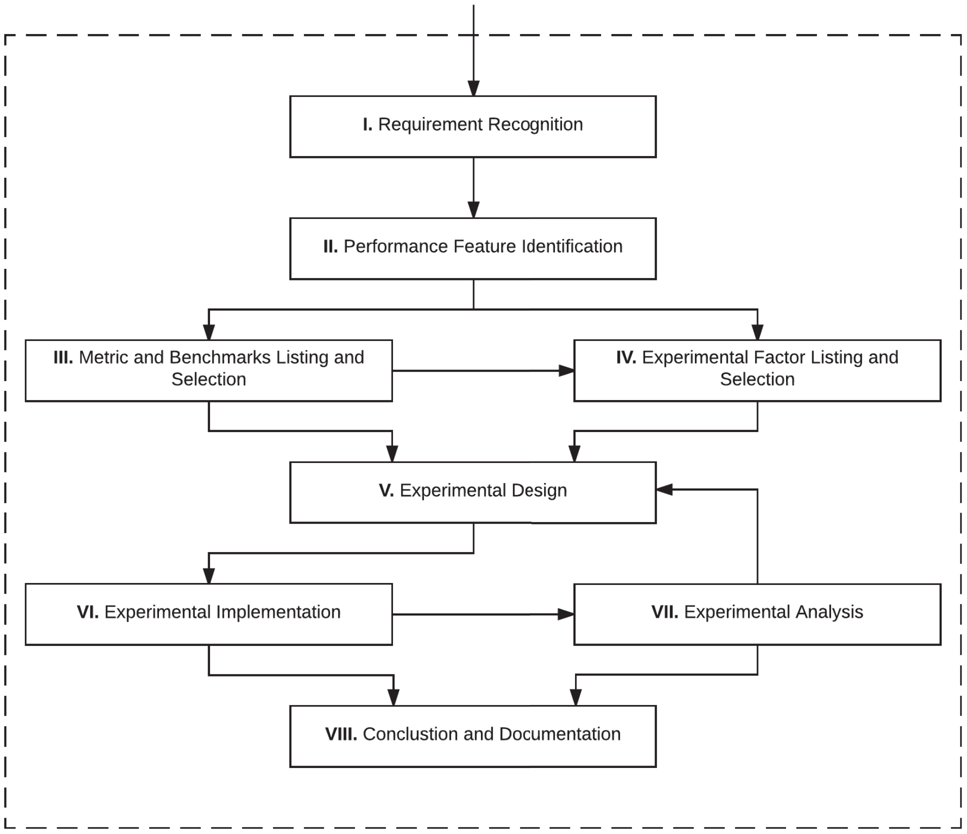

I. Requirement Recognition
II. Performance Feature Identification
III. Metric and Benchmarks Listing and Selection
IV. Experimental Factor Listing and Selection
V. Experimental Design
VI. Experimental Implementation
VII. Experimental Analysis
VIII. Conclustion and Documentation

Figure 4.2: The Step Skeleton of Design of Experiment

4.4 Evaluation of Data Mining Tools

A data mining workflow tool is a special type of software that aims for facilitating data mining jobs (53). It is a core component of the architecture depicted in Figure 4.1 to analyze the monitoring data collected from base stations. The evaluation of data mining workflow tools focuses on comparing alternative tools according to both quantitive and qualitative requirements. We apply the process of evaluation in the aforementioned eight steps.

4.4.1 Requirement Recognition

Driven by the evaluation method, the requirement recognition is both to understand the real-world problem and to achieve clear statements of the corresponding evaluation purpose. In fact, it is the recognized evaluation requirement that essentially drives the remaining evaluation steps. In this case, an engineering team with both telecom engineers and software engineers brainstormed the selection criteria on data mining tools as follows:

- It should be able to read data either from files (e.g., CSV or Excel files) or provide capabilities to read data from the database systems.
- It should provide appropriate support for data manipulation and transformation.
- It should provide a useful statistical model or Machine learning support for data processing.
- The application should have interoperability.
- It should accommodate a large volume of data, for example, a data set with approximately 2 million data points.
- It should provide effective data exportation and visualization.
- It should support other Scripting languages, such as R, Python, etc.
- It should process data effectively and efficiently.
- The interface of the tool should be user-friendly and easy to learn.

The main evaluation requirement becomes:

RQ: Given the selection criteria, the key difference between a pair of data mining tools on the same sets of data.

Table 4.1: Identified Features of Data Mining Tools

Category	**Feature**
Quantitative Performance	Data Mining Latency
	Data Mining Resource Usage
Qualitative Indicator	Data Import Support
	Data Export Support
	Node IO Limit
	Scripting Language Support
	Visualization Support
	Data Manipulation and Transformation
	Interoperobility in Batch Mode
	Interface Usability

It is clear that the selection criteria of data mining tools include both quantitative (like mining job performance) and qualitative (like operational capabilities) concerns. Since the evaluation method encourages defining a set of specific requirement questions to be addressed by potential evaluation experiments, we further distinguish between those quantitative and qualitative criteria and clarify the above requirement as follows:

rq1: How fast does a tool perform data mining jobs?

rq2: How many resources does a tool need for data mining jobs?

rq3: How well does a tool match the qualitative selection criteria?

4.4.2 Tool Feature Identification

According to the clarified evaluation requirement and selection criteria, we identify the features of data mining tools to be evaluated. There are two quantitative performance features and eight qualitative indicators, as listed in Table 4.1. We mainly focus on the experimental evaluation of the performance features in the following subsections, while leaving the qualitative discussions to the end of this study.

4.4.3 Metrics & Benchmarks Listing and Selection

We consider two typical types of jobs in processing telecommunication service monitoring data, namely forecasting job and clustering job. We developed solutions for each type of jobs to benchmark. For example, to evaluate data mining tools regarding forecasting, we used the scripting language R to implement a data forecasting algorithm as a baseline. The forecasting algorithm employs Seasonal Naive and BATS forecasting models and predicts the data point for next three days. As for the clustering job, we used K-means clustering algorithm to run clustering analysis on a dataset.

The two metrics we use to measure the performance of data mining tools are the data mining jobs' execution time and memory usage. Execution Time is the time required for job completion, as formulated in Equation (8).

$$Time_{Exe} = TimeStamp_{End} - TimeStamp_{Start} \tag{8}$$

Memory Load is the percentage of memory used during the job execution, which is supposed to be monitored every second.

4.4.4 Experimental Factors Listing and Selection

Knowing the relevant factors (also called parameters or variables) is a tedious but necessary task of designing proper experiments (37). Our evaluation method distinguishes among the workload-related, resource-related and capacity-related factor types. We identified experimental factors and their levels for evaluating data mining tools, as listed below.

- Workload-related factors:
    - *Data Mining Job:* Forecasting and Clustering
    - *Workload Size:* 1 month, 6 months, 1 year, and 2 years of data
- Resource-related factors:
    - *Tool Brand:* RapidMiner and KNIME
- Capacity-related factors:

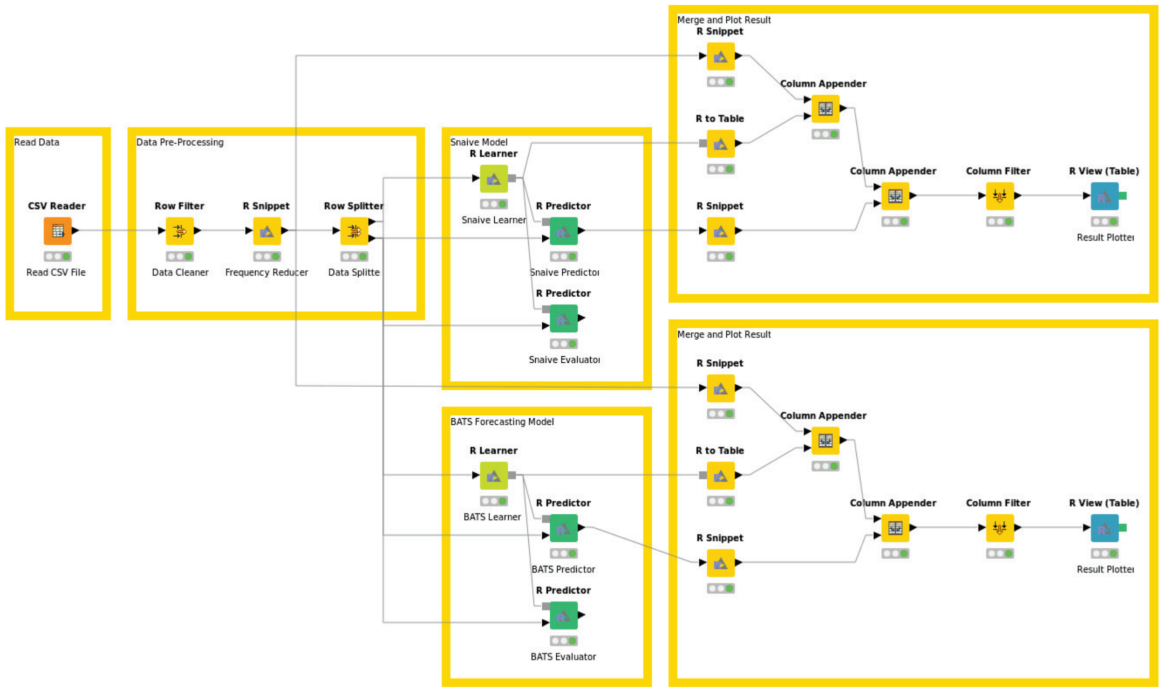


Figure 4.3: Workflow of implementing the forecasting job with KNIME.

- *Latency:* Execution Time
- *Memory:* Memory Usage

These benchmarks for forecasting and clustering are employed to vary data mining jobs by different sizes of workload. The workload of 1 month, 6 month, 1 year and 2 year of data have 2880, 17280, 34560 and 69120 data records respectively. We regard data mining tools as service resources to be evaluated, and then treat different tool brands as different resource types.

4.4.5 Experimental Design

Evaluation experiments are subsequently prepared and designed by utilizing the selected experimental factors. Here we employ the Full-Factorial Design technique (56). Since the factors *Data Mining Job*, *Workload Size* and *Tool Brand* have two, four and two values respectively, we have $2 \times 4 \times 2 = 16$ types of experimental trials. Note that the capacity factor *Latency* plays a response role in this design. Considering that *Randomization* and *Replication* are two principles of applying Design of Experiments (56), we decided to repeat 50 times each of the 16 types of experimental

Table 4.2: A General Full-Factorial Design*

Trial	Data Mining Job	Workload Size	Tool Brand	Execution Time & Memory Usage
1	Clustering	1 month	RapidMiner	?**
2	Forecasting	1 year	RapidMiner	?
3	Forecasting	2 years	KNIME	?
4	Clustering	2 years	RapidMiner	?
5	Forecasting	1 year	KNIME	?
6	Clustering	1 year	RapidMiner	?
7	Clustering	6 months	RapidMiner	?
8	Clustering	1 month	KNIME	?
9	Forecasting	6 months	RapidMiner	?
10	Forecasting	1 month	RapidMiner	?
11	Forecasting	2 years	RapidMiner	?
12	Clustering	2 years	KNIME	?
13	Forecasting	1 month	KNIME	?
14	Clustering	6 months	KNIME	?
15	Forecasting	6 months	KNIME	?
16	Clustering	1 year	KNIME	?

*The design matrix was generated by using Minitab.

**The "?" denotes a placeholder for *Response* value.

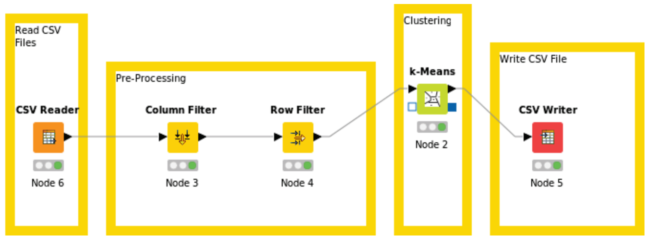


Figure 4.4: Workflow of implementing the clustering job with KNIME.

trials, and randomized those trials into a design matrix (please see Table 4.2) by using Minitab[1].

4.4.6 Experimental Implementation

The evaluation experiments were implemented following the full-factorial design. The experimental environment is consistent for running both forecasting and clustering jobs on RapidMiner and

[1]Minitab: https://www.minitab.com/en-us/

KNIME.

In the case of the forecasting job run by RapidMiner, the workflow includes three different phases: (1) The first phase reads data from the CSV files, and then passes the data to the next phase. (2) The preprocessing phase prepares the data and extracts the training and testing sets from the entire data set. (3) The last phase is composed of three major processes, namely data processing, validation, and plotting.

The prepared training data set is fed to the forecasting models, and the testing dataset is used to validate the forecasting results based on the validation metrics, such as root-mean-square error (RMSE) and mean absolute error (MAE) (14). This phase also plots the results for the purpose of visualization.

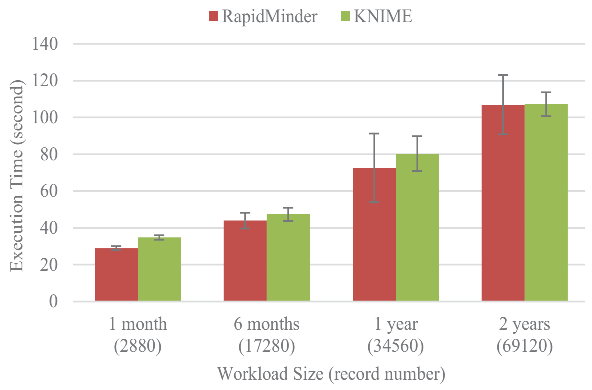
Along with the changing workloads, RapidMiner's execution time of running the forecasting job varies ranging from about 29 seconds to 107 seconds, on average, for the data sizes from 1-month to 2-year.

Unlike RapidMiner, each of the KNIME nodes supports one input node only. As a result, KNIME's forecasting workflow has four phases, namely reading data, preprocessing, data processing, and merging and plotting. In particular, the third phase provides data processing and validation, while the last phase provides data merging and plotting. We show the workflow of KNIME's forecasting job, as illustrated in Figure 4.3.

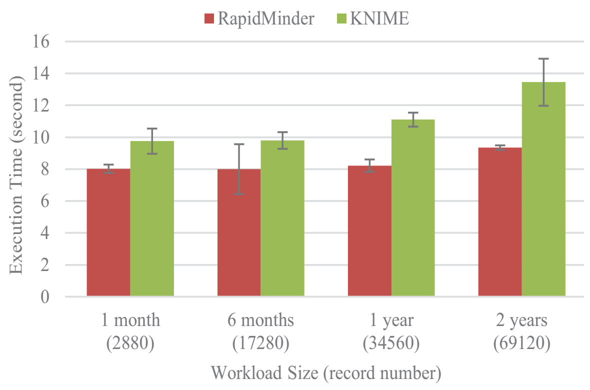
Given the above experimental setup, a KNIME's forecasting job takes approximately from 35 seconds for processing 1 month data to around 107 seconds for 2 year data.

As for the clustering job, the corresponding workflow is divided into four phases including reading, preprocessing, clustering and writing data for both data mining tools. Similarly, here we only show the workflow of KNIME's clustering job, as illustrated in Figure 4.4. Since the clustering job is simpler than the forecasting job, the two tools' execution time changes between around 8 seconds and about 13 seconds for different sizes of workloads. We further visualize the response time given the workload size shown in Figure 4.5a and 4.5b.

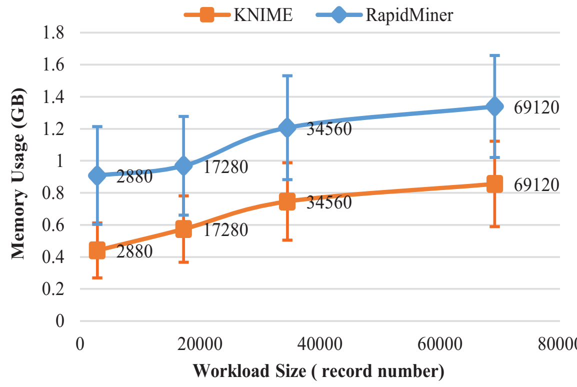
Likewise, we evaluate the memory usage of running jobs in both the forecasting and clustering workflows as shown in Figure 4.5c and Figure 4.5d. Both jobs have a steady increase of the memory usage, indicating that the algorithms applied in each job do not have a big size of intermediate results



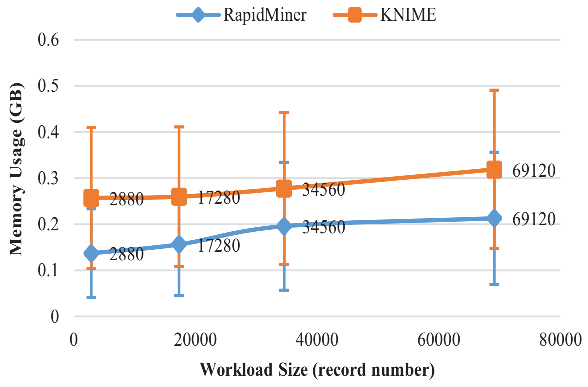
(a) Average execution time of the forecasting job



(b) Average execution time of the clustering job



(c) Average memory usage of the forecasting job against different sizes of workloads



(d) Average memory usage of the clustering job against different sizes of workloads

Figure 4.5: Average execution time & memory usages of jobs. The error bars indicate the corresponding standard deviations.

which could lead to intensive demands of memory usage. This is an important indicators to consider for the workflow design, since an intermediate step can contribute to the capacity related factors, while the workload related factors (such as the job types and workload sizes) and the resource-related remain the same.

