# Scalable Data Processing and Analytical Approach for Big Data Cloud Platform

by

**Bikash Agrawal**

A dissertation submitted in partial satisfaction of
the requirements for the degree
PHILOSOPHIAE DOCTOR (PhD)

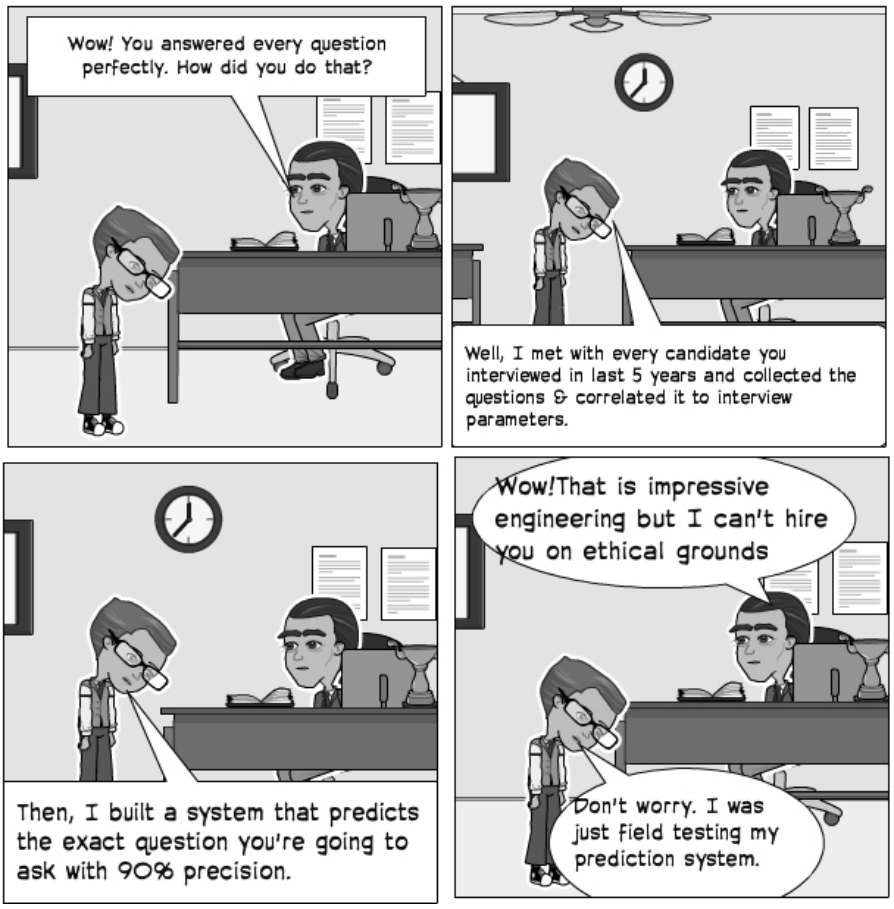

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science
March 2017

University of Stavanger
N-4036 Stavanger
NORWAY
www.uis.no



ISBN 978-82-7644-704-0
ISSN 1890-1387

PhD Thesis UiS no. 338




Big Data Humor: Hiring a Data Scientist. © Daniel Gutierrez.

## Summary

In the current "information explosion era", data is increasing dramatically every year. Tackling the challenges posed by collecting large-scale data is trending to a superabundance of data management systems characterized by horizontal scalability.

Data has become a critical commodity in organizations and data analysts are struggling to make optimal business decisions, due to the challenges in Big Data management: volume, velocity, variety, veracity, and value. Large-scale data analytics is turning into a computational resource paradigm due to already unprecedented yet fast growing requirements such as scalability, data intensiveness, high availability, fault tolerance, and the ability to handle diverse data structures. As these problems grow in scale, parallel computing resources are required to meet computational and memory requirements. The growing demand for large-scale data analysis and mining applications has resulted in both industry and academia designing highly scalable data-intensive computing platforms.

Large-scale analysis requires clusters of connected computers to allow high performance in environments such as cloud computing. As cloud computing clusters grow in size, the following key challenges have arisen: heterogeneity of the system, hidden complexities, time limitations, and scalability. Other challenges such as failure prediction and anomaly detection of components in the cluster components are also important factors that must be addressed while running Big Data applications. Although, some solutions to these challenges are already available for small-scale systems, a scalable approach for effective performance diagnosis and prediction is needed. This dissertation addresses these needs and proposes solutions (like a distributed scalable analytics framework) that enrich the cloud platform and enable high-performance processing on a large scale. The contribution of this dissertation is demonstrated by applying state-of-the-art processing framework and consequently improving data center overall performance by predicting failures and detecting anomalies. The framework enables Big Data applications to gain an advantage using cloud computing such as scalability and elasticity.

Additionally, it proposes a state-of-art solution for permanently deleting data stored by Big Data applications. Herein solutions for securely deleting cloud data are evaluated, and a novel secure deletion tool for Hadoop clusters is also proposed.

# Preface

This dissertation is submitted in partial fulfillment of the requirement for the degree of Philosophiae Doctor (PhD) at the University of Stavanger, Norway. The research presented has been and are carried out at the University of Stavanger during the period from September 2013 to August 2016, and at the Purdue University, West Lafayete, Indiana; USA during the period from October 2015 to March 2016.

The dissertation is based on the research papers listed below. The papers have been reformatted to correspond with the format of the dissertation, and in relation to this, all papers have been reviewed for any spelling, grammar and formatting issues. Please note, the content is that of the original publications is self-contained.

## Acknowledgements

I would like to express my profound gratitude to my supervisor Dr.Tomasz Wiktorski. He has always been available for dialogue, offered constructive feedback through the course of the work and provided a plenitude of development opportunities that had key impact on this and further work.

I would also like to thank my co-supervisor Prof. Chunming Rong for providing necessary feedback, inspiration, and guidance throughout my PhD work.

I would also like to thank all that had influence on the contents of this dissertation: Prof. Raymond Hansen, and Prof. Thomas Hacker of Purdue University. I would further like to thank Dr.Klaus Petritsch (for proof reading), and Mr.Michael J. Salerno for reviewing and providing necesssary feedback on this dissertation.

Last but not least, I would like to thank my friends and family, for their love and support during this work.

*Bikash Agrawal, March 2017*

# Contents

# List of Papers

The following papers are included in this thesis:

- **Paper I**

  **R2Time: a framework to analyse OpenTSDB timeseries data in HBase.**
  B. Agrawal, A. Chakravorty, C. Rong, T. Wiktorski
  Published in the proceedings of 2014 IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom).

- **Paper II**

  **Analyzing and Predicting Failure in Hadoop Clusters Using Distributed Hidden Markov Model**
  B. Agrawal, C. Rong, T. Wiktorski
  Published in the proceedings of 2015 6th International Conference on Cloud Computing and Big Data in Asia.

- **Paper III**

  **SD-HDFS: Secure Deletion in Hadoop Distributed File System**
  B. Agrawal, R. Hansen, C. Rong, T. Wiktorski
  Published in the proceedings of 2016 IEEE 5th International Congress on Big Data (BigData Congress).

- **Paper IV**

**Adaptive Anomaly Detection in Cloud using Robust and Scalable Principal Component Analysis**
B. Agrawal, T. Wiktorski, C. Rong
Submitted to Concurrency and Computation: Practice and Experience and is under review. It is an extended version of the paper published in the proceedings of 2016, 15th International Symposium of Parallel and Distributed Computing (ISPDC2016).

- **Paper V**

**AFFM: Auto Feature Engineering in Field-Aware Factorization Machines for Predictive Analytics**
L. Selsaas, B. Agrawal, C. Rong, T. Wiktorski
Published in the proceedings of 2015 IEEE International Conference on Data Mining Workshops (ICDM Workshops).

- **Paper VI**

**Enrichment of Machine Learning based Activity Classification in Smart Homes using Ensemble Learning**
B. Agrawal, A. Chakravorty, T. Wiktorski, C. Rong
Accepted to the proceedings of 2016 ACM 3rd International Workshop on Smart City Clouds: Technologies, Systems, and Applications (SCCTSA).

# Chapter 1

# Introduction

## 1.1 Motivation

Despite the fact that the term Big Data has become part of mainstream vocabulary, there is no comprehensive definition of what the phrase "Big Data" truly means. Data scientists posit the idea that the Extraction, Transformation and Load (ETL) for a massive data is the best definition of the of Big Data concept [12] [28] [13]. This depiction of Big Data depends on five information characteristics: volume, velocity, variety, variability and value (the 5Vs) [3] [9]. As the number of devices, sensors, and people connected to the global network increases, the amount of data and the need to communicate, share, and access the data increases. This increase in the data volume cannot be processed using traditional methods.

The Internet Data Center assessed estimated that the development amount of information would grow by a factor of 300 between 2005 and 2020, while expecting to increase data amount from 130 Exabytes to 40,000 Exabytes [15].

Futhermore, the discovery of data insights refocused information management on a new analytic paradigm- *Data Science* in which applications need to scale and ensure they do not overwhelm the 5Vs (volumes, velocities, varieties, variabilities, or values). This results in what is called "*the Big Data phenomenon*".

The existing Big Data tools (e.g., HDFS, MapReduce, Spark, Flink etc.) require partitioning and distribution of data for processing across multiple data centers. The need to process the data depends on data size and the location of data sources. The main challenge will be to enable faster execution times for Big Data processing [2]. In other words, the

main features of data-intensive paradigm are scalability and efficiency.

Cloud computing is another factor that accelerates the revolution of Big Data. Large, geo-located data centers of clouds enable the computational power of infrastructure, while on-demand scaling provides opportunities for Big Data scenarios. The cloud infrastructure allows users to avoid the burden of managing complex distributed hardware and also provides an infinitely scalable infrastructure. Therefore, users focus directly on renting and scaling their services for better resource utilization and can compile according to the application's processing needs.

However, problems with cloud infrastructure availability and performance can lead to extensive financial losses. Therefore, it is crucial to address performance as an explicit objective. There is also a need for automated failure diagnostics and predictive analytics, which enables the cloud providers to manage their data center proactively. This dissertation provides a framework for autonomic and scalable analytics, which makes it possible to explore a large amount of data for both automated failure diagnostics and anomaly detection that accelerates cloud performance and availability.

This dissertation proposes a diversified and efficient extensions to cloud-based Big Data framework as a key milestone. This dissertation provide a framework that can help to improve the accuracy and speed of data center by predicting the failure of computing nodes and jobs in Hadoop clusters as well as detecting anomalous behavior of Big Data applications running in cloud infrastructure. Finally, this dissertation provides a secure deletion technique which enhances transparency to the user when dealing with a distributed file system in a massive infrastructure.

### 1.1.1 Organization

This dissertation is organized as a collection of research papers and divided into two parts. In the first part, the motivation, relevant background, and a discussion is presented. The second part, consists of research articles that contribute to the dissertation.

Part 2 consists of a collection of six research papers: Paper I proposes a novel distributed data storage and processing framework. Papers II, and IV demonstrate large-scale analytics to predict failure and to detect anomalies in a data center. Paper III presents a state-of-art solution for deleting the data generated for papers II, and IV permanently. Finally, paper V and VI demonstrate the data analytics application.

# Chapter 2

# Background

This chapter provides an overview of different technologies and terminology used in the enclosed papers.

## 2.1 Big Data

In "information explosion era", a massive amount of data is continually generated at an unprecedented and ever increasing scales. This enormous volume of data is collected and studied in various domains. Data generated from a variety of connected devices are growing at an exponential rate. In 2011, digital information grew nine times in volume in just five years [25] and its amount is projected to reach 35 trillion gigabytes by 2020 [33]. That torrent of data is emerging from a wide and ever growing array of sources. Some examples of large datasets and their respective growth rates are listed below [21]:

- The New York Stock Exchange generates about one terabyte of data per day.
- Facebook hosts approximately 10 billion photos, taking up one PetaByte of storage.
- Ancestry.com, stores around 2.5 petabytes of data.
- Ebay generates more than 50 terabytes of data per day.
- Google generates 40000 search queries per second.
- The internet archive stores around 2 petabytes of data, and is increasing at a rate of 20 terabytes per month.

- The Large Hadron Collider produces about 15 petabytes of data per year
- Radiology data produces 69 petabytes per year.
- Internet of Things: 25-50 billion connected devices will be on the Internet by 2020
- Self-driving car will produce 100 million megapixel images which are almost 100 terabytes to train in deep learning.
- Exascale simulation will generate terabytes per second.
- The Square Kilometer Array Telescope will produce 100 terabits/second ( 400 exabytes per year)

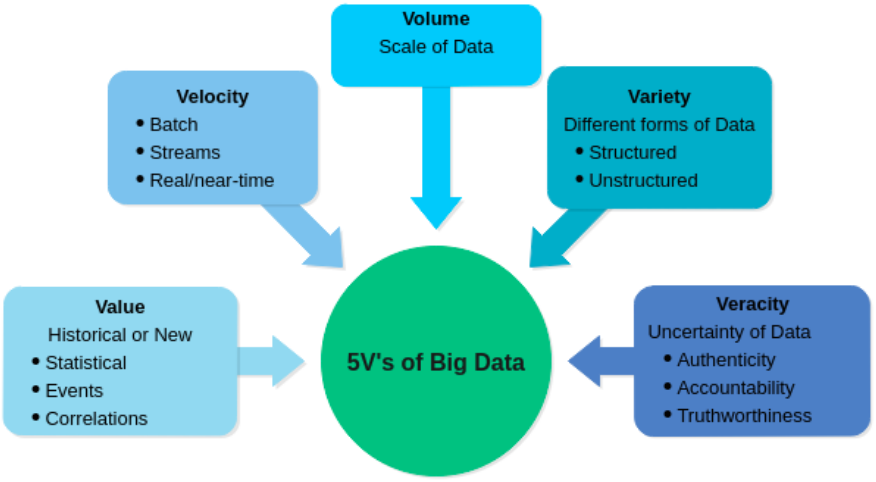



**Figure 2.1:** The 5 key challenges in Big Data

As more and more systems generate data, which is produced in ever-increasing quantities, the scalability of algorithms is essential for successful data management. This is the movitation for development and study of "Big Data".

The National Institute of Standards and Technology (NIST) describes Big Data as "a collection of extensive datasets primarily in the characteristics of volume, variety, velocity, and/or variability that require a scalable architecture for efficient storage, manipulation, and analysis" [8].

These are the key characteristics of Big Data shown in Figure 2.1, and are commonly referred to as the **V**s' of Big Data.

**Table 2.1:** 5**V**s of Big Data

| | |
|---|---|
| Volume | quantity, from terabytes to zettabytes |
| Velocity | batch processing to real-time processing |
| Variety | structured, semi-structured and unstructured |
| Veracity | quality, relevance, trustworthiness, accountability |
| Value | predictive value |

- **Volume**: The volume of Big Data is described as the amount of data coming in. The volume typically ranges from gigabytes to exabytes and beyond. Facebook alone generates 10 billion messages, 4.5 billion click events, and 350 million photograph uploads every day. This amount of volume is not only storage challenge, but also a massive computational goal for data analysis.

- **Variety**: Describes the organization of data. Although the development of "Big Data" technology produced structured, unstructured, and semi-structured data, today , 80% of the world's data is unstructured and therefore cannot be entered easily into relational databases.

- **Velocity**: Velocity means the flow rate at which data is produced, stored, and analyzed. In "data explosive era", data is analyzed and stored in real or near real time. The increasing number of connected devices known as the Internet of Things (IoT), creates additional challenges for real-time data analytics. Some industries (e.g. telecommunications) have processed a high volume and limited time interval data for years. However, with the horizontal scalability of Big Data it becomes possible to handle such data efficiently.

- **Variability**: Variability refers to any change in data over time, including the flow rate, the format, and the composition. The collected data with different fields has a high probability of containing incomplete information. These incomplete, uncertainty and diverse data sources significantly influence the quality of data. Therefore,

data validation and provenance become an important step during data processing to solve this problem [11].

- **Value**: The ability to understand and manage data sources and integrate them into a larger data network can provide previously unknown insights from data. The rise of Big Data is driven by the rapid development of machine learning algorithms, data mining techniques, and artificial intelligence. Moreover, it is motivated by a process of analyzing the data, extracting information into knowledge and action for desired values based on historical knowledge [10]. It involves a process to use the machine learning algorithms to achieve the value of data to make a business decision.

## 2.2 The Large-Scale Distributed Processing

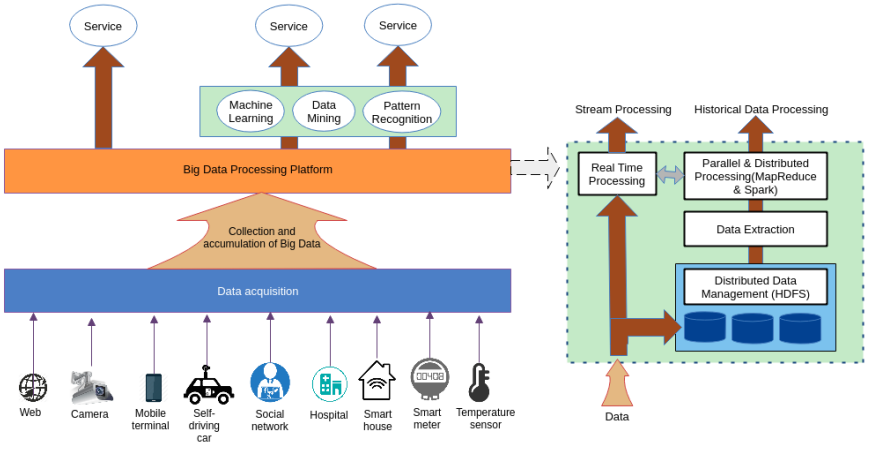



**Figure 2.2:** Large-scale distributed data processing framework

The analysis of large volume of data using conventional methods is exponentially expensive. The advent of horizontal scale [32] (adding more machines) has reduced the cost for constructing systems to process massive amounts of data. Large-scale data processing enables the use of different types of data in a cloud to provide diverse analytic services (see Figure 2.2). A large-scale data processing platform serves as a framework for both storing and processing of a large volume of data (batch and stream) in a distributed environment.

Big Data processing can be classified into two types: batch processing and stream processing. A framework for Big Data processing and analytics projects resembles that of a traditional business analytics projects. The key difference is how the processing is executed in the Big Data environment. In a traditional framework, analysis is performed with a business intelligence tool on a stand-alone system. In Big Data, processing is executed across multiple nodes.

The concept of distributed processing is not new. However, the availability and rapid growth of open-source platforms such as Hadoop and Spark have encouraged many organizations to use Big Data analytics in various domains. The challenge in their use is that Big Data tools are incredibly complex and to utilize them successfully requires a variety of skills.

Figure 2.2 shows detailed steps involved in Big Data processing and analytics. Data acquisition is performed using various sources such as: the Internet, cameras, mobile terminals, self-driving cars, social networks, hospitals, smart homes, smart meters, and sensors. The collected data are stored in a distributed file system such as HDFS. These data has to be retrieved, processed and analyzed with various tools.

Data-intensive processing tools such as Spark and MapReduce are used in conjunction along with machine-learning techniques to provide analytics to end users. Additionally, the data warehouse method [57] is also used whereby various data from an array sources are aggregated and processed. Before processing, the data is cleaned and made ready for use by way of ETL (extract, transform, and load). Structured and unstructured data is processed via data-intensive tools [23, 20].

The next step is the analytics layer wherein machine learning, data mining, and pattern recognition techniques are used to provide insights. The value is presented in a report or the form of charts.

## 2.3 Big Data Technologies

The following subsections provide an overview of different technologies used to conduct the research performed in the enclosed papers.

### 2.3.1 Hadoop

Hadoop [1] [21] is an open-source framework for distributed storage and data-intensive processing, first developed by Yahoo! [2]. It consisted of two core projects: Hadoop Distributed File System (HDFS) [35] and MapReduce programming model [41]. HDFS is a distributed file system that splits and stores data on nodes throughout a cluster, with a number of replicas. It provides an extremely reliable, fault-tolerant, consistent, efficient and cost-effective way to store a large amount of data.

HDFS is designed to store large-scale data in terabytes in an efficient way. It is based on Google file system (GFS) [48]. Each node in Hadoop has namenodes and a cluster of datanodes to form the HDFS cluster. Clients use RPC to communicate each other. HDFS stores large files (64MB to 512MB chunks of a single logical file), across the cluster. It mainly separates file's metadata and application data.

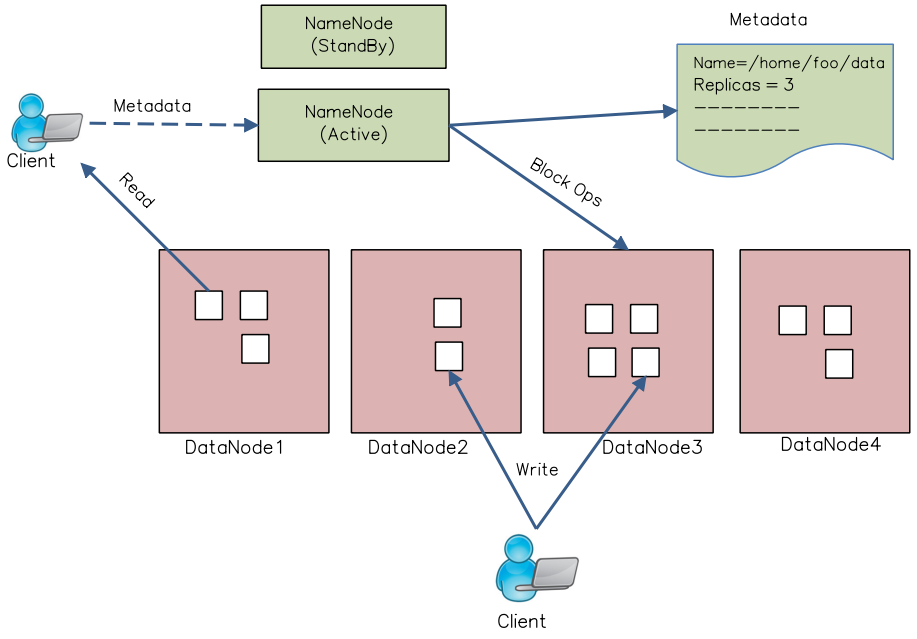



**Figure 2.3:** Architecture of HDFS.

The NameNode is the master node on which the job tracker runs. It

[1] Hadoop; [http://hadoop.apache.org/]

[2] Yahoo! Developer Network, (2014), Hadoop at Yahoo!, [http://developer.yahoo.com/hadoop/]

contains the metadata (information about data blocks stored in DataNodes - the location, size of the file, etc.). It maintains and manages the data blocks, which are present on the DataNodes, where the actual data is stored. The DataNode runs three main types of daemon: Read Block, Write Block, and Write-Replicated Block. The NameNode and the DataNode maintain their own logging format. Each node records events/activities related to reading, writing, and the replication of HDFS data blocks. When a file is written in HDFS, it is divided into blocks of a fixed size. The client first contacts the NameNode, which gets the list of DataNodes where actual data can be stored. The data blocks are distributed across the Hadoop cluster. Figure 2.3 shows the architecture of the Hadoop cluster node used for both computation and storage.

MapReduce is parallel programming model introduced by Google in 2004 that processes large data on clusters of computers. It has proven to be very attractive for parallel processing of arbitrary data [41]. It consists of two user-defined functions, Mapper and Reducer. The Mapper processes input data splits in parallel through different map tasks and then sends sorted, shuffled outputs to the Reducers that in turn group and process them using a reduce task for each group. MapReduce manages scheduling of the task across clusters, fault-tolerance, splitting the input data managing communication between nodes.

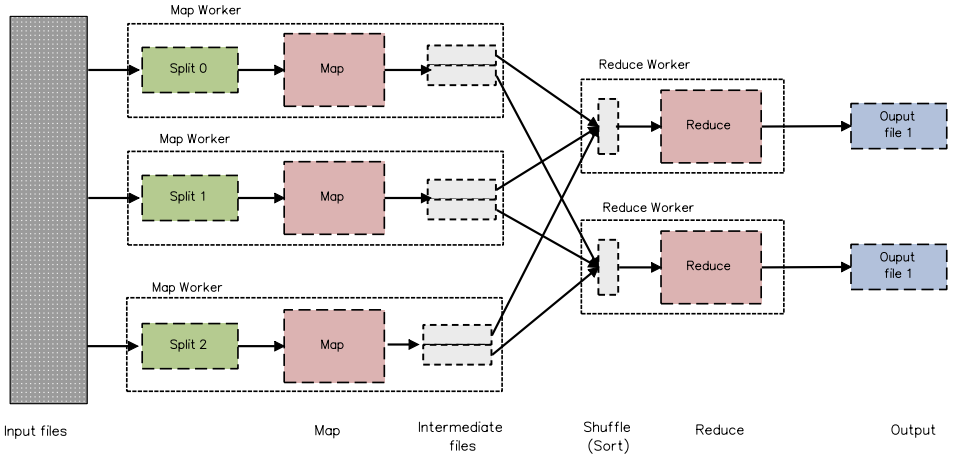



**Figure 2.4:** MapReduce simplified Flowchart.

A MapReduce program takes a set of key/value pairs as input and produce a set of key/value pairs as output. It splits input file into independent fixed-sized chunks called input splits. A set of key/value pairs

is read from each chunk of data. Each chunk of data is processed by the mapper process in a distributed environment as shown in Figure 2.4. A map function is user defined function, which reads key/value pairs from the input file. A map function generates another intermediate key/value pairs. This intermediate key/value pairs (output of map function) is written to a local disk. The values obtained are associated with the intermediate key and are grouped together by Mapper function. The reduce function then merges the values and finally outputs a set of key/value pairs.

Hadoop framework reduces the cost of cluster construction by creating clusters of inexpensive commodity hardware for distributed data storage and processing. It provides an interface for organizations to extract, store and analyze data in large scale. Hadoop can store an enormous amount of data whenever and whatever form is needed, simply by adding more servers (commodity machines with relatively less price) to an existing Hadoop cluster. This makes data storage with Hadoop cheaper than traditional methods.

### 2.3.2 HBase

HBase [3] [26] is a column-oriented and distributed database that uses HDFS as its storage layer and is modeled after Google's BigTable [40]. HBase is NoSQL database that provides random read/write access in real-time to large datasets [21]. In HBase, a table is physically divided into many regions, which are in turn served by different Region Servers. One of its uses is to combine real-time HBase queries with batch MapReduce jobs, using HDFS as a shared storage platform. HBase table can integrate with Hadoop to serve as a source or destination of MapReduce jobs.

### 2.3.3 OpenTSDB

OpenTSDB [4] is an open source, distributed and scalable time-series database, developed by StumbleUpon. It supports a real-time collection of data points from various sources. It is designed to handle terabytes of data with high performance for different monitoring needs. It stores, indexes and serves metrics at a large scale. Data is stored in HBase in two different tables: the *tsdb* table provides storage and query support over time-series data and the *tsdb-uid* table maintains an index of globally unique values for all metrics and tags.

[3] Hbase; [https://hbase.apache.org/]
[4] OpenTSDB; [http://opentsdb.net/]

### 2.3.4 R and RHIPE

R [5] [27] is a language and environment widely used among statisticians and data scientists. It provides a wide range of libraries for analysis and visualization. R Hadoop Integrated Processing Environment (RHIPE) [6] is a library for integrating R with the Hadoop DFS. Using the MapReduce programming model, RHIPE computes massive data sets across different nodes within a cluster. It works from the R environment using standard R programming idioms [16, 17].

### 2.3.5 Apache Spark

Apache Spark [7] is an open-source distributed framework that has recently become popular for data analytics. Similar to Hadoop, it is fault-tolerant and supports distributed computation systems to process fast and large streams of data. It uses Hadoop distributed file system to store and read data. It provides in-memory cluster computing that allows user to load data into a cluster's memory, which in turn makes it perform up to 100 times faster than Hadoop MapReduce. Apache Spark introduced the concept of Resilient Distributed Datasets (RDD) [22], which is a distributed memory abstraction that allows in-memory computation on large distributed clusters with high fault-tolerance. It enables efficient data reuse that let users explicitly persist intermediate results in memory. RDDs are a good fit for many parallel applications. RDDs is used in iterative in-memory operations where data is read multiple times and manipulated using a rich set of operators. [1].

### 2.3.6 Apache Kafka

Apache Kafka [8] [14] is publish-subscribe messaging also considered as a distributed commit log. It is scalable, fast, distributed, durable and offers high throughput [29]. Kafka contains feeds of messages in categories called topics. The process that publishes messages are producers, and the process that feeds published messages are consumers. Kafka consists of a cluster comprised of one or more servers each of which is called a broker. In Kafka, producers send messages over a network to the cluster which serves to consumers as shown in Figure 2.5.

[5] R; [https://www.r-project.org/]
[6] RHIPE; [https://github.com/saptarshiguha/RHIPE]
[7] Apache Spark; [http://spark.apache.org/]
[8] http://kafka.apache.org/

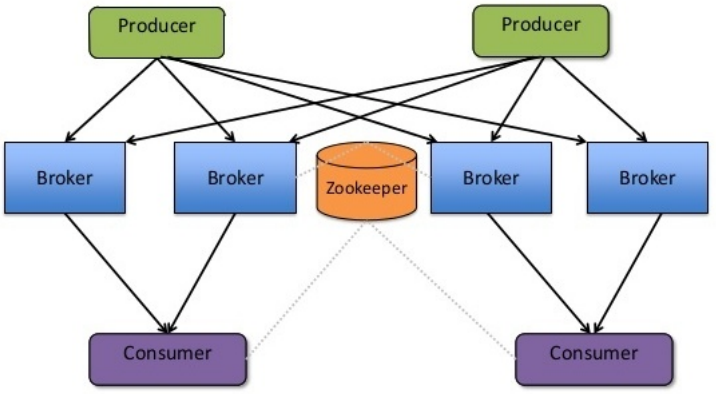



**Figure 2.5:** Kafka Architecture

## 2.4 Data Science

The National Institute of Standards and Technology define Data Science as "extraction of actionable knowledge directly from data through a process of discovery, or hypothesis formulation and hypothesis testing" [8]. Data Science is emerging as the fourth paradigm of science [37].

Data science is an interdisciplinary area which extracts knowledge or insights from data in various forms using algorithms from different fields like statistics, machine learning, data mining, and predictive analytics.

The explosion of data production, storage capabilities, computational capacity, and cloud technologies make data science a critical component in high-growth economic sectors such as healthcare, industrial production, retail, banking, government, and others.

## 2.5 Machine Learning

Machine learning is a collection of techniques that iteratively learns from data to complete a task, or to make accurate predictions. It uses set of computer algorithms and observations of data for learning. In general, machine learning enables a computer to improve future performance by learning from historical experience.

There are many real examples where machine learning is effective:

- spam filtering: identify email messages as spam or non-spam.

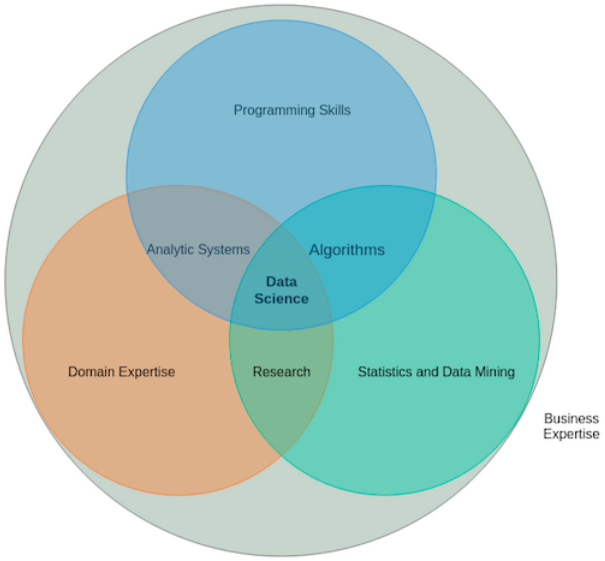



**Figure 2.6:** Definition of Data Science [8]

- face and image detection: find faces in images.
- recommendation's engine: suggestions based on data points from past selections (Amazon and Netflix).
- fraud detection: identify credit card transactions.
- weather prediction: predict whether it will rain tomorrow or not?
- medical diagnosis: diagnose a patient whether it suffers from a disease or not.
- optical character recognition: categorize images of handwritten characters.

With the advent of Big Data, collection of data is so large and complex that it is difficult to process with using traditional methods and models. As a result, some traditional statistical methods and signal processing techniques are unsuitable to satisfy the requirements of complex event processing and storage for Big Data. Thus, this needs to explore machine learning techniques with the power of data-intensive computing and distributed storage to analyze large-scale data.

The data are steadily growing in size. And the additional data help to train the model to achieve high accuracy even with simple algorithm [56]. There is one of the most famous quotes by Peter Norvig claiming that "We don't have better algorithms. We just have more data." [36]. The effect that Norvig et. al proposed in their paper was already captured years before on natural language processing with different machine learning algorithms in a famous paper by Microsoft [51]. The simple algorithm can achieve higher accuracy with large sample size. This paper addresses the problem on sentence disambiguation. For example, the relationship between two variables is linear - the number of pages viewed on a website and percent unique visitors on a website. Having more data points would improve the accuracy and confident estimation of two variables. Therefore, applying machine learning algorithms on a large amount of data requires Big Data tools for storing and processing.