4.4.7 Experimental Analysis

Given the results from the experimental implementation, we further conduct quantitative and qualitative analyses for those features respectively.

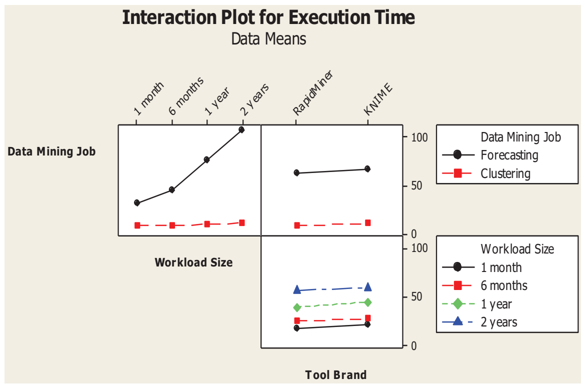
Analysis for Quantitative Features

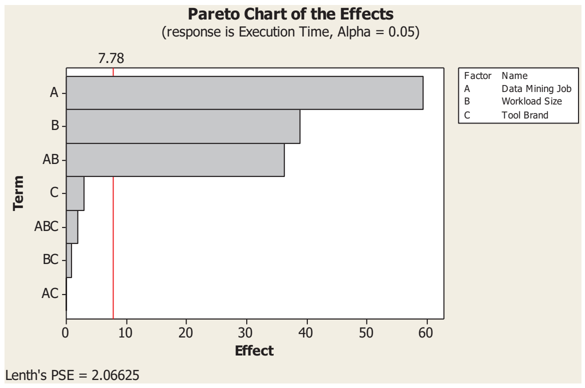
For evaluating quantitative features, DoE strongly suggests employing suitable statistical methods for experimental analysis. Note that the statistical methods do not directly prove any factor's effect (56) in the context of factorial experimental design. However, statistical analysis can add objectivity to drawing conclusions and to the potential decision-making process.

In this case, we first investigate if the effect of data mining tools on job execution could be influenced by the other factors, by statistically analyzing the interactions between different experimental factors against the response *Latency*. Benefiting from Minitab, we use the Interaction Plot to visualize the factor interactions, as shown in Figure 4.6a. In an interaction plot, the greater the difference in slope between two lines, the higher the degree of interaction between the corresponding factors while parallel lines indicate no interaction. It is then clear that there is a potential interaction between the factors *Data Mining Job* and *Workload Size* but no interaction between these two factors and *Tool Brand*. In other words, changing the value of *Data Mining Job* and *Workload Size* will not influence the effect of *Tool Brand* on job execution time.

Secondly, we investigate how significant changing data mining tools would impact on job execution time, by statistically analyzing the influences of different experimental factors on the response *Latency*. We employ the Pareto Chart to visualize the effects of the various experimental factors and their combinations, as shown in Figure 4.6b. In a Pareto chart, the absolute effect values are drawn with a reference line. Any effect that extends beyond this reference line indicates a potentially important factor or factor combination. As can be seen, the factors *Data Mining Job* and *Workload Size* and their combination have potentially significant effects on the execution time of data mining jobs, whereas *Tool Brand* seems not to be a major factor regarding the response *Latency*.

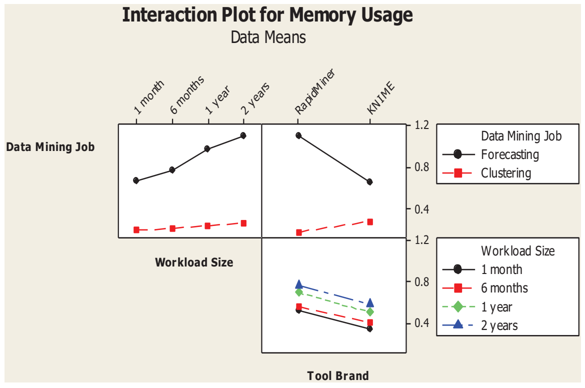
In terms of memory usage, the analysis in Figure 4.6c and Figure 4.6d shows two different tool brands have a clear cross trend concerning the data mining jobs, More specifically, RapidMiner uses less memory for the clustering jobs, while KNIME uses less memory for the forecasting jobs.

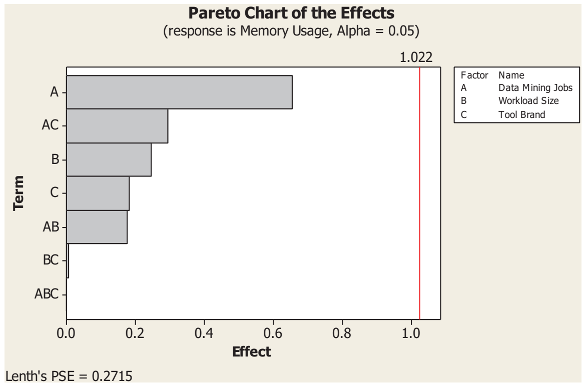




(a) Interactions between different experimental factors with respect to Execution Time (generated by Minitab)

(b) Effects of different factors and factor combinations on Execution Time (generated by Minitab).





(c) Interactions between different experimental factors with respect to Usage (generated by Minitab).

(d) Effects of different factors and factor combinations on Memory Usage (generated by Minitab).

Figure 4.6: Factor Impact Analysis

Analysis for Qualitative Features

For the qualitative features, we directly discuss them one by one to compare those two data mining tools, and the discussions are based on our experiences of implementing the aforementioned experiments.

- *Data Import Support:* Both RapidMiner and KNIME support importing data from a wide range of file formats and databases, such as CSV files, Excel files, Sequence files, etc. In addition, they also support model import and data streaming from various databases.
- *Data Export Support:* Similar to data importation, both KNIME and RapidMiner support enhanced data exportation support for a wide range of file formats, models, and databases. However, the open source version of RapidMiner has limitations to the support.
- *Node IO Limit:* Another desirable feature is the capability of a node to accept multiple input from and output flow connections to other nodes in a workflow management system. While RapidMiner allows multiple inputs and output flow capabilities in the workflow designing, KNIME has a limitation regarding input ports. Since a single node in KNIME has one or maximum two input port(s), making the workflow design complicated for complex workflows.
- *Scripting Language Support:* KNIME and RapidMiner both have efficient support for R and Python scripting languages. The advanced Node IO feature (as discussed in the above point) provides additional flexibility for external scripting languages in the workflow.
- *Visualization Support:* RapidMiner has exclusive dynamic data visualization support over KNIME. This feature provides the support to a wide range of charts for visualization, including scatter plot, heat maps, multiline plots, etc. However, this feature exacerbates its performance while visualizing a large amount of data.
- *Data Manipulation and Transformation:* KNIME and RapidMiner have similar number of nodes to support data manipulation, statistical modeling, and machine learning algorithms. They support machine learning algorithms including Decision Trees, K-Means, Support Vector Machine (SVM), and Bayesian Networks.
- *Interoperability in Batch Mode:* KNIME and RapidMiner support batch mode processing,

allowing them to run faster with loading GUI every time a workflow execution takes place.

- *Interface Usability:* KNIME has the same legacy interface of popular Eclipse IDE, making it easier for developers to interact efficiently with it. RapidMiner's interface is more user-friendly and interactive, and provides effective tutorials which make it easier for a novice to learn.

4.4.8 Conclusion and Documentation

As regulated by the principles of DoE, we summarize answers to the predefined requirement questions before drawing conclusions, as outlined below.

rq1: How fast do RapidMiner and KNIME perform data mining jobs respectively?

- *When running the forecasting job with different workload sizes, the execution time of KNIME varies ranging from about 35 to 107 seconds on average and of RapidMiner takes from nearly 30 to around 105 seconds (See Figure 4.5a). As for the clustering job with different workload sizes, the latency range of KNIME is from about 9.8 to 13.4 seconds and of RapidMiner's is from 8 to 9.3 seconds (See Figure 4.5b). RapidMiner shows comparatively lower latency in both the cases.*

rq2: How many resources do RapidMiner and KNIME need for data mining jobs respectively?

- *Since the data size is consistent for running both data mining tools, the implementation of algorithms by each tool is a key factor on the memory consumption of each type of job. In our case, KNIME shows less memory utilization in case of forecasting, whereas RapidMiner show less memory usage in case of clustering jobs, as shown in Figure 4.5c and Figure 4.5d. RapidMiner relatively requires less number of nodes to design a workflow, which entails less memory consumption.*

rq3: How well do RapidMiner and KNIME match the qualitative selection criteria?

- *Given the qualitative discussion in Section 4.4.7, KNIME beats RapidMiner in terms of "Data Import Support" and "Data Export Support". RapidMiner is better than*

KNIME in terms of "Node IO Limit", "Scripting Language Support", "Visualization Support" and "Interface Usability". KNIME and RapidMiner have equivalent capacity with respect to "Data Manipulation and Transformation" and "Interoperability in Batch Mode".

Overall, these answers reveal little difference between KNIME and RapidMiner regarding their quantitative features. The experimental analysis also confirms that *Tool Brand* is not a significant factor for executing data mining jobs, although RapidMiner runs slightly faster than KNIME in this study. Assuming the data analysis architecture has a specific emphasis on features such as "Interface Usability", we can select data mining tools based on their qualitative features. In this example, RapidMiner has more advantages over KNIME for interface usability. Therefore, RapidMiner can be chosen.

We have documented our evaluation logic and activities including directly reusable codes and scripts, which would facilitate telecom software engineers to replicate this evaluation study and evaluate other data mining tools.

4.5 Evaluation of Analysis Methods

The concept architecture in Figure 4.1 is further used to evaluate outlier detection methods with the workflow step of Process Data. Similar to the evaluation of a data mining tool for the whole workflow, case study, we employ the eight steps of DoE to compare methods.

4.5.1 Requirement Recognition

After thoroughly brainstorming the requirements with telecommunication engineers, we have developed the following five requirements for the analysis method:

- **Distribution Independent:** Several outlier detection methods, especially statistical methods, which are based on estimating standard deviation, make assumptions about underlying data distribution. For instance, GrubbFL s test (**?**) is an outlier detection technique that assumes the underlying dataset to be normally distributed. However, it is not possible to assume the distribution of all the features in a multivariate dataset. Such dataset contains several features having distinct data distributions. Thus, the outlier detection approach should be independent of the distribution.

- **Non-Parametric:** An outlier detection method should also be non-parametric. Parametric models have multiple fixed set of parameters. They need to be set to an appropriate value to achieve efficient detection. Estimating these values for parameters requires adjustment for every data distribution.

- **No Explicit Training:** Patterns observed in telecommunication data constantly change due to the addition of new features or continuous upgrading. Training detection models with an obsolete dataset entails inefficient results. Therefore, the outlier detection algorithm should be capable of detecting outliers without explicit training.

- **Applicable on Multivariate Dataset:** The dataset is multivariate that consists of several independent features. Thus, the detection method should be applicable on a multidimensional dataset.

- **Applicable on Large Dataset:** The outlier detection methods should scale as the size of data grows.

Following the DoE principles, the main evaluation requirements are identified as follows:

rq1: How efficient the outlier detection method is on a given telecommunication dataset?

rq2: How much is the resource utilization of the method?

rq3: How well does a method match the qualitative selection criteria?

4.5.2 Methodology Feature Identification

Based on requirements, the selection criteria are divided into qualitative indicators and one quantitative indicator that is performance. The selection criteria is listed in Table:4.4. We map these selection criteria to a set of outlier detection methods as shown in Table4.6 . The qualitative indicators help us to filter we out methods from further evaluation steps. For instance, since Grubb's test is distribution dependent, and its assumption of normal distribution contradict the requirements listed previously, this technique is not selected for further evaluation. Qualitative analysis are discussed later in section.

Table 4.3: Selection Criteria

Category	Features
Qualitative Indicator	Distribution Independent Non-Parametric No Explicit Training Multivariate Dataset Large Data Volume
Quantitative Indicator (Performance)	Efficiency Latency Memory Usage

Table 4.4: Selection Criteria

4.5.3 Metrics & Benchmarks Listing and Selection

The outlier detection method is supposed to extract data from the staging area 1 as shown in Figure 4.1 and process the data to determine outliers. Afterwards, the data is labeled and stored for further analysis in the staging area 3. Hence the execution of the outlier detection method reply on the entire phases of Extraction, Transform and Loading (ETL). We consider the latency and resource usage of the outlier detection method in the context of the whole ETL job. Hence the latency is measured by means of *execution time* defined as the time required for a ETL job completion using Equation (8). We also consider the resource usage in term of memory usage as the size of data grows.

4.5.4 Experimental Factors Listing and Selection

We identified experimentation factors and categorized them into four groups, namely workload-related, approach-related, efficiency-related and capacity-related factors.

- Workload-related factors:
  - *Job:* Outlier detection method

Table 4.5: Outlier Detection Techniques

Category	Outlier Detection Technique	Distribution Independent	Non-Parametric	No Explicit Training	Multivariate Dataset	Large Data Volume
Statistical Approach	Grubb's Test	x	x		x	x
	Chi Square Test	x	x		x	x
	Kernel Density Estimation (KDE)	x	x			x
	General Extreme Student Estimate	x	x		x	x
Classification Approach	One Class Support Vector Machine (SVM)			x		
	Bayesian Naive			x		
	Random forest			x		
Clustering Approach	K-Means			x		
	DBSCAN					
	OPTICS					
	Self-Organized Mapping (SOM)			x		
Nearest Neighbour Approach	Global KNN					
	Local Outlier Factor (LOF)					

Table 4.6: Outlier Detection Techniques

  - *Data size/Number of records:* 10,000 , 25,000, 50,000, 100,000, 800,000

- Efficiency-related factors:
  - *Threshold*
  - *False Positive/Detection Rate*
  - *True Positive/Detection Rate*
- Capacity-related factors,
  - *Latency:* Execution Time
  - *Resource usage:* Memory Usage

4.5.5 Experimental Design and Implementation

The experiment design utilize the experimental factors identified in the previous step to prepare for experimentation. Based on the metrics, benchmarks and experimental factors, two tables are designed as shown in table 4.7 and table 4.8.

Table 4.7: Efficiency Selection Metric

Trial	Method	Threshold	False Positive/Detection Rate	True Positive/Detection Rate
1	DBSCAN	?	?	?
2	DBSCAN	?	?	?
3	DBSCAN	?	?	?
4	OPTICS	?	?	?
5	OPTICS	?	?	?
6	OPTICS	?	?	?
7	LOF	?	?	?
8	LOF	?	?	?
9	LOF	?	?	?
10	GKNN	?	?	?

However, experiment implementation involves several tasks for data preparation before commencing the experimentation. These tasks are,

- Data Cleaning: It is a process which removes any sort of syntactic errors, semantic errors and coverage anomalies present in the dataset prior to its processing. For instance, in our case rows containing lexical errors such as *!DIV0!* present in the dataset are eliminated from the dataset. We manually removed the error by writing R data mining scripts as the amount of error present was limited and known.

- Data Labelling: The next step is to label the dataset after removing all the removal of unwanted and bad data rows from the dataset. We tag the data record in the dataset based on domain knowledge provisioned by telecommunication experts. For instance, we know that the success rate always lies between 0 and 1. Any value less than zero and more than one is an anomaly and thus tagged.