The following subsections provide an overview of the different machine learning techniques used for various experiments performed in the enclosed papers.

### 2.5.1 Hidden Markov Models

HMM [63] is based on Markov Models, a tool for representing probability distributions over sequences of observations. It is a probabilistic model, in which system is assumed to be a Markov process [65] (memoryless process) with hidden (unobserved) states. HMM consists of unobserved states, and each state is not directly visible, but output and dependent on the state are visible. It has a set of states each of which has a number of transitions and emissions state probability as shown in Figure 2.7. HMM typically used to solve three types of problem: detection or diagnostic problem, decoding problem and learning problem. Forward-backward algorithm [45] solves diagnostic problem. Similarly, Viterbi algorithm [64] solves decoding problems and Baum-Welch algorithm [61] solves learning problem.

### 2.5.2 Robust PCA (Principal Component Analysis)

PCA is a linear transformation that maps a given set of data points into new axis (i.e. principal components). It is used in the dimensional reduction technique. In the classical PCA, the eigenvectors and eigenvalues [62] are calculated from the sample covariance matrix using Euclidean distances between sample data points [46]. In RPCA, robust covariance estimation is used for eigen decompositions. The decomposition of a low-rank matrix

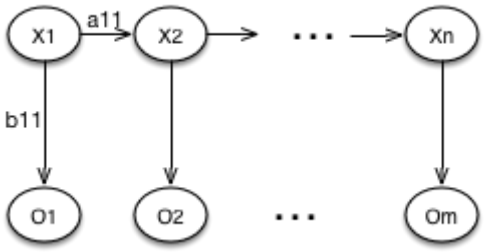



**Figure 2.7:** Hidden Markov Model.

from a set of observations with gross sparse errors is known as robust *principal component analysis* (RPCA) [24]. The robust PCs represent the data effectively in a lower-dimensional space. The anomalies are detected in this lower-dimensional space using distance measured; orthogonal distance, which is the distance of an observation to the PCA space. It has many applications in computer vision, image processing, and data ranking. In addition, if the observed data has been contaminated by a dense noise in addition to gross sparse errors, RPCA is used to get a low-rank matrix.

### 2.5.3 Ensemble

Ensemble methods combine a diverse set of individual models merged together to improvise on stability and predictive power [34]. Ensemble methods use learning algorithms to construct a set of classifiers and then classify new data points by taking an average weight of their predictions accuracy [54]. Ensemble techniques can provide high accuracy, but in some cases they over-fit the training data more than a single model.

### 2.5.4 FM (Factorization Machines)

Factorization machines are a new model class that combines the advantages of Support Vector Machines (SVM) [58] with factorization models. Steffen Rendle [19] introduced this model in 2010. This model exhibits similar properties of SVM. But FM is used as general prediction tasks. Like other models, non-linear SVMs or decision trees, FM includes interactions between predictor variables. For example, FMs can learn that young users prefer to access data from mobile devices, whereas of older users prefer a desktop enviroment.

### 2.5.5 Random Forest

Random Forest is an ensemble learning method operated by constructing a multitude of decision trees at training time for classification and regression [52]. Random Forests grow many classification trees. To classify a new target from an input vector, each vector is added down the trees in the forest. Each tree gives a classification and all trees votes for that particular class. The forest chooses the classification having the most votes. For example; in a case of movie recommendation, Netflix recommends you a set of the movie whenever you want to watch a movie. To figure out best recommendation, Netflix creates a set of questions like "Is X a romantic movie?", "Does Johnny Depp star in X?", and so on. It figures out a bunch of informative question first and gives a yes/no answer at the end. This process created a labeled training set. Moreover, it will look into other users data based on that it will build an ensemble classifier, (a forest).

### 2.5.6 TF-IDF

TF-IDF stands for term frequency-inverse document frequency, and it is a weighting scheme often used in information retrieval and text mining [55]. This weight specifies how important a word is to a given document in the overall collection. The importance of word increases proportionally to the number of times it appears in the document. The objective of TF-IDF is to model each document into a vector space, without caring much about ordering of the words in the document. It is composed of two terms: (i) Term Frequency (TF), which is the number of occurence of a word appears in a document, denoted in Equation 2.1 (ii) Inverse Document Frequency (IDF), is the logarithm of the number of documents in the corpus divided by the number of documents where the term $t_i$ appears in, denoted in Equation 2.2 [38]. TF-IDF are often used by search engines for page ranking. For example, consider a document of 50 words wherein the word data appears 5 times. The term frequency (i.e., tf) for data is then (5 / 50) = 0.1. Now, assume corpus of documents is 10 million and the word data appears in 1000. Then, the IDF is $log(10,000,000/1,000) = 4$. Thus, the TF-IDF is the product of $tf * idf$: 0.1 * 4 = 0.4.

$$TF(t) = \frac{\text{Number of term t appears in a document}}{\text{Total number of terms}} \tag{2.1}$$

$$IDF(t) = log_e(\frac{\text{Total number of documents}}{\text{Number of documents with term t in it}}) \tag{2.2}$$

### 2.5.7 Naive Bayes

Naive Bayes is a classifier that is based on the popular Bayes' probability theorem. They are known for creating simple yet well-performing models, especially in the area of document classification [53]. Naive Bayes classifier is scalable and very efficient for large datasets since it is less computationally intensive ( CPU and memory) [49]. Moreover, it requires less training data and training time.

### 2.5.8 K-nearest neighbors

K-nearest neighbors are one of the simplest algorithms that classify new classes based on distance functions (e.g., Manhattan, Euclidean, Minkowski, and Hamming) [39]. K-NN has been widely used as a non-parametric technique used in statistical estimation and pattern recognition since the 1970's. In most basic term it assigns label of its nearest neighbor to an observation. The label is assign by calculating distance between paris of observations.

The problem with nearest neighbor classification is that the estimation can be nosiy if the data is nosiy or has irrelevant features [42]. For example, if a spam email is labeled as nonspam then all emails which are similar to this email will classified as nonspam. However, it can yield excellent performance when used in proper and with a good distance functions. For example, the winnder of Netflix progress prize [44] was essentially based on nearest neighbours based.

### 2.5.9 XGBoost

XGBoost is "Extreme Gradient Boosting" [6] [4]. Usually, it is similar to Random Forest with multiple trees in series. It uses the combination of a tree that is so-called tree ensemble model, and that sums the production of multiple trees together. In Gradient Boosting, the error of one tree is adjusted to another, using parameter gradient to offer a better prediction for the new tree. XGBoost is primarily a tree ensemble model.

### 2.5.10 Neural Network

A neural network [50], is a network of interconnected processing elements (neurons) that work in unison to solve a particular problem. They have been inspired by the way a biological nervous system works and processes information. The goal of the neural network is to solve problems in the

same way that human brain would solve. Neural networks are typically consist of large number of highly interconnected processing elements (neurones) working in parallel to solve a specific problem.

### 2.5.11 Feature Engineering

Feature engineering is the process of creating features that make machine learning more accurate and efficient using domain knowledge of the data. It is an important step of data preparation process, where new and meaningful variables take place. Feature Engineering can drastically affect the performance and accuracy of a model. In some cases, it reduces feature space by ignoring some useless features, thus reducing overall training time. Some of the common techniques for feature engineering are discussed below:

- *Data Standardization*: There might be an issue if the distributions of different features are radically different. It is solved by transforming the data toward zero mean and unit variance [59].
- *Data Normalization*: It is the process of scaling individual samples to have unit norm. It is done similarly as standardization process, moving the data toward zero mean and unit variance.
- *Encoding categorical features*: Turning categorical variables to a fixed-length vector. This step is also known as *One Hot Encoding* [18].
- *Feature Binarization*: Converting numerical features to get boolean values.
- *Feature Discretization*: Converting continuous features to discrete features. Typically, data is sampled into partitions of equal intervals [60].
- *Feature Regularization*: It is a technique used to solve the overfitting problem in machine learning algorithms [47].
- *Missing Data*: This is a reality that everybody has to reckon with. The missing data can be replaced with mean, median or sometimes it is useful to ignore it.

## 2.6 Cloud Computing

The National Institute of Standards and Technology (NIST) describes cloud computing as “a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources

(e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [30].

The development of cloud computing has resulted in a large migration of individuals and organizations into cloud storage systems. Motivated by benefits such as shared services, storage, and computation, among a massive number of users, today enterprise applications both generate and require to handle huge volumes of data on a regular basis. This is appropriately referred to as data intensive computing. Cloud computing allows users to focus on renting and scaling services for an optimal resource utilization.

The core concept of cloud computing is based on five essential characteristics: (i) *On-demand self-service*: enable a consumer to automatically and independently provision computing and storage capabilities according to their needs. (ii) *Broad network access* capabilities: allow users to access data from anywhere and on any device (mobile, tablets, laptops, and workstations). (iii) Cloud provider *resource pools*: serve mul tiple consumers, with different physical and virtual resources dynamically allocating according to consumer demand. (iv) The storage and computing capability can be *elastically provisioned* to scale rapidly outward and inward commensurate with demand. (v) A *metering capability* allow users to control and optimized resource automatically. Resource usage can be monitored and controlled, to provide transparency for both the cloud user and provider.

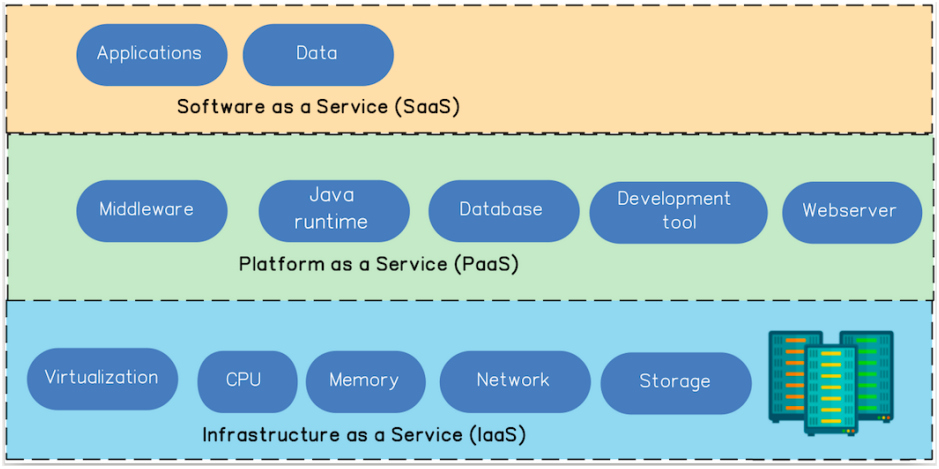



**Figure 2.8:** Cloud Computing Architecture [7].

Cloud computing is categorized into three different services as shown in Figure 2.8, and provide the following functionalities:

- **Infrastructure as a Service (IaaS):** Cloud vendor provides resources with minimum support, with raw storage and computing resources (i.e., close to the bare hardware). A typical example of IaaS cloud is Amazon Elastic Compute Cloud (EC2) [9]. Cloud vendors allocate their large storage and compute capability as demanded by users to provide ad-hoc virtual environments [43].

- **Platform as a Service (PaaS):** It allow users to create, run, and manage a set of applications without the complexity of managing or building underlying cloud infrastructure including storage, network, servers, operating systems, or storage. However, the user has freedom to deployed applications and configure the application for hosting environment. The most common examples of PaaS cloud are the Google Apps Engine [10] and Microsoft Azure [11].

- **Software as a Service (SaaS):** User, can use cloud applications running on a cloud infrastructure. Cloud vendor can host their services and provide online, most commonly through APIs and web-browser interfaces. The main advantages of this model are that it frees the user from any administrative management, software installation, and updating. Some general examples for this model are the Google Docs [12] and Microsoft Office 365 [13] tools.

Cloud computing can be deployed in several ways. A *Private Cloud* is a cloud model that is exclusively used by a single organization. It may be owned and managed by the organization, owned and manage by a third party, or may be a combination of both. A *Public Cloud* is a deployment model where a service is owned, managed and operated by an organization that makes the cloud available for open use by the general public. A *Community Cloud* provides services for exclusive use by a particular community that have shared concerns. A *Hybrid Cloud* is an integration of two or more distinct cloud entities that are bound

[9] Amazon Web Services: http://aws.amazon.com/
[10] Google Apps Engine; https://cloud.google.com/
[11] Microsoft Azure; http://azure.microsoft.com/en-us/
[12] Google Docs; https://docs.google.com/
[13] Microsoft Office 365; https://products.office.com/en/office-365-home

together for data and application portability (e.g., a transactional order entry system which when demand is high).

Cloud computing provides compute and storage resource that can be acquired on a "pay per use" basis. Typically, it requires no expenses to buy hardware/software upfront, thus cloud consumer can use operating expense budgets (OPEX) to fullfill their requirements, giving them plenty of budgeting flexibility [31]. The alternative approach is to invest in hardware and software, thus capitalizing assets (CAPEX) [5]. For example; If you need a car only for a week, you can rent it for a week. You only pay for the week that you used for and not needed to invest in purchasing the car. In an OPEX model, the maintenance expenses is built into "as-a-Service" cost.

# Chapter 3

# Contributions

This dissertation proposes an extension to cloud-based Big Data framework that provide a scalable, efficient, and accurate approach to improve the performance of data center by predicting failures and detecting anomalies.

This dissertation addresses these needs and proposes solutions (like a distributed scalable analytics framework) that enrich the cloud platform and enable high-performance processing on a large scale. The contribution of this dissertation is demonstrated by applying state-of-the-art processing framework and consequently improving data center overall performance by predicting failures and detecting anomalies. The framework enables Big Data applications to gain an advantage using cloud computing such as scalability and elasticity.

The contribution of this dissertation is scalable anomaly detection and failure prediction for a Big Data cloud platform to increase the performance of data centers. These techniques are based on a scalable machine learning algorithms to provide extensions to cloud-based Big Data framework. Moreover, this dissertation provides a state-of-art solution to permanently delete data stored by Big Data applications. Figure 3.1 shows the research outline of this dissertation. The research is divided into four different areas: security, storage, processing, and analytics. Paper III, describes a supporting tool that provides secure deletion of the data collected during the analytic step, which is described in the remaining four papers (papers I, II, and IV). Paper I, provides a processing and storage framework for large-scale data generated in the cloud. Papers II, IV, and V deal with distributed processing and scalable analytics to improve the efficiency and performance of a data center. In particular, papers V and VI provide machine learning techniques to improve the effectiveness of

the classification algorithm. Papers II and IV present scalable machine learning techniques to improve the performance of data centers.

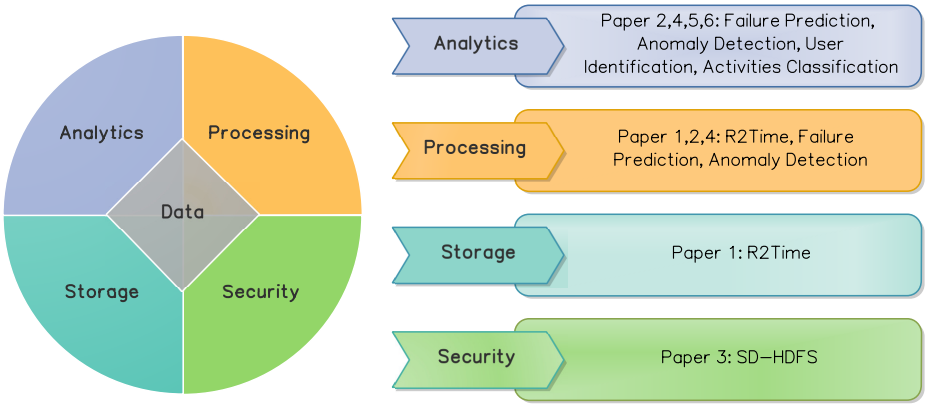



**Figure 3.1:** contribution of papers

Summary of the papers contribution are listed below.

Paper I proposes an efficient distributed computing framework: R2Time to process the time-series data in the Hadoop environment. R2Time allows R users to interact directly with OpenTSDB data and perform statistical analysis using the MapReduce programming model. R2Time allows analysts to work on massive datasets within a popular, well supported, and powerful analysis environment.

Paper II introduces a novel algorithm for failure prediction, using the MapReduce programming framework, for improved scalability and failure prediction probability. It accomplishes this by predicting the failure of jobs and computing nodes, which also boosts the performance of the data center.

Paper III [1] introduces an approach for data deletion that solves data leakage in HDFS that can be caused by node failure and data sniffing using forensic tools. The application of an additional component called *checkerNode* ensures that all blocks are deleted. In the case where it is impossible to delete a block, the component provides detailed reports of hardware components that might contain sensitive data. The deletion process is extended by physically locating each copy of the block and overwriting it to prevent data spillage using forensic tools.

Papers IV proposes an automated, self-adaptive, anomaly-detection technique in a distributed environment. It uses Spark as a framework to

[1] This research was conducted at Purdue University.

detect anomalies in the cloud infrastructure. An adaptive algorithm is introduced, which uses reconstruction errors to determine the sample size and update the threshold value.

Papers V and VI demonstrate "*analytics as a Service*" on different data sets obtained from Drawbridge devices and smart homes. Using machine learning techniques such as FFM to identify the user across cross-devices, as well as ensemble learning to classify activities in the smart home.

## 3.1 Research Questions

This dissertation contributes the following research questions:

(1) How to store and process large amounts of data from cloud monitoring logs effectively and efficiently?

(2) How to improve data center efficiency, performance, transparency, and availability?

(3) How to improve the accuracy of operations such as, failure detection, anomaly detecton, activity classification, and user identification using machine learning techniques?

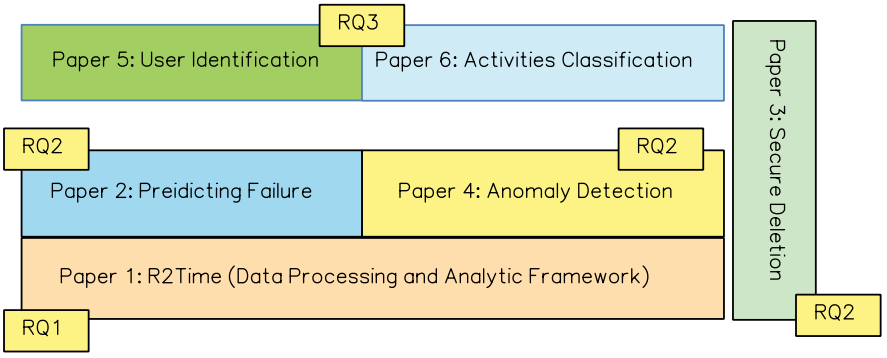



**Figure 3.2:** Contextual relationship between papers.

Figure 3.2 demonstrates the contextual relationship between different research questions and the respective papers included in this dissertation.

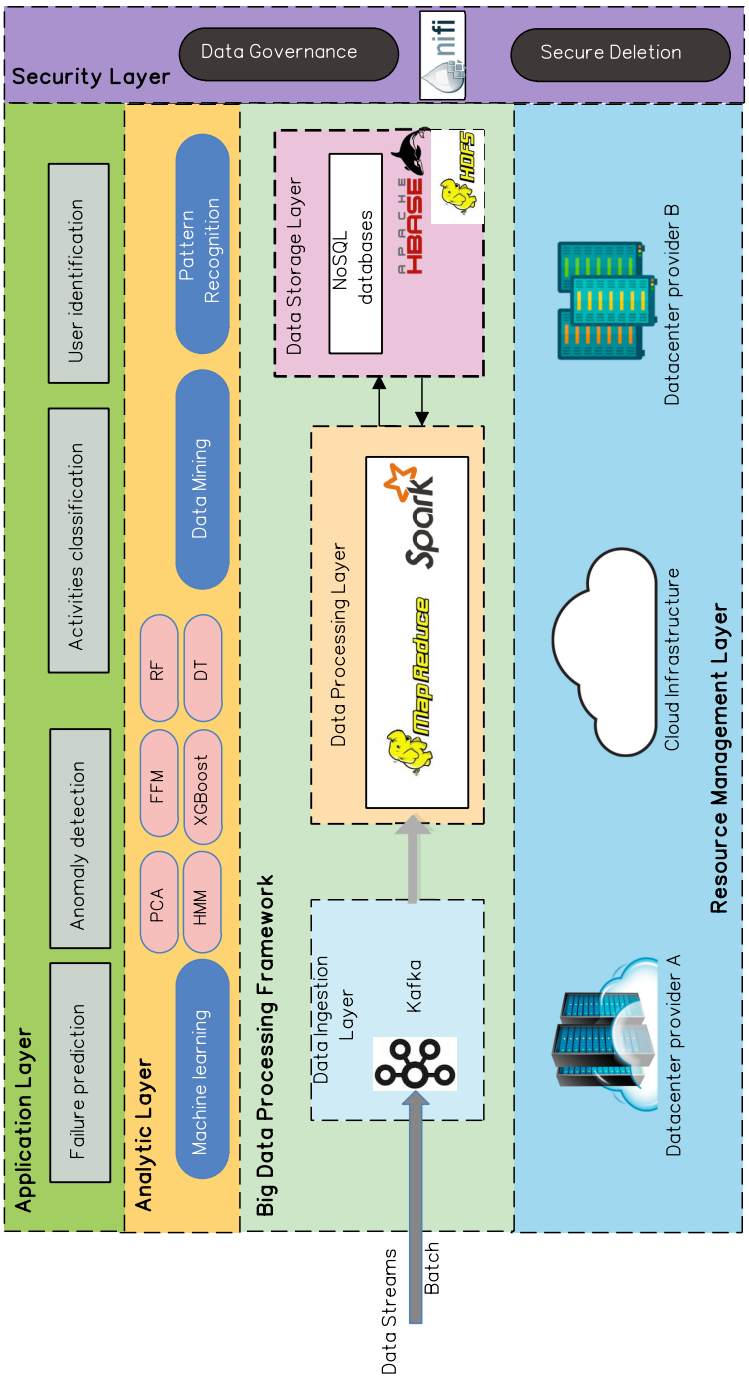



**Figure 3.3:** High-level architecture of Big Data ecosystem used in this dissertation.

## 3.2 Overview

A scalable framework for processing data from the data center and IoT devices is needed. Figure 3.3 demonstrates the Big Data ecosystem high-level architecture that is used during the research for this dissertation. The work is categorized into five layers: Application, Analytic, Processing, Resource, and Security.

### 3.2.1 Application Layer

This layer provides business insights derived from the analytic layer. It establishes that the insights gleaned through the analysis layer are provided to the user who can take action to make decisions. This layer provides results in the form of reports, charts, figures and key recommendations. The outcome of the analysis is important for various users within the organization as well as entities such as customers, vendors, partners, and suppliers. The insights can be used for various purposes such as product-targeting of customers and delivery of personalized offers. Correlating insights can also be used to target fraud in real-time. Indeed, a customer can be notified as soon as a fraudulent transaction occurs so that corrective actions can be taken immediately.

The application layer provides a visual interface to the user in the form of dashboards, charts, or reports that allow stakeholders and customers to make correct decisions and to design appropriate reparative strategies. Papers II, IV, V and VI describe an interface for a user to obtain analytic reports in the form of charts. Papers II and IV describe a pro-active measure through push notification for end users.

### 3.2.2 Analytic Layer

Stored data needs to be processed and analyzed to extract it's value. The most common approach is to use MapReduce and Spark for distributed processing. In this approach, the data elements are selected for analysis and modified to a format from which their value can be derived.

Paper II, discusses a framework for detecting failure in large systems. Failures in a large system can and will happen at various levels. Examples include network/bandwidth issues, disk failures, and node failure. A large platform should be able to recover from all these failures and be capable to resume from the last successful state without distorting the result. Moreover, it is preferable to employ a system that can predict failure

before it happens. In this paper, we propose a Hidden Markov Model, which is executed on top of MapReduce. This model is trained by using the prior failure record of the system. Looking at the failure pattern, our model can predict when and how future failure is likely to happen.

Paper IV, provide proactive anomaly detection that enhances the performance of a cloud platform. It uses a combination of SVD and RPCA beneath machine learning techniques. Papers V and VI, provide user identification and activity classification by using machine learning techniques like FFM and an ensemble model.

### 3.2.3 Big Data Processing Layer

A big data processing framework is often referred to as "software middleware". It helps a distributed computing environment to store and analyze large-scale data. This is done by processing, analyzing, and visualizing the data, as described in papers I, II, and IV. The framwework consists of three main components:

- Accepting real-time and batch data. *Data ingestion* acquires data from sources such as sensors, online services, email archives, sales records, social media channels, and log files.

- *Data processing.* This is another important aspect for Big Data management. Data processing consists of a scalable distributed environment, which can process stream/batch data. It consists of a framework like MapReduce [2] and Spark [3], which is used for data-intensive computing.

- *Data storage.* As the volume of data generated and stored by devices explodes, traditional systems have become inconvenient. Big Data tools such as Hadoop Distributed File System (HDFS) have emerged as an effective alternative to older systems. Consisting of a NoSQL database to store the data and intermediate results, the data can be stored in a distributed file system such as HDFS. A database system is needed that can organize and categorize the data in a way that people can understand. Hadoop has it own NoSQL known as HBase [4],

[2] http://wiki.apache.org/Hadoop/MapReduce
[3] http://spark.apache.org/
[4] https://hbase.apache.org/

but there are others like Amazon's DynamoDB [5], MongoDB [6] and Casandra [7].

Paper I is a framework that consists of three Big Data components. It proposes a framework called R2Time to enable distributed processing of large time-series data across a Hadoop cluster from the R environment. The R2Time framework allows a data scientist to work on the data collected from sensors using a standard processing scheme. It also efficiently stores time-series data by using OpenTSDB in a cost-effective way. Furthermore, papers II, IV, and V use the Big Data processing framework to predict failure and detect anomalous behavior in a data center.

### 3.2.4 Infrastructure Layer

This layer provides a horizontal scaling infrastructure for efficient large-scale data storage and processing. It also enables users to access virtual machines to store, and analyze Big Data. This layer acts as a major data source and computing done in this dissertation. Hadoop cluster data and cloud platform workload were generated by various applications running in the cloud platform.

### 3.2.5 Security Layer

Data privacy associated with Big Data collection is a significant challenge. Real time processing of large data complicates real-time synchronization. Conversely, demands for data privacy system architecture and computing power. Infrastructures that process large-scale data is gigantic and because traditional computation methods and security were designed for small scale data, these methods became inadequate to meet the demands of managing Big Data.

Billions of users leave data footprints when they access social networks, search engines, and networked websites. Extracting data and processing of data footprints can cause privacy problems. During real-time processing, finding a method to maintain data security and to accelerate processing speed is a critical issue for a Big Data framework.

Paper III, provides a tool for securely deleting data from a distributed file system. Storing data in a large distributed system can lead to data

[5] https://aws.amazon.com/documentation/dynamodb/
[6] https://www.mongodb.com/
[7] http://cassandra.apache.org/

leakage in two ways: undeleted blocks due to node failure and data sniffing using forensic tools. The SD-HDFS tool provides an improved deletion technique in HDFS. The use of additional components called *checkerNode* ensures that all blocks are deleted. In cases where it is impossible to delete data, the *checkerNode* component provides detailed reports on hardware that might contain sensitive data. Furthermore, we extend the deletion process by physically locating each copy of the block and overwriting it to prevent data spillage using forensic tools.

In future work, a tool for providing data governance needs to be developed using Apache NiFi [8]. Governed data are more reliable, secure, and ready to use, while data from an ungoverned has little value for analytics and business operations. Data governance provides a framework to set policies and implement controls designed to ensure that information remains accurate, consistent, and accessible.

Summarize the main contributions of the paper:

## 3.3 Paper I: R2Time: A framework to analyse OpenTSDB timeseries data in HBase.

This paper was published in the proceedings of 2014 IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom).

**Motivation:**

The volume of generated time series data continues to grow exponentially and demands the analysis of massive time-series datasets produced by sensor networks, power grids, stock exchanges, social networks and cloud monitoring logs. Big Data storage and processing frameworks provide the capability to manage the volume, velocity and frequency attributes associated with time-series data. Though OpenTSDB provides a platform to store time-series data, it has limited functionality to perform complex analytical functions like regression, correlation, and ARIMA. To solve this problem, a tool is developed that can perform sophisticated analytics in a scalable and distributed environment.

[8] https://nifi.apache.org/

**Method:**

In this paper, we develop a bridge between HBase and R to enable data-intensive computing. A composite row key is constructed using a start and end date provided by the user. HBase scans the data from region servers using row keys. The analytic user program written in R is executed using a Mapper and Reducer function of the MapReduce paradigm.

**Results and Conclusions**

The basic statistical and classifier functions were tested in the framework. They appear to be supra-linear in nature. This is due to the algorithm being scaled using the MapReduce paradigm. Moreover, the read latency in HBase is higher because of its log-structure based storage. Read latency in HBase can be optimized through compaction. The load is well distributed across a large number of nodes to improve performance. R2Time has improved execution time compared to OpenTSDB. R2Time performs by almost 50% better with two nodes clusters in comparison with OpenTSDB.

## 3.4 Paper II: Analyzing and Predicting Failure in Hadoop Clusters Using Distributed HMM.

This paper was published in the proceedings of 2015 6th International Conference on Cloud Computing and Big Data in Asia.

**Motivation:**

The enormous amount of data obtained from sources like sensor networks, power grids, stock exchanges, social networks, and cloud monitoring logs must be stored and processed. Hadoop, a pioneer in Big Data processing and storage provides this capability. Many enterprises and organizations use the Hadoop cluster to store and process Big Data. The prime objective of the Hadoop cluster is to maximize processing performance by employing data-intensive computing. The Hadoop cluster consists of several nodes, and failure in any node can result in higher execution time. However, if the prediction of a failure node and job type can be analyzed, the performance of the cluster can be improved.

**Method:**

In this paper, Hidden Markov Models (HMMs) are used to learn the characteristics of log messages and use them to predict failures. The model is based on a stochastic process with a failure probability of the previous state. Because the faults are unknown and cannot be measured, they produce error messages during their detection (i.e. present in log files). Our prediction model is divided into four main parts: First, the identification of error sequences and differentiating types of errors from the log files. Second, the usage of the clustering algorithm. The labeled training data are used to evaluate a maximum likelihood sequence that is used to update the parameters for our model. Last, the prediction of system failure through the observation of an error sequences.

The main idea of our approach is to predict failures by analyzing error event patterns that imitates failure. Each error event consists of a timestamp, error ID, and error type, which determine the type of errors. Both failure and non-failure information are extracted from error sequences to create a transition matrix. The observation symbols $O_1 = \{e_1, e_2, e_3, e_4, e_5, e_6\}$ are referred to error events of the system, and failures are represented as hidden state of HMM.

**Results and Conclusion:**

Because failures in the cluster systems are more prevalent, the ability to predict failures is becoming a critical need. To address this need, we collected Hadoop logs from a Hadoop cluster and developed our algorithm on the log messages. The messages in the logs contain both error and non-error information. The messages in the log were represented using error IDs that indicate message criticality. The different types of error messages (operational, software, resource errors) were classified. Six different types of errors were detected on different DataNodes: network connection, memory overflow, security setting, unkown, Java I/O error, and NameNode failure. Based on error sequences in the observation state in HHM, the model was trained. Training of the model was done using past observation. Viterbi's algorithm performs the prediction of the hidden state. Experimental results using Hadoop log files provide an accuracy of 91% and F-measure of 92% for two days prediction time. These results indicate that it is useful to use the HMM method together with MapReduce to predict failure.

## 3.5 Paper III: Secure Deletion in Hadoop Distributed File System.

This paper was published in the proceedings of 2016 IEEE, International Congress on Big Data (BigData Congress).

**Motivation:**

Data stored in a large Hadoop cluster does not provide transparency to the user. When a user uploads the data to HDFS, the underlying storage layers essentially keep the data immutable, only allowing concurrent appends. HDFS does not support in-place updates. Many applications that are write-intensive and require file modifications need to overwrite all file content, even if very few changes were made. Therefore, changing any file content requires the recreation of entire data blocks, which effectively increases the overall write and update performance of the system.

**Method**

A secure deletion framework propagates file information to the block management layer via an auxiliary communication path, so that file deletion can be honored throughout the data path. A checkerNode is added in the HDFS environment, which receives summary reports from the DataNodes and compares the block stored in the DataNodes with metadata in the NameNode. A file in HDFS is divided into a chunk of blocks stored in the file system. The data content of a file is deleted via its truncate function which involves updating the inode to set the file size to zero. Multiple rounds of truncation are suggested for secure deletion of content.