After data preparation, we feed the dataset into the workflow architecture to evaluate different

Table 4.8: Performance Selection Metric

Experiment	Mtethod	Data Size	Execution Time	Memory Usage
1	DBSCAN	10,000	?	?
2	DBSCAN	25,000	?	?
3	DBSCAN	50,000	?	?
4	DBSCAN	100,000	?	?
5	DBSCAN	800,000	?	?
6	OPTICS	10,000	?	?
7	OPTICS	25,000	?	?
8	OPTICS	50,000	?	?
9	OPTICS	100,000	?	?
10	OPTICS	800,000	?	?
11	LOF	10,000	?	?
12	LOF	25,000	?	?
13	LOF	50,000	?	?
14	LOF	100,000	?	?
15	LOF	800,000	?	?
16	GKNN	10,000	?	?
17	GKNN	25,000	?	?
18	GKNN	50,000	?	?
19	GKNN	100,000	?	?
20	GKNN	800,000	?	?

anomaly detection techniques. The experimentation takes place by gradually increasing the amount of data points (or records). We plotted Receiver operating characteristic ROC curves to measure and compare the efficiency of these outlier detection techniques. ROC curve is a graphical evaluation method used for assessing machine learning algorithms, especially, binary classifiers. ROC curve is obtained by calculating True Positive Rate (TPR) and False Positive Rate (FPR) of the detection algorithms. Where TPR is a measure of correct positives out of all the positives detected by an algorithm. On the other hand, FPR is a measure of incorrect positives out of all the positives detected. The TPR and FPR represents the *y-axis* and *x-axis* of the ROC curve and are formulated using eq 9 and eq 10, respectively.

$$TPR = \frac{True\ Positive}{(True\ Positive + False\ Negative)} \tag{9}$$

$$TPR = \frac{False\ Positive}{(False\ Positive + True\ Negative)} \tag{10}$$

ROC curve for an outlier detection method is obtained by running it under different parameter. The performance of each outlier detection technique depends on the value assigned to its parameters. Each parameter has different impact on the performance. For instance, the accuracy of the DBSCAN's prediction highly depends on the value of epsilon and minimum points (i.e. the maximum distance between two samples for them to be considered as in the same neighbourhood and number of data points to consider to form a dense cluster, respectively) (**?**). Therefore, we carefully selected only those parameters which have maximum impact on algorithms performance i.e. epsilon in case off DBSCAN. After running each outlier detection technique, we compare the results (i.e. prediction labels) produced during the detection process with the original labels to calculate TPR and FPR. Then, we plot ROC curve for each outlier detection technique and compared them thorough analysis. In addition, we also profile execution time and memory usage of detection techniques and plotted the results.

4.5.6 Experimental Analysis

Analysis of Quantitative features

- *Efficiency:* In the previous step, we carefully executed the experimentation process and plotted the results for analysis. We plotted ROC curve for the outlier detection techniques as shown in Figure **??**. The ROC curve of Local Outlier Factor (LOF) method shows the best performance among other detection techniques. It shows the maximum TPR of 0.7 attained with minimum FPR of 0.15, approximately. Whereas, OPTICS and DBSCAN have almost same area under the curve and shows similar behaviour for obvious reasons.

- *Execution Time and Memory Usage:* We scrutinize the outliers detection methods to estimate average execution time and average memory usage. For this purpose, ten readings were taken for execution time and memory usage by gradually increasing the data size, i.e. 10,000, 25,000, 50,000, 100,000 and 800,000 records, at the end of the each cycle of ETL job. We employ Pareto Chart to visualize effects of the different experimental factors and their combinations. We compare LOF with DBSCAN and OPTICS, as shown in Figure 4.9 and Figure 4.10. In Figure 4.9, the analysis shows that the factor Data Size has a relatively significant

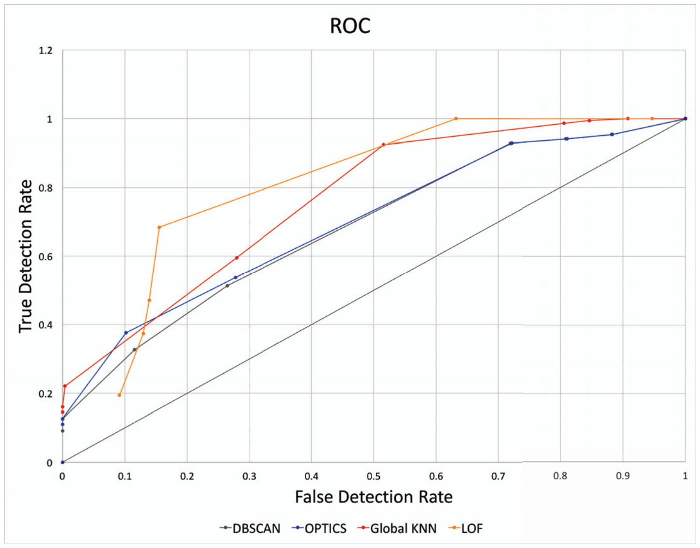
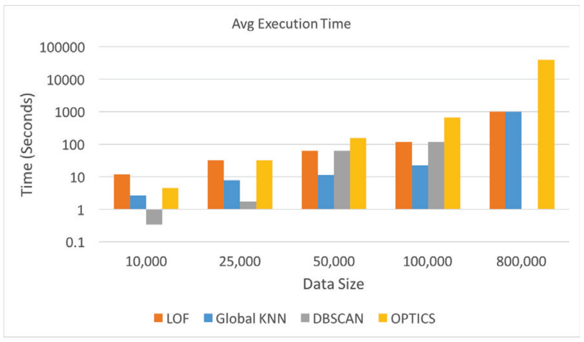
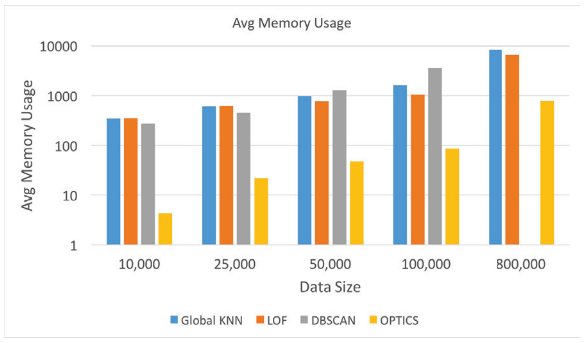


Figure 4.7: Efficiency Analysis Result



(a) Execution Time Analysis



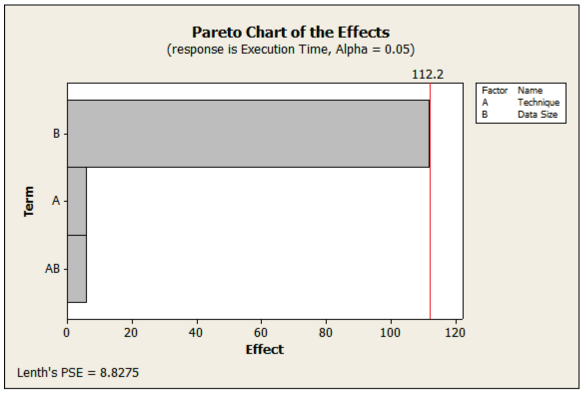
(b) Memory Usage Analysis

Figure 4.8: Average execution time and memory usage of outliers detection methods.

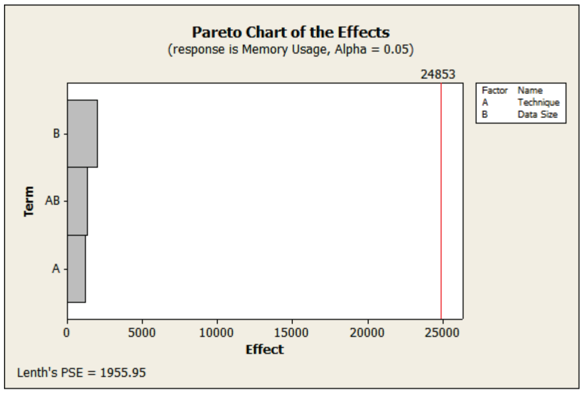
influence on the experimental results. However, in Figure 4.10 neither Technique nor Data Size has significant influence on the experimental results

Analysis for Qualitative Features

In the case of outlier detection techniques, the quality attributes are used as selection criteria. The selection process is a subtask and forms a basis quantitative analysis. The outlier detection technique which complies all these qualitative criteria will be assessed. Each qualitative criteria is analyzed independently (i.e. without observing the effect of on property on other) for each outlier detection technique. The study results are illustrated in Table 4.6.

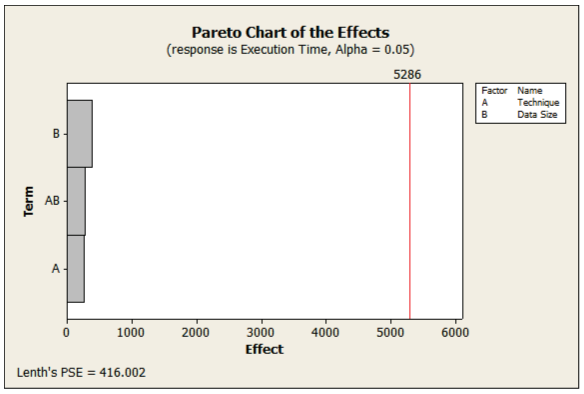


(a) Execution Tim Analysis

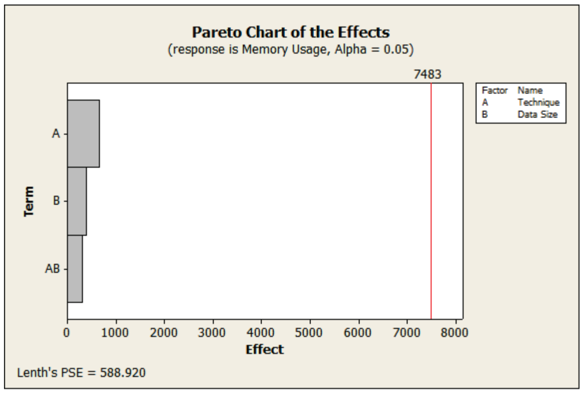


(b) Memory Usage Analysis

Figure 4.9: LOF Vs DBSCAN



(a) Execution Time Analysis



(b) Memory Usage Analysis

Figure 4.10: LOF Vs OPTICS

- *Distribution Independent:* Most of the statistical outliers detection techniques, such as GrubbFL s test and Chi-Square test, are distribution dependent. These statistical approaches rely on the underlying data distribution, and most of them are only applicable on a normally distributed dataset. In our case, these techniques are inapplicable as the base stations or eNodeBFLs generates data with distinct data distributions. After scrutinizing the outlier detection methods in consideration, we found that GrubbFL s Test, Chi Square Test, Kernel Density Estimation (KDE), General Extreme Student Estimate are distribution dependent whereas the remaining techniques are independent of underlying data distribution.
- *Non-Parametric:* Similarly, distribution independent methods are also non-parametric. Therefore, based on our analysis GrubbFL s Test, Chi Square Test, Kernel Density Estimation

(KDE), General Extreme Student Estimate techniques are parametric, and remaining detection techniques show non-parametric behaviour.

- *No Explicit Training:* Telecommunication companies generate a large volume of data. The large volume makes training data preparation a tedious task. Therefore, outlier detection technique should not need any prior data preparation. Clustering and classification detection techniques such as One-class support vector machine (SVM), Bayesian naive, Random forest, K-Means, Self-Organized Mapping (SOM) are effective methodologies for outlier detection but need explicit training. Training these techniques require preparation of training data which is not feasible in our case. However, outlier detection techniques such as GrubbFL s Test, Kernel Density Estimation (KDE), DBSCAN, OPTICS, Global KNN, Local Outlier Factor (LOF) do not require any training data, thereby, more suitable for analysis.
- *Multivariate Data:* Statistical outlier detection techniques i.e. (GrubbFL s Test, Chi Square Test, General Extreme Student Estimate techniques) are only applicable on a univariate dataset. Whereas, Kernel density estimation (KDE) has an implementation for both univariate and multivariate dataset. As far our knowledge is concerned, most of the machine learning algorithms apply to both univariate and multivariate data. However, they tend to perform better on multivariate data than univariate data.
- *Large Volume of Data:* As it has been already mentioned several times, that the base stations generate a large volume of data. Hence, the analysis technique should be capable of processing large data. In such case, GrubbFL s Test, Chi Square Test, General Extreme Student Estimate are inapplicable for large datasets, but the remaining techniques work fine on large datasets.

Based on the above analysis, only four outlier detection techniques are assessed quantitatively. These techniques are DBSCAN, OPTICS, LOF and Global KNN.

4.5.7 Conclusion and Documentation

As regulated by the DoE principles, we summarize answers to the predefined requirement questions before drawing conclusions, as outlined below.

rq1: How efficient the outlier detection method is on a given telecommunication dataset?

- *The ROC curve shows LOF's performance to be quite promising on this particular dataset with TPR of 0.7 with minimum FPR of 0.15, approximately. Whereas, the Global KNN methods has TPR of 0.37 with FPR as low as 0.1. DBSCAN and OPTICS shows almost similar performance with TPR 0.5 and FPR 0.25, approximately.*

rq2: How much is the resource utilization of the method?

- *The analysis results indicate that Global KNN is the fastest among others outperforming them with execution time of approximately 2 seconds to process 100,000 data points. However, its execution time eventually becomes similar to LOF with increasing data size becoming 3 seconds for 800,000 data points. GKNN and LOF also consume almost same amount of memory for analysis. However, in case of memory usage, OPTICS has the minimum utilization for 800,000 data points. Although, the method has maximum execution time but its memory consumption is comparatively very low. DBSCAN is unable to process large data volume due to excessive memory use.*

rq3: How well does a method match the qualitative selection criteria?

- *Given the qualitative discussion in Section 4.4.7, eliminating the detection methods that do not meet our criteria led us to extract only four methods out of the selected ones. These methods are DBSCAN, OPTICS, Global KNN and Local Outlier Factor. These methods are non-parametric, distribution independent, multivariate and does not require explicit training.*

4.6 Motivation for Architecture Design

The evaluation process has allowed us to explore and understand several challenges associated with monitoring data analysis. The monitoring data exhibits all the five VFLs of Big Data: Volume, Velocity, Variety, Value, and Veracity (51) which make the data exploration and itFLs analysis bit complicated. In such a scenario, general, traditional tools and analysis methods are inefficient and ineffective. Firstly, its requires a scalable storage for accommodating such huge amount of data. Secondly, the storage system should be available to guarantee the reliability of data i.e. data should be missing because of storage node failure. It also ensures that the data is always available for active monitoring. Thirdly, there is a need to provide advanced analytics such as forecasting for monitoring data analysis. To overcome this monitoring analysis issue we proposed a monitoring data analysis architecture. The architecture is designed to provide advanced analytics for monitoring analysis in an available and scalable environment. The architecture design is discussed in detail in the next chapter.

4.7 Conclusion

In this chapter, we present our design of experiment to evaluate data mining tools and outlier detection approaches into two different case studies. The aim of this evaluation is to observe them both quantitatively and qualitatively when running data analytic jobs. The analytics helps to understand the number of connected users per cell on the base stations in the trial mobile network. Since the data is constantly generated and accumulated, the workload size given the period of time datasets are collected become one experimental factor in both the cases. The evaluation becomes complicated because of Combination of several factors. Therefore, applying DOE principles to our evaluation study clearly make this practice in a structured eight-step procedure. At the end, the evaluation study allows us to address the evaluation requirements. The threats of validation is mainly in the evaluation environment. We used default settings in each study. Each case can be further optimized to achieve a better response time and memory utilization. Our purpose is not to compare any data mining tools or outlier detection approach in an absolute measurement, instead, we focus on the evaluation method that allows observing the features and metrics of these tools or approaches, respectively. The insights help to make the decision on how a data mining tools or outlier detection approaches should be fit into the big data analysis architecture. We believe this evaluation study based on design of experiment can be extended to other data mining tools or outlier detection approaches of interests.

Chapter 5

CloudAnalyzer: An Architecture Design for Big Monitoring Data

5.1 Introduction

A generic monitoring system performs four tasks to monitor IT infrastructure (*i.e. cluster of nodes*). This process involves: 1) data collection, 2) data storage, 3) data analysis, and 4) data visualization. Data collection is a process of gathering resource information (i.e. CPU Usage, Memory Usage, etc.) from each node of a cluster and storing it in a data storage space. Information gathered could be stored in different formats, such as in a flat file or a row in a database. Stored data is analyzed to comprehend different aspects of the cluster through effective visualization. The monitoring process becomes complicated when it involves monitoring huge volume of data (*i.e. Big Data*).In such a scenario, traditional monitoring systems are inefficient for analyzing monitoring data. An efficient big monitoring data analysis system requires reliable resource information collection, essential analysis methodologies, and effective visualization: it demands data storage space which is fault-tolerant, scalable and highly available. Also, It should have advanced and reliable analytics methods than simple aggregation. Besides, there should be an interface for generating reports and displaying the results for effective documentation and information sharing.

We propose an envisioned architecture design to analyze big monitoring data gathered from deployed IT infrastructure. It has advanced analytics for workload forecasting and pattern matching for efficient monitoring. The core of the architecture contains a central component consists of Representation State Transfer (or REST) API which integrates a storage and visualization components.

In this chapter, we explain this proposed architecture design (Section II) followed by its deployment (Section III). We discuss a case study of analysis of a time series dataset (Section IV), and end with a summary (Section V).

5.2 CloudAnalyzer: A Cloud Monitoring Architecture

We design a cloud monitoring infrastructure called *CloudAnalyzer*, to address the above mentioned issues. It adopts a layered architecture style as shown in Figure 5.1. The architecture is parted by an extension line into *Core* and *Extension*. The *Core* enables the user to perform advance analysis such as workload prediction and workload pattern matching on small datasets. As the name specifies, the *Extension* adds the ability to run distributed computing algorithms on large datasets. The architecture is discussed in detail in the following sections.

5.2.1 *CloudAnalyzer*: Core Architecture

The *Core* is composed of three essential layers: 1) a *CloudAnalyzer Extension*, 2) a *CloudAnalyzer Server*, and 3) a *SolrCloud Cluster*. The interaction between these elements allows the monitoring system (i.e. *CloudAnalyzer*) to process the workload traces and monitor the patterns that have occurred.

The top layer, *CloudAnalyzer Extension*, is a SMW extension design that provides an interactive interface for underlying architectural layers. It enables a user to configure a wiki page for a particular trace analysis, set up parameters, launch requests of data access and processing, and finally, display the generated report. It uses the feature of *Special Pages* of the MediaWiki engine. These *Special Pages* are wiki pages that are created on demand to perform a particular function. Hence, the *CloudAnalyzer extension* is a SMW wiki special page created upon request to provision the web interface: it sends the configuration parameters set up by the user making a HTTP request to the layer below. Given the range of traces and the type of analysis method, the *CloudAnalyzer Extension* uses the annotations from SMW to structure the analysis results and generate corresponding wiki

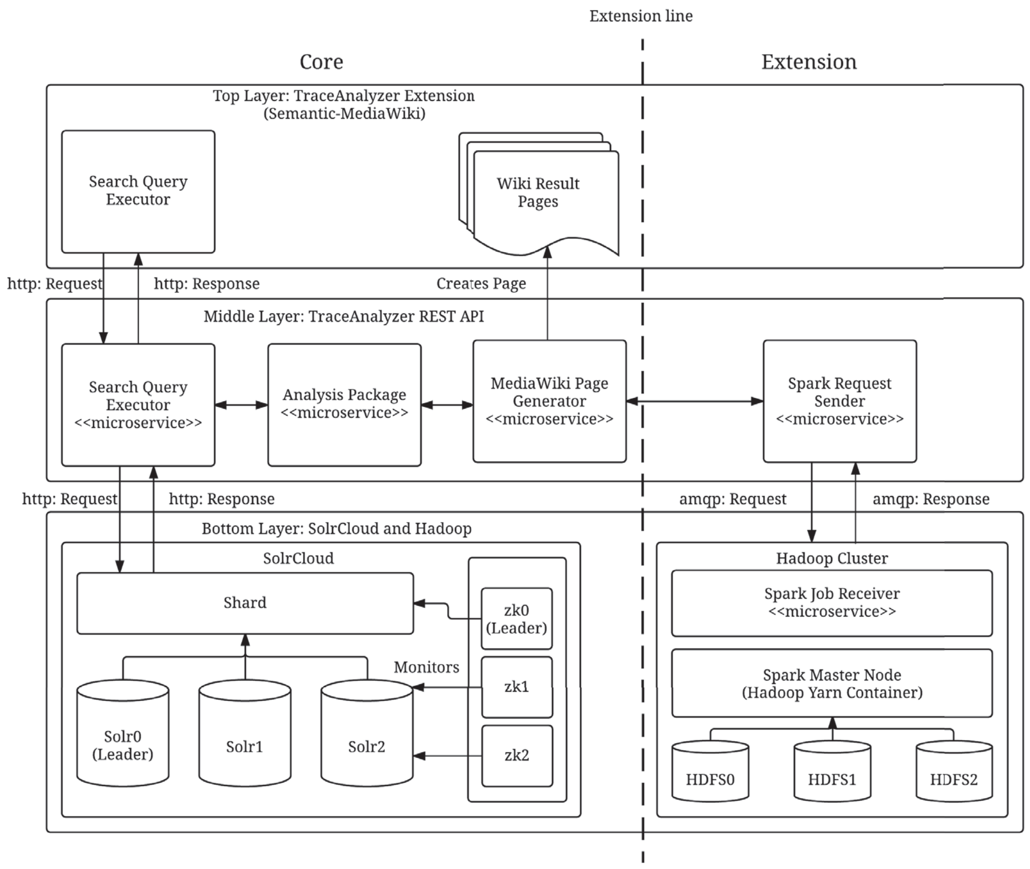

Extension line
Core
Extension
Top Layer: TraceAnalyzer Extension (Semantic-MediaWiki)
Search Query Executor
Wiki Result Pages
http: Request
http: Response
Creates Page
Middle Layer: TraceAnalyzer REST API
Search Query Executor <<microservice>>
Analysis Package <<microservice>>
MediaWiki Page Generator <<microservice>>
Spark Request Sender <<microservice>>
http: Request
http: Response
amqp: Request
amqp: Response
Bottom Layer: SolrCloud and Hadoop
SolrCloud
Shard
zk0 (Leader)
Monitors
zk1
zk2
Solr0 (Leader)
Solr1
Solr2
Hadoop Cluster
Spark Job Receiver <<microservice>>
Spark Master Node (Hadoop Yarn Container)
HDFS0
HDFS1
HDFS2

Figure 5.1: *CloudAnalyzer: Cloud Monitoring Architecture*

pages. In SMW, annotation *Category* provides automatic indices, similar to tables of contents. The architecture provides several analysis methods, such as Kernel Density Estimation (KDE), Forecasting, etc. The SMW categorizes the generated reports based on these analysis methods. Once the result returns from the *CloudAnalyzer* server, a wiki page, or report, is created and then semantically linked to its corresponding category for its respective analysis method.

The middle layer, *CloudAnalyzer REST API*, adopts the Representational State Transfer (REST) architectural style instead of using traditional HTTP methods. The advantages of REST include the following: it utilizes the Uniform Resource Locator (URL) to identify various web resources; it dissolves a very complex application into simple resources; it minimizes the network latency enabling better performance; and, it enables independence and scalability of component implementations in a distributed systems (25; 46). Thus, *CloudAnalyzer REST-API* acts as a bridge to connect *CloudAnalyzer Extension* and the underlying *SolrCloud Cluster*. The core of *REST-API* is built on three microservices, specifically, search, analysis, and page generator. The *CloudAnalyzer Extension* utilizes the *REST-API's* resources to send processing requests, invoking the analysis package, which utilizes the search microservice to extract datasets of interest. The search microservice then sends a search query to the shard of SolrCloud using SolrCloud's API via HTTP. Upon return of the data, the analysis package produces the result and sends success response to the *CloudAnalyzer*. On receiving a success response, it initializes another request to plot forecasted data for that particular dataset via HTTP POST method. Finally, the extension sends a request to generate a wiki page, calling the page generator. This microservice generates the analysis result and creates a new wiki page using MediaWiki's markup language. The *CloudAnalyzer REST-API* is discussed in detail in later section.

At the bottom layer consists of a cluster of Apache Solr nodes. This *SolrCloud Cluster* is responsible for effective data storage and provides availability and scalability to users. It contains the subset of optional features in Solr for indexing and searching. It also enables horizontal scaling of a search index using sharding and replication. The structure of SolrCloud cluster consists of the following components:

- *SolrCore* encapsulates a single physical index.
- *Node* is a single instance of Solr. A single Solr instance is bound to a specific port and can have multiple SolrCores.
- *Collection* contains a single search index distributed across multiple nodes. A collection has a name, a shard count and a replication factor. A replication factor specifies the number of copies of a document in a collection.
- *Shard* is a logical slice of a single collection. Each shard has a name, a hash range, a leader, and a replication factor.
- *Replica* hosts a copy of a shard in a collection, implemented as a single index on a SolrCore. One replica of every shard is designated as a leader to coordinate indexing for that shard. Each shard has one and only one leader at any time and leaders are elected using ZooKeeper.
- *Cluster* includes all the nodes that hosts SolrCores.

- *ZooKeeper* provides a distributed coordination service that provides the cluster state management and the leader-election mechanism for fault-tolerance. ZooKeeper nodes constantly monitor the Solr nodes in a cluster.

5.2.2 *CloudAnalyzer*: Extension Architecture

The *Extension* takes advantage of REST API and acts as a plugin to the *Core*. It incorporates additional components to facilitate complex computation of large datasets. It integrates frameworks such as Apche Hadoop, with the *Core*. In our case, it consists of an Apache Spark application running on a Hadoop Yarn cluster. The distributed architecture extension is illustrated in Figure 5.1.

The top layer, *CloudAnalyzer Extension*, is used by both *Core* and *Extension*. It provides the web interface and performs the same task of initiating a process. However, it adds another microservice *Spark Request Sender* to the REST API. It is responsible for comprehending the request initiated by *CloudAnalyzer Extension* and forwarding it to another microservice *Solr Request Receiver* running at the bottom layer. The interaction between sender and receiver mircoservices takes place using RabbitMQ. RabbitMQ is a message broker software that implements Advanced Message Queuing Protocol (AMQP). *Solr Request Receiver* receives the task information and transforms into Spark command to process it using Spark cluster.

The bottom layer is consists of Spark on Hadoop Yarn Cluster. Hadoop Yarn is the key feature of second generation Apache Hadoop. It is a resource management platform for managing and scheduling user application in a cluster. In *Extension*, the Hadoop Yarn cluster contains a Yarn resource manager, a Hadoop name-node and Spark running on a single node. It is also connected to other Hadoop file system (HDFS) data-nodes, where the user stores resource data as well as the results retrieved from Spark jobs. The HDFS ensure availability of the data. Also, it is scalable for accommodating the increasing data size.

5.3 *CloudAnalyzer*: Workflow

The basic workflow of *CloudAnalyzer* is illustrated in Figure 5.2. The process begins when a user interacts with the *CloudAnalyzer Extension* to send a monitoring request. The monitoring request with specified configuration parameters is received by the *CloudAnalyzer Server*, which interprets the request, and initiates a connection establishments process. It is responsible for locating the available Solr node for establishing a connection. The required amount of resource data is then retrieved from SolrCloud, and forwarded to the analysis package for processing. The analyzed results are transformed into wiki pages and the page is injected into SMW. Finally, a user can go through wiki pages to view or share the results.

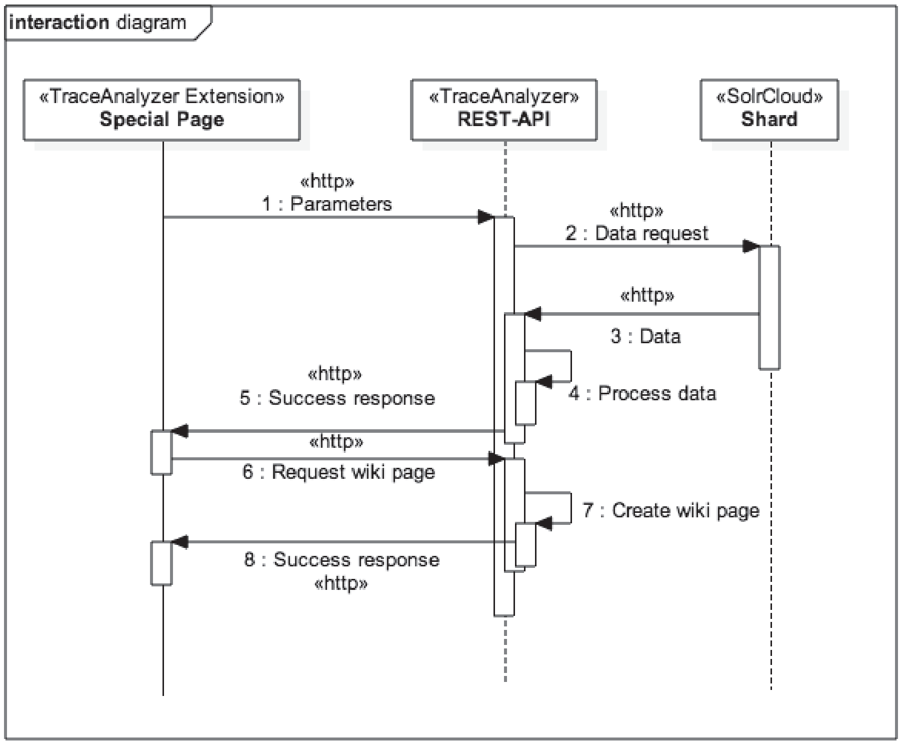


Figure 5.2: *CloudAnalyzer*: *Core* Workflow

As illustrated in the Figure 5.3, workflow of the extended architecture resembles the workflow of the *CloudAnalyzer Core Architecture*. The special wiki page accepts the required input parameters from the user. These parameters are then forwarded to the *CloudAnalyzer REST-API* in the form of a HTTP POST request. The *CloudAnalyzer REST-API* comprehends the request and forwards the parameters that has been recieved wrapped as JSON message to Spark cluster's main node. The node receives the JSON message and submits the job request to the Spark cluster. The message encoded in JSON format is passed to the Spark cluster's main node using the

Spark Request Sender microservice present in the *CloudAnalyzer REST-API*. The *Spark Request Receiver* microservice running on the Spark cluster's main node receives the message and decodes it, submitting the requested Spark job. The messages are passed between nodes using the Advanced messaging queueing protocol (AMQP). AMQP is a standard protocol for message-passing between different hosts in a cluster. The spark job after successful completion, the spark job saves the result and transfers it to the *CloudAnalyzer REST-API* via secure-shell (ssh). In the end of this sequence, *CloudAnalyzer REST-API* sends a success response to the *CloudAnalyzer extension*, which in turn sends another request to *CloudAnalyzer REST-API* to generate a wiki page for the specific result. Currently, we have implemented KMeans clustering algorithm using the sparkFLs machine learning library to analyze the trace data by dividing it into required number of sets.

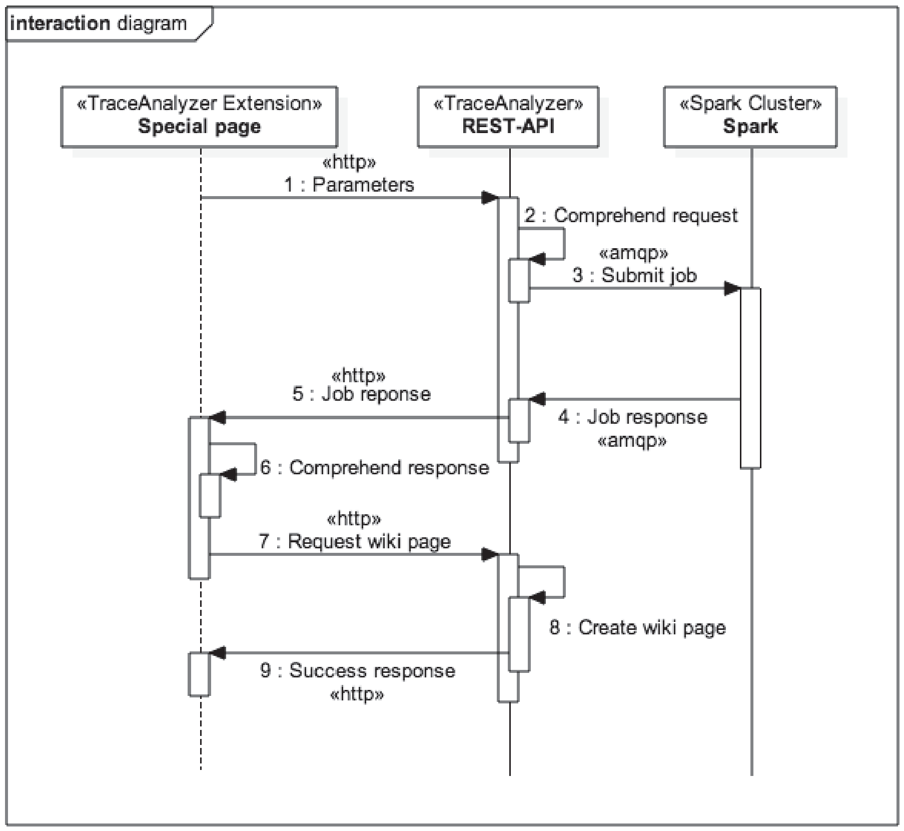


Figure 5.3: *CloudAnalyzer*: *Extension* Workflow

5.4 REST API

The *CloudAnalyzer REST-API* provides five important Restful resources as shown in Figure 5.4. These service resources provisions access to *CloudAnalyzer*'s core functionalities. The four RESTful service resources are describes as follows,