**Results and Conclusion:**

Several test scenarios allow to strategically track undeleted data within HDFS to investigate vulnerability and risk assessment. Data consistency test, where undeleting was monitored by CheckerNode, were deleted automatically. During the secure deletion test, the default delete operation in HDFS was tracked using the autopsy tool. It was later verified that the overwrite technique successfully prevents recovery of deleted data. In addition, the new framework allows Hadoop to integrate more easily into an existing high-performance computing environment.

## 3.6 Paper IV: Adaptive Anomaly Detection in Cloud using Robust and Scalable Principal Component Analysis.

Submitted to Concurrency and Computation: Practice and Experience and is under review. It is an extended version of the paper published in the proceedings of 2016, 15th International Symposium of Parallel and Distributed Computing (ISPDC2016).

**Motivation:**

Cloud computing has become increasingly popular, which has led many individuals and organizations towards cloud storage systems. This move is motivated by benefits such as shared storage, computation and, transparent service among a massive number of users. However, Cloud computing system requires maintaining complex and large-scale system that inherent various runtime problems caused by hardware and software faults. urrent data centers consist of thousands of virtual machines, which require dynamic resource scheduling to operate both efficiently and cost-effectively. These data centers need to meet the varying demands for different resources like CPU and memory, and the scheduler must allocate or re-allocate these resources dynamically. This necessitates monitoring of resource utilization to detect any abnormal behavior.

**Method:**

The data from cloudwatch is collected and stored in OpenTSDB in a near real-time. Different instances of Amazon EC2 services with various geo-location are created, and thousand of a different user application are simulated on those instances. Amazon CloudWatch collect all the server metrics and send to OpenTSDB. The data is loaded into the RSPCA model which detect an unusual pattern and alert the user. In this paper, we propose an adaptive anomaly detection mechanism which investigates principal components of the performance metrics. It transforms the performance metrics into a low-rank matrix and then calculates the orthogonal distance using the Robust PCA algorithm. The proposed model updates itself recursively learning and adjusting the new threshold value to minimize reconstruction errors. A predefined threshold $\delta$ is used to determine an update operation. A matrix decomposition algorithm decomposes the input matrix X into a sum of three parts $X = L + S + E$ using Robust

Principal Component Pursuit. The low-rank matrix L is calculated using the SVD of X using a threshold as singular values.

**Results and Conclusion:**

This paper presents a real-time adaptive anomaly detection technique in cloud infrastructure. Different the performance metrics from Amazon CloudWatch logs is collected and normalized. Fast Fourier Transformation is used to detect a trend in input time series and convert the time series into a matrix. This paper presents a self-adaptive threshold approach is used. The threshold is updated during a learning phase. RPCA uses an efficient approach to decompose into low-rank representation using Spark as the underlying framework. The test was performed on different datasets and with different algorithms such as SVM, DBSCAN, and Increment PCA. It shows that RSPCA performs more accurately. RSPCA uses an efficient approach to decompose data into a low-rank representation using Spark. This model achieves an accuracy of 87.24% with real-time monitoring.

## 3.7 Paper V: AFFM:Auto Feature Engineering in FFM for Predictive Analytics.

This paper was published in the proceedings of 2015 IEEE, International Conference on Data Mining Workshop (ICDM).

**Motivation:**

Users are employing many devices to complete online tasks or browse the Internet. For example, a user wants to plan a holiday trip: and reads a travel blog, books flight tickets, searches for a preferred restaurant, or downloads a travel book. The user completes these tasks using a variety of devices. As users move across the devices to complete their online tasks, their identity becomes fragmented. The ads, recommendations, and messages are not always able to determine whether activities on a specific device are tied to the same or a different user. A model is required that can predict user behavior as they switch between their respective devices (websites/mobile apps).

**Method:**

Learning with the FFM model is improved by interacting with the property feature, which is updated once for each unique property. For large-scale datasets, where the properties of a field are massive, learning rates tend to be slow. Our approach solves this by dividing the learning rate by the number of features in the opposite field. This approach fits the training data with extreme precision. Another key challenge is to overwhelm the over-fit that occurs when small weights are adding up. In this case, Regularization is needed to reduce the impact of the lower weights.

**Result and Conclusion:**

This paper presents an FFM model with auto feature-engineering capability. The model can be used on any datasets with inbuilt options to calculate features in the field to reduce learning time. The prediction accuracy for user identification across devices is 86.48%.

## 3.8 Paper VI: Enrichment of Machine Learning based Activity Classification in Smart Homes using Ensemble Learning.

This paper was accepted to the proceedings of 2016, ACM 3rd International Workshop on Smart City Clouds: Technologies, Systems, and Applications (SCCTSA).

**Motivation:**

The data from various Internet-Of-Things (IOT) enabled sensors in smart homes provide an opportunity to develop predictive models that offer actionable insights in the form of preventive care to the resident of a smart home. Applying machine learning on such data allows the detection of patterns and activities that enable preventive care. Behavior patterns that lead to preventive care constitute a series of activities. Behavior classification can lead to preventive care in Aging-In-Place (AIP) by accurate labeling of the activities done inside the house. The accurate classification of activities can help detect whether the user is eating, sleeping, standing, or jumping.

**Method:**

An ensemble model is used with a combination of Random Forest, KNN, and XGBoost to classify activities in smart homes. In ensemble methods, different models are combined to produce a more precise and robust model. The most common approach uses either the average or weighted average of each model.

To evaluate the accuracy of activity recognition in smart homes, an ensemble model with a 2 level ensemble is created. In a level-2 step, 20 XGBoost runs are used to achieve better accuracy. The input data consists of 100 different features that combine the data of 3 sensors: accelerometer, video, and environment. Additionally, other features such as mean, standard deviation, minimum, median, and maximum values of the sensors are also extracted. The normalization of data between 0 and 1 is performed with a standardization technique. The output of the different models is then combined and fed as a feature to another level.

**Result and Conclusion:**

The accuracy of five different models was evaluated by applying them to the raw and feature extracted datasets. The best three models were then chosen to construct the ensemble model. The evaluation of the ensemble model was done using their Brier score as the outcome of activities for the probabilistic forecast. The dataset was then split into 80% for training and 20% for testing of the ensemble model. The overall Brier score of our ensemble model is 0.164. It can precisely classify rare activities such as $p_{squat}$, $a_{jump}$.

# Chapter 4

# Conclusion and Future Work

## 4.1 Conclusion

This dissertation proposes an extension to cloud-based Big Data framework that provide a scalable, efficient, and accurate approach to improve the performance of data center.

It provides an overview for performance management in cloud infrastructure. It provides highly flexible and scalable tools to monitor and analyze events in data center. A distributed scalable architecture is proposed by combining a NoSQL database with an analytic platform to predict failure. This was achieved by using a scalable machine-learning algorithm and a non-intrusive metrics collection mechanism. The analytics component leverages complex event processing (CEP) technology to monitor and improve the performance in datacenters.

Papers I, II, and IV, describe the idea of distributed processing for anomaly detection, failure prediction, and data storage. To achieve scalability, the Big Data tools such as MapReduce and Spark were used. Moreover, this dissertation also describe a scalable machine-learning algorithm to obtain faster execution times for the analysis of Big Data. The contribution of this dissertation is scalable anomaly detection and scalable failure prediction for Big Data cloud platforms. The techniques used in papers II and IV are based on scalable machine-learning algorithms and are used to create the described analytics framework. Moreover, this dissertation helps to improve the accuracy and performance of data center by predicting failure of computing nodes and jobs in Hadoop clusters as

well as detecting anomalous behavior of Big Data applications running on cloud infrastructure.

Finally, this dissertation provides a supporting tool that enables secure deletion of the data collected during the analytic step that is described in papers I, II, and IV. A secure deletion technique is introduced which enhances transparency to the user when dealing with a distributed file system in a massive infrastructure.

## 4.2 Future Work

An immediate extension of this work is to extend paper II, to provide a proactive measure in a real-time inside Hadoop ecosystem. Moreover, it can be extended to include the analysis of dataflow. Tracking data flow requires domain knowledge for interpreting the fields in log files. For example, $block_id$ in HDFS logs can be used to detect a failure in DataNode while accessing data that may affect other services running on Hadoop ecosystem such as; HBase region server.

Further, paper IV can also be extended to provide anomaly detection in Big Data framework. It can leverage the relationships between the behavior of various components to verify the potential factors that cause the system failure in data centers. Exposing relationships between different components helps in root cause analysis of a failure. Analysis of system metrics alone cannot provide detail analysis to the users for root cause of a failure because a fault in one component can be detected as unusual patterns for many metrics.

In future work, a security framework for providing data governance needs to be developed. Governed data are more reliable, secure, and ready to use, while data from an ungoverned has little value for analytics and business operations. Data governance provides a framework to set policies and implement controls designed to ensure that information remains accurate, consistent, and accessible.

# References


[1] *Apache Spark - Lightning-fast cluster computing*. URL: `https://spark.apache.org/`.

[2] Mai Abdrabo, Mohammed Elmogy, Ghada Eltaweel, and Sherif Barakat. "Enhancing Big Data Value Using Knowledge Discovery Techniques." In: (2016).

[3] IBM Blogs. *The 5 Vs of Big Data*. 2016. URL: `https://www.ibm.com/blogs/watson-health/the-5-vs-of-big-data/` (visited on 09/18/2016).

[4] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system." In: *arXiv preprint arXiv:1603.02754* (2016).

[5] Michael McKinnie. "Cloud Computing: TOE Adoption Factors By Service Model In Manufacturing." In: (2016).

[6] Tianqi Chen and Tong He. "xgboost: eXtreme Gradient Boosting." In: *R package version 0.4-2* (2015).

[7] Kevin Jackson, Cody Bunch, and Egle Sigler. *OpenStack cloud computing cookbook*. Packt Publishing Ltd, 2015.

[8] PWG NBD. *NIST Big Data interoperability framework*. 2015.

[9] Shen Yin and Okyay Kaynak. "Big Data for Modern Industry: Challenges and Trends [Point of View]." In: *Proceedings of the IEEE* 103.2 (2015), pp. 143–146.

[10] Min Chen, Shiwen Mao, and Yunhao Liu. "Big data: a survey." In: *Mobile Networks and Applications* 19.2 (2014), pp. 171–209.

[11] Dunren Che, Mejdl Safran, and Zhiyong Peng. "From big data to big data mining: challenges, issues, and opportunities." In: *International Conference on Database Systems for Advanced Applications*. Springer. 2013, pp. 1–15.

[12] Alfredo Cuzzocrea, Ladjel Bellatreche, and Il-Yeol Song. "Data warehousing and OLAP over big data: current challenges and future research directions." In: *Proceedings of the sixteenth international workshop on Data warehousing and OLAP*. ACM. 2013, pp. 67–70.

[13] Alfredo Cuzzocrea, Domenico Saccà, and Jeffrey D Ullman. "Big data: a research agenda." In: *Proceedings of the 17th International Database Engineering & Applications Symposium*. ACM. 2013, pp. 198–203.

[14] Nishant Garg. *Apache Kafka*. Packt Publishing Ltd, 2013.

[15] John Gantz and David Reinsel. "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east." In: *IDC iView: IDC Analyze the future* 2007 (2012), pp. 1–16.

[16] Saptarshi Guha, Ryan Hafen, Jeremiah Rounds, Jin Xia, Jianfu Li, Bowei Xi, and William S Cleveland. "Large complex data: divide and recombine (d&r) with rhipe." In: *Stat* 1.1 (2012), pp. 53–67.

[17] Alex Holmes. *Hadoop in practice*. Manning Publications Co., 2012.

[18] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[19] Steffen Rendle. "Factorization Machines with libFM." In: *ACM Trans. Intell. Syst. Technol.* 3.3 (May 2012), 57:1–57:22. ISSN: 2157-6904. DOI: 10.1145/2168752.2168771. URL: http://doi.acm.org/10.1145/2168752.2168771.

[20] Arvind Sathi. *Big data analytics: Disruptive technologies for changing the game*. Mc Press, 2012.

[21] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[22] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." In: *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association. 2012, pp. 2–2.

[23] Paul Zikopoulos, Krishnan Parasuraman, Thomas Deutsch, James Giles, David Corrigan, et al. *Harness the power of big data The IBM big data platform*. McGraw Hill Professional, 2012.

[24] Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright. "Robust principal component analysis?" In: *Journal of the ACM (JACM)* 58.3 (2011), p. 11.

[25] John Gantz and David Reinsel. "Extracting value from chaos." In: *IDC iview* 1142 (2011), pp. 1–12.

[26] Lars George. *HBase: the definitive guide*. " O'Reilly Media, Inc.", 2011.

[27] R. Kabacoff. *R in Action: Data Analysis and Graphics with R*. Manning Pubs Co Series. Manning, 2011. ISBN: 9781935182399. URL: http://books.google.no/books?id=qWpWRwAACAAJ.

[28] Ralph Kimball and Joe Caserta. *The Data WarehouseETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. John Wiley & Sons, 2011.

[29] Jay Kreps, Neha Narkhede, Jun Rao, et al. "Kafka: A distributed messaging system for log processing." In: 2011.

[30] Peter Mell and Timothy Grance. "The NIST definition of cloud computing (draft)." In: *NIST special publication* 800 (2011), p. 145.

[31] Christopher S Yoo. "Cloud computing: Architectural and policy implications." In: *Review of Industrial Organization* 38.4 (2011), pp. 405–421.

[32] Paul Zikopoulos, Chris Eaton, et al. *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.

[33] John Gantz and David Reinsel. "The digital universe decade-are you ready." In: *IDC iView* (2010).

[34] Lior Rokach. "Ensemble-based classifiers." In: *Artificial Intelligence Review* 33.1-2 (2010), pp. 1–39.

[35] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. "The hadoop distributed file system." In: *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE. 2010, pp. 1–10.

[36] Alon Halevy, Peter Norvig, and Fernando Pereira. "The unreasonable effectiveness of data." In: *IEEE Intelligent Systems* 24.2 (2009), pp. 8–12.

[37] Tony Hey, Stewart Tansley, Kristin M Tolle, et al. *The fourth paradigm: data-intensive scientific discovery*. Vol. 1. Microsoft research Redmond, WA, 2009.

[38] Justin Martineau and Tim Finin. "Delta TFIDF: An Improved Feature Space for Sentiment Analysis." In: *ICWSM* 9 (2009), p. 106.

[39] Leif E Peterson. "K-nearest neighbor." In: *Scholarpedia* 4.2 (2009), p. 1883.

[40] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. "Bigtable: A distributed storage system for structured data." In: *ACM Transactions on Computer Systems (TOCS)* 26.2 (2008), p. 4.

[41] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[42] Alex Smola and SVN Vishwanathan. "Introduction to machine learning." In: *Cambridge University, UK* 32 (2008), p. 34.

[43] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. "A break in the clouds: towards a cloud definition." In: *ACM SIGCOMM Computer Communication Review* 39.1 (2008), pp. 50–55.

[44] Robert M Bell and Yehuda Koren. "Lessons from the Netflix prize challenge." In: *ACM SIGKDD Explorations Newsletter* 9.2 (2007), pp. 75–79.

[45] Md Rafiul Hassan, Baikunth Nath, and Michael Kirley. "A fusion model of HMM, ANN and GA for stock market forecasting." In: *Expert Systems with Applications* 33.1 (2007), pp. 171–180.

[46] Roland Kwitt and Ulrich Hofmann. "Robust Methods for Unsupervised PCA-based Anomaly Detection." In: *Proc. of IEEE/IST WorNshop on Monitoring, AttacN Detection and Mitigation* (2006), pp. 1–3.

[47] Andrew Y Ng. "Feature selection, L 1 vs. L 2 regularization, and rotational invariance." In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 78.

[48] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system." In: *ACM SIGOPS operating systems review*. Vol. 37. 5. ACM. 2003, pp. 29–43.

[49] Jin Huang, Jingjing Lu, and Charles X Ling. "Comparing naive Bayes, decision trees, and SVM with AUC and accuracy." In: *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*. IEEE. 2003, pp. 553–556.

[50] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[51] Michele Banko and Eric Brill. "Scaling to very very large corpora for natural language disambiguation." In: *Proceedings of the 39th annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2001, pp. 26–33.

[52] Leo Breiman. "Random forests." In: *Machine learning* 45.1 (2001), pp. 5–32.

[53] Irina Rish. "An empirical study of the naive Bayes classifier." In: *IJCAI 2001 workshop on empirical methods in artificial intelligence*. Vol. 3. 22. IBM New York. 2001, pp. 41–46.

[54] Thomas G Dietterich. "Ensemble methods in machine learning." In: *International workshop on multiple classifier systems*. Springer. 2000, pp. 1–15.

[55] Joel Larocca Neto, Alexandre D Santos, Celso AA Kaestner, Neto Alexandre, D Santos, et al. "Document clustering and text summarization." In: (2000).

[56] Damien Brain, G Webb, D Richards, G Beydoun, A Hoffmann, and P Compton. "On the effect of data set size on bias and variance in classification learning." In: *Proceedings of the Fourth Australian Knowledge Acquisition Workshop, University of New South Wales*. 1999, pp. 117–128.

[57] Chuck Ballard, Dirk Herreman, Don Schau, Rhonda Bell, Eunsaeng Kim, and Ann Valencic. *Data modeling techniques for data warehousing*. IBM Corporation International Technical Support Organization, 1998.

[58] Marti A. Hearst, Susan T Dumais, Edgar Osman, John Platt, and Bernhard Scholkopf. "Support vector machines." In: *IEEE Intelligent Systems and their Applications* 13.4 (1998), pp. 18–28.

[59] Boris Mirkin. "Mathematical classification and clustering: From how to what and why." In: *Classification, data analysis, and data highways*. Springer, 1998, pp. 172–181.

[60] Huan Liu and Rudy Setiono. "Feature selection via discretization." In: *IEEE Transactions on knowledge and Data Engineering* 9.4 (1997), pp. 642–645.

[61] Arthur P Dempster, Nan M Laird, and Donald B Rubin. "Maximum likelihood from incomplete data via the EM algorithm." In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1977), pp. 1–38.

[62] Carl S Rudisill. "Derivatives of eigenvalues and eigenvectors for a general matrix." In: *AIAA Journal* 12.5 (1974), pp. 721–722.

[63] Leonard E Baum, JA Eagon, et al. "An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology." In: *Bull. Amer. Math. Soc* 73.3 (1967), pp. 360–363.

[64] Andrew J Viterbi. "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm." In: *Information Theory, IEEE Transactions on* 13.2 (1967), pp. 260–269.

[65] Evgeniui Borisovich Dynkin. "Markov processes." In: *Markov Processes*. Springer, 1965, pp. 77–104.

# Paper I:
# R2Time: a framework to analyse OpenTSDB timeseries data in HBase.

# R2Time: a framework to analyse OpenTSDB timeseries data in HBase.

**B. Agrawal**[1], **A. Chakravorty**[1], **C. Rong**[1], **T. Wiktorski**[1]

[1] Department of Electrical Engineering and Computer Science, University of Stavanger

**Abstract:**
In recent years, the amount of time series data generated in different domains have grown consistently. Analyzing large time-series datasets coming from sensor networks, power grids, stock exchanges, social networks and cloud monitoring logs at a massive scale is one of the biggest challenges that data scientists are facing. Big data storage and processing frameworks provides an environment to handle the volume, velocity and frequency attributes associated with time-series data. We propose an efficient and distributed computing framework - R2Time for processing such data in the Hadoop environment. It integrates R with a distributed time-series database (OpenTSDB) using a MapReduce programming framework (RHIPE). R2Time allows analysts to work on huge datasets from within a popular, well supported, and powerful analysis environment.

Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conf. on

# 1 Introduction

With the rapid, broad infuse of technology in various sectors of life, the amounts of data being generated are moving towards enormous proportions. Many real-world application domains as diverse as: epidemiology, biostatistics, engineering, geo-science, oceanography, neuroscience and computer science all generate continuous streams of successive data points in time, spaced at regular intervals. Such sequences of data are termed as time-series data. The frequency and quantity of generated time-series in various environments, makes traditional data-storage solutions inefficient in handling them. NoSQL big data solutions [13] are competent ways to store and process unstructured time-series data [3]. OpenTSDB [1] is an open source distributed and scalable time-series database for storing and indexing time-series metrics in HBase [2]. It is also necessary to be able to analyse and visualize the data. R [4] is a well-accepted open source environment for statistical explorations and data visualization.

In this paper, we present a framework called R2Time that uses a distributed computing model to analyse OpenTSDB time-series data stored in HBase from the R environment. R2Time allows R users to interact directly with OpenTSDB data and perform statistical analysis using the MapReduce [20] programming model. It provides methods for distributed handling of composite keys, allowing analysis of massive amount of time-series data in an efficient and scalable manner. The R2Time platform enables statistical and data mining operations using MapReduce programming model, through user defined map and reduce tasks for different analytical requirements. The practicality of this framework was verified through evaluation of basic statistical functions.

Section 2 gives an overview of the background. Section 3 introduces the design and implementation of the framework. Section 4 evaluates the performance and presents the results. Section 5 introduces the related work and section 6 concludes the paper.

# 2 Background

The various technologies that would facilitate creation of our data-intensive time-series processing framework are summarized in the following sections.

[1] OpenTSDB, [http://opentsdb.net/]

## 2.1 Hadoop

Hadoop [8] is an open-source framework for distributed storage and data-intensive processing first developed by Yahoo [2]. It has two core projects: Hadoop Distributed File System (HDFS) [17] and MapReduce [20] programming model. HDFS is a distributed file system that splits and stores data on nodes throughout a cluster, with a number of replicas. It provides an extremely reliable, fault tolerant, consistent, efficient and cost effective way to store a large amount of data. The MapReduce model consists of two key functions: Mapper and Reducer. The Mapper processes input data splits in parallel through different map tasks and sends sorted, shuffled outputs to the Reducers that in turn groups and processes them using reduce tasks for each group.

## 2.2 HBase

HBase [12] is a distributed column-oriented and NoSQL database that can be stored in distributed file systems such as HDFS and is modelled after Google's BigTable [19]. HBase provides real-time read/write random-access to very large datasets [8]. In HBase, a table is physically divided into many regions, which are in turn served by different Regional Servers. One of its biggest utility is of combining real-time HBase queries with batch MapReduce jobs, using HDFS as a shared storage platform.

## 2.3 OpenTSDB

OpenTSDB is an open source distributed and scalable time-series database, developed by StumbleUpon. It supports real time collection of data points from various sources. It is designed to handle terabytes of data with better performance for different monitoring needs. It stores, indexes and serves metrics at a large scale. Data is stored in HBase in two different tables: the *tsdb* table provides storage and query support over time-series data and the *tsdb-uid* table maintains an index of globally unique values for all metrics and tags.

## 2.4 R and RHIPE

R [14] is a language and environment from the open-source community widely used among statisticians and data scientists. It provides a wide

[2] Yahoo! Developer Network, (2014), Hadoop at Yahoo!, [http://developer.yahoo.com/hadoop/]

range of libraries for analysis and visualization. R Hadoop Integrated Processing Environment (RHIPE) is a library for integrating R with the Hadoop DFS. Using the MapReduce programming model it computes massive data sets across different nodes within a cluster. It works from the R environment using standard R programming idioms [4, 6].

# 3 Design And Implementation

To interpret, analyse and visualize large time-series data, it is necessary to have an efficient storage mechanism. Time-series data can be stored in different ways [5]. HBase provides multi-dimensional storage through usage of unique composite row keys. OpenTSDB makes use of such keys comprising of timestamps, metrics and tags to store its data. Our framework helps to analyse time-series data stored by OpenTSDB. R2Time figure 1 acts as a bridge between RHIPE and OpenTSDB.

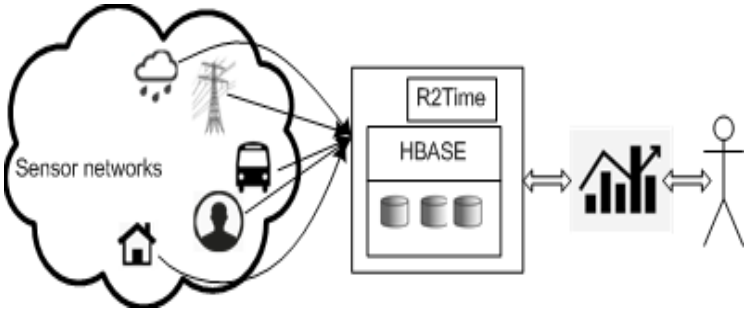



**Figure 1:** R2Time acts as a bridge between OpenTSDB and RHIPE

## 3.1 Row Key Design

OpenTSDB stores data in HBase using two tables *tsdb* and $tsdb - uid$ for storage and lookup respectively. The lookup table maintains an index of globally unique identifiers and values of all metrics and tags for data points collected by OpenTSDB. The tsdb table stores data using composite row keys comprising of metric id, base timestamp (hourly), tag-ids and tag-values. Each column is grouped under one column family consisting of column qualifiers representing delta elapse from base time, as shown in table 4.1.

An OpenTSDB row key design consists of a minimum 13 bytes with at least one tag. If there are multiple tags, they are appended at the end of

| | **T:0** | **T:10** | **T:20** | **T:30** | **timestamp** |
|---|---|---|---|---|---|
| row1 | 14.44 | 15.44 | 17.24 | 34.34 | 1368187868217 |
| row2 | 13.65 | 12.44 | 15.04 | 24.44 | 1368187868317 |
| row3 | 12.22 | 14.44 | 15.40 | 14.34 | 1368187868417 |
| ... | ... | ... | ... | ... | ... |

**Table 4.1:** OpenTSDB: 'tsdb' table sample

the row key. Timestamps in the row keys are rounded down to an hourly boundary. After every hour, values are created for a new row key for the metric and its tags. The composite row key format is shown in figure 2. With each additional tag, the row key size increases by 6 bytes.

The column qualifier is of 2 bytes. The first 12 bits are used to store an integer value, which is delta elapse (in seconds) from the base timestamp in the row key. The remaining 4 bits are of type flag, the first bit indicates whether the value is an integer or a floating point number. The remaining 3 bits are reserved for variable-length encoding. The measure/value of a data point is stored as the cell value and is reserved to 8 bytes. As example, a data point at timestamp "3456123232" will be stored with its base timestamp as "3456123000" in its row key and column qualifier as the delta value "232". R2Time retrieves data from the *tsdb* table for a given


**Figure 2:** OpenTSDB row key format

metric, tags and time ranges using a set of generated row keys. The row keys are generated in two phases. The first phase generates the start and end row key for the given metric and time range. Each generated row key consists of their respective metric and base timestamp. The combination of metric and base timestamp represents the first 7 bytes of the row key. These sequences of bytes are fixed, as a metric is created before being able to store data points. As mentioned earlier, for each metric and tag, a unique id (uid) is registered in the *tsdb-uid* table. A data point is stored in the *tsdb* table, with its metric uid retrieved from tsdb-uid table. It occupies the first 3 bytes of a row key, with next 4 bytes being the base timestamp calculated from given start/end time range.

In the second phase, filters are created for specified tags. The numbers of tags for data points vary and could range from single to multiple. For each tag, a regular expression of 6 bytes is created and appended together. An example of regular expression for filtering the specified tags is: Let binary representation of two tags be [0 0 1 0 0 2] and [0 0 2 0 0 4] respectively. A regular expression for filtering them would be: $\backslash Q \backslash 000 \backslash 000 \backslash 001 \backslash 000 \backslash 000 \backslash 002 \backslash E$ and $\backslash Q \backslash 000 \backslash 000 \backslash 002 \backslash 000 \backslash 000 \backslash 004 \backslash E$. Data stored in HBase are sorted on row keys, which makes performance with regular expression efficient. For a specified query and date-time range, data from HBase is read using tag filter regular expressions.

Once the row keys and filters are created, R2Time uses the scan object of HBase to point at the start and end data blocks. HBase provides several API for filtering of data. Using RowFilter, CompareFilter and RegexStringComparator the data between the start and end blocks are filtered with the generated regular expressions.

### 3.2 Data Retrieval

Any distributed operations for data stored in HBase are performed using MapReduce tasks. A MapReduce job is defined through R2Time, which creates the required row keys from the specified start/end timestamps, metric and tags. It also provides an extended *RecordReader* class called *RHHBaseReader* to read data from the *tsdb* table with its row keys as "keys" and delta columns as "values". In turn, it uses RHIPE to distribute the user-defined *map()* and *reduce()* tasks through the underlying MapReduce framework.

The overview of this framework is shown in figure 3 and a pseudo-code for a simple MapReduce program to count the number of data-points is provided in listing 4.1.

## 4 Result And Analysis

Our cluster is comprised of 11 nodes with CentOS linux distro, one node each for Namenode, Secondary Namenode, HBase Master, Job Tracker, and Zookeeper. The remaining 6 nodes act as Data Nodes, Regional Severs and Task Trackers. All nodes have an AMD Opteron(TM) 4180 six-core 2.6GHz processor, 16 GB of ECC DDR-2 RAM, 3x3 TeraBytes secondary storage and HP ProCurve 2650 switch. Experiments were conducted using OpenTSDB-1.0, Hadoop-0.20, Hbase-0.90 Apache releases. Our default

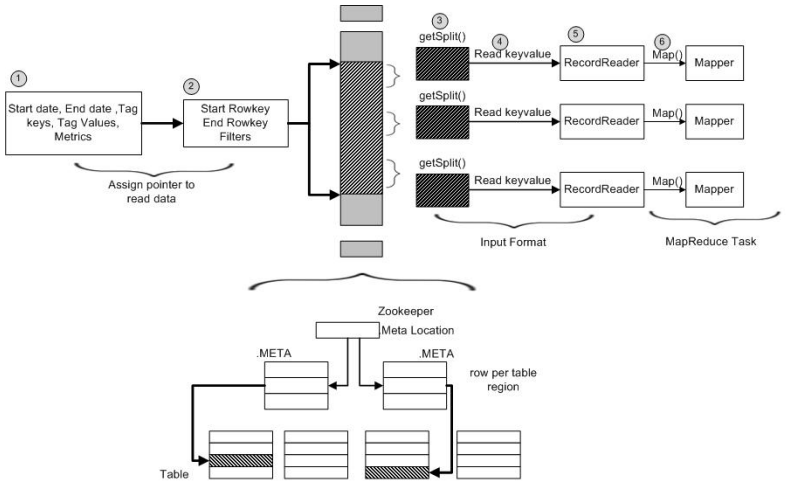



**Figure 3:** R2Time architecture. Step1: user supplies input parameters. Step2: R2Time converts input parameters into start/end row key and filters. Step3: R2Time defines its RecordReader, getSplit() to get numbers of split to run map tasks. Step4: RecordReader of R2Time to read <key,value> pairs. Step5: ready to run map and reduce task.

HDFS configuration had a block size of 64 MB and the replication factor 3.

**Listing 4.1:** Calculation for counting number of data-points using R2Time

```
# Load Rhipe and R2Time library
library(Rhipe)
library(r2time)
rhinit() #Initialize RHIPE framework
r2t.init() #Initialize R2Time framework

# Mapper function in R
map <- expression({
   cnt <- lapply(seq_along(map.keys), function(r) {
      len <- length(map.values[[r]]);
   })
   rhcollect(1, sum(unlist(cnt))) })
# Reduce function
reduce <- {
   count <- count + sum(sapply(reduce.values, function(x)
        sum(x)))
   },
   post={ rhcollect(reduce.key, count) })
# Submit MapReduce job to R2Time framework
r2t.job(startdate, enddate, metric, tagkeys, tagvalues,
        zkquorum, output, map, reduce)
```

## 4.1 Performance of Statistical Functions

The performance of different statistical functions was evaluated using the R2Time framework. Figure 4 shows the computation time for count, mean, linear regression and k-means for different data size.

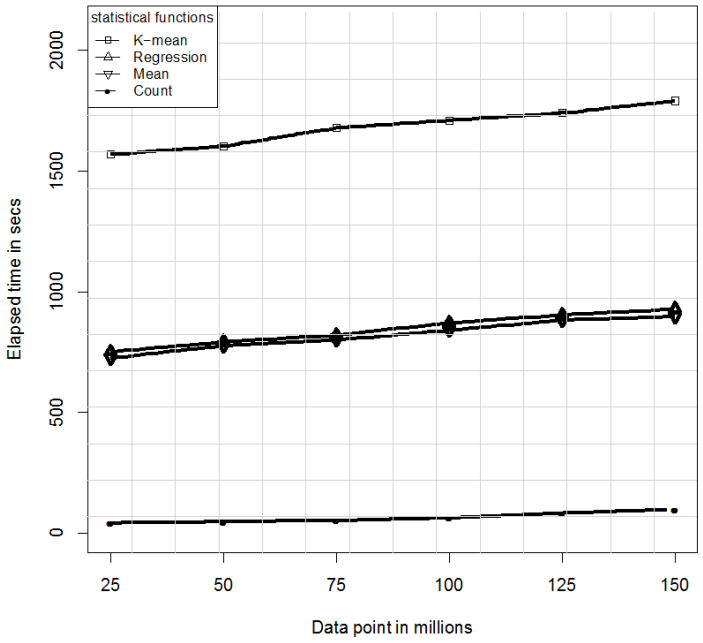



**Figure 4:** Performance based on different statistical functions

The count of the total number of data point present in the tsdb table took just over 50 seconds for processing 25 million data points, comprising of 250,000 rows with 100 columns per row. Each cell has a size of 8 bytes and an additional 2 bytes for its column qualifiers. Each row occupied disk space of 1013 bytes $[(100 \times 10bytes) + 13\ bytes]$ or $[(no.\_of\_cols. \times col.\_size) + row\_key\_size]$. The mean function is similar in nature but more expensive than the count function, due to its internal conversion of all data point values from serialized bytes to numeric int/float type. Moreover, the column-oriented format of the stored data requires it to be counted iteratively, over all columns in each row. Regression performs further matrix operations of multiplication and transposition,

thus requiring additional computations. Linear regression is calculated using basic implementation of $(X'X) - 1$ [24]. Regression is performed on each map function and later combined to obtain the final result in the reduce function. K-mean is the most expensive operation out of all the four functions, due to extra computation cost in calculation as well as storing/reading the initial centroids. Moreover, it also consists of computational overhead in terms of calculation of Euclidean distances [7, 22].