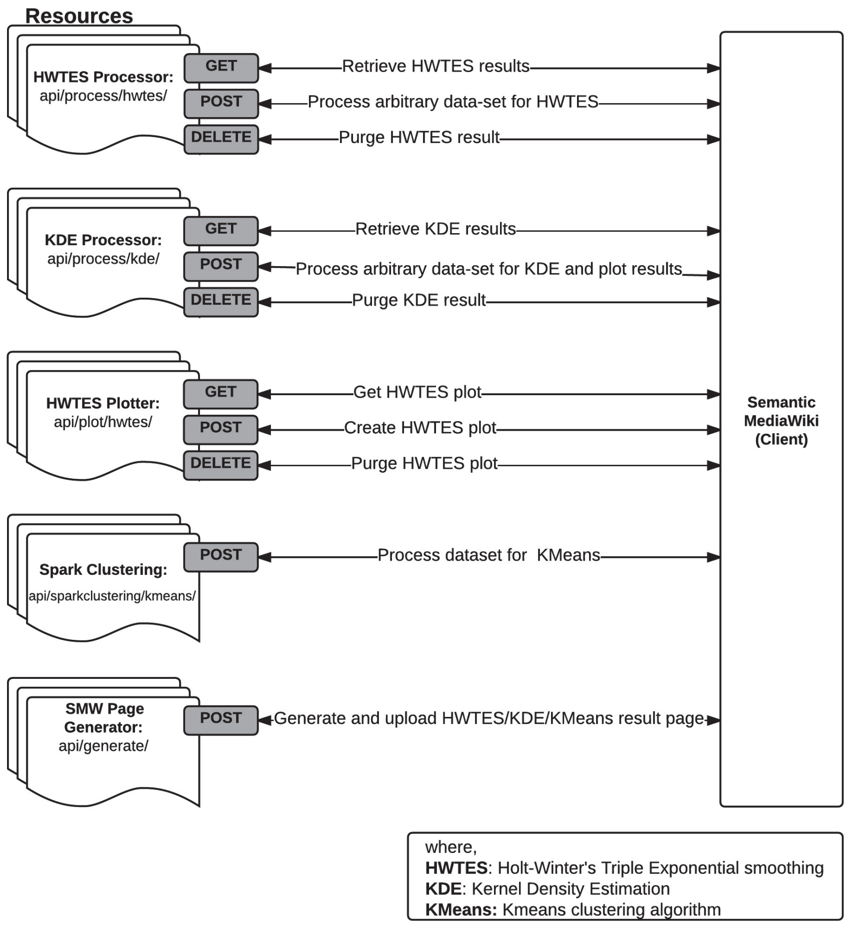

Resources
HWTES Processor: api/process/hwtes/
GET
Retrieve HWTES results
POST
Process arbitrary data-set for HWTES
DELETE
Purge HWTES result
KDE Processor: api/process/kde/
GET
Retrieve KDE results
POST
Process arbitrary data-set for KDE and plot results
DELETE
Purge KDE result
HWTES Plotter: api/plot/hwtes/
GET
Get HWTES plot
POST
Create HWTES plot
DELETE
Purge HWTES plot
Spark Clustering: api/sparkclustering/kmeans/
POST
Process dataset for KMeans
SMW Page Generator: api/generate/
POST
Generate and upload HWTES/KDE/KMeans result page
Semantic MediaWiki (Client)
where,
HWTES: Holt-Winter's Triple Exponential smoothing
KDE: Kernel Density Estimation
KMeans: Kmeans clustering algorithm

Figure 5.4: *CloudAnalyzer REST API*

- *HWTES Processor* enables the client to process a specified range of a dataset to determine workload forecast using Holt Winter's exponential smoothing (HWTES) method, which runs forecasting on data stored in both the file system as well as on SolrCloud. Forecast results can be pulled out anytime using the unique dataset identifier, the initial slice index.
- *KDE Processor* provides services to determine data points lying outside a particular threshold

for an arbitrary dataset. It utilizes the Kernel density estimation (KDE) method to determine workload patterns. Similar to *HWTES Process*, results can be retrieved from both the file system as well as SolrCloud. It also contains an implicit plotting service that plots the analysis results and stores them within the server. Both workload patterns and plots can be accessed using the initial slice index.

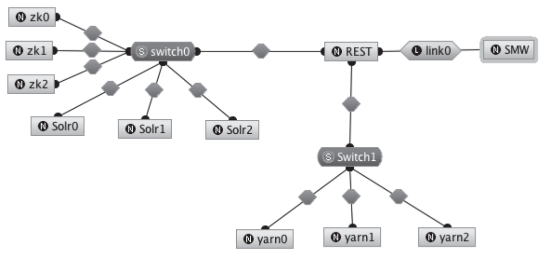
- *HWTES Plotter* provides functionality to plot the forecast data produced by *HWTES Processor*.
- *Spark Clustering* service runs clustering algorithms on large datasets stored in HDFS to get better a insight of the dataset. It also plots results obtained for visualization.
- *SMW Page Generator* creates SMW pages to display the analysis results. It automatically uploads the pages to SMW client.

The complete documentation of these resources is provided in APPENDIX A.1.

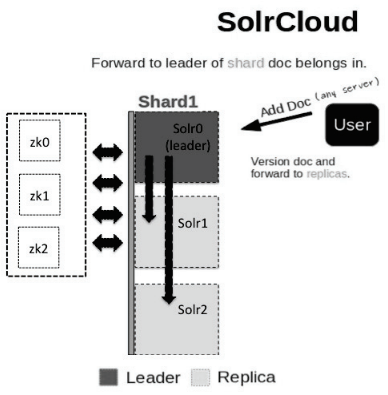
5.5 Architecture Deployment

We deploy the *CloudAnalyzer*'s prototype using a cluster of Emulab nodes. Emulab is *IaaS* provided by the University of Utah (31). We acquire eight nodes in total to set up the testbed. Out of eight nodes, six nodes are used to deploy *SolrCloud Cluster*. SolrCloud requires a ZooKeeper ensemble for monitoring or manage the Solr nodes, thereby three of these six nodes are set up as ZooKeeper nodes to monitor the other three Solr nodes. The remaining two nodes are used for SMW and *CloudAnalyzer Server* each. The network configuration of the *CloudAnalyzer* is shown in Figure 5.5a.

SolrCloud Cluster is configured to use a single shard and two replicas in a collection. Figure 5.5b shows the configuration settings for *SolrCloud Cluster*. Initially, resource information (CPU, Memory, etc.) gathered by monitoring agents is transferred to the leader node. The stored information is replicated and stored in the other two replica nodes. Upon failure of the leader, information is available from the replicas. The figure also illustrates the ZooKeeper ensemble *zk0*, *zk1* and *zk2* monitoring shard that consists of three Solr nodes. This ensemble is also responsible for re-electing the leader node in case of failure.



(a) Emulab Nodes Configuration



(b) Single shard and two replicas configuration

Figure 5.5: Architecture Deployment Configuration

In addition, we also deploy 3 additional nodes, *yarn0*, *yarn1* and *yarn2*, to the*CloudAnalyzer Core architecture* extension in order to add another component, the Spark cluster. One node is specified as the Hadoop Yarn resource manager. The Hadoop Yarn's namenode contains a resource manager and a Spark application. It also includes a microservice for receiving requests from the *CloudAnalyzer REST-API*. There are two more nodes connected to the Hadoop Yarn and serve the purpose of data-nodes for HDFS storage.

5.6 Case Study on Monitoring Data Analysis

We consider a case study on analyzing the trace data to predict the workload and understand patterns in a time series dataset that has levels, trends and seasonality factors. For this purpose, we use the traceversion1 dataset release by Google (63) as the input to the monitoring system. Google's traceversion1 is a collection of resource information gathered from IaaS. It contains the CPU and memory usage of jobs acquired by a cluster of machines at a particular timestamp. The data is completely anonymized in several ways. There are no task or jobs name, only numeric identifiers. Timestamps are relative to the start of data collection. The consumption of CPU and memory is obscured using a linear transformation (63). We presume the data is normally distributed based on *central limit theorem*, which states that the mean of an arbitrary dataset of a large size from an arbitrary distribution has approximately normal distribution.

5.6.1 Data Loading

The traceversion1 dataset consists of unstructured data with no unique features. However, SolrCloud requires a unique feature to index the dataset in order to provide its unique functionalities. Therefore, we use SolrCloud's implicit indexing functionality that automatically creates a unique id for each data record while uploading. The automatically generated unique id is configured using Solr schema file as shown below:

```
<fieldType name="uuid" class="solr.UUIDField" indexed="true"/>
<uniqueKey>id</uniqueKey>
```

An example of automatically generated unique id is also given below:

```
<str name="id">b6f17c35-e5ad-4a09-b799-71580ca6be8a</str>
```

In addition, SolrCloud also supports several data uploading methods, termed as data handlers, as specified in (**?**). These data handlers include: 1) index handlers for uploading data from files; 2) data import handlers for uploading data from database; and 3) HTTP POST for uploading data over the network. As the traceversion1 dataset is in CSV format, we used CSV index handler to directly upload the dataset to SolrCloud.

SolrCloud does not support interfacing with Hadoop Yarn cluster, and it does not have capabilities to retrieve data from HDFS. However, a single Solr instance can be run over HDFS, but it retains its high availability and scalability. To avoid this situation, another copy of trace data is moved to HDFS periodically for processing. The file system is dedicated to the Spark cluster and provides highly available and scalable storage capabilities.

5.6.2 Data Query

In *CloudAnalyzer*, data queries are sent via HTTP POST requests. These requests contain the parameters for specific analysis methods. For instance, if a user intends to request 10,000 data records from SolrCloud, a sample query is shown below:

```
se = SearchOptions();
se.commonparams.q('*:*').rows('100000')
response = trace_collection.search(se)
```

The queried data is received in JSON format via HTTP from SolrCloud. The returned data is further processed by the analysis microservice.

5.6.3 Workload Prediction

The analysis microservice of the *CloudAnalyzer* implements two main analysis methods, the HWTES and KDE, explained in detail in Chapter 2. The analysis methods are implemented using the Python programming language. We use *HWTES* to forecast the CPU usage of a Google cluster. Prediction of the future values of CPU usage enables us to trigger an alert in case of critical events. For this purpose, we periodically analyze a data chunk of 10,000 data points., split into a number of seasons, each with duration *L*. In our case, the 10,000 data points are split into 4 seasons, *4L*, whereby *L* contains 2,500 data points. Each season and trend are predicted at the end of each period of *L* based on the growth of the previous season and trend. As a result, the analysis method provides forecasting for time series data of length *4L*. Figure 5.6 illustrates the*HWTES* result for a data chunk of 10,000 data points of CPU usage from the trace version1 of a Google trace. This particular result is predicted with smoothing constants alpha(α) = 0.2, beta(β) = 0.2 and gamma(γ) = 0.05.

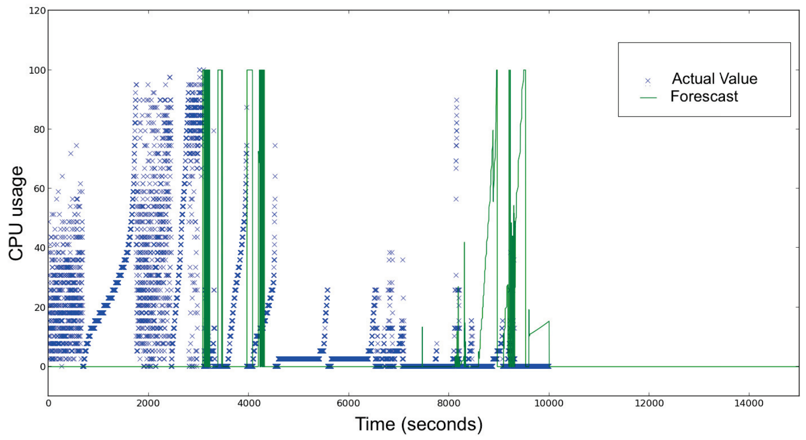


Figure 5.6: Forecast using *HWTES*

5.6.4 Workload Pattern Matching

We use *KDE* for determining the workload pattern in traceversion1 monitoring data. Firstly, *HWSES* is used to decide the upper and lower thresholds, which are calculated by adding and subtracting the standard deviation from the produced mean, respectively. The smoothing analysis of the CPU usage for 10,000 data points from traceversion1 is shown in Figure 5.7. Once the threshold is set, the kernel density is estimated for outliers. Outliers are data points lying above the upper threshold and below the lower threshold. The kernel density estimated for 4 seasons is shown in Figure 5.8 for outliers above the upper threshold and Figure 5.9 for those below the lower threshold.

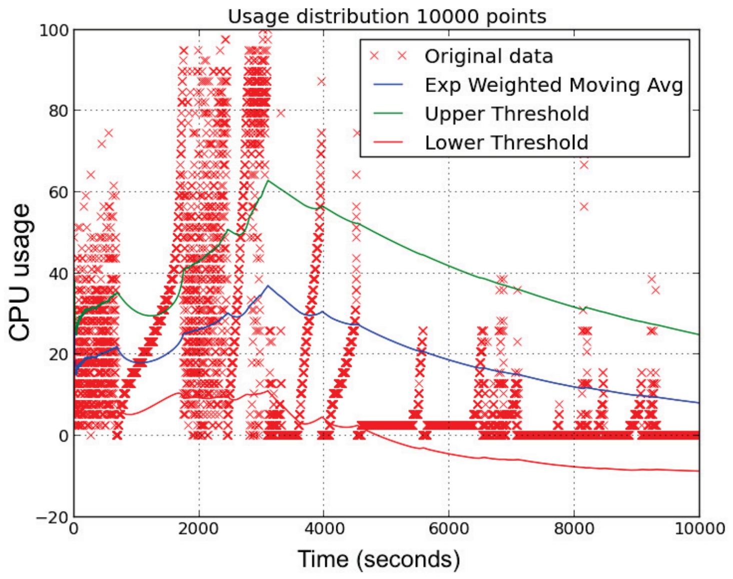


Figure 5.7: Moving average using Holt-Winters single exponential smoothing

By calculating the Pearson Correlation of two sequential seasons, we can determine if the outlier distribution changes significantly or not. The Pearson Correlation is a measure of how well two datasets are related to each other. The relation is shown in the following Eq.11,

$$r = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{\sqrt{[n\Sigma x^2 - (\Sigma x)^2][n\Sigma y^2 - (\Sigma y)^2]}} \tag{11}$$

where Pearson's *r* value can range from -1 to 1, such that linear relationship between variables is the following: a perfect negative linear relationship if -1; no linear relationship if 0; perfect positive

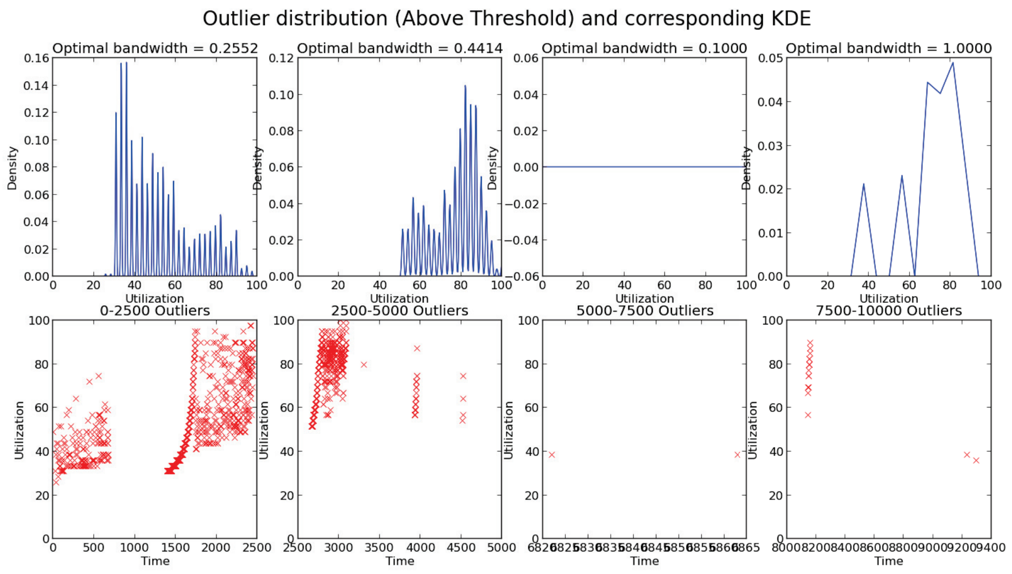


Figure 5.8: KDE for outliers above threshold

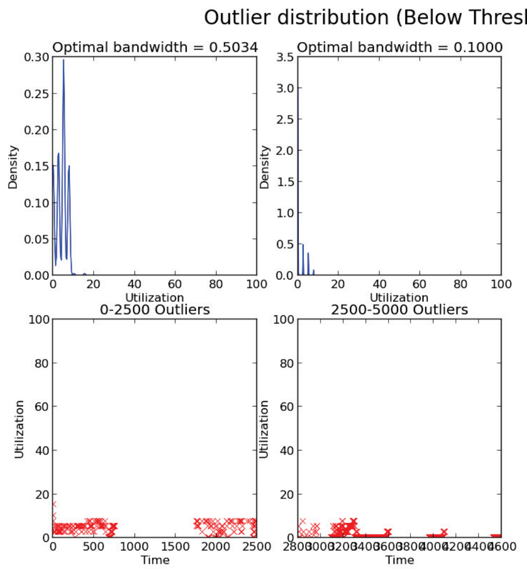


Figure 5.9: KDE for outliers below threshold

linear relationship if +1; and a high correlation if 0.5 and 1. Figure 5.8 depicts the KDE analysis for outliers above the threshold for each season in Figure 5.7. The *r* value is 0.53, 0.74, and 0.76, for the correlation values of two consecutive seasons in Figure 5.8. The correlation value between the

2nd and 3rd season, and the 3rd and 4th season are high due to the low number of outliers. In both cases, zeros are default values for those spots where there are no outliers. Therefore, statistically, the correlation values of the seasons are high. This is intuitively meaningful as the low number of outliers also indicates the likelihood of the upper or lower threshold being crossed is low. Hence, the changes of workload over these last three partitions are negligible.