The aforementioned functions were chosen because they are constituted to be some of the core and basic functionalities for any analytics [23, 10]]. Based on these functions, further analysis can be performed in terms of trend analysis [9], seasonal variation [9] and clustering [21].

Due to the distributed nature of MapReduce, loads are equally distributed between different nodes. The methods used for distributed handling of composite keys allow the achievement of optimal performance, limited only by general algorithmic complexities and regular Hadoop overheads. Even for reasonably sized datasets (125 million), performance goals for distribution of calculations outweighed all the overheads. For larger datasets (>1 billion), the performance is expected to follow intrusive complexity of each algorithm with minor Hadoop overheads.

As observed from the evaluation, the functions appear to be supralinear in nature. In particular, this is due to algorithms being scaled using the MapReduce paradigm. As mentioned earlier, the framework supports composite keys in an optimal manner and does not introduce added complexities to any implementation of algorithms. The read latency in HBase is higher because of its log-structure based storage [1]. HBase flushes its memory tables to disk in separate files and needs to search each file for fragments of records [15]. Read latency in HBase can be optimized through compaction [12].

## 4.2 Scalability Test

With a larger number of nodes, the loads are further distributed across which leads to improve in performance as shown in figure 5. Moreover, R2Time provides improved execution time compared to OpenTSDB. R2time outperforms OpenTSDB by almost 50% for a cluster with 2 nodes. With a larger number of nodes in the cluster the performance tends to further improve. The test was performed on 75 million data points for reading data from HBase. R2Time is built using Hadoop and HBase Java API. It does not comprise performance despite allowing R users enrich

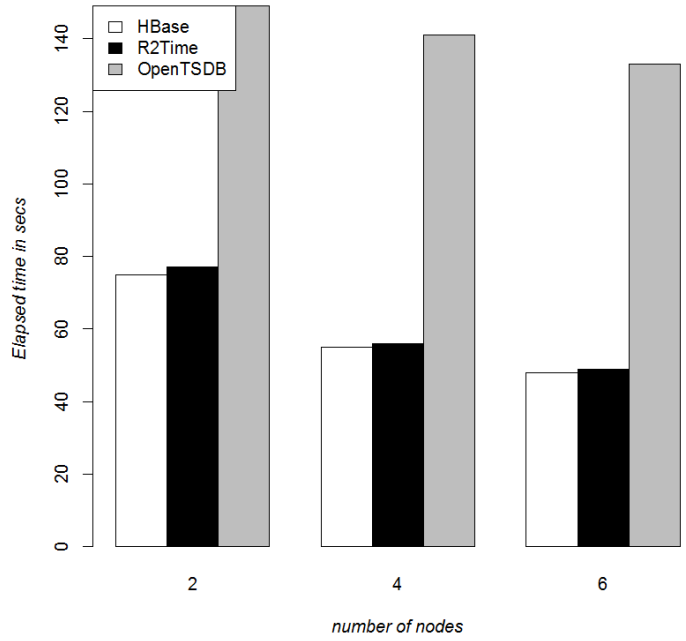



**Figure 5:** Runtime in seconds for reading data on different cluster sizes

statistical functionality.

## 4.3 Performance based on Scan Cache

*scanCache* and *setBatch* are HBase settings that control the number of rows and columns fetched from regional servers. scanCache and setBatch control transfer of rows and columns over network and retain them in memory for quick access. Increasing them to the maximum may not necessarily bring any improvements, due to increased memory and bandwidth overheads. Hadoop daemons use RPCs (Remote Procedure Calls) to talk to each other. Examples include: data nodes communicating with name nodes and task trackers communicating with job trackers. R2Time allows users to have custom setting for scan cache and batch sizes. The number of RPC calls needed for processing a particular amount of data, can be calculated as $RPC = \frac{Row \times ColsPerRow}{Min(ColsPerRow \times BatchSize)} \times \frac{1}{scanACache}$.

By default, $BatchSize$ is 1. From figure 6, it can be observed that

increasing BatchSize to the number of column per row, results in better execution time, whereas BatchSize>=ColsPerRow also lowers the number of RPCs, leading to lesser execution time. Lesser number of RPCs maintains less interconnection between regional servers. Due to network bandwidth and memory leakage, the ideal case of RPC=1 cannot be achieved [12].

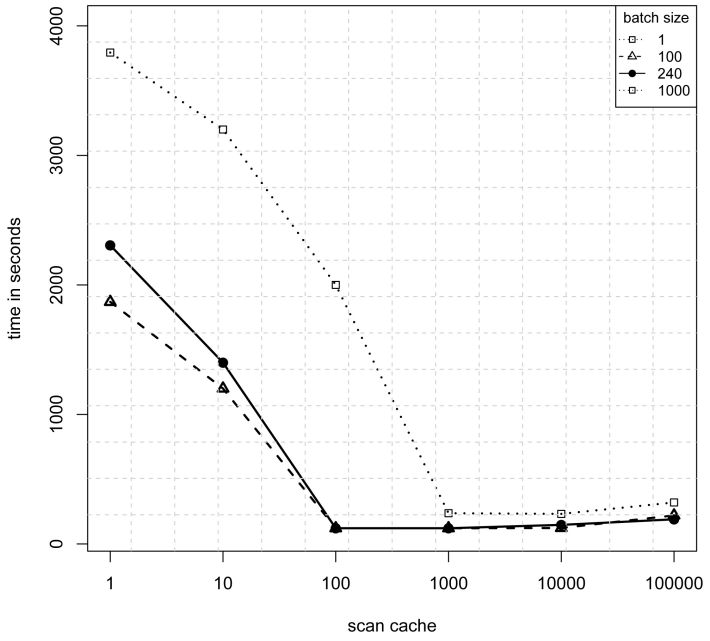



**Figure 6:** Performance based on scan cache

Case 1: When $BatchSize < ColsPerRow$, $BatchSize$ has inverse effect on $RPCs$. Here $K1 = Rows \times ColsPerRow$ and $RPC = \frac{K1}{BatchSize \times scanCache}$
Case 2: When $BatchSize >= ColsPerRow$, $BatchSize$ has no effect on RPCs. Here $K2 = Rows$ and $RPCs = \frac{K2}{scanCache}$

Figure 7, presents RPCs variation achieved with a constant BatchSize=ColsPerRow and varying ScanCache. The x-axis represents the number of RPCs calls and the y-axis represents the execution time for the statistical function row counter. It can be observed that RPCs is indirectly proportional to ScanCache. When RPCs called were 40,000, the execution time to complete the task was 46 seconds. Tuning the number of RPCs needed from 40,000 to 40 improved the performance by 65%.

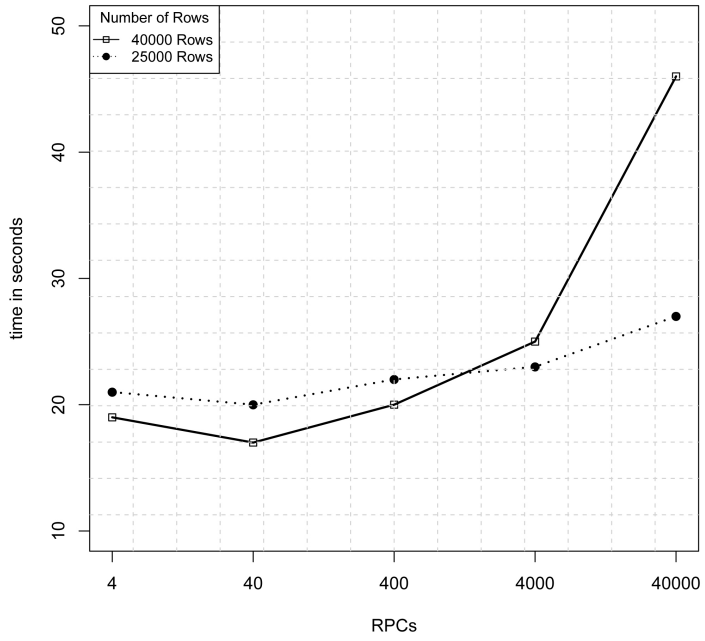



**Figure 7:** Performance based on RPCs

# 5 Related Work

Existing solutions in R such as astsa, tsa3, tseries and temp [18] provide analytical functionalities for time-series data but lacks operability over NoSQL based solutions. opentsdbr [3] and StatsD [4] uses HTTP/JSON APIs to query data from OpenTSDB and import them into R. However, these solutions are inherently centralized and non scalable, creating performance bottlenecks due to large requirements of memory, processing and network throughput. rbase [5] and rhbase [6] provides distributed functionalities for non time-series data in HBase. RHIPE [16] allows possibility to analyse

[3]D. Holstius, [https://github.com/holstius/opentsdbr]

[4]StatsD OpenTSDB publisher backend, [https://github.com/emurphy/statsd-opentsdb-backend]

[5]S. Guha, [https://github.com/saptarshiguha/rbase]

[6]Revolution Analyltics, [https://github.com/RevolutionAnalytics/RHadoop/ wiki/rhbase]

simple time-series data that are stored in HDFS, however has limitations in directly reading data from HBase. RHIPE can read data from HBase with the help of third-party libraries [16], but has shortcomings in reading composite keys [5] as used by OpenTSDB. RHadoop [7] provides similar functionalities as RHIPE, but has significantly poorer performance [11] due to distribution of required R libraries over the network to every node in the cluster participating in processing the data.

# 6 Conclusion

This paper presents our implementation and evaluation of framework R2Time that integrates R, OpenTSDB and RHIPE. It allows analysing OpenTSDB data stored in HBase using MapReduce programming model from within the R environment. OpenTSDB provides a well-accepted solution for storage of time-series data preserving its properties of time dependence, quantity, frequency and speed but lacks advance analytical functionalities. R is a powerful, open source statistical platform with a wide variety of analytical functionalities including time-series analysis. RHIPE enables an extension of R functionalities through MapReduce on data in HDFS and HBase, but remains incompatible with the data format of OpenTSDB. The R2Time framework fills this gap, integrating these solutions together enabling distributed processing of large time-series data across a Hadoop cluster.

# References

[1] Wei Tan, Sandeep Tata, Yuzhe Tang, and Liana L Fong. "Diff-Index: Differentiated Index in Distributed Log-Structured Data Stores." In: *EDBT*. 2014, pp. 700–711.

[2] Nick Dimiduk, Amandeep Khurana, Mark Henry Ryan, and Michael Stack. *HBase in action*. Manning Shelter Island, 2013.

[3] Luca Deri, Simone Mainardi, and Francesco Fusco. *tsdb: A compressed database for time series*. Springer, 2012.

[4] Saptarshi Guha, Ryan Hafen, Jeremiah Rounds, Jin Xia, Jianfu Li, Bowei Xi, and William S Cleveland. "Large complex data: divide and recombine (d&r) with rhipe." In: *Stat* 1.1 (2012), pp. 53–67.

[7] Revolution Analyltics, [https://github.com/RevolutionAnalytics/RHadoop/]

[5] Dan Han and Eleni Stroulia. "A three-dimensional data model in hbase for large time-series dataset analysis." In: *Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2012 IEEE 6th International Workshop on the*. IEEE. 2012, pp. 47–56.

[6] Alex Holmes. *Hadoop in practice*. Manning Publications Co., 2012.

[7] Richa Loohach and Kanwal Garg. "Effect of distance functions on simple k-means clustering algorithm." In: *International Journal of Computer Applications* 49.6 (2012).

[8] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[9] Theodore Wilbur Anderson. *The statistical analysis of time series*. Vol. 19. John Wiley & Sons, 2011.

[10] George EP Box, Gwilym M Jenkins, and Gregory C Reinsel. *Time series analysis: forecasting and control*. Vol. 734. John Wiley & Sons, 2011.

[11] Rui Maximo Esteves, Rui Pais, and Chunming Rong. "K-means clustering in the cloud–a Mahout test." In: *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*. IEEE. 2011, pp. 514–519.

[12] Lars George. *HBase: the definitive guide*. " O'Reilly Media, Inc.", 2011.

[13] Jing Han, Meina Song, and Junde Song. "A novel solution of distributed memory NoSQL database for cloud computing." In: *Computer and Information Science (ICIS), 2011 IEEE/ACIS 10th International Conference on*. IEEE. 2011, pp. 351–355.

[14] R. Kabacoff. *R in Action: Data Analysis and Graphics with R*. Manning Pubs Co Series. Manning, 2011. ISBN: 9781935182399. URL: `http://books.google.no/books?id=qWpWRwAACAAJ`.

[15] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. "Benchmarking cloud serving systems with YCSB." In: *Proceedings of the 1st ACM symposium on Cloud computing*. ACM. 2010, pp. 143–154.

[16] Saptarshi Guha. "Computing Environment for the Statistical Analysis of Large and Complex Data." AAI3449757. PhD thesis. West Lafayette, IN, USA, 2010. ISBN: 978-1-124-57405-9.

[17] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. "The hadoop distributed file system." In: *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE. 2010, pp. 1–10.

[18] Georgi N Boshnakov. "Time Series Analysis With Applications in R Series: Springer Texts in Statistics." In: *Journal of Time Series Analysis* 30.6 (2009), pp. 708–709.

[19] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. "Bigtable: A distributed storage system for structured data." In: *ACM Transactions on Computer Systems (TOCS)* 26.2 (2008), p. 4.

[20] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[21] Thomas Hill, Pawel Lewicki, and Pawel Lewicki. *Statistics: methods and applications: a comprehensive reference for science, industry, and data mining*. StatSoft, Inc., 2006.

[22] Tan Pang-Ning, Michael Steinbach, Vipin Kumar, et al. "Introduction to data mining." In: *Library of Congress*. 2006, p. 74.

[23] Stanley Wasserman and Katherine Faust. *Social network analysis: Methods and applications*. Vol. 8. Cambridge university press, 1994.

[24] Jeremy D Finn. *A general model for multivariate analysis*. Holt, Rinehart & Winston, 1974.

# Paper II: Analyzing and Predicting Failure in Hadoop Clusters Using Distributed Hidden Markov Model

# Analyzing and Predicting Failure in Hadoop Clusters Using Distributed Hidden Markov Model

**B. Agrawal[1], C. Rong[1], T. Wiktorski[1]**

[1] Department of Electrical Engineering and Computer Science, University of Stavanger

**Abstract:**

In this paper, we propose a novel approach to analyze and predict failures in Hadoop cluster. We enumerate several key challenges that hinder failure prediction in such systems: heterogeneity of the system, hidden complexity, time limitation and scalability. At first, the clustering approach is applied to group similar error sequences, which makes training of the model effectual. Subsequently Hidden Markov Models (HMMs) are used to predict failure, using the MapReduce programming framework. The effectiveness of the failure prediction algorithm is measured by precision, recall and accuracy metrics. Our algorithm can predict failure with an accuracy of 91% with 2 days in advance using 87% of data as training sets. Although the model presented in this paper focuses on Hadoop clusters, the model can be generalized in other cloud computing frameworks as well.

# 1 Introduction

The cluster system is quite commonly used for high performance in cloud computing. As cloud computing clusters grow in size, failure detection and prediction become increasingly challenging [24]. The root causes of failure in such a large system can be due to the software, hardware, operations, power failure and infrastructure that support software distribution [26]. The cluster system dealing with a massive amount of data needs to be monitored and maintained efficiently and economically. There have been many relevant studies on predicting hardware failures in general cloud systems, but few on predicting failures in cloud computing frameworks such as Hadoop [10]. Hadoop is an open-source framework for distributed storage and data-intensive processing, first developed by Yahoo. Hadoop provides an extremely reliable, fault-tolerant, consistent, efficient and cost-effective way of storing a large amount of data. Failure in storing and reading data from the large cluster is difficult to detect by human eyes. All the events and activities are logged into their respective application log files. Logs provide information about performance issues, application functions, intrusion, attack attempts, failures, etc. Most of the applications maintain their own logs. Similarly, HDFS system consists of DataNode and NameNode logs. The logs produced by NameNode, secondary NameNode and DataNode have their individual format and content.

The prime objective of the Hadoop cluster is to maximize the job processing performance using data-intensive computing. Hadoop cluster normally consists of several nodes and can execute many tasks concurrently. The job performance is determined by the job execution time. The execution time of a job is an important metric for analyzing the performance of job processing in the Hadoop cluster [11]. As Hadoop is a fault-tolerant system if the nodes fail, then the node is removed from the cluster in the middle of the execution and the failed tasks are re-executed on other active nodes. However, this assumption is not realistic because the master node can crash. Many researchers reported that the master node crash is a single point of failure and needs to be handled [4] [17]. The failure of the data node results in higher job execution time, as the job needs to re-execute in another node. Failure nodes are removed from the cluster so that the performance of the cluster improves. Prediction methods operate mostly on continuously available measures, such as memory utilization, logging or workload, to identify error pattern. Our analysis in this paper is mostly only on time of occurrence of different types of error events that

ultimately cause failure. This might also help in root cause analysis [15] for automatic triggering of preventive actions against failures.

We used Hidden Markov Models (HMMs) [2] to learn the characteristics of log messages and use them to predict failures. HMMs have been successfully used in speech, handwriting, gesture recognition, and also in some machine failure prediction. HMM is well suited to our approach as we have observations of the error messages, but no knowledge about the failure of the system, which is "hidden". Our model is based on a stochastic process with a failure probability of the previous state. As faults are unknown and cannot be measured, they produce error messages on their detection (i.e. present in log files).

Our prediction model is divided into four main parts; First, identifying error sequences and differentiating types of error from the log files. Second, using the clustering algorithms [1] like K-means [27]. Third, training the model. Given the labeled training data, HMM method is used to evaluate maximum likelihood sequence that is used to update the parameters of our model. Last, predicting failure of the system based on the observation of an error sequences. The idea of our approach is to predict failures by analyzing the pattern of error events that imitates failure. Experimental results for this method can predict failure with 91% accuracy for two days in advance (prediction time). It also shows that our approach can compute on the massive amount of datasets. Ultimately, our approach can be used to improve the performance and reliability of the Hadoop cluster.

## 1.1 Related work

A significant number of studies have been done on the performance evaluation and failure diagnosis of systems using log analysis. However, most of the prediction methods focus only on the system logs, but not on the application logs. Many studies have been done on predicting hardware failure in the cluster. For example, studies in [16], [26] and [7] provide a proactive method of predicting failure in the cluster, based on system logs. These methods provide failure in hardware level but fail to provide failure of a node in the Hadoop clusters.

Konwinski et al. [19] used X-trace [14] to monitor and improve Hadoop jobs. X-trace allows path-based tracing on Hadoop nodes. Additionally, by using X-trace in conjunction with other machine-learning algorithms, they were able to detect failure with high accuracy. Similarly, SALSA [20] is another tool in which system logs are analyzed to detect failure using distributed comparison algorithm. Also, it shows information on

how a single node failure can affect the overall performance of the Hadoop cluster. All of these papers present failure detection algorithm in the Hadoop cluster but lacks prediction algorithm.

Fulp et al. [18] demonstrated failure prediction in the hard disk using SVMs (Support Vector Machines) with an accuracy of 73% with two days in advance. On the other hand, Liang et al. [24] uses RAS event logs to predict failure in IBM BlueGene/L. They compared their results with Support Vector Machines (SVMs), a traditional Nearest Neighbor method, a customized Nearest Neighbor method and a rule-based classifier, and found that all were out-performed by the customized Nearest Neighbor method. However, these all provide failure prediction algorithm in the different areas, but still lacks to provide precise model.

Hidden Markov Models have been used in pattern recognition tasks such as handwriting detection [12], gene sequence analysis [30] [23], gesture recognition [29], language processing [28] [31], hard drive failure [9] or machine failure [16]. HMM is a widely used model due to it's flexibility, simplicity, and adaptivity. Indeed, as mentioned earlier, Hadoop log data have challenging characteristics, which thus require expert knowledge to transform data into an appropriate form.

## 1.2 Our Contribution

We proposed a novel algorithm for failure prediction algorithm using MapReduce programming framework, thus achieving better scalability and better failure prediction probability. The proposed model is based on distributed HMM through MapReduce framework in a cloud-computing environment. Through this paper, we also present our idea to increase the performance of the Hadoop Cluster by predicting failure. The accuracy of our model is evaluated using performance metrics (precision, recall, F-measure).

## 1.3 Paper Structure

Section II provides an overview of the background. Section III introduces the design and approach of our analysis. Section IV evaluates our algorithm and presents the results. Section V concludes the paper.

# 2 Background

## 2.1 Hadoop

Hadoop [10] is an open-source framework for distributed storage and data-intensive processing, first developed by Yahoo. It has two core projects: Hadoop Distributed File System (HDFS) and MapReduce [25] programming model. HDFS is a distributed file system that splits and stores data on nodes throughout a cluster, with a number of replicas. It provides an extremely reliable, fault-tolerant, consistent, efficient and cost-effective way to store a large amount of data. The MapReduce model consists of two key functions: Mapper and Reducer. The Mapper processes input data splits in parallel through different map tasks and sends sorted, shuffled outputs to the Reducers that in turn groups and processes them using reduce tasks for each group.

## 2.2 Hidden Markov Models

HMM [33] is based on Markov Models, a tool for representing probability distributions over sequences of observations. It is a probabilistic model, in which system is assumed to be a Markov process [35] (memoryless process) with hidden (unobserved) states. HMM consists of unobserved states, and each state is not directly visible, but output and dependent on the state are visible. It has a set of states each of which has a number of transitions and emissions state probability as shown in Figure 2.7. HMM typically used to solve three types of problem: detection or diagnostic problem, decoding problem and learning problem. Forward-backward algorithm [21] solves diagnostic problem. Similarly, Viterbi algorithm [34] solves decoding problems and Baum-Welch algorithm [32] solves learning problem.

# 3 Approach

In this section, we describe how the useful information from different logs are extracted and the use of HMMs to predict failure from those log files.

The proposed method deals with all the log files associated with Hadoop cluster (HDFS): DataNode and NameNode logs. The log files are collected from the different nodes associated with the cluster. The logs generated from 11-node clusters are stored in HDFS system using Apache Flume collector [13]. The log files contain all unwanted and wanted information that makes it difficult for the human to read. For this reason, pre-processing

of logs is needed before storing to HDFS system. In the pre-processing steps, all the log messages are extracted and unwanted and noisy messages are removed. The stored data is further analyzed using HMM model. Failure prediction algorithm is used to detect a failure and ignore defective node before running any task.

HDFS system consists of NameNode and DataNode. NameNode is the master node on which job tracker runs. It consists of the metadata (information about data blocks stored in DataNodes - the location, size of the file, etc.). It maintains and manages the data blocks, which are present on the DataNodes. The DataNode is a place where actual data is stored. The DataNode runs three main types of daemon: Read Block, Write Block, and Write-Replicated Block. NameNode and DataNode maintain their own logging format. Each node records events/activities related to reading, writing, and replication of HDFS data blocks. Each log is stored on the local file-system of the executing daemon. Our analysis is based on some important insights about the information available through Hadoop logs. Block ID in DataNode log provides a unique identifier, which represents the HDFS blocks that consist of raw bytes of the data file.

Before using log messages to build the model, we structured and appended all the log files into systematized forms. Four steps are involved in our approach: pre-processing, clustering, training, and predicting as shown in figure 1. In the first step, all useful information, such as timestamp, error status, error type, node ID and user ID, are extracted, and new log template is created. Since different logs reside on the local disk in different nodes, it is necessary to collect and attach all the log information into a one-log template. In the second step, we use the clustering algorithm to differentiate various types of errors. With the clustering technique, real error types that propagate to failure are recognized. And the third and fourth step is the training and prediction algorithm using HMM model, which is discussed in detail below.

We adopted Hidden Markov models (HMMs) for this approach. HMM applies machine-learning techniques to identify whether a given sequence of the message is failure-prone or not. HMM models parameters can be adjusted to provide better prediction. The sequences of an error event are fed into the model as an input. Each error event consists of a timestamp, error ID and error type, which determine the types of error. Failure and non-failure information are extracted from error sequences to create a transition matrix. HMM is characterized by the following modules: hidden states $X = \{x_1, x_2, x_3\}$, observations state $Y = \{y_1, y_2, y_3\}$, transition

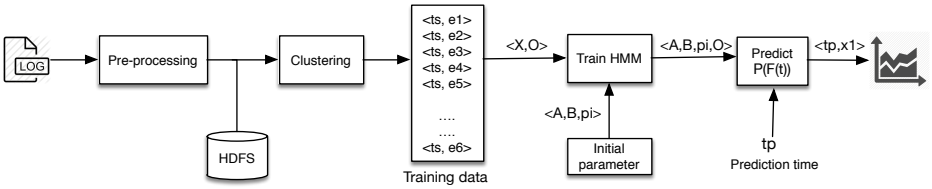



**Figure 1:** Workflow of failure prediction.

probabilities A $= a_{ij} = \{P[q_{t+1} = x_j | q_t = x_j]\}$ and emission probabilities B $= b_{ij}$. HMM ($\lambda$) is denoted as

$$\lambda = (\pi, A, B) \tag{4.1}$$

Where, A is the transition matrix whose elements give the probability of transitioning from one state to another, B is the emission matrix giving $b_j(Y_t)$ the probability of observing $Y_t$. $\pi$ is initial state transition probability matrix.

The observation symbols $O_1 = \{e_1, e_2, e_3, e_4, e_5, e_6\}$ are referred to error events of the system, and failures are represented as hidden state of HMM as shown in figure 2. Error patterns are used as training set for HMM model if the model transits to a failure state each time a failure occurs in the training data. Two steps are necessary for obtaining training sequences for the HMM. The first step involves the transformation of error types

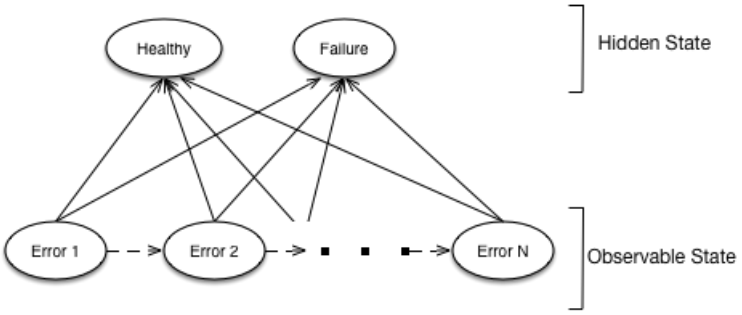



**Figure 2:** Mapping Failure and errors to Hidden Markov Model.

into a special error symbol in order to classify different types of error. Error event timestamps and error categories form an *error sequence* (event-driven sequence). HMM applies machine-learning techniques in order to identify characteristic properties of an error sequence. It also helps to

detect whether a given error sequence is failure-prone or not. Moreover, we trained the model using past error sequence. The model adjusts its parameters based on those records. The trained model is then applied to the new error sequences to predict failure. Such an approach in machine learning is known as "*supervised learning*". To extract error sequence, timestamp and error ID are extracted in preprocessing step as mentioned earlier. Let "$e$" represent different types of error in the log files. The series of messages that appear in "e" form a time-series representation of events that occurred. In this paper, all categories of "$e$" are identify using *k-means clustering technique* [27]. Six different types of error are distinguished from the given log files and the set $e$ would be $(e_1, e_2, e_3, e_4, e_5, e_6)$. This error set is known as *error sequence* or observation for our model.

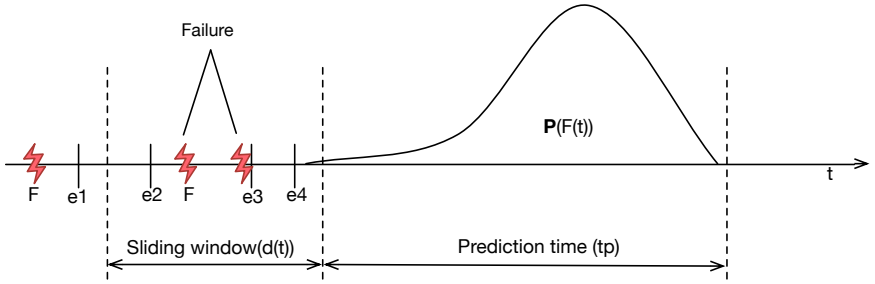



**Figure 3:** Failure prediction, where e1, e2, e3 and e4 are error sequences, and F is the failure in the system.

In the next step, the model is defined using error sequences. Error sequence consists of failure and non-failure information that has occurred within a sliding window of length $\Delta t$ as shown in figure 3. F is the failure in the system and $e_1, e_2, e_3, e_4$ represent the error events in the log files. Failure $t_p$ is predicted based on $\Delta t$ error sequence. HMM models are trained using error sequences. The main goal of training is that failure characteristics are generated from the error sequences. The non-failure model stays rather remains unspecific. Once the models are trained, new upcoming failure is evaluated from the error sequences. To predict upcoming failure, sequence likelihood is computed from HMM models.

In this paper, there is a sequence of log data over timestamp, which we needed to train our HMM model using a set of the sequence of log output as observation O = (info, warn, error, fatal). N= 4 for HMM model is denoting the stages in time that are allowed in different transitions in the HMM training. Each error sequence may result in failure-prone or

non-failure system. The Failure-prone system has a similar pattern of errors, which result in failure. The probability of log data is computed using Forward-Backward Algorithm as proposed in [32].

**Training the model:** First, from the log template, sequence of an error message is extracted to form an output sequence composed of 1,2,3,4,5 and 6's, one number for each time by rounding the timestamp of logs of a job to the nearest integer. Thus, the state transition in the HMM takes place every time step until the absorption state is reached. The choice of time step determines the speed of learning and its accuracy. A log sequence of a job always starts from the state $x_1$ and ends at $x_2$, and the initial probabilities for $\pi$ are fixed to be 0.5. With the output sequence as described, we compute the most likely hidden state transition sequence and the model parameters $\lambda = (A, B, \pi)$.

We have adapted Expectation-Maximization algorithm for training purpose. During training, the HMM parameters $\pi$, A, B are optimized. These parameters are maximized in order to maximize sequence likelihood. For initial steps, a number of states, a number of observations, transition probability, and emission probability are pre-specified. In this experiment, initial parameters are calculated from the past observation, such that the model can predict accurately from the initial phase. As the training of model progress, the parameter value gets closer to the actual value. Training in HMM is done using Expectation-Maximization algorithm, where backward variable $\beta$ and forward variable $\alpha$ are evaluated. This algorithm helps to maximize the model parameters based on maximum likelihood. If the model started randomly from a pre-specified HMM parameter, it would take several iterations to get superior parameters, which best fit, the model for prediction. The goal of training datasets is to adjust the HMM parameters in such a way that error sequences are best represented and that the model transits to a failure state each time a failure occurs in the training datasets.

There are a few existing methods such as; Baum-Welch algorithm and gradient-descent techniques that use iterative procedures to get the locally maximized likelihood. However, this iterative procedure might be significantly slow if the observed sequence is large. In this paper, we proposed a slightly different algorithm to train data, which is significantly faster than the traditional method. The idea is to formulate the probability of the observation sequence $O_t, O_{t+1}$ pairs and then to use Expectation-Maximization algorithm to learn for this model $\lambda$.