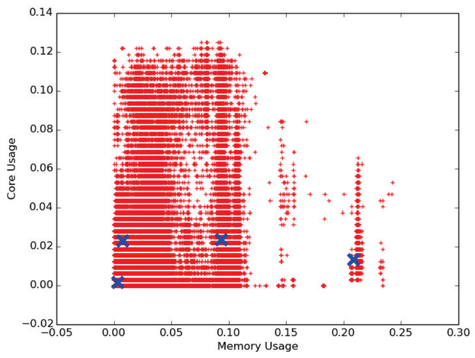
5.6.5 Workload Clustering

We used a basic KMeans clustering algorithm for classification of the time series workload data. For this purpose, we utilized spark's inbuilt machine learning library to run KMeans, which is the most commonly used workload classification method. The algorithm divides the dataset into a specified number of clusters, k, and assigns each datapoint to its closest cluster centroid by identifying the shortest distance between the data point and the centroids. We have used Euclidean metric to estimate the distance between data points.

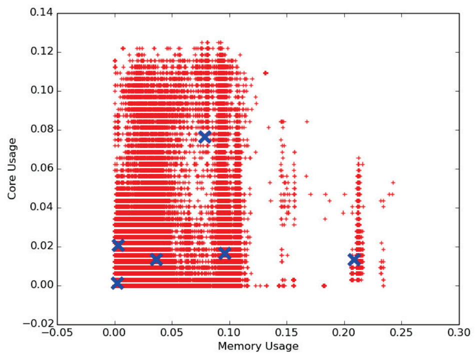
Spark's KMeans clustering algorithm requires multiple input parameters to classify the dataset, as described below:

- *k* decides the number of clusters required from the dataset.
- *Runs* is the number of times KMeans clustering algorithm should iterate to obtain convergence.
- *Max iterations* restricts the maximum number of iterations allowed for clustering.
- *Initialization mode* provides two initialization modes, either random initialization or initialization via k-means ||.
- *Initialization steps* provides the estimation of number of steps in k-means ||.
- *Epsilon* is the distance threshold to decide convergence.

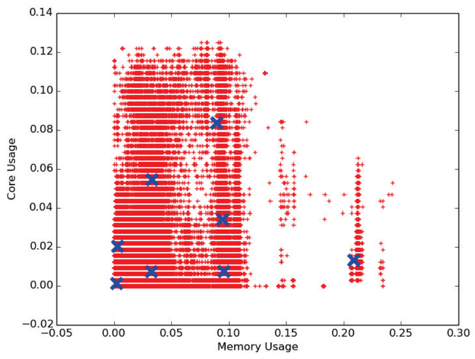
To demonstrate how the framework works, we classified the trace data for different values of k. The workload classification intends to find insight of the task event in the trace data. We ran the clustering algorithm on 1000k data points for different values of k, i.e. 2, 4, 6, 8, 10 and 12. We used k-means || as the initialization mode and kept the convergence threshold / epsilon constant as 0.6. The example results are produced using the framework are illustrated in Figure 5.10.



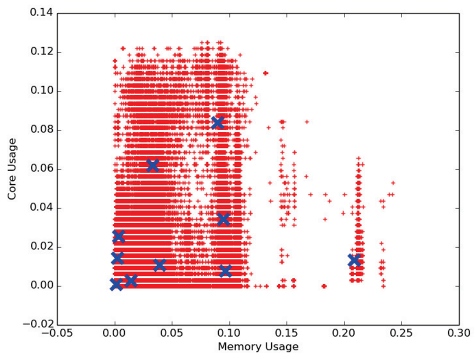
(a) $k = 4$



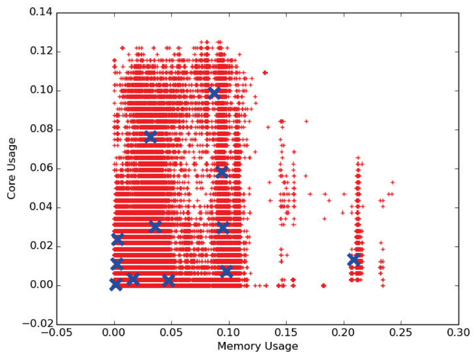
(b) $k = 6$



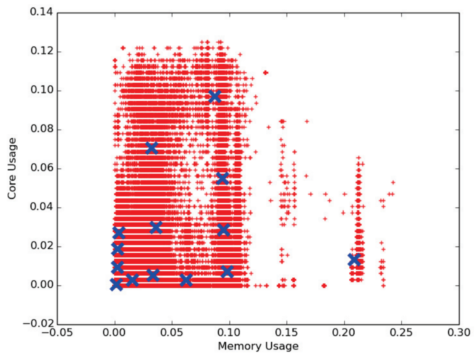
(c) $k = 8$



(d) $k = 10$



(e) $k = 12$



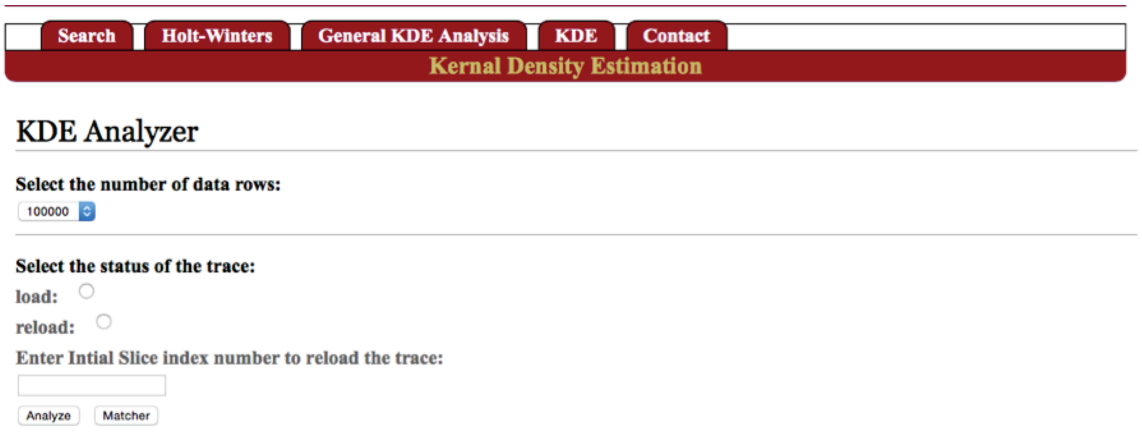
(f) $k = 14$

Figure 5.10: KMeans clustering for different values of k

We scrutinized classification results for different values of k. We observed that most of the data points representing the task's CPU and memory usage in the monitoring dataset have a considerably low value, utilizing approximately 0.0016 and 0.0031 CPU and memory on average, respectively. Only some tasks have high CPU and memory usage, i.e 0.287 and 0.018, respectively. This shows that the cluster may contain similar task events. Due to anonymity of trace data, the observations are not very conclusive. However, the main focus of the paper is to implement an architecture that provides such services.

5.6.6 Result Display in SMW

As previously described, the page generator microservice in *CloudAnalyzer Server* uploads the analysis results to SMW. The uploading process takes place in three steps. First, the results are uploaded to SMW as image files. SMW authenticates the page generator microservice as a valid user. After authentication, SMW grants the page generator microservice access to the upload feature. In the second step, a new page is created on SMW. Finally, the content of the page is structured using wiki markup language to represent the results in a comprehensive form. The markup content in Figure 5.11b shows how results are organized as wiki pages. These wiki pages can be used in various ways to structure the results so that it helps documentation and sharing of the workload information of the system.

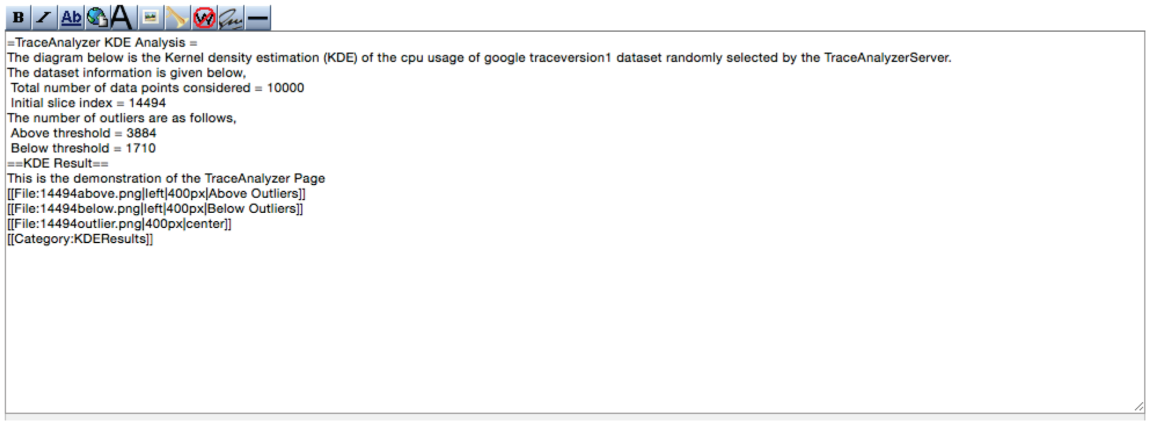


KDE Analyzer

Select the number of data rows:
100000

Select the status of the trace:
load: ○
reload: ○
Enter Intial Slice index number to reload the trace:

(a) Data query wiki page



```
=TraceAnalyzer KDE Analysis =
The diagram below is the Kernel density estimation (KDE) of the cpu usage of google traceversion1 dataset randomly selected by the TraceAnalyzerServer.
The dataset information is given below,
 Total number of data points considered = 10000
 Initial slice index = 14494
The number of outliers are as follows,
 Above threshold = 3884
 Below threshold = 1710
==KDE Result==
This is the demonstration of the TraceAnalyzer Page
[[File:14494above.png|left|400px|Above Outliers]]
[[File:14494below.png|left|400px|Below Outliers]]
[[File:14494outlier.png|400px|center]]
[[Category:KDEResults]]
```

(b) Wiki-markup for example result page for KDE analysis

Figure 5.11: Semantic-Mediawiki (SMW) Interface

5.7 Conclusion

In this chapter, we propose an architecture that integrates distributed systems and semantic MediaWiki to support the exploration of monitoring data. This architecture benefits from a web-based MediaWiki interface, and allows a user to define the access to monitoring data and organize the processing results. The search-based cluster built on SolrCloud enables indexing of large size of data, and thus makes the whole architecture suitable to explore and monitor the ever-accumulating data such as the traces produced from data centres. The architecture also includes an extension, which runs Spark on Yarn cluster to deploy efficient analysis methods for large data sets. It utilizes the spark's MapReduce paradigm to identify the cluster in the dataset using KMeans clustering method.

The architecture provides a rationale to develop cloud monitoring applications with advance algorithms for forecasting data and identifying workload patterns. Our future work includes developing more analysis methods for the architecture extension so that this architecture can be generalized for various applications.

Chapter 6

Empirical Evaluation of *CloudAnalyzer* using DoE

Performance evaluation is an important step in software design quality, since it ensures that the designed architecture meets all the requirements.We use the edesign of experiment (DoE) evaluation methodology to assess the performance attributes of the proposed *CloudAnalyzer* architecture. We additionally assess the fault-tolerance of the system.

6.1 Empirical Evaluation Using Design of Experiments

We use the previously described DoKnowMe evaluation methodology to evaluate *CloudAnalyzer* architecture.

6.1.1 Requirement Recognition

We begin by brainstorming the requirements for evaluation. The proposed architecture should meet the following criteria:

(1) It should be able to automate report generation and information distribution.

(2) It should provide advanced analytics to process monitoring data.

(3) It should be highly available.

Table 6.1: Identified Features for Architecture Evaluation

Category	**Feature**
Performance Features	Latency
	Availability

(4) It should be capable of processing a large volume of data.

Based on the above requirements, the following are the requirement questions:

RQ1: How fast does the architecture process monitoring data?

RQ2: How fault-tolerant is the architecture in case of node failure?

6.1.2 Feature Identification

We examine the requirement questions to determine the relevant performance features. According to the requirements, we focused on three performance features, as in Table 6.1. We mainly focus on the experimental evaluation of the performance features in this study.

6.1.3 Metric & Benchmark Listing and Selection

We consider three jobs to be performed by the *CloudAnalyzer* architecture: 1) workload prediction, 2) workload pattern matching, and 3) workload clustering. We have used the Google trace data as a monitoring input to benchmark. For instance, we run the workload prediction task to observe future events using HWTES. The purpose of prediction is to forecast critical events and prevent failure before it takes place.

Based on these jobs, we decide to observe execution time/latency, fault-tolerance/availability and scalability of the monitoring architecture. Therefore, execution time, fault-tolerance and scalability are the three metrics selected for performance evaluation. Execution time of each job is evaluated based on the following formula given in equation 12.

$$ExecutionTime(\Delta t) = ResponseTime(t') - FetchTime(t) \tag{12}$$

The *Response Time(t)́* is the total time taken by the *CloudAnalyzer* server to send a response back to the user, whereas the *Fetch Time (t)* is the time taken by the *CloudAnalyzer* server to retrieve data from the bottom storage layer. Therefore, the *Execution Time(Δt)* is the total time taken to execute processing, *Response Time($\acute{t}$)*, subtracted from *Fetch Time (t)*. The availability and scalability of the system are also important considerations for evaluation. Availability is evaluated by observing the cluster while deliberately failing cluster nodes. [TODO: discuss scalability]

6.1.4 Experimental Factors Listing and Selection

Our evaluation method distinguishes among three-factor types, including workload-related, resource-related and capacity-related factors. Accordingly, we have identified experimental factors and their levels for evaluating data mining tools, as listed below.

(1) Workload-related factors:

- Jobs: Workload Prediction, Workload Pattern and Workload Clustering
- Workload Size: 100k, 200k, 300k, 400k, 500k, 600k, 700k, 800k

(2) Resource-related factors:

- Monitoring Tool: *CloudAnalyzer Architecture*

(3) Capacity-related factors:

- Latency: Execution Time
- Availability
- Scalability

Performance is evaluated by employing the previously mentioned benchmark jobs to vary increasing workload size (i.e. 100k, 200k,..., 800k).

6.1.5 Experimental Design and Implementation

Performance Experimentation

We organized our requirements as factors, designing the experiment with Full-Factorial Design, as shown in Table 6.2. We design experiments and measure the end-to-end delay under various workloads. A sequence diagram in Figure 6.1 shows the time broken down for operations in order, from a request being initialized in a SMW page to the result displayed on the SMW page. The response time provides performance indication of the system in response to different situations like system failure and scaling during workload changes. Therefore, we conduct experiments to determine the response time to test performance, scalability and availability of the system architecture. Here, we explain the experiments in detail.

The aforementioned time metrics of the architecture iare measured by changing the amount of data being retrieved and processed. We increase this workload incrementally from 10,000 data

Table 6.2: A General Full-Factorial Design*

Trial	Job	Workload Size	Monitoring Tool	Execution Time	Availability	Scailaibility
1	Workload Prediction	100k	*CloudAnalyzeer*	?	?	? **
2	Workload Pattern	100k	*CloudAnalyzeer*	?	?	?
3	Workload Clustering	100k	*CloudAnalyzeer*	?	?	?
4	Workload Prediction	200k	*CloudAnalyzeer*	?	?	?
5	Workload Pattern	200k	*CloudAnalyzeer*	?	?	?
6	Workload Clustering	200k	*CloudAnalyzeer*	?	?	?
7	Workload Prediction	300k	*CloudAnalyzeer*	?	?	?
8	Workload Pattern	300k	*CloudAnalyzeer*	?	?	?
9	Workload Clustering	300k	*CloudAnalyzeer*	?	?	?
10	Workload Prediction	400k	*CloudAnalyzeer*	?	?	?
11	Workload Pattern	400k	*CloudAnalyzeer*	?	?	?
12	Workload Clustering	400k	*CloudAnalyzeer*	?	?	?
13	Workload Prediction	500k	*CloudAnalyzeer*	?	?	?
14	Workload Pattern	500k	*CloudAnalyzeer*	?	?	?
15	Workload Clustering	500k	*CloudAnalyzeer*	?	?	?

*The design matrix was generated by using Minitab.

**The "?" denotes a placeholder for *Response* value.

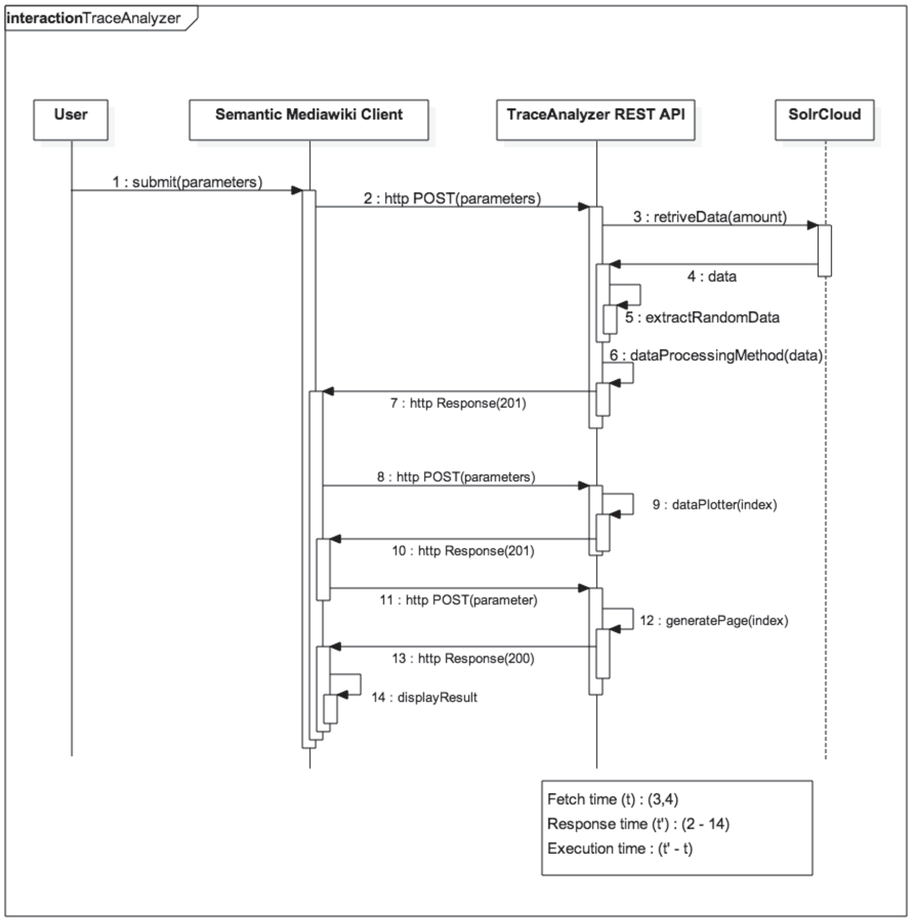


Figure 6.1: *CloudAnalyzer*: Core System Sequence Diagram

records to 50,000 data records, and perform experiments for both analysis methods. The broken-down times are plotted for both HWTES and KDE, respectively.

We have also evaluated the performance aspect of the clustering functionality provided by *Core architecture* extension using a similar approach. In case of clustering, the trace data is stored in HDFS instead of SolrCloud, and does not explicitly require time for extraction. Hence, the *Fetch Time* is replaced by *Job Execution time* and is shown in Figure 6.2. The *Job Execution time* is the total time taken by Spark cluster to process the data. The performance of KMeans clustering is measured on the basis of the following Eq.13

$$ExecutionTime(\Delta t) = ResponseTime(t') - JobExcutionTime(t) \tag{13}$$

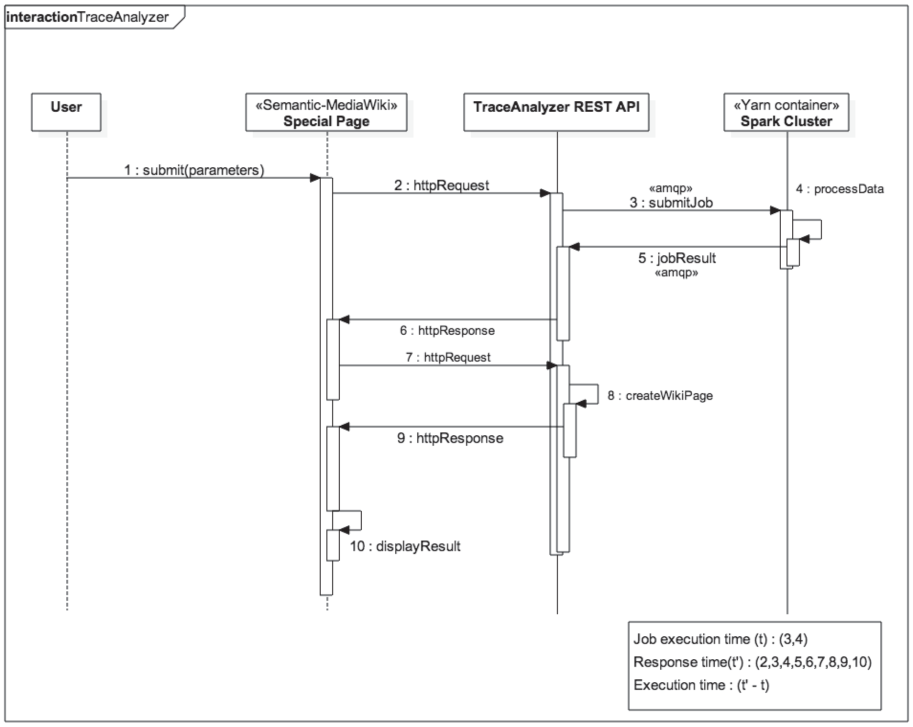
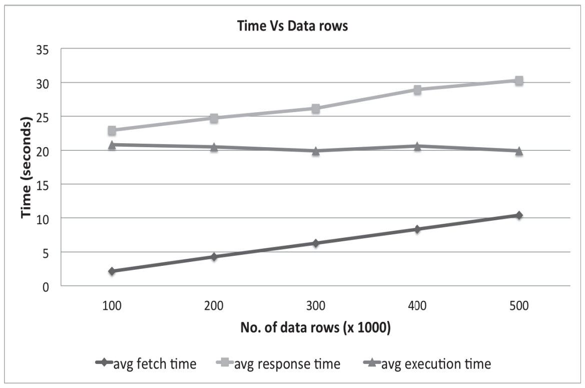


Figure 6.2: *CloudAnalyzer*: Extension System Sequence Diagram)

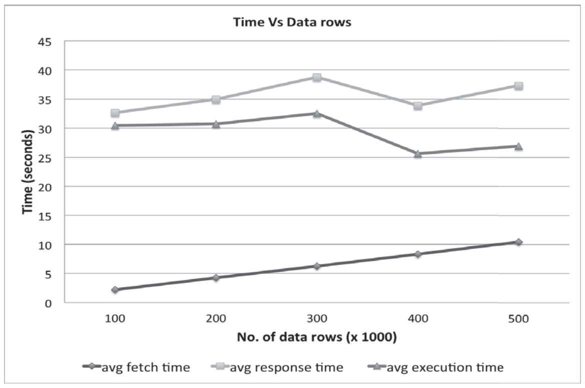
The response time of the KMeans clustering methods is measured by examining processing time on varying amounts of data. Initially, we gradually increased the size of data sets form 100,000 data points to 500,000 data points. Later, we jumped from 500,000 to 800,000 data points. We obtained multiple readings and estimated the average time taken by the KMeans clustering methods on different datasets.

Fault-Tolerance Experimentation

We design experiments to evaluate the fault tolerance of this architecture by leveraging the fault tolerant mechanisms supported by SolrCloud. We deliberately put down the nodes of SolrCloud and the nodes of ZooKeeper. Meanwhile, we measure the *Fetch Time* and observe any delays in data accessing. The initial reading for the *Fetch Time* is observed when all the nodes are alive. Then, deliberately a node is killed, and its effect on the system's behaviour is observed by measuring the *Average Fetch Time*. These observations are plotted for both HWTES and KDE, as in Figure 6.3a and Figure 6.3b, respectively.



(a) Holt-Winters Triple Exponential Smoothing Response Time



(b) Kernel Density Estimation Response Time

Figure 6.3: *CloudAnalyzer*: Latency Analysis Result

6.1.6 Experimental Analysis

Latency

The *Average Execution Time* is steady regardless of the change in workload size, as seen in figure 6.3a. It is independent of any factors, and solely dependent upon the dataset in consideration. Also, we observed that the *Average Response Time* is closely related to *Average Fetch Time*. Its linear growth is determined by a gradual increase in the *Average Fetch Time*, whose growth has directly proportional to workload size. In contrast, KDE analysis does not show any relationship between workload size and its *Average Response Time* (*see figure 6.3b*). It is dependent on the number of outliers detected, rather than the *Average Fetch Time*.

We analyzed the performance of the *Extension* architecture by observing its execution time while running a KMeans clustering job. The KMeans clustering results show a behaviour similar to that of HWTES, with a steady response time that is unaffected by the amount of data. However, the effect of increase in data size is quite prominent in *Average Job Execution time*.

Fault-Tolerance

Figure 6.5b, shows the system is continuously responding when nodes of SolrCloud and Zookeeper are down. The node failure of SolrCloud, however, has a larger effect on the responsiveness of the architecture. It is observed that the *Fetch Time* slightly increases with spikes in both HTWES and

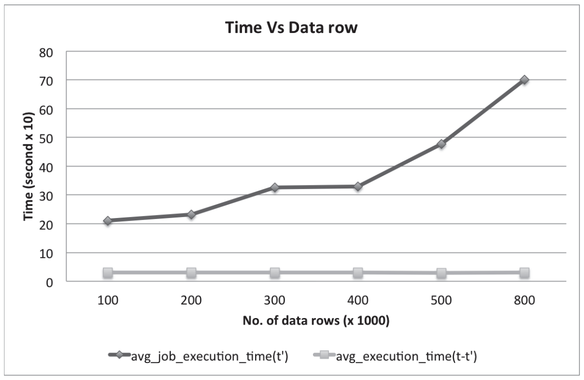


Figure 6.4: Response time and Execution time for KMeans clustering

KDE. The observed spikes make approximately 5% to 15% more of the *Fetch Time*. This is mainly due to the election of a new leader node from the remaining nodes when a failure occurs in either the ZooKeeper ensemble or Solr nodes, and there is an overhead associated to stabilize the cluster.

6.1.7 Conclusion and Documentation

The evaluation of the cloud architecture provides insights for the following performance aspects:

- How fast does the architecture process?

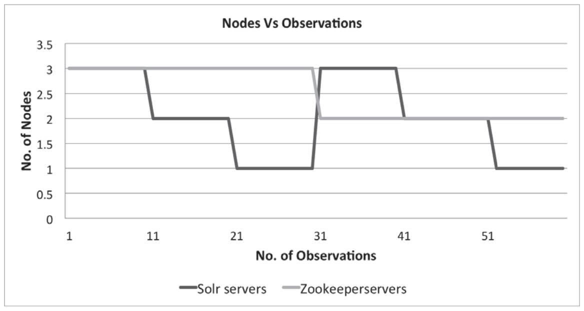
  The processing time is directly proportional to data size, taking approximately 22.5 seconds and 21 seconds to process 100,000 data points for HWTES and KMeans, respectively. However, the KDE analysis is workload size independent and depends upon the number of outliers. It takes approximately 33 seconds to process 100,000 data points.

- How fault-tolerant is the architecture is case of node failure?

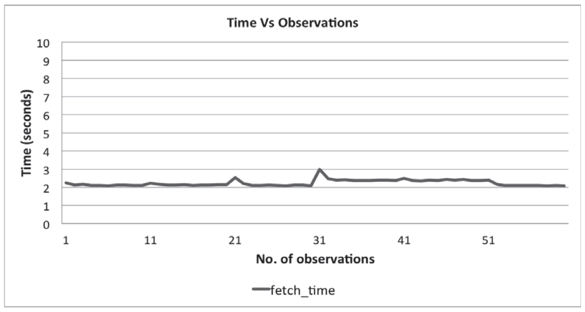
  SolrCloud and Hadoop Clustering provides Scalability and availability in the monitoring system. The analysis results shows seamless response of the SolrCloud in case of failure. The storage system remains functional unless all the Solr or ZooKeeper nodes are down.

6.2 Conclusion

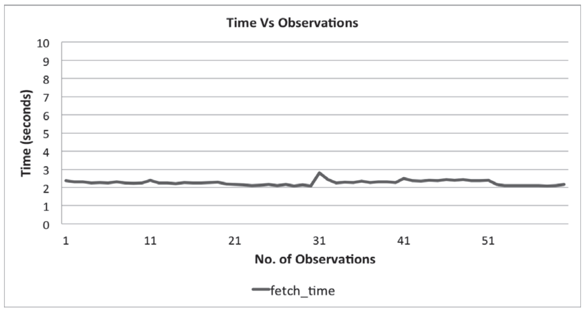
In this paper, we evaluate the performance and fault-tolerance of the architecture by using the design of experiment (DoE) evaluation method. Our observations demonstrate that our architecture is



(a) Nodes Vs Observation



(b) HWTES: Fetch Time Vs Observations



(c) KDE: Fetch time vs total number of observation

Figure 6.5: Responsiveness under node failure

responsive under node failure. The overhead incurred to handle the node failure in the storage cluster can result in extra delays of retrieving data. Since SolrCloud is optimized for indexing and data retrieval, the overall response time of this architecture is mainly determined by running analysis methods. We also evaluated the architecture extension for its performance and observed an increase in responsiveness with an increasing amount of data.

Chapter 7

Discussion and Conclusion

In this thesis, we present our empirical evaluation method based on design of experiment (DoE). It is generic assessment approach to evaluate software applications, such as analysis tools, workflow designs, analysis algorithms, etc. We validated the generic behaviour of the evaluation approach through two case studies. First case study was conducted to assess both functional and non-functional features of several data mining tools based on a specific set of requirements. The analysis process reveals a number of connected users per cell on the base stations in a trial mobile network. Since the data is continuously generating and accumulating the workload size for a given period of time, the collected datasets become one experimental factor. However, because several factors are leading to that data, there may be confounds, for example, data mining jobs and tool brands in case study I. Similar, the second case study evaluates the analysis algorithms (i.e. outlier detection techniques) to get hidden insight. Our evaluation study aims to minimize such confounds by applying DoE principles to make this practice into a structured eight-step procedure that addresses specific requirements. The threats of validation are mainly in the evaluation environment. We used default settings in each study. Each case can be further optimized to achieve a better response time and memory utilization. Our purpose is not to compare any data mining tools or outlier detection approach in an absolute measurement; instead, we focus on the evaluation method that allows observing the features and metrics of these tools or approaches, respectively. These insights will aid decision-making regarding data mining tools or outlier detection approaches as part of the big data analysis architecture. We believe this evaluation study, based on experimental design, can be extended to other data mining tools or outlier detection approaches of interests.

The DoE inspired us to design an architecture to tackle the issues concerning cloud monitoring. For this purpose, we propose an architecture that integrates search-based clusters and semantic media wiki to support the exploration of cloud monitoring data. This architecture benefits from a web-based MediaWiki interface and allows a user to define the access to monitoring data and to organize the processing of results. It also provides advanced monitoring solutions, such as workload prediction and workload matching. The search-based cluster built on SolrCloud enables indexing of large size of data, which makes the entire architecture suitable to explore and monitor ever-accumulating data, such as traces produced from data centers. We evaluate the performance and fault-tolerance of the architecture by experiments. Our observations demonstrate that the architecture is responsive to node failure. The overhead incurred to handle the node failure in a SolrCloud cluster can result in extra delays of retrieving data. Since SolrCloud is optimized for indexing and data retrieval, the overall response time of this architecture is mainly determined by running analysis methods. The architecture provides a rationale to develop cloud monitoring applications with advanced analysis algorithms for forecasting and identifying workload patterns. Furthermore, the architecture is enhanced by realizing Restful services instead of mere data transfer over HTTP protocol. It is extended to include distributed analysis for processing a large volume of data. The extension runs Spark on Yarn cluster for deploying efficient analysis methods for large data set. It utilizes the spark's MapReduce paradigm to identify the cluster in the dataset using KMeans clustering method. We also evaluated the architecture extension for its performance and the observed increase in responsiveness with the increasing amount of data.

In future, we will conduct more case studies to assess and validate the generic nature of the evaluation approach. we will explore more distributed analysis algorithms used to process data in a distributed environment. Also, we will inculcate more machine learning algorithms for workload clustering. We will also modify the architecture design to non-data centric architecture design and induce automatic load balancing.