In order to train the model, there is a need to find the repetitive error

**Algorithm 1** Failure State Prediction Algorithm

Initialized $O = \{o_1, o_2, ..o_6\}$ ▷ different types of error as observation
$S = \{Healthy, Failure\}$ ▷ two hidden state
$m = 2$ ▷ m is number of hidden state
$n = 6$ ▷ n is number of different types of errors
Initialized $A_{ij}, B_{ij}$ ▷ emission matrix $B_{ij}$ stores the probability of observable sequences $o_j$ from state $s_i$ ▷ transition matrix $A_{ij}$ stores the transition probability of transiting from state $s_i$ to state $s_j$
Initialized $\Pi$ ▷ an array of initial probabilities
$Y = \{y_1, y_2...y_k\}$ ▷ an error sequence of observation
**Map:**
Initialized $StatePathProb$
Update $A_{ij}, B_{ij}$
$PathProb = StatePathProb * B_{ij}$
**for** each state $s_j$ **do**
    $StatePathProb[j, i] \leftarrow \max_k (StatePathProb[k, i-1] \cdot A_{kj} \cdot B_{jy_i})$
    $PathProb[j, i] \leftarrow \arg\max_k (PathProb[k, i-1] \cdot A_{kj} \cdot B_{jy_i})$
**end for**
$z_i \leftarrow \arg\max_k (StatePathProb)$
$x_i \leftarrow S_i$
**for** $i \leftarrow T-1, ..., 1$ **do** ▷ T is length of observable sequence
    $z_i \leftarrow PathProb[z_i, i]$
    $x_i \leftarrow zi$
**end for**
$emit(timestamp, x)$

sequence in the data. To do so first, we need to compute the likelihood of raw data in the desired range. This problem is computed using EM algorithm. The EM consists of two steps: an expectation (E) step, which creates a function for calculating log-likelihood from the current estimate, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood calculated on the E step. This EM algorithm is carried on a map and reduce task.

After getting all the parameter of the model ($\lambda$), the prediction algorithm is used to get all the hidden states that are calculated using maximum likelihood of the past sequences.

**Prediction**: Failure is the hidden layer in our HMM model. The

Hidden state is calculated using Viterbi algorithm. There is a sequence of observations $0 = O_1, O_2....O_n$ with given model $\lambda = (A, B, \pi)$. The aim of Viterbi algorithm is to find optimal state sequence for the underlying Markov chain, and thus, reveal the hidden part of the HMM $\lambda$. The final goal of Viterbi is to calculate the sequence of states (i.e $S = \{S_1, S_2, ...S_n\}$), such that

$$S = argmax_s P(S; O, \lambda) \tag{4.2}$$

Viterbi algorithm returns an optimal state sequence of S. At each step t, the algorithm allow S to retain all optimal paths that finish at the N states. At t+1, the N optimal paths get updated and S continues to grow in this manner. Figure 4 shows details architecture of Viterbi algorithm implementation. The goal is to predict hidden state from the given observation $0 = \{O_1, O_2....O_n\}$ . No reducer is used and on each mapper, a local maximum is calculated and state path based on maxima are observed. Two states defined in algorithm 1: healthy and failure are determined based on error sequence.

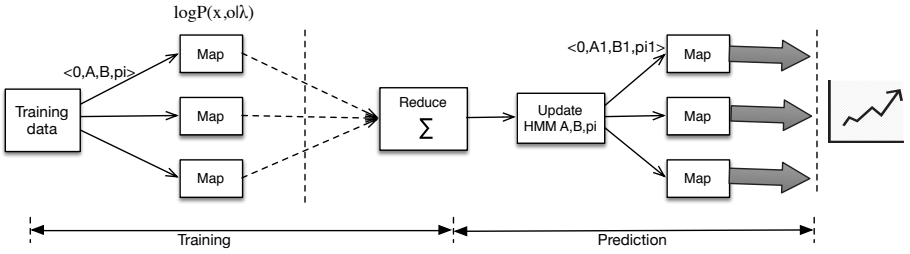



**Figure 4:** Architecture of an algorithm predicting failure state using MapReduce programming framework.

# 4 Result

*Setup*: Our cluster is comprised of 11 nodes with CentOS Linux distro, one node each for Namenode, Secondary Namenode, HBase Master, Job Tracker, and Zookeeper. The remaining 6 nodes act as Data Nodes, Regional Severs, and Task Trackers. All nodes have an AMD Opteron(TM) 4180 six-core 2.6GHz processor, 16 GB of ECC DDR-2 RAM, 3x3 TeraBytes secondary storage and HP ProCurve 2650 switch. Experiments were conducted using RHIPE, Hadoop-0.20, Hbase-0.90 Apache releases.

Our default HDFS configuration had a block size of 64 MB and the replication factor of 3.

The prediction techniques presented in this paper have been applied to the data generated while performing operations in the Hadoop Cluster. With 1 month of Hadoop log data, we trained HMM model using sliding windows varying from 1 to 2 hours in length. A Large amount of Hadoop log data was generated using SWIM [8], a tool to generate arbitrary Hadoop jobs that emulate the behaviors of true production jobs of Facebook. We used AnarchyApe [3] to create different types of failure scenarios in Hadoop cluster. AnarchyApe is an open-source project, created by Yahoo! developed to inject failures in Hadoop cluster [5].

## 4.1 Types of error

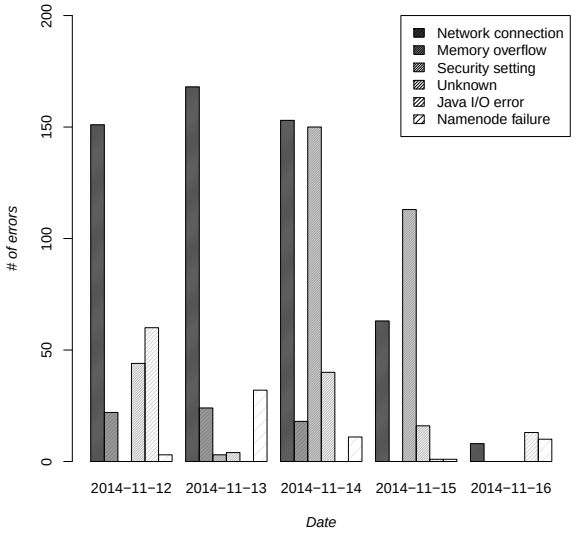



**Figure 5:** Different types of error in Hadoop cluster during 5 days interval.

Errors such as operational, software, configuration, resource and hardware are present in Hadoop cluster. In this analysis, hardware failure was not considered. Operational, software and resource errors are taken into consideration to detect a failure in the software level of Hadoop cluster. Operation errors include missing operations and incorrect operations, and they are easily identified in log messages by operations: $HDFS\_READ$,

$HDFS\_WRITE$, etc. Resource errors (memory overflow, node failure) refer to resource unavailability occurring in the computing environment, while software errors (Java I/O errors, unexceptional) refer to software bugs and incompatibility. These types of error are detected on different DataNodes. Log messages are classified into six different types of error: Network connection, Memory overflow, Security setting, Unknown, Java I/O error, NameNode failure as shown in Figure 5. Errors like network connection and security setting are most occurring errors in the Hadoop cluster. In the pre-processing step, each log message is tagged with certain error ID and using the clustering algorithm like k-means, different types of error are analyzed.

## 4.2 Predicting failure state in Hadoop cluster.

Different types of error are used as input observation in our model i.e.$O = \{O_1, O_2....O_n\}$. $O_1$ to $O_6$ are error sequences, and O7 is a non-error sequence from the log template. This observation is used in the model $\lambda = (\pi, A, B)$ to detect $S = \{Healthy, Failure\}$. Based on the error sequence in the HMM model, with the help of Viterbi algorithm, hidden state sequences are generated which are shown in gray and red line in figure 6. The red line indicates failure state and gray indicates non-failure state. Similarly, black and gray line shows actual failure state. Based on the probability of the previous state and HMM parameters, the failure, and non-failure states are determined. Error sequences are predicted using EM algorithm, and based on predicted error sequences, hidden states (failure or non-failure) are predicted. Error in prediction is calculated by differencing the actual and predicted value as shown in the graph. At first step, our model is trained from the previous record. As the time passes, the model gets more accurate. The training of the model depends solely on the initial parameter. For this example, initial parameters are calculated from the past record. That is why this model has similar behavior from the initial point, but not an accurate prediction.

The models' ability to predict failure precisely is evaluated by four metrics: precision, recall, false positive rate and F-measure. These metrics are frequently used for prediction evaluation and have been used in many relevant researches as in e.g. [22]. Four metrics are defined in table 4.1:

From the above observation, we used log entries of 800 hours out of which; first 650 hours is used for training and last 150 hours is used for prediction. In total, we have 24000 observations for 150 hours of prediction time. Different cases for prediction is shown in table 4.2. The accuracy of

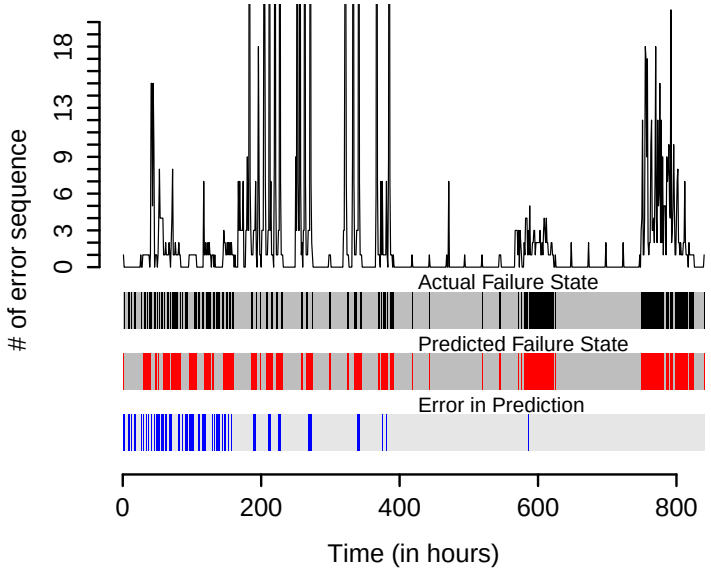



**Figure 6:** Using HMM to predict failure and normal state. Error sequences are the observation or observable state of HMM and gray-black bar indicate the hidden state, gray line indicate non-failure state and black indicate failure state. Similarly, red line indicates predicted failure state. And blue line indicates prediction error.

| Metric | Definition | Calculation |
|---|---|---|
| Precision | $p = \frac{TP}{TP+FP}$ | 0.93 |
| Recall | $r = \frac{TP}{TP+FN}$ | 0.91 |
| False positive rate | $fpr = \frac{FP}{FP+TN}$ | 0.091 |
| F-measure | $F = \frac{2pr}{p+r}$ | 0.92 |
| Accuracy | $accuracy = \frac{TP+TN}{TP+FP+FN+TN}$ | 0.91 |

**Table 4.1:** Definition of metrics.

the model is 91.25 %$(9000 + 12900/24000)$. And the precision and recall are 0.93 and 0.91 respectively.

Higher precision ensures fewer false positive errors, while a model with high recall ensures lesser false negative errors. Ideal failure prediction

| Predicted failure state | Predicted non-failure state |
|---|---|
| TN: 9000 | FP: 900 |
| FN: 1200 | TP: 12900 |

**Table 4.2:** Observation for different cases.

model would achieve higher precision and recall value, i.e. precision = recall = 1. However, both high recall and precision are difficult to obtain at the same time. They are often inversely proportional to each other. Improving recall in most cases lowers the precision and vice-versa. F-measure ensures that the model is accurate or not, and makes sure both precision and recall are reasonably high. In HMM method, a threshold value allows the control of true positive rate and false positive rate. This is one of the big advantages of HMM, method over other techniques.

## 4.3 Scalability

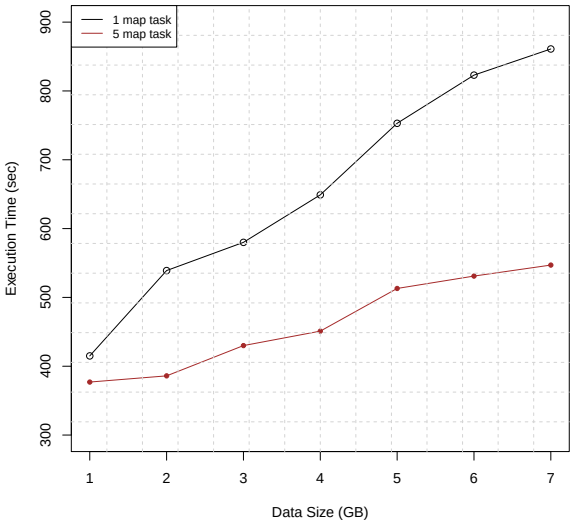



**Figure 7:** Scalability with increases in data size

To test the implementation of MapReduce HMM model in the cluster,

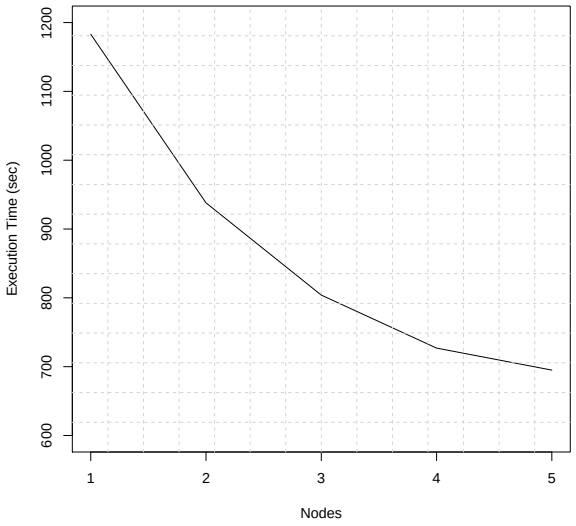



**Figure 8:** Scalability with increases in cluster size.

we fixed the number of nodes in the cluster to be 6. And, then tested HMM by varying the number of data size from 1 GB (85 million error sequences) to 7 GB (571 million error sequences). Figure 7 demonstrates the scalability of the algorithm. It shows a steady increase in execution time with the growth in data size. The brown and black lines in the graph represent the parallel and sequential execution of map task. It is obvious that parallel execution outperforms.

In the figure 8, we set the number of nodes participating in the MapReduce calculations to 1, 2, 3, 4, or 5. The algorithm was then tested on the dataset of size 5 GB (138 million error sequences). The experimental result shows that the execution time improves with an increase in the number of nodes. This increase can significantly improve the system processing capacity for the same scale of data. By adding more nodes to the system, the performance improves and computation can be distributed across the nodes [6]. We conclude that this performance improvement would be even more noticeable with large-scale data involving many more nodes. The typical scalability behavior would illustrate a linear line in the graph. However, it is impossible to realize this ideal behavior due to many factors such as network overheads.

# 5 Conclusion

As failures in cluster systems are more prevalent, the ability to predict failures is becoming a critical need. To address this need, we collected Hadoop logs from Hadoop cluster and developed our algorithm on the log messages. The messages in the logs contain error and non-error information. The messages in the log were represented using error IDs, which indicate message criticality. This paper introduced a novel failure prediction method using distributed HMM method over distributed computation. The idea behind this model is to identify the error pattern that indicates an upcoming failure. A machine learning approach like HMM has been proposed here, where the model is trained first using previously pre-processed log files and then it is used to predict the failures. Every log entry is split into equal intervals, defined by sliding window. These entries are separated into error sequence and non-error sequence. Training of the model is done using past observation. Viterbi's algorithm does the prediction of hidden state. Experimental results using Hadoop log files provide an accuracy of 91% and F-measure of 92% for 2 days of prediction time. These results indicate that it is promising to use the HMM method along with MapReduce to predict failure.

# References

[1] Adil Fahad, Najlaa Alshatri, Zahir Tari, Abdullah Alamri, Ibrahim Khalil, Albert Y Zomaya, Sebti Foufou, and Abdelaziz Bouras. "A survey of clustering algorithms for big data: Taxonomy and empirical analysis." In: *IEEE transactions on emerging topics in computing* 2.3 (2014), pp. 267–279.

[2] George EP Box, Gwilym M Jenkins, and Gregory C Reinsel. *Time series analysis: forecasting and control*. John Wiley & Sons, 2013.

[3] David. *anarchyape*. 2013. URL: `https://github.com/david78k/anarchyape` (visited on 11/08/2014).

[4] Pedro de Botelho Marcos. "Maresia: an approach to deal with the single points of failure of the MapReduce model." In: *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2013). DOI: `http://hdl.handle.net/10183/65635`.

[5] Faraz Faghri, Sobir Bazarbayev, Mark Overholt, Reza Farivar, Roy H Campbell, and William H Sanders. "Failure scenario as a service (FSaaS) for Hadoop clusters." In: *Proceedings of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management*. ACM. 2012, p. 5.

[6] Richard Mccreadie, Craig Macdonald, and Iadh Ounis. "MapReduce Indexing Strategies: Studying Scalability and Efficiency." In: *Inf. Process. Manage.* 48.5 (Sept. 2012), pp. 873–888. ISSN: 0306-4573. DOI: 10.1016/j.ipm.2010.12.003. URL: http://dx.doi.org/10.1016/j.ipm.2010.12.003.

[7] Florin Dinu TS Eugene Ng. "Analysis of Hadoop's Performance under Failures." In: *Rice University* (2012).

[8] SWIMProjectUCB. *SWIMProjectUCB/SWIM*. 2012. URL: https://github.com/SWIMProjectUCB/SWIM (visited on 11/08/2014).

[9] Teik-Toe Teoh, Siu-Yeung Cho, and Yok-Yen Nguwi. "Hidden Markov Model for hard-drive failure detection." In: *Computer Science & Education (ICCSE), 2012 7th International Conference on*. IEEE. 2012, pp. 3–8.

[10] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012. ISBN: 9781449311520.

[11] Hyunseok Chang, Murali Kodialam, Ramana Rao Kompella, TV Lakshman, Myungjin Lee, and Sarit Mukherjee. "Scheduling in mapreduce-like systems for fast completion time." In: *INFOCOM, 2011 Proceedings IEEE*. IEEE. 2011, pp. 3074–3082.

[12] Thomas Plotz and Gernot A Fink. *Markov Models for handwriting recognition*. Springer, 2011.

[13] Apache. *Apache Flume*. 2010. URL: https://flume.apache.org/FlumeUserGuide.html (visited on 11/08/2014).

[14] Rodrigo Fonseca. *X-Trace*. 2010. URL: https://github.com/rfonseca/X-Trace (visited on 11/08/2014).

[15] Hamzeh Zawawy, Kostas Kontogiannis, and John Mylopoulos. "Log filtering and interpretation for root cause analysis." In: *ICSM*. IEEE Computer Society, 2010, pp. 1–5. ISBN: 978-1-4244-8630-4. URL: http://dblp.uni-trier.de/db/conf/icsm/icsm2010.html#ZawawyKM10.

[16] Allen H Tai, Wai-Ki Ching, and Ling-Yau Chan. "Detection of machine failure: Hidden Markov Model approach." In: *Computers & Industrial Engineering* 57.2 (2009), pp. 608–619.

[17] Feng Wang, Jie Qiu, Jie Yang, Bo Dong, Xinhui Li, and Ying Li. "Hadoop high availability through metadata replication." In: *Proceedings of the first international workshop on Cloud data management*. ACM. 2009, pp. 37–44.

[18] Errin W. Fulp, Glenn A. Fink, and Jereme N. Haack. "Predicting Computer System Failures Using Support Vector Machines." In: *Proceedings of the First USENIX Conference on Analysis of System Logs*. WASL'08. San Diego, California: USENIX Association, 2008, pp. 5–5. URL: `http://dl.acm.org/citation.cfm?id=1855886.1855891`.

[19] Andy Konwinski, Matei Zaharia, Randy Katz, and Ion Stoica. *X-tracing Hadoop*. 2008.

[20] Jiaqi Tan, Xinghao Pan, Soila Kavulya, Rajeev Gandhi, and Priya Narasimhan. "SALSA: Analyzing Logs as StAte Machines." In: *WASL* 8 (2008), pp. 6–6.

[21] Md Rafiul Hassan, Baikunth Nath, and Michael Kirley. "A fusion model of HMM, ANN and GA for stock market forecasting." In: *Expert Systems with Applications* 33.1 (2007), pp. 171–180.

[22] Felix Salfner and Miroslaw Malek. "Using hidden semi-markov models for effective online failure prediction." In: *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*. IEEE. 2007, pp. 161–174.

[23] Alessandro Daidone, Felicita Di Giandomenico, Andrea Bondavalli, and Silvano Chiaradonna. "Hidden Markov models as a support for diagnosis: Formalization of the problem and synthesis of the solution." In: *Reliable Distributed Systems, 2006. SRDS'06. 25th IEEE Symposium on*. IEEE. 2006, pp. 245–256.

[24] Yinglung Liang, Yanyong Zhang, Anand Sivasubramaniam, Ramendra K Sahoo, Jose Moreira, and Manish Gupta. "Filtering failure logs for a bluegene/l prototype." In: *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*. IEEE. 2005, pp. 476–485.

[25] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." In: *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*. OSDI'04. San Francisco, CA: USENIX Association, 2004, pp. 10–10. URL: http://dl.acm.org/citation.cfm?id=1251254.1251264.

[26] Ramendra K. Sahoo, Anand Sivasubramaniam, Mark S. Squillante, and Yanyong Zhang. "Failure Data Analysis of a Large-Scale Heterogeneous Server Environment." In: *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2004), p. 772. DOI: http://doi.ieeecomputersociety.org/10.1109/DSN.2004.1311948.

[27] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. "An Efficient k-Means Clustering Algorithm: Analysis and Implementation." In: *IEEE Trans. Pattern Anal. Mach. Intell.* 24.7 (July 2002), pp. 881–892. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2002.1017616. URL: http://dx.doi.org/10.1109/TPAMI.2002.1017616.

[28] Xuedong Huang, Alex Acero, Hsiao-Wuen Hon, and Raj Foreword By-Reddy. *Spoken language processing: A guide to theory, algorithm, and system development*. Prentice Hall PTR, 2001.

[29] Andrew D Wilson and Aaron F Bobick. "Parametric hidden markov models for gesture recognition." In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 21.9 (1999), pp. 884–900.

[30] Richard Durbin. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

[31] Lawrence Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition." In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286.

[32] Arthur P Dempster, Nan M Laird, and Donald B Rubin. "Maximum likelihood from incomplete data via the EM algorithm." In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1977), pp. 1–38.

[33] Leonard E Baum, JA Eagon, et al. "An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology." In: *Bull. Amer. Math. Soc* 73.3 (1967), pp. 360–363.

[34] Andrew J Viterbi. "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm." In: *Information Theory, IEEE Transactions on* 13.2 (1967), pp. 260–269.

[35] Evgeniui Borisovich Dynkin. "Markov processes." In: *Markov Processes*. Springer, 1965, pp. 77–104.

# Paper III: SD-HDFS: Secure Deletion in Hadoop Distributed File System

# SD-HDFS: Secure Deletion in Hadoop Distributed File System

**B. Agrawal[1], R. Hansen[2], C. Rong[1], T. Wiktorski[1]**

[1] Department of Electrical Engineering and Computer Science, University of Stavanger

[2] Department of Computer and Information Technology, Purdue University, West Lafayette, IN, USA

**Abstract:**

Sensitive information that is stored in Hadoop clusters can potentially be retrieved without permission or access granted. In addition, the ability to recover deleted data from Hadoop clusters represents a major security threat. Hadoop clusters are used to manage large amounts of data both within and outside of organizations. As a result, it has become important to be able to locate and remove data effectively and efficiently. In this paper, we propose *Secure Delete*, a holistic framework that propagates file information to the block management layer via an auxiliary communication path. The framework tracks down undeleted data blocks and modifies the normal deletion operation in the Hadoop Distributed File System (HDFS). We introduce *CheckerNode*, which generates a summary report from all DataNodes and compares the block information with the metadata from the *NameNode*. If the metadata do not contain the entries for the data blocks, unsynchronized blocks are automatically deleted. However, deleted data could still be recovered using digital forensics tools. We also describe a novel secure deletion technique in HDFS that generates a random pattern and writes multiple times to the disk location of the data block.

# 1 Introduction

The emergence of cloud computing paradigm has made individuals and organizations are moving towards cloud storage systems; this move is motivated by benefits such as shared storage, computation, and services transparently among a massive number of users. Many users and enterprise applications today generates and needs to handle huge volumes of data on a regular basis, and are appropriately referred to as data intensive applications. However, to perform data-intensive computation, two major pre-conditions need to be satisfied: (i) the application should run in a parallel algorithm to utilize the available resources; and (ii) there should be an appropriate parallel runtime support to implement it. There are several cloud technologies such as Hadoop[8], and Spark[9] that can perform a data-intensive computation.

Hadoop [14] is an open-source framework that implements the MapReduce [20] parallel programming model. The Hadoop framework is composed of a MapReduce engine and Hadoop Distributed File System (HDFS). The HDFS manages storage resources across a Hadoop cluster by providing global access to any file [19]. HDFS is implemented by two services: NameNode and DataNodes. The NameNode is responsible for maintaining the HDFS directory tree and metadata of all files. Clients contact the NameNode in order to perform common file system operations, such as open, close, rename, and delete. The NameNode does not store actual data itself, but rather maintains a mapping between HDFS filename, a list of blocks and the DataNode(s) on which these blocks are stored.

The scaling of storage systems involves the manipulation of thousands of computation and storage nodes, which leads to a higher probability of node failure. Hadoop has become a hugely popular platform for large-scale data analysis. This popularity poses even greater demands when it comes to scalability and functionality and has revealed an essential architectural limitation and challenges of its underlying file system. To provide better reliability and availability, the traditional design of HDFS replicates each data block into three copies to protect against node failures. While this mechanism provides better availability than keeping a single copy of data, it suffers issues during deletion of data in dead nodes [2]. For example: when a user sends the delete command to remove data and the data is replicated on three different nodes, but the data is not completely deleted

[8] https://hadoop.apache.org/
[9] http://spark.apache.org/

though if one of the nodes is dead, this can then result in major data spillage of sensitive information [7]. Even though HDFS delete command confirms data is deleted successfully, there can be some undeleted blocks or traces of data which can lead to major security threats. In this paper, we introduce the checkerNode, that collects a summary report from all the DataNodes and compares the block information with the metadata information in the NameNode. If the block information does not exist in the metadata, the block is automatically deleted, and the notification is sent to the user.

In HDFS, the data can be recovered from the HDFS deletion command using forensic tools. Most of the users are not aware that after deleting a file it can still be recovered. There are many forensic tools available which can recover deleted data stored on Linux, Windows, and Macintosh machines. The simple solution for deletion is to overwrite the memory segment with a random pattern. However, overwritten data can still be recovered using techniques like Magnetic force microscopy (MFM) and scanning tunneling microscopy (STM) [29]. Hence multiple processes of overwriting are needed. However, a number of overwrite processes involve with an extra I/O overhead.

When a user uploads the data to HDFS, the underlying storage layers essentially keep the data immutable, only allowing concurrent appends. While this method exploits workflow parallelism to a certain degree, it does not provide a true file system capability e.g., support for users to randomly, update the contents of the files. Since HDFS does not support in-place updates, many applications that are write-intensive and require file modifications need to overwrite all the file contents, even if very small changes were made. Therefore, changing any file content requires recreating whole data blocks, which effectively increases the overall write and update performance of the system. However, Finger [9] provides novel erasure coding scheme by breaking the large block size (64 or 128 MB) into smaller chunks and also added an in-place update. It is designed to mitigate extra reads when performing erasure coding on a large block update and maintains the minimal metadata size. HDFS-RAID [6] developed by Facebook, reduces the disk I/O operation. However, it has a computation overhead due to encoding and decoding process on the sub-block granularity and does not encounter any extra I/O regardless of the block size.

Currently, there is no component in Hadoop that could overwrite the data blocks with a random pattern for complete deletion of data [10]. Secure Deletion is the technique where data is deleted from a system in

such a manner that the system is not able to recover the deleted data. In this paper, we introduce a overwrite technique in a block with a random pattern of 1s and 0s.

## 1.1 Our Contribution

The existing approach for data deletion in HDFS can lead to data leakage in two ways: undeleted blocks due to node failure and data sniffing using forensic tools. In this paper, we propose an improved data deletion framework for HDFS. Through an additional component called checkerNode which ensure that all blocks are deleted in the case where it is impossible to delete, the component provides detailed reports on hardware components that might contain sensitive data. Further, we extend the deletion process by physically locating each copy of the block and overwriting it to prevent data spillage using forensic tools.

## 1.2 Related Work

Storing files using HDFS as an underlying platform for cloud storage has been of significant interest in recent years. This typically follows one of two possible approaches: extending Hadoop's built-in FileSystem abstract class with *shim* code that supports the distributed file system, or using built-in Java APIs to expose the distributed file system. HFAA (Hadoop Filesystem Agnostic API) [15] provides Hadoop to integrate with any distributed file system over TCP sockets. This API allows the user to integrate any file system with Hadoop without knowing details about it.

There are several research projects carried out on secure deletion in the normal file system, but none of these address the issue of secure deletion in a distributed file system. Bauer et al. [28] used an asynchronous overwriting process and ultimately provided a far more usable and complete secure deletion facility in Linux file system Ext3. Systematization of Knowledge (SoK) [11] showed the existing approach in terms of its interface to a physical medium. It shows user-level approaches are generally limited and cannot securely delete data in all cases. However, it proves overwriting can be used as a user-space approach. Reardon et al. [18] provided three approaches for secure deletion on log-structured file systems; purging, ballooning, and zero overwriting. Out of the three approaches, zero overwriting guarantees immediate secure deletion. Sun et al. [22] proposed a hybrid model of secure deletion in flash memory using a technique of zero overwriting and block cleaning. However, the hybrid model consists

of block cleaning that involves moving a block from one memory location to another. This can be an expensive operation for moving large block size. Lee et al. [21] modified YAFFS, a log-structured file system for NAND flash, to handle secure file deletion. The modified YAFFS encrypts files and stores each file's key along with its metadata. Whenever a file is deleted, its key is erased, and the encrypted data blocks remain. Joukov et al. [26] introduced *Purgefs* techniques in a file system which provide better performance for secure deletion. TrueErase [13] is a secure-deletion framework that deletes sensitive data from the electronic storage. True deletion is a compatible full-storage-data-path framework that performs per file secure deletion and works with common file systems and solid-state storage. Daniel et al. [17] used Kepler+Hadoop framework to presents a data model that is capable of capturing provenance inside a MapReduce job.

All these approaches provide a solution for secure deletion in Linux file system or Windows, where the most common approaches used for secure deletion is overwriting the kernel metadata and memory location. In 1996, Gutmann [29] proved that simple overwriting does not delete the data completely. The data can be recovered using Magnetic force microscopy (MFM) and scanning tunneling microscopy (STM) techniques. Gutmann proved that for complete deletion, it requires 35 passes of an overwriting process. However, in 2006 NIST Special Publication 800-88 [25] showed that most of the today's media can be effectively cleared by a single overwrite process. We found that existing solutions are either inconvenient, incompetent, or insecure. We have designed secure deletion in Hadoop as a file system extension to Hadoop that transparently overwrites files on the per-delete basis.

### 1.3 Paper Structure

Section II gives an outline for this paper. Section III introduces the design and approach of our method. Section IV evaluates our approach and presents the results. Section V concludes the paper.

## 2 Background

### 2.1 Hadoop:

Hadoop [14] is an open-source framework for distributed storage and data-intensive processing, first developed by Yahoo!. It has two core projects:

Hadoop Distributed File System (HDFS) and MapReduce programming model [20]. HDFS is a distributed file system that splits and stores data on nodes throughout a cluster, with a number of replicas. It provides an extremely reliable, fault-tolerant, consistent, efficient and cost-effective way to store a large amount of data. The MapReduce model consists of two key functions: Mapper and Reducer. The Mapper processes input data splits in parallel through different map tasks and then sends sorted, shuffled outputs to the Reducers that in turn group and process them using a reduce task for each group.

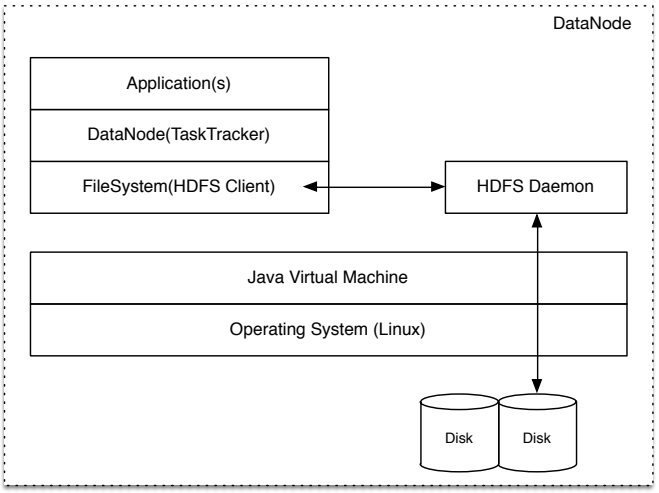



**Figure 1:** Example of HDFS cluster Node.

Hadoop uses HDFS for reliable storage. When a file is written in HDFS, it is divided into blocks of a fixed size. The client first contacts the NameNode, which gets the list of DataNodes where actual data can be stored. The data blocks are distributed across the Hadoop cluster. Figure 1 shows the architecture of the Hadoop cluster node used for both computation and storage. The MapReduce engine (running inside a Java virtual machine) executes the user application. When the application reads or writes data, requests are passed through the Hadoop *org.apache.hadoop.fs.FileSystem* class that provides a standard interface for distributed file systems, including the default HDFS. An HDFS client is then responsible for retrieving data from the distributed file system by contacting a DataNode. In the common case, the DataNode that is running on the same node requires no external network traffic. The DataNode, also running inside a Java

virtual machine, accesses the data stored on local disk using normal file I/O functions.