Appendix A

Appendix

A.1 REST API Resources

A.1.1 HWTES Processor

- **Resource URI :** `http://host:port/api/process/hwtes/`

  POST

  evaluates forecast data for arbitrary dataset of 10,000 data points using Holt-Winter's triple exponential smoothing (HWTES). The dataset is extracted either from a trace-data file or SolrCloud cluster.

Parameters	Description
alpha	constant estimated in such a way that the mean square error (MSE) is minimum
beta	same as above
gamma	same as above
state	’–l’ or ’–r’ are arguments to decides to load a new dataset or reload the existing one in the system, respectively.
path/index	path variable is specifies alongside load state to locate dataset. For example, `/solr` would request to retrieve data from SolrCloud, whereas, path `some-folder/some-file.csv` would extract data from specified file location. However, index variable is specified alongside reload state, to retrieve data from existing dataset

- **Resource URI : `http://host:port/api/process/hwtes/{index}/`**

GET

returns the list of forecast data for a particular dataset in JSON format.

Parameters	Description
index	it is the initial dataset index used as an id to find the location of forecast data list.

DELETE

removes the forecast dataset package for the a particular data.

Parameters	Description
index	it is the initial dataset index used as an id to find the forecast list request.

A.1.2 KDE Processor

- **Resource URI :** `http://host:port/api/process/kde/`

  POST

  determines the outliers i.e. data points lying above and below the estimated threshold value and plot the results.

Parameters	**Description**
state	'–l' or '–r' arguments decides to load a new dataset or reload the existing one.
path/index	path variable is specified alongside load state to locate dataset. For example, `/solr` would request to retrieve data from SolrCloud, whereas, path `some-folder/some-file.csv` would extract data from specified file location. However, index variable is specified alongside reload state, to retrieve data from existing dataset

- **Resource URI :** `http://host:port/api/process/kde/{index}/`

  GET

  returns the number of outliers i.e. data points lying above/below a specific threshold.

Parameters	**Description**
index	it is the initial dataset index used as an id to find the forecast list request.

  DELETE

  purge the KDE results for a particular dataset.

Parameters	**Description**
index	it is the initial dataset index used as an id to find the kde data lists location.

A.1.3 HWTES Plotter

- **Resource URI :** `http://host:port/api/plot/hwtes/`

  POST

  plots the forecast data for a particular dataset.

Parameters	**Description**
index	it is the initial dataset index used as an id to find the forecast list location.

- **Resource URI :** `http://host:port/api/plot/hwtes/{index}`

  GET

  returns the forecast plot for particular dataset.

Parameters	**Description**
index	it is the initial dataset index used as an id to find the forecast list location.

  DELETE

  deletes the forecast plot for the specified index.

Parameters	**Description**
index	it is the initial dataset index used as an id to find the forecast list location.

A.1.4 Spark Clustering

- **Resource URI :** `http://host:port/api/sparkclustering/kmeans/`

  POST

  provides the information regarding the clusters present in the given dataset.

Parameters	Description
input	input path for the file location. The data can be retrieved from HDFS or local file system by specifying in the URL as hdfs:/// or file:///, respectively
output	to save the result obtained after processing.
k	it defines the number of clusters to be obtained in the dataset.
iteration	number of times data needs to be iterated to get convergence.
size	the size of data to utilized for processing.

A.1.5 SMW Page Generator

- **Resource URI :** `http://host:port/api/generate/`

  POST

  structures the analysis results and creates a SMW page using MediaWiki markup language

Parameters	Description
index/path	the initial dataset index used as an id to find the forecast list location required for KDE and HWTES only. However, output path of the result location is required in case of creating KMeans clustering result page.
method	'hwtes' , 'kde ' and 'kmeans' string arguments to create holt winter forecast, to generate kde outliers estimation page and 'kmeans' to generate KMeans clustering page, respectively.
username	Semantic-Mediawiki username for access.
password	same as above.

References

Aceto, G., Botta, A., De Donato, W., and Pescapè, A. (2013a). Cloud monitoring: A survey. *Computer Networks*, 57(9):2093–2115.

Aceto, G., Botta, A., De Donato, W., and Pescapè, A. (2013b). Cloud monitoring: A survey. *Computer Networks*, 57(9):2093–2115.

Alam, M., Shakil, K. A., and Sethi, S. (2015). Analysis and clustering of workload in google cluster trace based on resource usage. *arXiv preprint arXiv:1501.01426*.

Amazon, E. (2014). Cloudwatch.

Antony, J. (2003). *Design of Experiments for Engineers and Scientists*. Butterworth-Heinemann, Burlington, MA.

Babar, M. A. and Gorton, I. (2004). Comparison of scenario-based software architecture evaluation methods. In *Software Engineering Conference, 2004. 11th Asia-Pacific*, pages 600–607. IEEE.

Barbosa de Carvalho, M., Pereira Esteves, R., da Cunha Rodrigues, G., Zambenedetti Granville, L., and Rockenbach Tarouco, L. M. (2013). A cloud monitoring framework for self-configured monitoring slices based on multiple tools. In *Network and Service Management (CNSM), 2013 9th International Conference on*, pages 180–184. IEEE.

Barrett, D. J. (2008). *MediaWiki*. " O'Reilly Media, Inc.".

Bergner, K., Rausch, A., Sihling, M., and Ternité, T. (2005). Dosam–domain-specific software architecture comparison model. In *Quality of Software Architectures and Software Quality*, pages 4–20. Springer.

Bhat, A. Z. and Ahmed, I. (2016). Big data for institutional planning, decision support and academic excellence. In *2016 3rd MEC International Conference on Big Data and Smart City*

(ICBDSC), pages 1–5. IEEE.

Blackburn, S. M., McKinley, K. S., Garner, R., Hoffmann, C., Khan, A. M., Bentzur, R., Diwan, A., Feinberg, D., Frampton, D., Guyer, S. Z., et al. (2008). Wake up and smell the coffee: evaluation methodology for the 21st century. *Communications of the ACM*, 51(8):83–89.

Borges, L. C., Marques, V. M., and Bernardino, J. (2013). Comparison of data mining techniques and tools for data classification. In *Proc. 6th Int. C* Conf. Comput. Sci. & Softw. Eng. (C3S2E 2013)*, pages 113–116, Porto, Portugal. ACM Press.

Celebi, M. E., Kingravi, H. A., and Vela, P. A. (2013). A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Systems with Applications*, 40(1):200–210.

Chai, T. and Draxler, R. R. (2014). Root mean square error (RMSE) or mean absolute error (MAE)? *Geosci. Model Dev. Discuss.*, 7(1):1525–1534.

Chaves, D., Aparecida, S., Uriarte, R. B., and Westphall, C. B. (2011). Toward an architecture for monitoring private clouds. *Communications Magazine, IEEE*, 49(12):130–137.

CloudWatch, A. (2013). monitoring for aws cloud resources.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

Denning, P. J. (1981). Performance evaluation: Experimental computer science at its best. *ACM SIGMETRICS Perform. Eval. Rev.*, 10(3):106–109.

Di, S., Kondo, D., and Franck, C. (2013). Characterizing cloud applications on a Google data center. In *42nd International Conference on Parallel Processing (ICPP)*, Lyon, France.

Dobrica, L. and Niemelä, E. (2002). A survey on software architecture analysis methods. *Software Engineering, IEEE Transactions on*, 28(7):638–653.

Dunietz, I. S., Ehrlich, W. K., Szablak, B. D., Mallows, C. L., and Iannino, A. (1997). Applying design of experiments to software testing: experience report. In *Proc. 19th Int. Conf. Softw. Eng. (ICSE 1997)*, pages 205–215, Boston, Massachusetts, USA. ACM Press.

Elagib, S. B., Najeeb, A. R., Hashim, A. H., and Olanrewaju, R. F. (2014). Big data analysis solutions using mapreduce framework. In *Computer and Communication Engineering (ICCCE), 2014 International Conference on*, pages 127–130. IEEE.

Enterprises, N. (2012). Nagios-the industry standard in it infrastructure monitoring. *Online http://www. nagios. org/last accessed*, 3:29.

Feitelson, D. G. (2007). Experimental computer science. *Commun. ACM*, 50(11):24–26.

Fielding, R. (2000). Fielding dissertation: Chapter 5: Representational state transfer (rest). *Recuperado el*, 8.

Fiutem, R. and Antoniol, G. (1998). Identifying design-code inconsistencies in object-oriented software: a case study. In *Software Maintenance, 1998. Proceedings., International Conference on*, pages 94–102. IEEE.

Fuad, A., Erwin, A., and Ipung, H. P. (2014). Processing performance on apache pig, apache hive and mysql cluster. In *Information, Communication Technology and System (ICTS), 2014 International Conference on*, pages 297–302. IEEE.

Ghosh, S. and Dubey, S. K. (2013). Comparative analysis of k-means and fuzzy c-means algorithms. *International Journal of Advanced Computer Science and Applications*, 4(4):34–39.

Hadoop, A. (2009). Hadoop.

Hansen, B. E. (2009). Lecture notes on nonparametrics. *Lecture notes*.

Hibler, M., Ricci, R., Stoller, L., Duerig, J., Guruprasad, S., Stack, T., Webb, K., and Lepreau, J. (2008). Large-scale virtualization in the emulab network testbed. In *USENIX Annual Technical Conference*, pages 113–128.

Honnutagi, P. S. (2014). The hadoop distributed file system. *International Journal of Computer Science & Information Technologies*, 5(5):6238–6243.

Hu, H., Wen, Y., Chua, T.-S., and Li, X. (2014). Toward scalable systems for big data analytics: A technology tutorial. *IEEE Access*, 2:652–687.

Hyndman, R. J. and Athanasopoulos, G. (2014). *Forecasting: principles and practice*. OTexts.

IBM. Big data at the speed of business.

Ionita, M. T., Hammer, D. K., and Obbink, H. (2002). Scenario-based software architecture evaluation methods: An overview. *ICSE/SARA*.

Jean-Yves Le Boudec (2011). *Performance Evaluation of Computer and Communication Systems*. EFPL Press, Lausanne, Switzerland.

Kalekar, P. S. (2008). Time series forecasting using holt-winters exponential smoothing. dec. 2004.

Kanwal Rekhi School of Information Technology, 3.

Kazman, R., Bass, L., Abowd, G., and Webb, M. (1993). Analyzing the properties of user interface software. Technical report, DTIC Document.

Kazman, R., Bass, L., Webb, M., and Abowd, G. (1994). Saam: A method for analyzing the properties of software architectures. In *Proceedings of the 16th international conference on Software engineering*, pages 81–90. IEEE Computer Society Press.

Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., and Carriere, J. (1998). The architecture tradeoff analysis method. In *Engineering of Complex Computer Systems, 1998. ICECCS'98. Proceedings. Fourth IEEE International Conference on*, pages 68–78. IEEE.

Khoudali, S., Benzidane, K., Sekkaki, A., and Bouchoum, M. (2014). Toward an elastic, scalable and distributed monitoring architecture for cloud infrastructures. In *Next Generation Networks and Services (NGNS), 2014 Fifth International Conference on*, pages 132–138. IEEE.

Krötzsch, M., Vrandečić, D., and Völkel, M. (2006). Semantic mediawiki. In *The Semantic Web-ISWC 2006*, pages 935–942. Springer.

Kuć, R. (2013). *Apache Solr 4 Cookbook*. Packt Publishing Ltd.

Kuhn, D. R. and Reilly, M. J. (2002). An investigation of the applicability of design of experiments to software testing. In *Proc. 27th Annu. NASA Goddard/IEEE Softw. Eng. Workshop (SEW-27 2002)*, page 91. IEEE Computer Society.

Li, L. and Chou, W. (2011). Design and describe rest api without violating rest: A petri net based approach. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 508–515. IEEE.

Li, Z., O'Brien, L., and Kihl, M. (2016). DoKnowMe: Towards a domain knowledge-driven methodology for performance evaluation. *ACM SIGMETRICS Perform. Eval. Rev.*, in press.

Lindvall, M., Tvedt, R. T., and Costa, P. (2003). An empirically-based process for software architecture evaluation. *Empirical Software Engineering*, 8(1):83–108.

Marzouk, Y. M. and Ghoniem, A. F. (2005). K-means clustering for optimal partitioning and dynamic load balancing of parallel hierarchical n-body simulations. *Journal of Computational Physics*, 207(2):493–528.

Massie, M. L., Chun, B. N., and Culler, D. E. (2004). The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840.

McAfee, A., Brynjolfsson, E., Davenport, T. H., Patil, D., and Barton, D. (2012). Big data. *The management revolution. Harvard Bus Rev*, 90(10):61–67.

Meng, X., Bradley, J., Yuvaz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al. (2016). Mllib: Machine learning in apache spark. *JMLR*, 17(34):1–7.

Mikut, R. and Reischl, M. (2011). Data mining tools. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(5):431–443.

Mishra, A. K., Hellerstein, J. L., Cirne, W., and Das, C. R. (2010). Towards characterizing cloud backend workloads: insights from Google compute clusters. *SIGMETRICS Perform. Eval. Rev.*, 37(4):34–41.

Montes, J., Sánchez, A., Memishi, B., Pérez, M. S., and Antoniu, G. (2013). Gmone: A complete approach to cloud monitoring. *Future Generation Computer Systems*, 29(8):2026–2040.

Montgomery, D. C. (2009). *Design and Analysis of Experiments*. John Wiley & Sons, Inc., Hoboken, NJ, 7th edition edition.

Murphy, G. C., Notkin, D., and Sullivan, K. (1995). Software reflexion models: Bridging the gap between source and high-level models. *ACM SIGSOFT Software Engineering Notes*, 20(4):18–28.

Nair, V. N., James, D. A., Ehrlich, W. K., and Zevallos, J. (1998). A statistical assessment of some software testing strategies and application of experimental design techniques. *Statistica Sinica*, 8(1):165–184.

Newman, H. B., Legrand, I. C., Galvez, P., Voicu, R., and Cirstoiu, C. (2003). Monalisa: A distributed monitoring service architecture. *arXiv preprint cs/0306096*.

Pardo-Castellote, G. (2003). Omg data-distribution service: Architectural overview. In *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, pages 200–206. IEEE.

Povedano-Molina, J., Lopez-Vega, J. M., Lopez-Soler, J. M., Corradi, A., and Foschini, L. (2013). Dargos: A highly adaptable and scalable monitoring architecture for multi-tenant clouds. *Future Generation Computer Systems*, 29(8):2041–2056.

Reiss, C., Tumanov, A., Ganger, G. R., Katz, R. H., and Kozuch, M. A. (2012). Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, page 7. ACM.

Reiss, C., Wilkes, J., and Hellerstein, J. L. (2011). Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA, USA.

Sefika, M., Sane, A., and Campbell, R. H. (1996). Monitoring compliance of a software system with its high-level design models. In *Proceedings of the 18th international conference on Software engineering*, pages 387–396. IEEE Computer Society.

Shanmugapriya, P. and Suresh, R. (2012). Software architecture evaluation methods-a survey. *International Journal of Computer Applications*, 49(16).

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010). The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE.

Tamhane, D. S. and Sayyad, S. N. (2015). Big data analysis using hace theorem. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume*, 4:2278–1323.

Trosset, M. W. (2009). *An introduction to statistical inference and its applications with R*. CRC Press.

Tvedt, R. T., Lindvall, M., and Costa, P. (2002). A process for software architecture evaluation using metrics. In *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE*, pages 191–196. IEEE.

Vilas, K. S. (2012). Big data mining. *Int J Comput Sci Manag Res*, 1(1):12–17.

Villars, R. L., Olofson, C. W., and Eastwood, M. (2011). Big data: What it is and why you should care. *White Paper, IDC*.

Wahbeh, A. H., Al-Radaideh, Q. A., Al-Kabi, M. N., and Al-Shawakfa, E. M. (2011). A comparison study between data mining tools over some classification methods. *Int. J. Adv. Comput. Sci. Appl.*, Special Issue(Artificial Intelligence):18–26.

Ward, J. S. and Barker, A. (2014). Observing the clouds: a survey and taxonomy of cloud monitoring. *Journal of Cloud Computing*, 3(1):1–30.

Wikipedia. Kernel density estimation — wikipedia, the free encyclopedia? [Online; accessed 12-May-2016].

Wilkes, J. (2011). More Google cluster data. Google research blog.

Wu, X., Zhu, X., Wu, G.-Q., and Ding, W. (2014). Data mining with big data. *IEEE transactions on knowledge and data engineering*, 26(1):97–107.

Xu, X. and Tang, M. (2015). A new approach to the cloud-based heterogeneous mapreduce placement problem.

Yongdnog, H., Jing, W., Zhuofeng, Z., and Yanbo, H. (2013). A scalable and integrated cloud monitoring framework based on distributed storage. In *Web Information System and Application Conference (WISA), 2013 10th*, pages 318–323. IEEE.

Zanikolas, S. and Sakellariou, R. (2005). A taxonomy of grid monitoring systems. *Future Generation Computer Systems*, 21(1):163–188.