The Hadoop frameworks consist of two key component: the MapReduce and the Hadoop Distributed File System (HDFS). The HDFS system consists of the NameNode and the DataNode. The NameNode is the master node on which the job tracker runs. It contains the metadata (information about data blocks stored in DataNodes - the location, size of the file, etc.). It maintains and manages the data blocks, which are present on the DataNodes, where the actual data is stored. The DataNode runs three main types of daemon: Read Block, Write Block, and Write-Replicated Block. The NameNode and the DataNode maintain their own logging format. Each node records events/activities related to reading, writing, and the replication of HDFS data blocks.

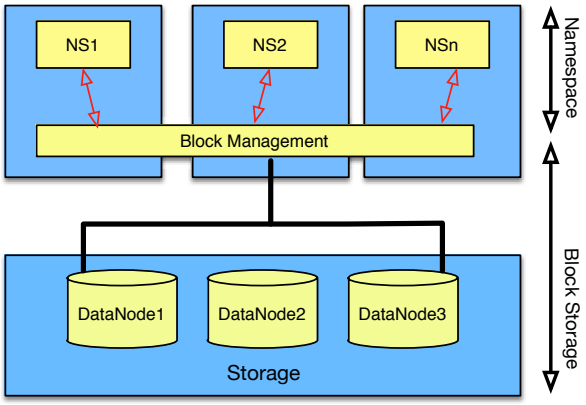



**Figure 2:** HDFS architecture.

The NameNode maintain the directory tree of all files in the file system and keeps track of the data stored across the cluster. The HDFS clients contact the NameNode in order to perform common file system operations, such as open, close, rename, and delete. The NameNode does not store data itself, but rather maintains a mapping between HDFS filename, a list of blocks in the file, and the DataNode(s) on which those blocks are stored. Each DataNode stores data as HDFS blocks (64MB to 512MB chunks of a single logical file). Each data block is saved as a separate file in the DataNode's local file system, which uses a native file system like Ext4. The data blocks are created or destroyed on DataNodes at the request of the NameNode, which validates and processes requests from clients. Although

NameNodes manage the namespace, clients communicate directly with DataNodes to read or write data at the HDFS block level.

Figure 2 shows the architecture of a Hadoop cluster used for storage. The user application requests Hadoop *org.apache.hadoop.fs.FileSystem* class for reading and writing data. The NameNode decides where to put each block using the block placement policy. The NameNode stores the file system namespace in memory for fast access. The DataNodes store the actual file blocks and are responsible for serving read and write requests from clients.

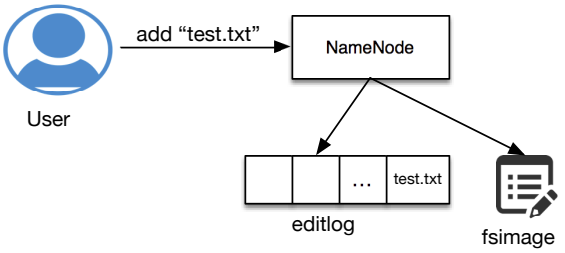



**Figure 3:** Namenode updates editlog and fsimage.

The NameNode metadata represents the structure of HDFS directories and files in a tree. It also includes the various attributes of directories and files, such as ownership, permissions, quotas, and replication factor. HDFS metadata is divided into two categories of files as shown in figure 3 **fsimage**: A fsimage file is an image that contains the complete state of the file system in HDFS at a point in time. Every file system modification is assigned a unique, monotonically increasing transaction ID. **editlog**: âĂŞ An edits file is a log that maintains changes (file creation, deletion or modification) that were made in the file system after the most recent fsimage. Metadata contains all the storage level (block management) information. These documents include information related to the file's block locations on the DataNodes. In addition to keeping track of data blocks, the NameNode manages information related to the data provenance.

The HDFS is designed to provide high sequential access throughput and fault tolerant storage on low-cost commodity hardware. HDFS aims to support WORM (write-once-read-many) type of workloads with large datasets in contrast with traditional POSIX (Portable Operating System Interface) distributed file systems. Distributed file systems tend to provide similar consistency as a POSIX model [12]. However, HDFS provide

a consistency level weaker than POSIX. For example, HDFS does not provide a solution to append and update on existing blocks. Similar to POSIX file systems, HDFS represents both files and directories as inodes in the metadata. Directory inodes contain a list of files inodes, and file inodes consist of a number of blocks stored in different DataNodes. A block is replicated on a number of different DataNodes (default=3).

### 2.2 Apache Common:

Apache Common[10] is an open source project by Apache Software Foundation. It consists of a set of common utility libraries needed by the Hadoop framework and modules. It has native shared libraries that are used for common I/O operations and is include in Java implementations for I/O utilities, logging, compression codecs, and error detection. It also includes interfaces and tools for configuration, authentication, data confidentiality, service-level authorization, and the Hadoop Key Management Server (KMS).

### 2.3 Fourth Extended Filesystem (Ext4):

Ext4 is the evolution of Ext3. It is a journaling file system for Linux. It provides backward compatibility extensions to Ext3, and it divides a storage media into an array of logical blocks to reduce bookkeeping overhead and to increase throughput by forcing larger transfer sizes. It supports large file system volume with size up to 1 exbibyte (EiB) and files with size up to 16 tebibytes (TiB).

## 3 Approach

We introduce *secure deletion*, a holistic framework that propagates file information to the block management layer through a communication path so that file deletion can be honored throughout the data path. Our system includes the following steps:

- The HDFS client specifies which files or directories are to be deleted.
- Tracking the information across different storage layers via a block management module and checkerNode.

[10]https://commons.apache.org/

- Enforcing secure deletion.
- Exploiting data-block consistency between different DataNodes and metadata for verification.

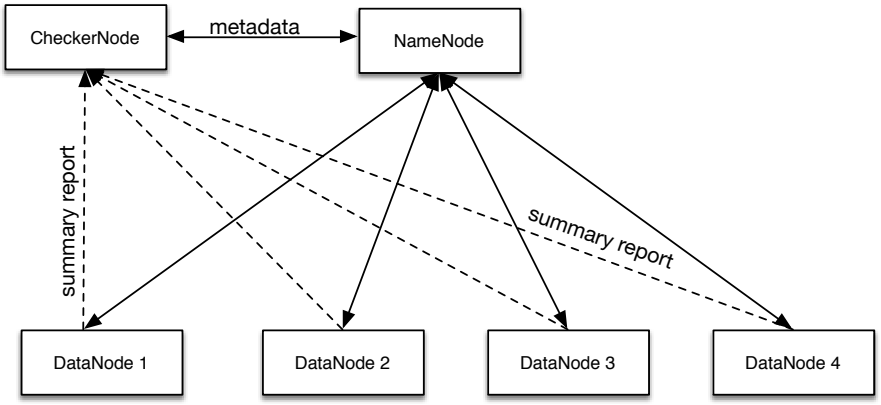



**Figure 4:** CheckerNode checks periodically with NameNode and DataNode.

Our framework consists of the *CheckerNode* that interacts with the NameNode at periodic intervals. They are set at a default of 2 minutes, but the user can reconfigure it. Figure 4, consists of the *CheckerNode* that receives summary reports from the DataNodes and compares the block stored in the DataNodes with the metadata in the NameNodes. If there a mismatch block in a summary report when compared with metadata, the CheckerNode sends the delete command to delete the inconsistent block and at the same time it informs the user about the hardware component containing that block.

In figure 5, the user stored a file *Test.txt* of size 248MB. The block size on each DataNodes is configured to be 128MB, so the file *test.txt* is stored in two blocks; B1 and B2 in three DataNodes (D1, D2, D3). The NameNode contains the block storage information. When the user sends the secureDelete command, the file block is overwritten and replaced with a random pattern and the memory location is released. In the meantime, checkerNode daemon is called, and it checks for inconsistent blocks present in any of the DataNodes. In figure 5, DataNode 3 consists of blocks B4 and B5, whose entries are not present in the metadata. The checkerNode gets all the information from DataNodes, and it finds blocks B4 and B5 are

not present in the metadata. The checkerNode sends a delete command (*delete[B4,B5]*) to delete the block from the DataNode.

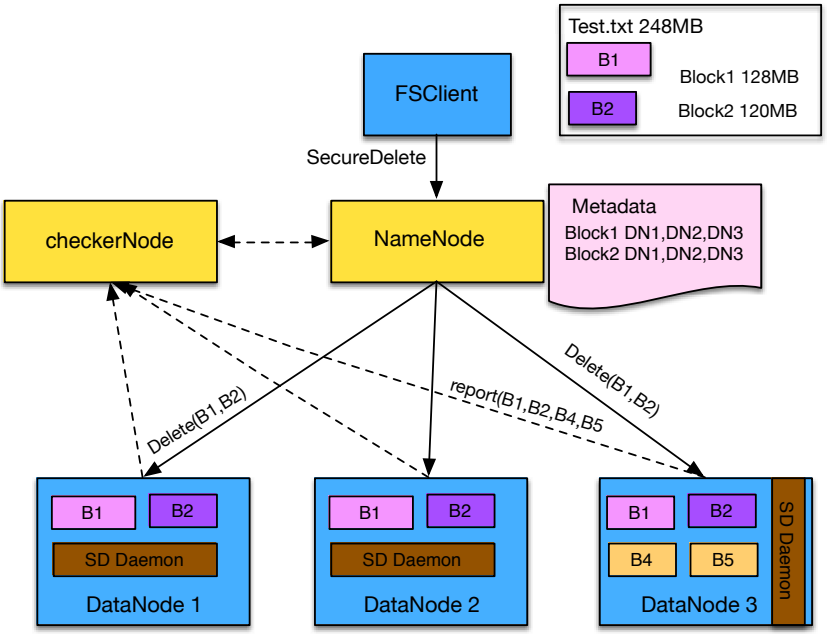



**Figure 5:** Secure Deletion command sent through HDFS client.

The chunk of blocks are stored as a file in the file system, and the data content of a file is deleted via its truncate function. This process involves updating the inode to set the file size to zero, removing the pointers to data blocks, overwriting the data blocks with a random pattern and freeing up the blocks for reuse. Multiple rounds of truncation may be required to securely delete the contents of a large file as suggested by Gutmann [29]. When a file is deleted in file systems including HDFS, only the reference to that data, called a pointer, is deleted but the data itself remains. In the FAT file system, for example, when a file is deleted the files directory entry is changed to reflect that the space occupied by data, and the data occupies is then unallocated. The first character of a file name is switched with a marker. The actual file data is still left unchanged. In the file system, the data traces are still present with an exception of overwriting with a random data [27]. Similarly, in HDFS, when a file is deleted, only the pointer to the file on the DataNode is deleted. The data remains unchanged in the memory location of the DataNode until that data is

overwritten.

Our approach for secure deletion is tools open the file from the user-space and overwrite its contents with a random pattern (e.g., all 1s and 0s or all zeros). The file is stored as a chunk of blocks in HDFS. Later, when the file is unlinked, only the content of most recent version is stored on the disk. To combat analog remnants, overwriting can be performed multiple times as proposed by Gutmann [29] but with at least one pass of random data. The overwrite function will update the *edit_logs* and $fs\_image$ information in the NameNodes. It overwrites the block attributes, such as the file's size, and access times. The framework relies on the NameNode metadata: each file block is stored at known locations pointed by the NameNode metadata, and when the file block is updated, all older versions are replaced with the new version. If the NameNode metadata fails to point the actual block location, the *checkerNode* will fix that issue.

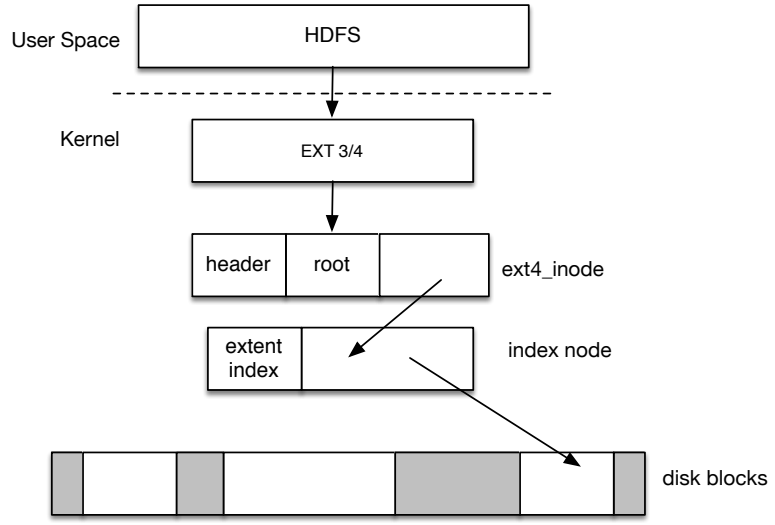



**Figure 6:** Structure of ext4 inode which stores hdfs data blocks.

Each block of data is treated as a separate file in the file system. It is stored above Ext4[24] in the latest version of Linux as shown in Figure 6. It uses an indirect block to index data blocks. Ext4 needs to pass through an extra level of indirect nodes to access data blocks as shown in Figure 6. Each HDFS data block is treated as a normal file by Ext4 and it is represented by inode. In a nutshell, a file's inode number can be found using *ls -i* command. An inode is the data structure that describes the

metadata of files and is composed of several fields such as ownership, size, modify time, mode, link count and data block pointers. When a user calls *secureDelete* command, the overwrite daemon is called. The overwrite function traces the location of the files where all the blocks are stored. The larger files are represented by the chunk of blocks located at different DataNodes. Each DataNode has a secure delete daemon which is triggered by a checkerNode.

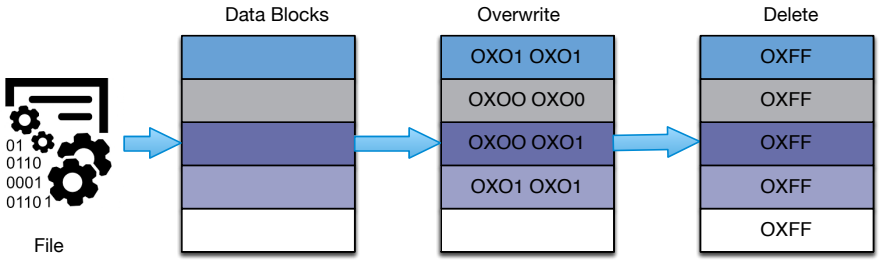



**Figure 7:** Process of overwrite and delete method.

Figure 7 shows the process of 1s and 0s overwrite and deletion in the memory location of the blocks. Overwrite is a deletion method that overwrites memory with the combination of 0x00 and 0x01 so that the original data can be securely deleted. In this paper, we introduced the model in which the whole block of data is overwritten and deleted. In contrast, entire data is deleted from the block. However, if some valid data remains in the block and is useful for other operations the data needs to move to another block, so it does not need additional computation (read/write). In order to minimize cost disk I/O, the entire blocks are overwritten, and the memory locations are released.

Figure 8 shows secure deletion process in Ext4. The client invokes *getblockLocation(file)* to get all the block locations of the files through NameNode. With the help of Java API, the client creates a TCP connection to the DataNodes to run overwrite daemon. Now, the daemon gets inode information and underlying block location from each block. A single data block is divided into smaller blocks whose metadata is maintained by the inode. The daemon opens the block in overwrite mode and loads the block with a random pattern of 1s and 0s. After the overwriting operation is completed, the daemon deletes command to release all the memory location occupied by the data blocks.

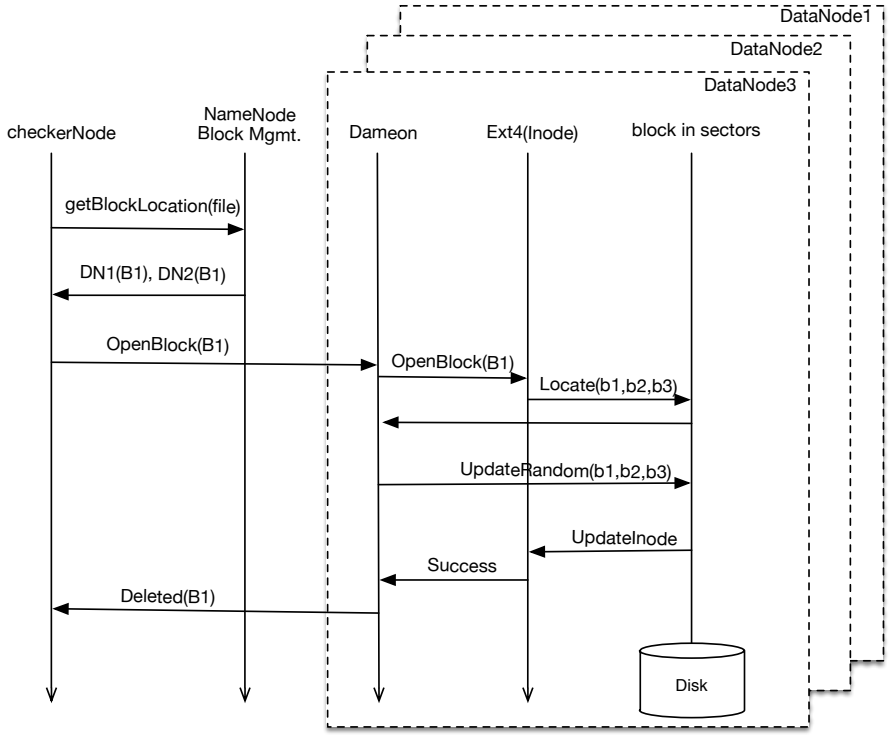



**Figure 8:** The sequence diagram shows how the update process is done in the data blocks of ext4 file system.

# 4 Result

*Setup:* Our cluster is comprised of 4 nodes with Ubuntu 14.04 Trusty Tahr, one node each for Namenode, and Secondary Namenode. The remaining 3 nodes act as DataNodes, and Task Trackers. All nodes have an Intel two-core 2.13GHz processor, 2 GB of DDR-2 RAM, 160 GigaBytes storage. Experiments were conducted using Hadoop-2.7.1 Apache releases. Our default HDFS configuration had a block size of 128 MB and a replication factor of 2.

In this section, we highlight the inconsistency issue during data deletion in information security assessment for HDFS. Our approach provides a plugin for HDFS, which is developed with several test scenarios that allow us to strategically track undeleted data within HDFS to investigate vulnerability and risk assessment. In this section, we test our data consistency

in all DataNodes and allow users to delete sensitive information from non-classified Hadoop Distributed File System (HDFS).

## 4.1 Data Consistency

At the system storage level, the NameNode provides commands that allow us to collect the location of file blocks in the system and their ID from its system log files. But there is no such command in Hadoop that can provide a report of undeleted data. Therefore, we introduced checkerNode that contains information about undeleted data traces and deletes those unsynchronized data. The checkerNode also helps in secureDeletion by establishing TCP connection with different DataNodes and executing a delete daemon in each DataNode.

```
5 block(s) Missing Replicas with Metadata:  OK
DataNode: 128.210.139.79
0. 128.210.139.79:blk_1073741854_1031 len=134217728
1. 128.210.139.79:blk_1073741855_1032 len=134217728
2. 128.210.139.79:blk_1073741856_1033 len=134217728
3. 128.210.139.79:blk_1073741856_1034 len=134217728
4. 128.210.139.79:blk_1073741856_1035 len=85222489
 Default replication factor:   2
 Average block replication:    2.0
 Number of data-nodes:         3
 Number of racks:              1
FSClient ended at Sun Jan 17 22:14:32 EST 2016 in 50 milliseconds
```

**Figure 9:** CheckerNode report with 5 blocks still present in one of the DataNode after deletion.

For example, HDFS client uploads 580 MB of data in the cluster, which has a replication factor of 2 and block size of 128MB. The total number of blocks used to store the data is 5. Our cluster has 3 DataNodes; DN1, DN2, and DN3. The DataNodes DN1 and DN2 are used to store all the blocks and its replication. After a while, DN2 is down. Now, data is replicated in DN3. A user sends the delete command. The data is deleted from node DN1 and DN3 along with its metadata information. After an hour, DN2 is back, and it contains the actual data blocks. The metadata in the NameNode does not have the information about the file anymore, so it is treated as deleted. Perhaps, still sensitive data is present in DataNode DN2 which is tracked by checkerNode in Figure 9.

The secure deletion framework runs the checkerNode daemon which now deletes this unsynchronized data in DataNode DN2. When the user sends delete command, the checkerNode sends the report that the data is still in the DN2.

## 4.2 Secure Deletion

Several steps are taken for the analysis phase to prepare the forensic environment. The first step is to attach a network-attached storage (NAS), or other large-scale storage solutions such as a cloud environment. The analysis is performed on the logical files or forensic images, with the focus being on extracting and collecting metadata information for different blocks. The method used for analyzing HDFS data is *data carving techniques* [8]. Data carving techniques frequently occur in a digital investigation when unallocated file system space is analyzed to extract files. The files are 'carved' from the unallocated space using file type-specific header and footer values. File system structures are not used during the process [23]. Simply stated, file carving is the process of extracting the strips of data from a greater storage space. Digital forensics examiners commonly look for data remnants in unallocated file system space. Beek et al. [16] wrote a white paper explaining data carving concepts in which it referred to several data carving tools were referred to. In the paper, Beek et al. also explained the difference between data carving and data recovery. Data recovery is the carving of data based on the file system structure, which would not be useful in a system format. Further, the file system used to retrieve data is not important to the data retrieval process. In the case examined by our research, we are dependent on the HDFS to identify the nodes that need to be carved.

The framework consists of commands that are used by the client for tracking the distributed undeleted blocks and secure deletion of the content in HDFS and are listed in table 4.1:

The NameNode is used to locate where and how a file is distributed within the DataNodes. There are other related components involved in deleting the sensitive information such as the trash folder and $fs\_image$ which captures the state of data stored within the DataNodes. Using forensic tools such as Forensic ToolKit (FTK) [1], EnCase [5] and Autopsy [3], the deleted data can be recovered as shown in Figure 10.

When a file is stored in HDFS, the NameNode maintains the metadata and the DataNode stores the actual data blocks. The disk image of each DataNode is created using the *dd* command in Linux. The single data

**Table 4.1:** List of commands for secure deletion.

| **Command** | **Info** |
|---|---|
| getBlockLocation | $blk_1073751858$<br>Host [$datanode1, datanode2,$<br>$datanode3$] |
| getUndeletedBlock | $blk_1073751851$ in datanode3<br>$blk_1073751852$ in datanode1 |
| rmr | deleted $blk_1073751858$ |
| secureDelete | Overwrite $blk_1073751858$<br>Deleted $blk_1073751858$ |
| randPattern | 101010101010111100110101 |
| Put | Added $blk_1073751858$ len=8615342 repl=3<br>[$datanode1, datanode2,$ $datanode3$] |

blocks in each DataNode is stored into multiple blocks where the metadata is maintained by inode. Figure 10 shows the block *blk_107374189* which is stored in different subblocks whose information is maintained by inode (*7925760,7925761,7925762,7925763, and 7925764*) in the disk sectors. The process for extracting HDFS data from a forensic image has several steps:

- Identify the location of the data in the DataNode.
- Locate the blocks and files in the operating system.
- Analyze the data blocks to identify the relevant files and its content.
- Extract the data into a file for analysis.

The location of the data in the DataNode is located in *hdfs-site.xml* file. In the Autopsy tool, we collected the image of the disk and analyzed it.

```
                    META DATA INFORMATION

inode: 1980976
Allocated
Group: 241
Generation Id: 2123065567
uid / gid: 1000 / 1000
mode: rrw-rw-r--
Flags: Extents,
size: 17167
num of links: 1

Inode Times:
Accessed:       2016-01-13 16:13:02.841541162 (EST)
File Modified:  2016-01-13 16:12:56.289422531 (EST)
Inode Modified: 2016-01-13 16:14:08.966737041 (EST)
File Created:   2016-01-13 16:12:56.109419273 (EST)

Direct Blocks:
7925760 7925761 7925762 7925763 7925764

File Type: no read permission
```

**Figure 10:** Autopsy shows the deleted block with its associated inode and subblocks information in Ext4 filesystem.

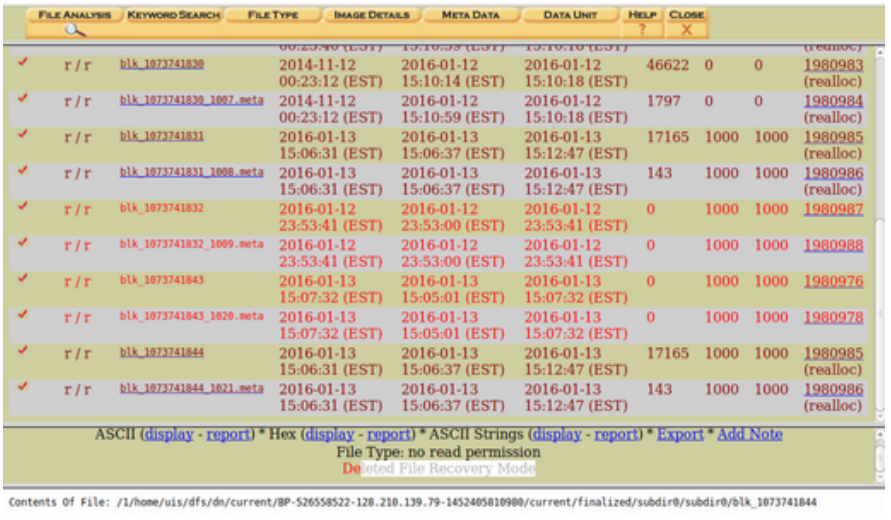



**Figure 11:** Autopsy shows all the deleted blocks from one of the DataNodes.

The Autopsy tool has capabilities to allow raw disk images to be loaded and processed as a normal file system.. The tool allows data blocks to be

recovered from the disk image in two ways: 1) navigate to the standard directories where HDFS data is stored, 2) run a keyword search for *hdfs-site.xml* and sort the results by filename for that file. The entry with the red color in Figure 11 recovers the deleted data block from DataNode 1. The files with the *.meta* file extension contains checksum information that is used by HDFS to verify the integrity of data blocks. The files without an extension are the actual data blocks, which contain sensitive information. The next step is to analyze the content of the files using a hex editor.


**Figure 12:** Autopsy recovers the deleted blocks from one of the DataNodes.

The *secureDelete* command issues *overwrite* and *delete* commands. The first phase of the secure delete command is an overwrite process, the framework gets all the block locations from the NameNode metadata. Now, it opens the block in write mode using the *FSDataInputStream* pointer. The pointer is moved to the beginning of the file and the actual content of the file is replaced with a random pattern of 1s and 0s. Figure 12 shows that the actual content of *test.txt* is replaced with 1s and 0s.

## 4.3 Execution Time

In this section, we analyze the data deletion costs of the HDFS using techniques like overwriting and deleting. We used a random pattern of 1s and 0s for an overwriting method. The cost of overwriting is slightly higher than of an ordinary write process in HDFS because the blocks need to be relocated using metadata information from the NameNode. As a result, the data deletion cost of multiple blocks can be written as:

$$Cost_w = N_{deletedBlocks} * (T_w + T_{meta_r}) \tag{4.1}$$

Where, $N_{deletedBlocks}$ is the number of blocks deleted, $T_w$ is the overwrite time for the blocks, and $T_{meta_r}$ is read time of the metadata from the

NameNode. The equation 4.1 indicates the cost of overwriting is directly proportional to the number of blocks to be deleted. But writing and reading time depends on the commodity hardware and network throughput. Let us assume that **K** is the constant for hardware and network throughput; the final equation is:

$$Cost_w = K * (N_{deletedBlocks} * (T_w + T_{meta_r})) \tag{4.2}$$

Equation 4.2 refers to execution time in each DataNodes. Our method follows a write-then-erase technique. The cost of overwriting also depends on the size of blocks and the number of overwrites. The cost of deletion is indirectly proportional to the block size. Equation 4.2 can be written as:

$$Cost_w = K * N_{pass} * (N_{deletedBlocks} * (T_w + T_{meta_r})) \tag{4.3}$$

$Cost_w \propto (1/S_{blocks})$ where, $S_{blocks}$ is size of blocks (64-512 MB) depend on configuration and $N_{pass}$ number of overwrite process. Figure 13 indicates the block size has very less effect on the execution time and thus in cost.

Equation 4.3 depends on the distribution of the data as well. Let $T_{cap}$ be the total capacity of a disk in a node and $T_{used}$ be the total used space. Therefore the ideal storage on each volume/disk should be $I_{storage} = T_{cap}/T_{used}$. The volume of data density is the difference between the ideal storage and the current DFS used ratio, in other words volume data density for one node is $DD_{volume} = I_{storage} - dfsUsedRatio$, where $DD_{volume}$ is volume of data density. A positive value of $DD_{volume}$ indicates that the disk is underutilized and a negative indicates that the disk is over-utilized. Now, we can calculate the data distribution around the data center. This is done by computing nodes with maximum skew from $I_{storage}$ values, for which we sum up all the absolute values of $DD_{volume}$. The node data density is calculated as: $node_{DD} = \sum_{d_i \epsilon DD_{volume}(I)} |d_i|$ [4], where $node_{DD}$ is the node data density. Lower $node_{DD}$ indicates a uniform distributed and higher values indicate a more skewed distribution. Now, the total cost considering the data distribution around the node looks as follows:

$$Cost_w = \frac{K * N_{pass} * (N_{deletedBlocks} * (T_w + T_{meta_r}))}{\sum_{d_i \epsilon DD_{volume}(I)} |d_i|} \tag{4.4}$$

In Figure 13, we assumed the number of overwrite process was 1 i.e ($N_{pass} = 1$). Larger blocks offer several advantages over smaller blocks. It reduces the number of interactions with NameNode and also reduces

a size of metadata that needs to be stored in the NameNode. It reduces extra network overheads by keeping a persistent TCP connection to the DataNode. In Figure 13, the execution time for writing and deleting the larger blocks is almost similar to deleting smaller blocks. However, HDFS blocks are larger in comparison to disk blocks, because they minimize the cost of seeks. Thus, in our case, all the interaction of deletion is done through a checkerNode. The checkerNode sends the delete command, and the rest is handled by overwriting daemon in each node. The overwrite daemon reads the metadata from inodes. It does not need to make a connection with NameNode or checkerNode. Thus, it reduces the extra network overhead and boosts the performance of the deletion operation.

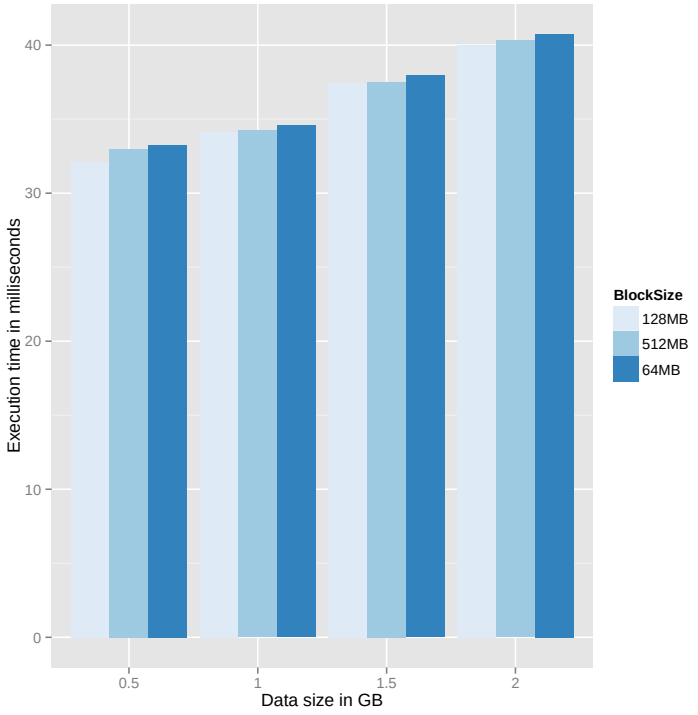



**Figure 13:** Secure deletion execution time: Block size does not has much effect in execution time.

# 5 Conclusion

Secure deletion is an important technique to prevent data leakage. People are unaware of the fact that data can be recovered even after they are

deleted from the storage device. However, we showed that data can be recovered using digital forensic tools.

We proposed a hybrid scheme of overwriting and deleting for secure deletion in HDFS. The default overwrite operation in Hadoop does not overwrite, it simply creates a new data block which writes data to a new location while the original data still exists. There is no actual overwrite method existing in Hadoop that could overwrite the memory location. In our proposed approach, we overwrote the memory location with a random pattern of 1s and 0s and released the memory location after the operation so that memory could be reused. We also introduced a checkerNode that tracks undeleted data blocks and deletes it. The framework provides the location of the undeleted blocks and failure components where the block is stored. Further, our framework allows Hadoop to integrate more easily into existing high-performance computing environments, where alternate distributed file systems are already present.

## References

[1] AccessData. *Forensic Toolkit (FTK).* `http://accessdata.com/solutions/digital-forensics/forensic-toolkit-ftk`. [Online; accessed 10-Dec-2015].

[2] Cloudera Engineering Blog. *Introduction to HDFS Erasure Coding in Apache Hadoop.* `http://blog.cloudera.com/blog/2015/09/introduction-to-hdfs-erasure-coding-in-apache-hadoop/`. [Online; accessed 1-Nov-2015].

[3] Basis Technology Corp. *Autopsy, a graphical interface to The Sleuth Kit.* `http://accessdata.com/solutions/digital-forensics/forensic-toolkit-ftk`. [Online; accessed 5-Dec-2015].

[4] Jira issue: HDFS-1312. *Re-balance disks within a Datanode.* `https://issues.apache.org/jira/browse/HDFS-1312/`. [Online; accessed 21-Dec-2015].

[5] Guidance Software. *EnCase Forensic, Pasadena, California.* `https://www2.guidancesoftware.com/products/Pages/encase-forensic/overview.aspx`. [Online; accessed 10-Dec-2015].

[6] Hairong Kuang Weiyan Wang. *Saving capacity with HDFS RAID.* `https://code.facebook.com/posts/536638663113101/saving-capacity-with-hdfs-raid/`. [Online; accessed 15-Nov-2015].

[7] Oluwatosin Alabi, Joe Beckman, Melissa Dark, and John Springer. "Toward a Data Spillage Prevention Process in Hadoop using Data Provenance." In: *Proceedings of the 2015 Workshop on Changing Landscapes in HPC Security*. ACM. 2015, pp. 9–13.

[8] Joe Sremack. *Big Data Forensics–Learning Hadoop Investigations*. Packt Publishing Ltd, 2015.

[9] Pradeep Subedi, Ping Huang, Benjamin Young, and Xubin He. "FINGER: A novel erasure coding scheme using fine granularity blocks to improve Hadoop write and update performance." In: *Networking, Architecture and Storage (NAS), 2015 IEEE International Conference on*. IEEE. 2015, pp. 255–264.

[10] George Trujillo, Charles Kim, Steve Jones, Rommel Garcia, and Justin Murray. *Virtualizing Hadoop: How to Install, Deploy, and Optimize Hadoop in a Virtualized Architecture*. en. VMWare Press, July 2015. ISBN: 9780133811131.

[11] Joel Reardon, David Basin, and Srdjan Capkun. "Sok: Secure data deletion." In: *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE. 2013, pp. 301–315.

[12] José Valerio, Pierre Sutra, Étienne Rivière, and Pascal Felber. "Evaluating the Price of Consistency in Distributed File Storage Services." In: *Distributed Applications and Interoperable Systems*. Springer. 2013, pp. 141–154.

[13] Sarah Diesburg, Christopher Meyers, Mark Stanovich, Michael Mitchell, Justin Marshall, Julia Gould, An-I Andy Wang, and Geoff Kuenning. "TrueErase: Per-file secure deletion for the storage data path." In: *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM. 2012, pp. 439–448.

[14] Tom White. *Hadoop: The definitive guide.* " O'Reilly Media, Inc.", 2012.

[15] Adam Yee and Jeffrey Shafer. "Hfaa: a generic socket api for hadoop file systems." In: *Proceedings of the 2nd Workshop on Architectures and Systems for Big Data*. ACM. 2012, pp. 15–20.

[16] Christiaan Beek. "Introduction to file carving." In: *White paper. McAfee* (2011).

[17] Daniel Crawl, Jianwu Wang, and Ilkay Altintas. "Provenance for MapReduce-based Data-intensive Workflows." In: *Proceedings of the 6th Workshop on Workflows in Support of Large-scale Science*. WORKS '11. Seattle, Washington, USA: ACM, 2011, pp. 21–30. ISBN: 978-1-4503-1100-7. DOI: 10.1145/2110497.2110501. URL: http://doi.acm.org/10.1145/2110497.2110501.

[18] Joel Reardon, Claudio Marforio, Srdjan Capkun, and David Basin. "Secure deletion on log-structured file systems." In: *arXiv preprint arXiv:1106.0917* (2011).

[19] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. "The hadoop distributed file system." In: *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE. 2010, pp. 1–10.

[20] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[21] Jaeheung Lee, Junyoung Heo, Yookun Cho, Jiman Hong, and Sung Y Shin. "Secure deletion for NAND flash file system." In: *Proceedings of the 2008 ACM symposium on Applied computing*. ACM. 2008, pp. 1710–1714.

[22] Kyoungmoon Sun, Jongmoo Choi, Donghee Lee, and Sam H Noh. "Models and design of an adaptive hybrid scheme for secure deletion of data in consumer electronics." In: *Consumer Electronics, IEEE Transactions on* 54.1 (2008), pp. 100–104.

[23] Simson L Garfinkel. "Carving contiguous and fragmented files with fast object validation." In: *digital investigation* 4 (2007), pp. 2–12.

[24] Avantika Mathur, Mingming Cao, Suparna Bhattacharya, Andreas Dilger, Alex Tomas, and Laurent Vivier. "The new ext4 filesystem: current status and future plans." In: *Proceedings of the Linux Symposium*. Vol. 2. Citeseer. 2007, pp. 21–33.

[25] Richard Kissel, Matthew Scholl, Steven Skolochenko, and Xing Li. "Guidelines for media sanitization." In: *NIST special publication* 800 (2006), p. 88.

[26] Nikolai Joukov and Erez Zadok. "Adding secure deletion to your favorite file system." In: *Security in Storage Workshop, 2005. SISW'05. Third IEEE International*. IEEE. 2005, 8–pp.

[27] Brian Carrier, Eugene H Spafford, et al. "Getting physical with the digital investigation process." In: *International Journal of digital evidence* 2.2 (2003), pp. 1–20.

[28] Steven Bauer and Nissanka Bodhi Priyantha. "Secure Data Deletion for Linux File Systems." In: *Usenix Security Symposium*. Vol. 174. 2001.

[29] Peter Gutmann. "Secure deletion of data from magnetic and solid-state memory." In: *Proceedings of the Sixth USENIX Security Symposium, San Jose, CA*. Vol. 14. 1996.

# Paper IV: Adaptive Anomaly Detection in Cloud using Robust and Scalable Principal Component Analysis

# Adaptive Anomaly Detection in Cloud using Robust and Scalable Principal Component Analysis

**B. Agrawal**[1], **T. Wiktorski**[1], **C. Rong**[1]

[1] Department of Electrical Engineering and Computer Science, University of Stavanger

**Abstract:**
Cloud computing has become increasingly popular, which has led many individuals and organizations towards cloud storage systems. This move is motivated by benefits such as shared storage, computation and, transparent service among a massive number of users. However, cloud-computing systems require the maintenance of complex and large-scale systems with practically unavoidable runtime problems caused by hardware and software faults. Large systems are very complex due to heterogeneity, dynamicity, scalability, hidden complexity, and time limitations. This paper proposes a scalable model for automatic anomaly detection on a large system like a cloud. The anomaly detection process is capable of issuing a correct early warning of unusual behavior in dynamic environments after learning the system characteristic of normal operation. In this paper, we propose an adaptive anomaly detection mechanism, which investigates principal components of the performance metrics. It transforms the performance metrics into a low-rank matrix and calculates the orthogonal distance using the Robust PCA algorithm. The proposed model updates itself recursively, while learning and adjusting the new threshold value, to minimize reconstruction errors. This paper also investigates robust principal component analysis in distributed environments using Apache Spark as the underlying framework. It specifically addresses cases in which normal operation might exhibit multiple hidden modes. The accuracy and sensitivity of the model were tested on Amazon CloudWatch datasets, and Yahoo! datasets. The model achieved an accuracy of 88.54%.

# 1 Introduction

Cloud computing is becoming increasingly pervasive and trending to be complicated. For the systems of this massive scale, reliability becomes a major concern for the system administrator who manages servers. Automation is needed to monitor such a large system. This requires monitoring to gain insights into the operation of the hardware, systems, and applications running in the cloud. Monitoring the system is the key to track system behavior and detect unusual behavior. Anomalies are identified as unusual behavior based on performance issues, failures (hardware or software), and configuration issues. An anomaly can cause unexpected behavior and result in reduced efficiency or even downtime of the data center. Current data centers use thousands of virtual machines that require dynamic resource scheduling to operate efficiently and cost-effectively. These data centers need to meet the changing demand for various resources like CPU and memory. A scheduler must allocate or re-allocate these resources dynamically. Therefore, knowing the resource utilization helps detect unusual behavior. In other words, it is critical to monitor the server metrics (e.g., latency, CPU, memory, disk I/O), represented by a time series, for any unusual behavior. Early detection of unusual behavior of performance metrics is critical to take preemptive action to protect users and provide a better user-experience.

The performance data generated by these data centers is unstructured, of high velocity, and a high volume that needs to be processed in an efficient way. Moreover, the high volume needs to be handled with ease and in a scalable manner. Scalability is a major requirement and a problem for most anomaly detection tools. Popular Big Data frameworks like Hadoop [16], MapReduce [31], HBase [21], Apache Spark [11], etc. address the scalability issue. Apache Spark performs in-memory computation and has an advanced DAG (Directed Acyclic Graph). Spark is 100 times faster than MapReduce in memory and 10 times faster on disk [1].

In this paper, we propose a self-adaptive anomaly detection method to detect unusual behavior. Our method analyzes the log files and calculates reconstruction errors that adjust the threshold value, which helps to detect anomalies accurately. Our method consists of 5 steps: (1) preprocessing, (2) metric collection (3) feature extraction, (4) prediction, and (5) anomaly detection. We introduce an efficient and distributed anomaly detection algorithm that uses mild assumptions at uncorrupted points.

[11] https://spark.apache.org/

The algorithm recovers the optimal low-dimensional subspace and identifies the corrupted points. Our technique employs matrix decomposition using SVD (Singular Value Decomposition). The PCA is transformed into a low-rank approximation to data matrix using the lower dimensional approximating subspace via SVD. The anomalous data are normally with a higher magnitude and variance in the projection plane. The evaluation of our model is done with data collected from Amazon CloudWatch, which consists of five different geo-located datacenters and many thousands of jobs and tasks. The accuracy of the model is calculated using precision and recall metrics. Our model achieves an accuracy of 87.24% when tested on Yahoo! time-series datasets [5], and an accuracy of 88.54% on Amazon CloudWatch server metrics.

## 1.1 Our Contribution

We propose a real-time and self-adaptive anomaly detection technique in a distributed environment using Spark as an underlying framework to detect anomalies in the cloud infrastructure. An adaptive algorithm is introduced, which uses reconstruction errors to determine the sample size and update the threshold value. The accuracy of our methodology is evaluated using AWS CloudWatch server metrics and Yahoo! anomaly datasets.

## 1.2 Related Work

Most of the monitoring tools in a data center use a fixed threshold technique to detect anomalies. Principal Component Analysis (PCA) has been used in many research work on anomaly detection [39], [37]. Our work is mainly based on the work done on KDD 99 datasets [39], for which the authors proposed and successfully employed a PCA based classifier, to filter out anomalies in a 34-dimensional connection record dataset. This method was used in the KDD Cup 1999 [3] - a classifier learning context. Furthermore, the authors took a first step forward towards robust approach for their detector. The major drawback of the approach is the feature engineering, which compiled features of KDD datasets that are hardly available in real-life. There are other traditional techniques to detect anomalies, which are used to monitor cloud infrastructures. These include threshold-based, statistical, and entropy-based techniques. MASF [42] is a threshold-based technique that operates on the hourly, daily or weekly data segments. However, this technique compromised on accuracy and false alarm rates.

Entropy-based Anomaly testing (EbAT) [29] is a novel technique for detecting anomalies in cloud computing systems and analyzes a metric's distribution. Many studies were done on anomaly detection, and they are typically based on statistical techniques [38], [24], [27], [33]. Unfortunately, most of these are not scalable and cannot operate at the scale needed for future data center and cloud computing. Moreover, most of them are fixed threshold techniques and require prior knowledge about applications and service implementations. In addition, a few of them deal only with a particular problem at specific levels of abstraction.

Pandeeswari et al [7] proposes a mixture of Fuzzy C-Means clustering algorithm and Artificial Neural Network (FCM-ANN) for anomaly detection in the cloud environment and also compare with Naive Bayes classifier and Classic ANN. Auto-regression(AR) based statistical methods for online monitoring time-series data to detect anomalies for applications running on networked cloud systems [6]. However, the evaluation of this approach is done on DARPA's KDD cup dataset 1999 [3] that does not match current cloud infrastructures. Yu et al [15] has proposed a scalable approach to anomaly detection based on hierarchical grouping and a non-parametric diagnostic mechanism using Hadoop and MapReduce. Similarly, Gupta et al. [14] used Hadoop to convert logs into time-series and applied data mining technique to extract anomalies. Their approaches achieve scalability, but are not readily extensible for real-time processing. Additionally, their approach deals with particular types of problems. Apache Spark, being faster than MapReduce, has been used as an underlying framework for our approach.

The proposed technique is distributed, scalable, and adaptive in nature. Moreover, it improves over time as it learns about the workload characteristics that enhances accuracy and reduces the number of false alarms. It is scalable, i.e. it meets the requirement of future data centers, and it can process the massive amount of logs. In our study, we investigate a five-step method, including prediction, classification, and RPCA (Robust Principal Component Analysis) for feature extraction to find anomalies.

## 1.3 Paper Structure

Section II gives an overview of the background. Section III introduces the design and approach. Section IV evaluates our algorithm and presents the results. Section V concludes the paper.

# 2 BACKGROUND

## 2.1 Robust PCA (Principal Component Analysis):

PCA is a linear transformation that maps a given set of data points into new axis (i.e. principal components). It is used in the dimensional reduction technique. In the classical PCA, the eigenvectors and eigenvalues [45] are calculated from the sample covariance matrix using Euclidean distances between sample data points [36]. In RPCA, robust covariance estimation is used for eigen decompositions. The decomposition of a low-rank matrix from a set of observations with gross sparse errors is known as robust *principal component analysis* (RPCA) [18]. The robust PCs represent the data effectively in a lower-dimensional space. The anomalies are detected in this lower-dimensional space using distance measured; orthogonal distance, which is the distance of an observation to the PCA space. It has many applications in computer vision, image processing, and data ranking. In addition, if the observed data has been contaminated by a dense noise in addition to gross sparse errors, RPCA is used to get a low-rank matrix.

## 2.2 Spark:

Apache Spark [12] is an open-source distributed framework that has recently become popular for data analytics. Similar to Hadoop, it is fault-tolerant and supports distributed computation systems to process fast and large streams of data. It uses Hadoop distributed file system to store and read data. It provides in-memory cluster computing that allows user to load data into a cluster's memory, which in turn makes it perform up to 100 times faster than Hadoop MapReduce. Apache Spark introduced the concept of Resilient Distributed Datasets (RDD) [17], which is a distributed memory abstraction that allows in-memory computation on large distributed clusters with high fault-tolerance [11]. It enables efficient data reuse that lets users explicitly persist intermediate results in memory. RDDs are a good fit for many parallel applications. RDDs is used in iterative in-memory operations where data is read multiple times and manipulated using a rich set of operators. [1].

[12] Apache Spark; [http://spark.apache.org/]

### 2.3 Hadoop:

Hadoop [13] [16] is an open-source framework for distributed storage and data-intensive processing, first developed by Yahoo! [14]. It consisted of two core projects: Hadoop Distributed File System (HDFS) [25] and MapReduce programming model [31]. HDFS is a distributed file system that splits and stores data on nodes throughout a cluster, with a number of replicas. It provides an extremely reliable, fault-tolerant, consistent, efficient and cost-effective way to store a large amount of data. The MapReduce model consists of two key functions: Mapper and Reducer. The Mapper processes input data splits in parallel through different map tasks and sends sorted, shuffled outputs to the Reducers that in turn groups and processes them using a reduce task for each group [8] [10].

## 3 APPROACH

To improve efficiency of cloud platforms, it is necessary to provide a high degree of transparency in the production systems that has a capability of horizontal scaling. Data from Amazon CloudWatch [15] is collected and stored in OpenTSDB [16] in a real-time. Figure 1 shows the steps involved in the metrics collection from Amazon CloudWatch into OpenTSDB. Different instances of Amazon EC2 services with various geo-locations are created, and different user applications are simulated on these instances. Amazon CloudWatch collects all the server metrics and sends them to OpenTSDB in near real-time. The data are loaded into our RSPCA model, which detects unusual patterns and alerts the user. The figure depicts the entire data collection pipeline starting from Amazon CloudWatch to the anomaly detection model. Different metrics received are staged in realtime, and then written to a queue (Apache Kafka [17]) for real-time using spark streaming as well as to OpenTSDB/HBase for longer-term storage. The anomaly detection model (RSPCA) consumes the dataâĂŤeither in real-time (Apache Kafka) or in batch (HBase). The overall architecture of the system follows lambda architecture [18].

[13] Hadoop; [http://hadoop.apache.org/]

[14] Yahoo! Developer Network, (2014), Hadoop at Yahoo!, [http://developer.yahoo.com/hadoop/]

[15] https://github.com/prelert/engine-python

[16] http://opentsdb.net/

[17] https://kafka.apache.org/

[18] http://lambda-architecture.net/

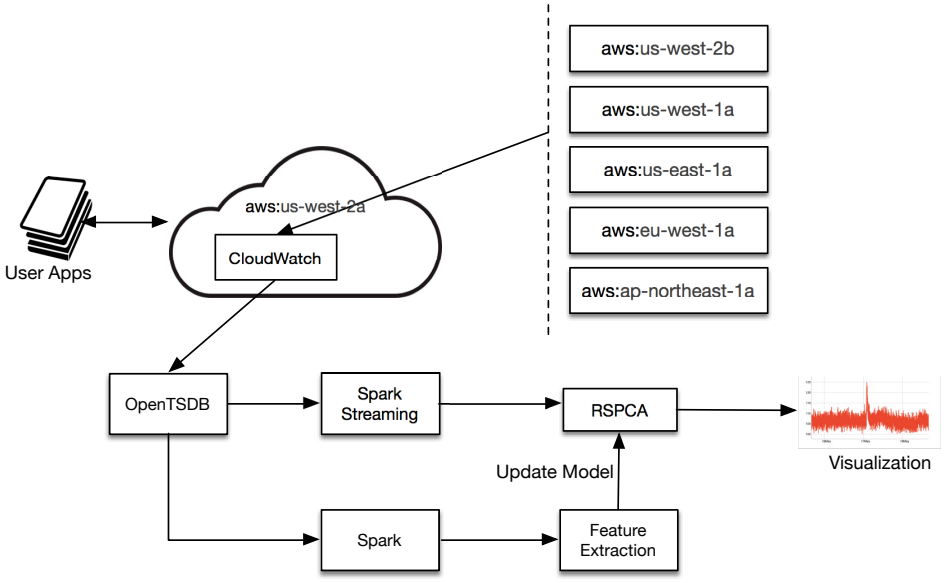



**Figure 1:** AWS CloudWatch metrics are collected and stored in OpenTSDB from Amazon CloudWatch, in real-time

We choose Spark streaming for real-time data and Spark for batch layer (see Fig. 1). The system reads the data from HBase in the batch layer. The batch jobs runs at a regular interval that computes and update the sample for our model. Spark Streaming is used for processing data streams that it receives directly from Amazon CloudWatch. The batch layer perform feature extraction and send to the model that detect anomalies and at the same time it update the sample from batch layer.The detailed steps of our model are explained later in this section.

Figure 2 illustrates the CPU and memory utilization of the tasks in Amazon CloudWatch. The data are sampled every second and collected over a period of 10 hours. Each task consumes different amounts of memory and CPU. However, there is a trend in the behavior of memory and CPU utilization. Any trends and periodicity in the graphs are detected using Fast Fourier Transformation [44]. An anomalous segment is identified when an application cannot explain the observed CPU and memory utilization. This may happen due to an unknown background process that consumes CPU and memory resources either at a constant rate or randomly. It is important to detect and filter out such events. Any unusual event also provides insight into different problems with the possibility to correct

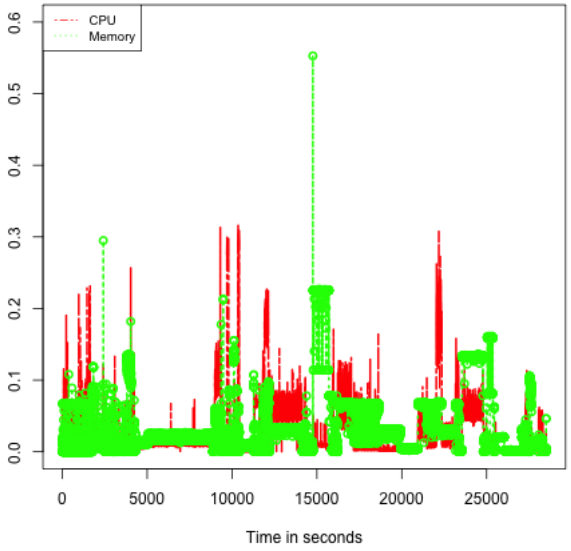



**Figure 2:** CPU and memory-utilization time-series of AWS CloudWatch.

them.

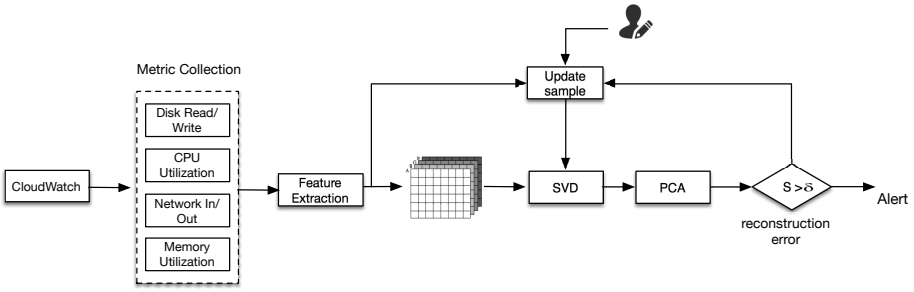



**Figure 3:** Steps involved in anomaly detection.

Our approach involves several important steps: pre-processing, metric collection, feature extraction, error rate calculation, and anomaly detection. Anomaly detection alone also is done in several steps - see Figure 3. The first step is the aggregation of log files and their storage in OpenTSDB. The log files are pre-processed into a form that can be read with our model. Different types of metrics are collected using the log files, e.g., for this paper,

we collected resource utilization (CPU utilization, memory utilization, and disk I/O). Pre-processing steps were performed for each type of metric. First, trends (seasonal, cyclic, and trend) on each metric were extracted using Fourier transformation and time series decomposition [32]. Each metric is then converted to a data matrix and decomposed to a low-rank representation using Singular Value Decomposition (SVD). Robust PCA is used to separate outlier and low-rank representations from the original data. An adaptive technique uses the mean reconstruction error as a measure to determine the update threshold value. A predefined threshold $\delta$ is used to determine when an update is required. If the value is greater than the threshold value ($\delta$), it is compared with detected anomalies, and the threshold value for anomalies is updated. The steps involved in the anomaly detection are:

*Pre-processing data:* The log data from each cloud instances are pre-processed into a form that is suitable for the model. Each sample value is transformed into a normalized form by dividing the sample value by the mean of all samples. After normalization has been completed, the normalized sample values are binned with each other. For example, CPU and memory are binned together for the same time interval. This produces a vector result, e.g. at time $t$ : $E_t =< C_t, M_t >$, where $C_t$ is the CPU utilization at time $t$ and $M_t$ is the Memory utilization at time $t$.

*Metrics collection and selection:* There are different metrics available in the data, and it is hard to identify all metrics, it is necessary to select an optimal subset of metrics. We collect a uniform set of metrics from the nodes and concatenate them into one matrix, $X$. In this paper, we collected several metrics from the data center, including CPU, memory, disk I/O, and page cache. For example, there might be a memory leakage that may affect the CPU utilization rate and other resources in the system. To detect an anomaly, we collected the traces from AWS CloudWatch âĂŞ see Table 4.1.

*Feature extraction:* The time series $X_t$ might consist of three components: a seasonal component, a trend-cycle component, and a remainder component. The time series is often decomposed into 3 sub-time series:

- ***Seasonal***: patterns that repeat with a constant period. For example, weather data have seasonality, and every winter and summer there is a similar pattern of temperature. The Fast Fourier Transform is a good tool to detect seasonality if enough historical data are provided.

**Table 4.1:** List of collected metrics.

| Index | Metric |
|---|---|
| 1 | CPU rate |
| 2 | Canonical memory usage |
| 3 | Assigned memory usage |
| 4 | Unmapped page cache |
| 5 | Total page cache |
| 6 | Maximum memory usage |
| 7 | Disk I/O time |
| 8 | Local disk space usage |
| 9 | Maximum CPU rate |
| 10 | Maximum disk I/O time |

- ***Trend***: the underlying trend or pattern of the metrics. For example, a stock-market trend, whichhas up or down patterns of stocks.
- ***Random***: (noise or irregular) is the residual of a time series after allocation into the seasonal and trends time series. This is an error term to determine if a time-series contains an anomaly. It is a remainder from the trend and the seasonality, which is an error term.

For example, in an additive model, the time series can be written as

$$X_t = S_t + T_t + E_t \tag{4.1}$$

where $X_t$ is the time series at period $t$, $S_t$ is the periodically component at period $t$, $T_t$ is the trend-cycle component at period t and $E_t$ is the remainder (irregular or error) component at period $t$.

*Anomaly detection:* We need to predict consumption to detect unusual behavior in the resource utilization of large-scale distributed systems such as cloud computing and data centers. Let $X_t$ be the vector that represents the measurement of CPU usage at time $t$. The data center consists of a large number of nodes ($n$ is large), and a vector is represented as vector $X_n$.

We use principal component analysis to identify the resource usage patterns, and then conduct a prediction on a subset of these principal components. PCA is a popular statistical technique for data analysis and dimensionality reduction. However, it is fragile with respect to the corrupted input data matrix, which often threatens its validity. A corrupted entry in $X$ could render the estimated low-rank representation $L'$ very different from the true low-rank representation $L$. PCA uses the Singular Value Decomposition (SVD) to find low-rank representations of the data. The robust version of PCA, RPCA, identifies a low-rank representation, random noise, and a set of outliers by repeatedly calculating the SVD and applying thresholds to the singular values as well as an error for each iteration. A matrix decomposition algorithm decomposes the input matrix $X$ into the sum of three parts $X = L + S + E$ using Robust Principal Component Pursuit [28]. Where, $L$ is a low-rank representation matrix illustrating a smooth $X$, $S$ is a sparse matrix containing corrupted data, and $E$ is noise. If a matrix $X$ consists of trends, we represent the trend in each column. For example, weekly seasonality would be where each row is a day of a week, and one column is one full week.

The low-rank matrix $L$ is calculated using the SVD of $X$ and using a threshold as the singular value [4]. This approach allows us to detect multiple anomalies simultaneously, which makes the method more robust. There are many techniques available for anomaly detection, but most of them e.g. regression and moving averages (ARIMA), are not robust when two or more different types of anomalies are present.

In the conventional approach, the first principal component corresponds to the projected observation with the largest variance. The accuracy of the conventional approach depends on the estimation of the covariance matrix from the data, which is very sensitive for unusual observations. We assume a large data matrix $X$ decomposes into $L$ and $S$ from the classical PCA definition given in Equation 4.2:

$$X = L + S \tag{4.2}$$

where,

- $L$ has a low-rank representation matrix.
- $S$ is a sparse matrix.
- $X$ is a data matrix.

We denote a $mxn$ data matrix as $X \epsilon \mathbb{R}^{mxn}$, $X_{i,j}$ denotes the $(i,j)_{th}$ entry of $X$. Singular value decomposition (SVD) is the most commonly used tools for low-rank decomposition. SVD decomposes matrix L into three factors: $U$, $V$, and $S$ shown in equation 4.3:

$$L_k = \sum_{i=1}^{l} (S_i U_i V_i^T) \tag{4.3}$$

where,

- $U$ is an $m$ x $m$ orthogonal matrix of the left singular vectors of $X$.
- $V$ is an $n$ x $n$ orthogonal matrix of the right singular vector of $X$.
- $S$ is the vector of singular values of $X$.
- $l = min(m, n)$
- $K$ is $0 \leq K \leq rank(X)$

**Algorithm 2** Robust Outlier Detection Algorithm

```
Input X = {x_1, x_2, ..x_n}                          ▷ data matrix
Input e                                  ▷ Maximum number of outlier
Input δ                                    ▷ Pre-defined threshold
while not converged do
    SVD:
    L = Σ_{i=1}^{l} (S_i U_i V_i^T)
    S = argmin_S ||E − S|| , E = X − L              ▷ Calculate anomalies
    if S > δ then
        if S > y.predicted then
            Update δ
        end if
    end if

end while
return S,L          ▷ S = Anomalies, L = Low-rank approximation
```

In a general form, low-rank matrix representation can be written as in equation 4.4

$$min_L \left\| X - L \right\|_F \tag{4.4}$$

where, $\|.\|_F$ is the Frobenius norm [43], L is the low-rank approximation to X, and K is the maximal rank of L. L is solved with the help of the following optimization problem in equation 4.5

$$\begin{aligned} &\min_{L,S} \|L\|_* + \lambda \|S\|_1 \\ &subj \qquad X = L + S \end{aligned} \tag{4.5}$$

where $\|\|_*$ and $\|L\|_1$ are the nuclear norm and $l_1$ norm, respectively, and $\lambda > 0$ is a balanced parameter. The optimization problem in Equation 4.2 can be solved as a convex optimization problem [20]. However, this process converges extremely slowly. It does not scale well for large matrices because they maintain the high-order information. To overcome the scalability problem, the first-order information is used [30]. By excluding the outliers from the effort of low-rank approximation, we can ensure the reliability of the estimated low-rank structure. The outlier presented in the sparse matrix $S$ contains significant variance that is calculated in the algorithm 2.

After the sample has been selected, the reconstruction error for the data is calculated. The ratio of the mean reconstruction error of the training set to the mean reconstruction error of the new sample is calculated. If $\sigma_{ratio} > \delta$, then the current data is not well-represented by the current model. The sample input is updated such that model can predict accurately.

# 4 EMPIRICAL EVALUATION

*Setup:* Our cluster is comprised of 6 nodes with Ubuntu 14.04: one node for Namenode, Job Tracker, Zookeeper and second node for Secondary Namenode. The remaining 4 nodes act as Data Nodes, and Task Trackers. All nodes have an AMD Opteron(TM) 4180 six-core 2.6GHz processor, 16 GB of ECC DDR-2 RAM, 3x3 TeraBytes secondary storage and HP ProCurve 2650 switch. Experiments were conducted using Apache Spark, Hadoop-0.20 releases, and OpenTSDB 2.2. Our default HDFS configuration had a block size of 64 MB and the replication factor of 3.

The real scenario dataset comes from Google [22], Amazon, and Yahoo! and it represents the various server metrics (e.g., memory usage, disk i/o, CPU). The time series in the real dataset consists of malicious activities. The detection techniques presented in this paper have been applied to the real world scenarios, Amazon cloud, which generates CloudWatch performance and events traces that are stored in HBase using OpenTSDB.

Moreover, the accuracy and sensitivity of the model are verified using Yahoo! datasets [5] and AWS server metrics[19]. An example of AWS metric includes CPU Utilization, and disk read bytes. Different kind of web application was running on Amazon EC2 for which we collect monitoring logs. When the CPU load is high, the CloudWatch stops responding. Some of the incidents are unpredictable and unavoidable.

## 4.1 Anomaly Detection:

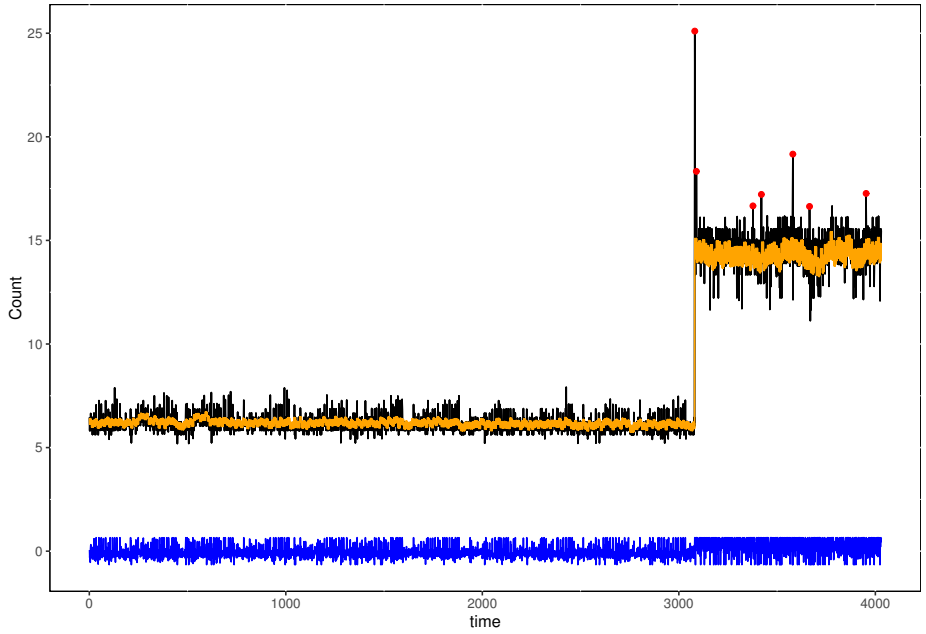



**Figure 4:** Anomaly detection on CPU utilization: The red dot indicates anomaly, the black line indicates original data (X), the orange line is random noise (E) and the blue line represents low rank signal (L).

In this section, we analyze the data we collected from the real-world production data centers during 29 days. Metrics such as CPU, disk I/O, and memory utilization were sampled every second. We chose CPU usage, disk I/O, and memory utilization to analyze the system behavior of the data center workloads. The CPU usage is defined by CPU utilization and IO wait ratio. It is also determined by the percentage of the time that a CPU, which waits for outstanding disk I/O, requests. Memory usage is

[19] Amazon; http://aws.amazon.com

defined as the percentage of memory used during CPU usage. In Figure 4, shows the number of anomalies detected by the model. The Robust PCA technique compares resource utilization with a lower and higher threshold bound. When the utilization exceeds the higher bound, unusual behavior is detected, and an "unusual activity" alarm is generated. The red dots are anomalies detected using the threshold value [27].

The complete historical data from OpenTSDB is loaded into our model to obtain the exact threshold value. From the historical trace, we calculate the lower and upper threshold bound, which represents unusual behavior outside the acceptable range. In other words, there is only about a 1% chance to have outliers from a normally-distributed time-series [34].

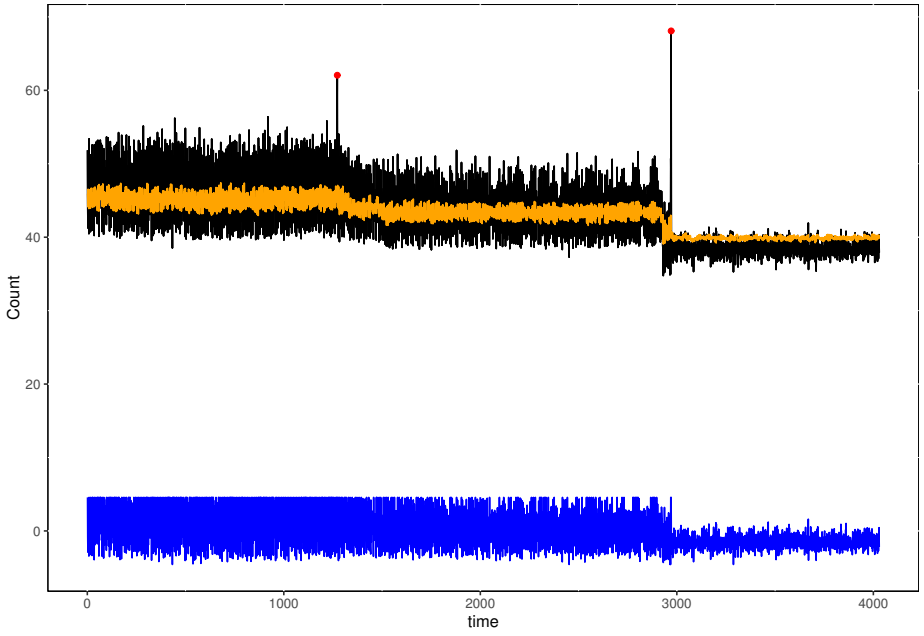
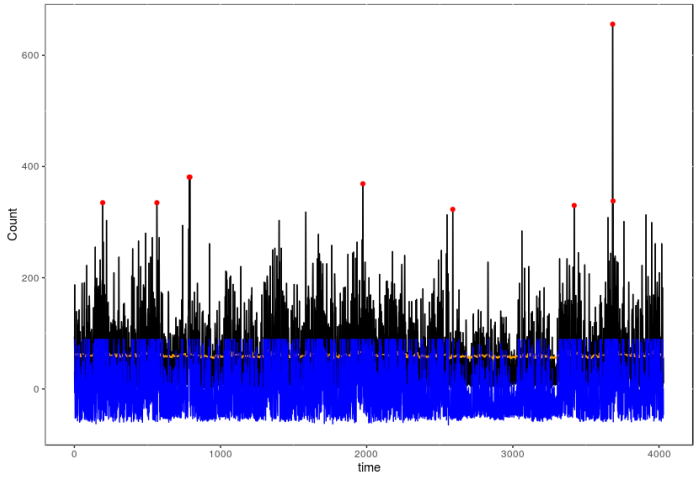
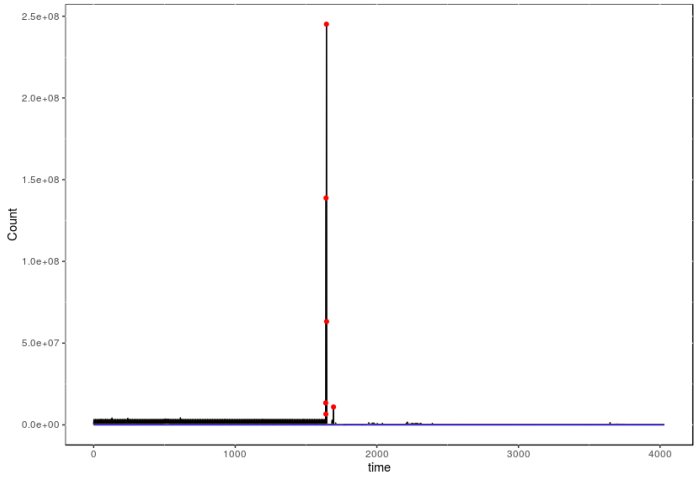



**Figure 5:** Memory usage anomaly detection: The red dot marks an anomaly, the black line indicates original data (X), the orange line is random noise (E), and the blue line represents a low rank signal (L).

Figure 5 illustrates memory-related anomaly detection. Like CPU utilization, anomaly detection in memory usage uses a threshold. There might be memory leakage in the system, which can lead to the detection of unusual behavior. The upper bound is detected using the threshold and the lower bound can be zero because the CPU can be in idle or sleep mode. The acceptance range is between 0 to the threshold value.

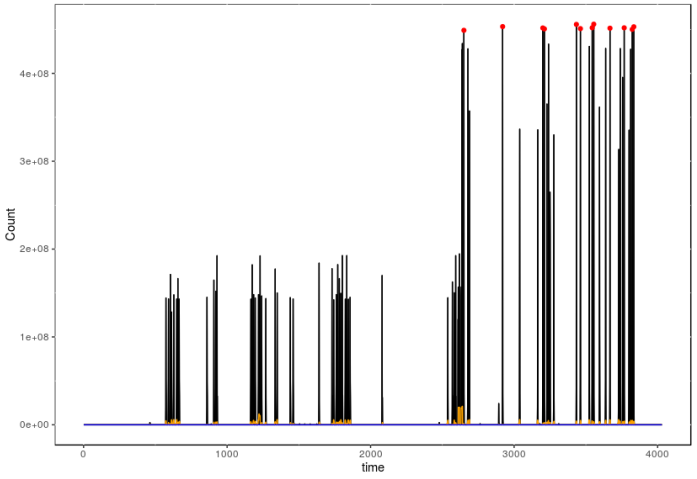



(a) Amazon EC2 number of request every 5 min.

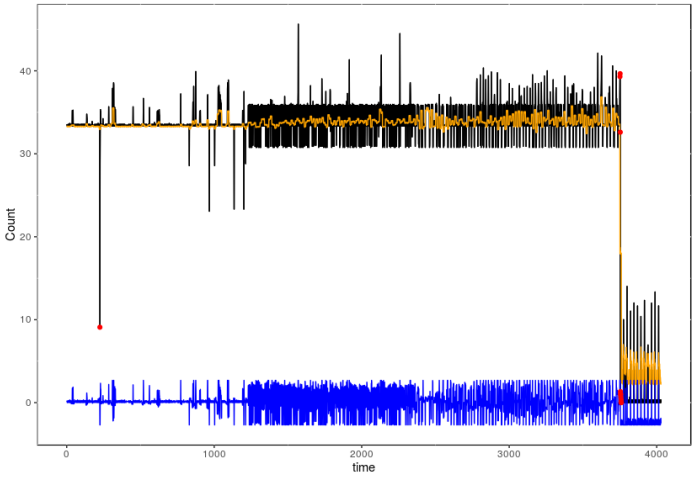



(b) AWS CloudWatch network traffic in.

**Figure 6:** Several metrics in AWS CloudWatch; red dot in the graph is abnormal behaviour. The accuracy of the overall system is 87%.




(a) AWS CloudWatch disk read in bytes.




(b) CPU and memory request together in CloudWatch.

**Figure 7:** Several metrics in AWS CloudWatch; red dot in the graph is abnormal behaviour. The accuracy of the overall system is 87%.

Figure 6 and 7 shows the anomaly detection in the Amazon CloudWatch data on different metrics. Figure 4.6(a) gives an overview of a number of requests made from various user applications on Amazon region us-west-2b. Similarly, Figure 4.6(b), 4.7(a) and 4.7(b) indicate abnormal behavior detected on network traffic, disk read/write, and resource request (CPU and memory) through CloudWatch logs.

## 4.2 Accuracy Test:

The models′ ability to precisely predict anomaly is evaluated by five metrics: precision, recall, false positive rate, true positive rate, and F-measure. These metrics are frequently used to evaluate the effectiveness of anomaly detection and have been used in many relevant types of research [35]. Five metrics are defined in table 4.2:

**Table 4.2:** Definition of the metrics.

| **Metric** | **Definition** |
|---|---|
| Precision | p = $TP/(TP + FP)$ |
| Recall | r = $TP/(TP + FN)$ |
| False positive rate | $fpr = FP/(FP + TN)$ |
| True positive rate | $tpr = TP/(TP + FN)$ |
| F-measure | $F = 2pr/(p + r)$ |

Higher precision ensures fewer false positive errors, while a model with high recall ensures fewer false negative errors. An ideal failure prediction model would achieve higher precision and recall value, i.e. precision = recall = 1. However, both high recall and precision are difficult to achieve at the same time. They are often inversely proportional to each other. Improving recall in most cases lowers precision and vice-versa. F-measure indicates whether the model is accurate or not. It ensures that both precision and recall are reasonably high.

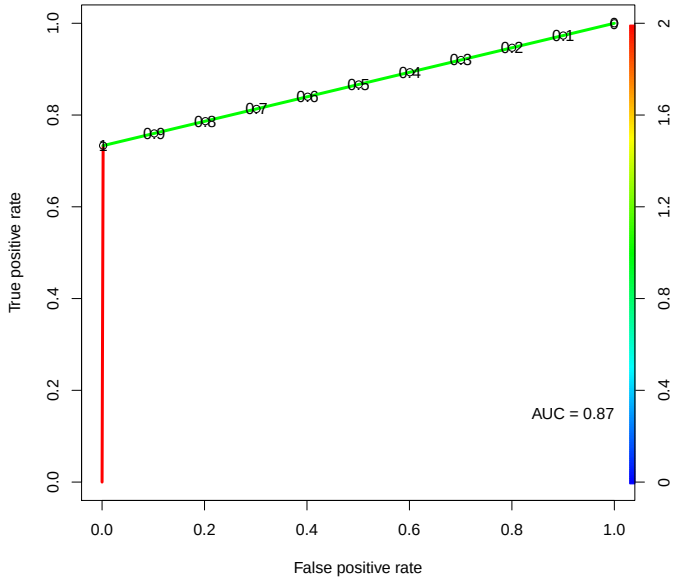



**Figure 8:** Performance of the proposed anomaly detector.

**Table 4.3:** Evaluation Metrics

| **Description** | **Value** |
|---|---|
| No. of anomaly detected | 13 |
| Total No. of anomaly | 15 |
| Recall | 0.80 |
| Precision | 0.85 |
| Accuracy | 87.24% |
| FPR | 0.1 |
| F-measure | 0.86 |

The anomaly detection is verified on labeled time-series Yahoo! datasets [5]. From the observation mentioned in Table 4.3, we used time-series for 1460 periods. In total, we have 1460 observations of which 13 anomalies

events have been detected using the model defined above. The actual number of anomaly present in the datasets is 15. The accuracy of the model is 87.24%. True positives are anomalies, which are identified correctly, and true negatives are anomalies that are not identified. Figure 8 shows the performance of our anomaly detection approach. The algorithm achieves a TPR of 73.3% and an FPR of 28.2%. The Area Under Curve (AUC) is 87.24%

### 4.3 Scalability test:

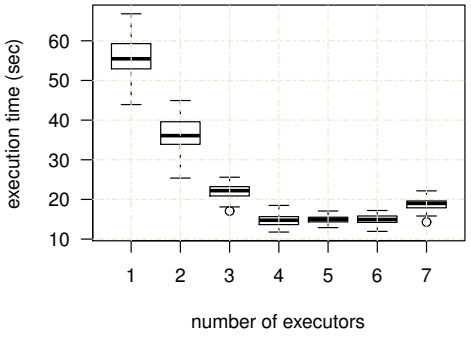
The technique proposed needs to be able to scale-out and efficient to cope with massive amounts of data generated by large system. To perform the evaluation, we vary the number of nodes (executors) and the volume of the incoming data along with historical data. In this experiment, we use two million time series of fixed-length (15GB), which were generated by Amazon CloudWatch using simulator to produce real-world scenario.

Distributed and parallel processing is a key feature of data-intensive applications. In order to evaluate the scalability of our approach, different number of nodes is used. Since we are using Spark on top of YARN, each executor needs to isolated memory partition.

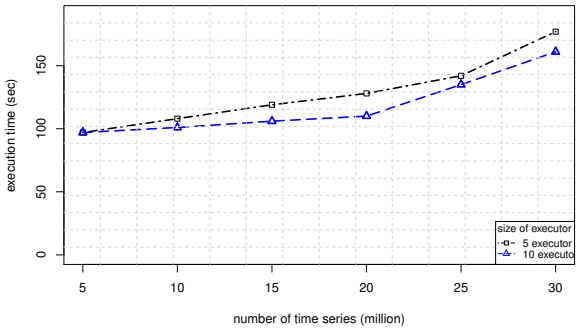
The data streams coming from Amazon CloudWatch is consumed by all the receivers in parallel. The receivers do not perform any task on the data. Since the data stream is coming every 10 minutes, we perform the experiment on micro-batches of data and historical data from OpenTSDB.

In order to evaluate the scalability of the model over large volumes of data, we compare different workloads. The number of executors was fixed to 7 and each executor was configured to 4GB of memory. Figure 4.9(a) shows the execution time with an increasing number of executors. Further, the processing time and the variance decrease when more executors are added. A low variance helps to guarantee an upper bound for the execution time. In fact, spark is fault tolerance which may require recomputing of some partitions and hence imposing extra overhead. The optimum execution time is reached at 6 executors. Increasing the parallelism beyond 7 executors does not increase the speed further.

Figure 4.9(b) evaluate the scalability of the algorithms over the incoming data streams with different workloads. We use the optimal number of parallel executors (5 and 10) for training our model with configuration of 4GB executor's memory. The incoming data varies from 5 to 30 million (the size of 10GB to 60GB). The execution time of varying workloads are displayed in Figure 4.9(b). The different configuration of executors observe




(a) Scalability and performance on data stream.




(b) Size-up experiment: execution time vs. number of collected data.

**Figure 9:** Scalability test with execution time per number of executors.

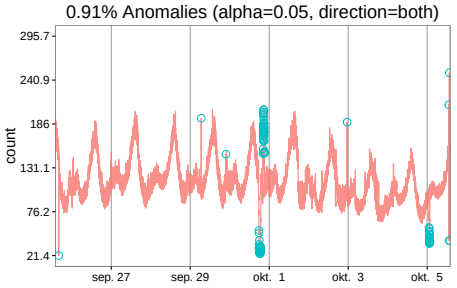
both configuration can scale and exhibits supra-linearly with the quantity of the time series. The total execution time of handling time-series takes less than 90 seconds to finish five million data points with optimized settings. The training process uses the full set of data from batch layer and generate new models each time while Spark Streaming runs periodical to process the data.

### 4.4 Benchmark test:

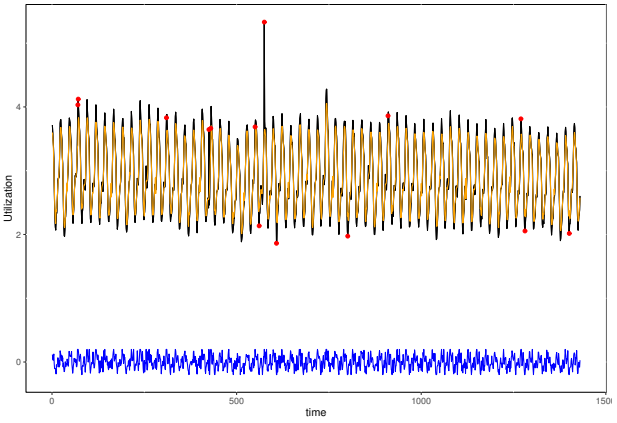
Two most popular anomaly detection algorithm were compared with our model. Figure 4.10(a) shows the anomaly detection technique [20] used by Twitter. Twitter uses Seasonal Hybrid ESD (S-H-ESD) as an underlying algorithm. It is built on the Generalized ESD test for detecting anomalies. The algorithm can detect both local as well as global anomalies. This is achieved by using a concept of time series decomposition and using robust statistical metrics together with ESD. For longer time series (say more than 6 months), the algorithm uses a piecewise approximation. It has an accuracy of 87% but has a high false-positive rate. Similarly, Netflix uses a Robust Anomaly Detection as a part of open source project called Surus [21]. Netflix uses PCA for detecting anomalies that has an accuracy of 76.42%.

The accuracy of our model is evaluated against other state-of-the-art anomaly detection techniques like SVM [41], DBSCAN [23] [2], and Incremental PCA [40]. Our approach first converts the principal components (PC)s into a low-rank matrix (L), and then separates it from a sparse matrix to distinguish noise. We compute the Mean Absolute Difference between time series to compare it with the baseline model. A low average similarity score time series is labeled as an anomaly [9]. For example, in Yahoo datasets, which consist of 1400 time series, 15 were unusual features. All the methods are evaluated in terms of false positive/negative and accuracy $= \frac{TP}{(TP+FP)} = \frac{No.ofcorrect}{Total}$ in real-world datasets. To evaluate the accuracy of our model, real-world data sets such as Yahoo! data, KDD intrusion detection dataset , ECG data and Amazon CloudWatch metrics were used. Figure 11 details the performance of other techniques such as SVM (Support Vector Machine) [13] [26], DBSCAN (Density-based spatial clustering of applications with noise) [19] and Incremental PCA [12] with compared to RSPCA. The figure shows that RSPCA is superior and more accurate. The RSPCA model performs better when the time-series data are random in nature. Our technique outperformed the baseline techniques because the low-rank space of the principal component and noise is well separated by its variance. We used popular conventional anomaly-detection algorithms as a baseline.

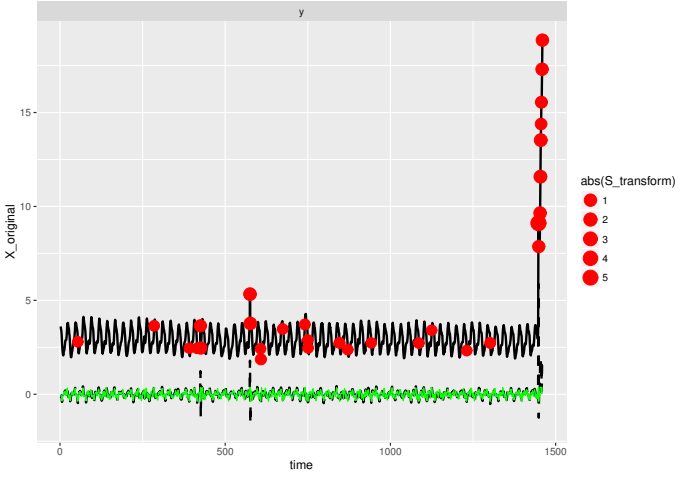
[20] https://github.com/twitter/AnomalyDetection
[21] https://github.com/Netflix/Surus
[21] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
[21] https://github.com/h2oai/h2o-2/tree/master/smalldata




(a) Twitter anomaly detection algorithm.




(b) Robust PCA algorithm.




(c) Netflix Robust Anomaly Detection algorithm using PCA.

**Figure 10:** Several anomaly detection algorithms tested on Yahoo datasets

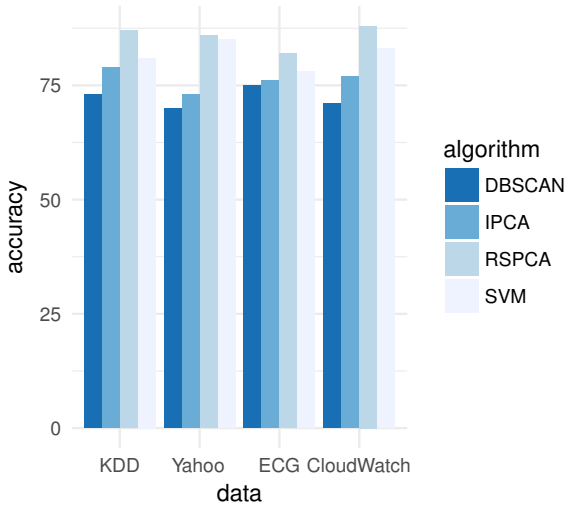



**Figure 11:** Average accuracy of our method compared to other approaches on different datasets.

## 5 Conclusion

Large-scale and complex cloud computing systems are susceptible to failures, which can significantly affect the cloud dependability and performance. In this paper, we present a real-time adaptive anomaly detection technique in cloud infrastructure. We collected all the performance metrics from Amazon CloudWatch logs and normalized it. Fast Fourier Transformation was used to detect a trend in the input time series and converted the time series into a matrix. Robust PCA was used to transform the original matrix into a low-rank representation using recursive SVD and a soft threshold. In this paper, a self-adaptive threshold approach is used. The threshold is updated during a learning phase. RPCA uses an efficient approach to decompose into low-rank representations using Spark as the underlying framework.

## References

[1] *Apache Spark - Lightning-fast cluster computing.* URL: `https://spark.apache.org/`.

[2] *DBSCAN clustering algorithm on top of Apache Spark.* URL: `https://github.com/alitouka/spark_dbscan`.

[3] *Knowlege Discovery and Data Mining Cup 1999 Data*. URL: http://www.ics.uci.edu/%20kdd/databases/kddcup99/kddcup99.html.

[4] MetaMarkets. *Algorithmic Trendspotting and the Meaning of Interesting*. URL: https://metamarkets.com/2012/algorithmic-trendspotting-the-meaning-of-interesting/.

[5] *Yahoo! labs*. URL: http://webscope.sandbox.yahoo.com/.

[6] Prathamesh Gaikwad, Anirban Mandal, Paul Ruth, Gideon Juve, Dariusz Król, and Ewa Deelman. "Anomaly detection for scientific workflow applications on networked clouds." In: *High Performance Computing & Simulation (HPCS), 2016 International Conference on*. IEEE. 2016, pp. 645–652.

[7] N. Pandeeswari and Ganesh Kumar. "Anomaly Detection System in Cloud Environment Using Fuzzy Clustering Based ANN." In: *Mob. Netw. Appl.* 21.3 (June 2016), pp. 494–505. ISSN: 1383-469X. DOI: 10.1007/s11036-015-0644-x. URL: http://dx.doi.org/10.1007/s11036-015-0644-x.

[8] Bikash Agrawal, Tomasz Wiktorski, and Chunming Rong. "Analyzing and Predicting Failure in Hadoop Clusters Using Distributed Hidden Markov Model." In: *Cloud Computing and Big Data: Second International Conference, CloudCom-Asia 2015, Huangshan, China, June 17-19, 2015, Revised Selected Papers*. Cham: Springer International Publishing, 2015. ISBN: 978-3-319-28430-9. DOI: 10.1007/978-3-319-28430-9_18. URL: http://dx.doi.org/10.1007/978-3-319-28430-9_18.

[9] Rob J Hyndman, Earo Wang, and Nikolay Laptev. "Large-scale unusual time series detection." In: *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. IEEE. 2015, pp. 1616–1619.

[10] Bikash Agrawal, Antorweep Chakravorty, Chunming Rong, and Tomasz Wiktor Wlodarczyk. "R2Time: A Framework to Analyse Open TSDB Time-Series Data in HBase." In: *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*. IEEE. 2014, pp. 970–975.

[11] Mohiuddin Solaimani, Mohammed Iftekhar, Latifur Khan, Bhavani Thuraisingham, and Joey Burton Ingram. "Spark-based anomaly detection over multi-source VMware performance data in real-time." In: *Computational Intelligence in Cyber Security (CICS), 2014 IEEE Symposium on*. IEEE. 2014, pp. 1–8.

[12] Meysam Alikhani and Mohammad Ahmadi Livani. "Dynamic anomaly detection by using incremental approximate PCA in AODV-based MANETs." In: *Journal of AI and Data Mining* 1.2 (2013), pp. 89–101.

[13] Yuting Chen, Jing Qian, and Venkatesh Saligrama. "A new one-class SVM for anomaly detection." In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE. 2013, pp. 3567–3571.

[14] Manish Gupta, Abhishek B Sharma, Haifeng Chen, and Guofei Jiang. "Context-aware time series anomaly detection for complex systems." In: *WORKSHOP NOTES*. 2013, p. 14.

[15] Li Yu and Zhiling Lan. "A scalable, non-parametric anomaly detection framework for hadoop." In: *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*. ACM. 2013, p. 22.

[16] Tom White. *Hadoop: The definitive guide.* " O'Reilly Media, Inc.", 2012.

[17] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." In: *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association. 2012, pp. 2–2.

[18] Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright. "Robust principal component analysis?" In: *Journal of the ACM (JACM)* 58.3 (2011), p. 11.

[19] Mete ÇElik, Filiz Dadaşer-çelik, and Ahmet Şakir Dokuz. "Anomaly detection in temperature data using dbscan algorithm." In: *Innovations in Intelligent Systems and Applications (INISTA), 2011 International Symposium on*. IEEE. 2011, pp. 91–95.

[20] Venkat Chandrasekaran, Sujay Sanghavi, Pablo A Parrilo, and Alan S Willsky. "Rank-sparsity incoherence for matrix decomposition." In: *SIAM Journal on Optimization* 21.2 (2011), pp. 572–596.

[21] Lars George. *HBase: the definitive guide.* " O'Reilly Media, Inc.", 2011.

[22] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. *Google cluster-usage traces: format + schema*. Technical Report. Revised 2012.03.20. Posted at URL `http://code.google.com/p/googleclusterdata/wiki/TraceVersion2`. Mountain View, CA, USA: Google Inc., Nov. 2011.

[23] Tran Manh Thang and Juntae Kim. "The anomaly detection by using dbscan clustering with multiple parameters." In: *Information Science and Applications (ICISA), 2011 International Conference on*. IEEE. 2011, pp. 1–5.

[24] Chengwei Wang, Krishnamurthy Viswanathan, Lakshminarayan Choudur, Vanish Talwar, Wade Satterfield, and Karsten Schwan. "Statistical techniques for online anomaly detection in data centers." In: *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*. IEEE. 2011, pp. 385–392.

[25] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. "The hadoop distributed file system." In: *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE. 2010, pp. 1–10.

[26] Vasilis A Sotiris, Peter W Tse, and Michael G Pecht. "Anomaly detection through a bayesian support vector machine." In: *Reliability, IEEE Transactions on* 59.2 (2010), pp. 277–286.

[27] Chengwei Wang, Vanish Talwar, Karsten Schwan, and Parthasarathy Ranganathan. "Online detection of utility cloud anomalies using metric distributions." In: *Network Operations and Management Symposium (NOMS), 2010 IEEE*. IEEE. 2010, pp. 96–103.

[28] Zihan Zhou, Xiaodong Li, John Wright, Emmanuel Candes, and Yi Ma. "Stable principal component pursuit." In: *Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on*. IEEE. 2010, pp. 1518–1522.

[29] Chengwei Wang. "EbAT: Online Methods for Detecting Utility Cloud Anomalies." In: *Proceedings of the 6th Middleware Doctoral Symposium*. MDS '09. Urbana Champaign, Illinois: ACM, 2009, 4:1–4:6. ISBN: 978-1-60558-852-0. DOI: `10.1145/1659753.1659757`. URL: `http://doi.acm.org/10.1145/1659753.1659757`.

[30] John Wright, Arvind Ganesh, Shankar Rao, Yigang Peng, and Yi Ma. "Robust principal component analysis: Exact recovery of corrupted low-rank matrices via convex optimization." In: *Advances in neural information processing systems*. 2009, pp. 2080–2088.

[31] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[32] Philippe Masset. "Analysis of financial time-series using Fourier and wavelet methods." In: *Available at SSRN 1289420* (2008).

[33] Hai Qiu, Neil Eklund, Xiao Hu, Weizhong Yan, and Naresh Iyer. "Anomaly detection using data clustering and neural networks." In: *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*. IEEE. 2008, pp. 3627–3633.

[34] Anthony McCluskey and Abdul Ghaaliq Lalkhen. "Statistics II: Central tendency and spread of data." In: *Continuing Education in Anaesthesia, Critical Care & Pain* 7.4 (2007), pp. 127–130.

[35] Felix Salfner and Miroslaw Malek. "Using hidden semi-markov models for effective online failure prediction." In: *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*. IEEE. 2007, pp. 161–174.

[36] Roland Kwitt and Ulrich Hofmann. "Robust Methods for Unsupervised PCA-based Anomaly Detection." In: *Proc. of IEEE/IST WorNshop on Monitoring, AttacN Detection and Mitigation* (2006), pp. 1–3.

[37] Khadija Houerbi Ramah, Hichem Ayari, and Farouk Kamoun. "Traffic anomaly detection and characterization in the tunisian national university network." In: *NETWORKING 2006. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*. Springer, 2006, pp. 136–147.

[38] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. "Using Magpie for Request Extraction and Workload Modelling." In: *OSDI*. Vol. 4. 2004, pp. 18–18.

[39] Mei-Ling Shyu, Shu-Ching Chen, Kanoksri Sarinnapakorn, and LiWu Chang. *A novel anomaly detection scheme based on principal component classifier*. Tech. rep. DTIC Document, 2003.

[40] Peter Hall, David Marshall, and Ralph Martin. "Adding and subtracting eigenspaces with eigenvalue decomposition and singular value decomposition." In: *Image and Vision Computing* 20.13 (2002), pp. 1009–1016.

[41] Binh Viet Nguyen. "An application of support vector machines to anomaly detection." In: *Research in Computer Science-Support Vector Machine, report* (2002).

[42] Jeffrey P Buzen and Annie W Shum. "Masf-multivariate adaptive statistical filtering." In: *Int. CMG Conference*. 1995, pp. 1–10.

[43] Changxue Ma, Yves Kamp, and Lei F Willems. "A Frobenius norm approach to glottal closure detection from the speech signal." In: *Speech and Audio Processing, IEEE Transactions on* 2.2 (1994), pp. 258–265.

[44] E Oran Brigham and Elbert Oran Brigham. *The fast Fourier transform*. Vol. 7. Prentice-Hall Englewood Cliffs, NJ, 1974.

[45] Carl S Rudisill. "Derivatives of eigenvalues and eigenvectors for a general matrix." In: *AIAA Journal* 12.5 (1974), pp. 721–722.

# Paper V:
# AFFM: Auto Feature Engineering in Field-Aware Factorization Machines for Predictive Analytics

# AFFM: Auto Feature Engineering in Field-Aware Factorization Machines for Predictive Analytics

**L. Selsaas[1], B. Agrawal[2], C. Rong[2], T. Wiktorski[2]**

[1] Gamut

[2] Department of Electrical Engineering and Computer Science, University of Stavanger

**Abstract:**
User identification and prediction is a typical problem with the cross-device connection. User identification is useful for the recommendation engine, online advertising, and user experiences. Extreme sparse and large-scale data make user identification a challenging problem. To achieve better performance and accuracy for identification a better model with short turnaround time, and able to handle extremely sparse and large-scale data is the needed. In this paper, we proposed a novel efficient machine learning approach to deal with such problem. We have adapted Field-aware Factorization Machine's approach using auto feature engineering techniques. Our model has the capacity to handle multiple features within the same field. The model provides an efficient way to handle the fields in the matrix. It counts the unique fields in the matrix and divides both the matrix with that value, which provides an efficient and scalable technique in term of time complexity. The accuracy of the model is 0.864845, when tested with Drawbridge datasets released in the context of the ICDM 2015 Cross-Device Connections Challenge.

**keywords– Predictive analytics, FFM, Factorization machines, cross-device connections**

# 1 Introduction

Predictive analytics is an important technique with applications in many fields ranging from business to science. Applications such as online adverting, e-commerce website, personalized search, social networks, etc., make use of predictive analytics and data mining to mine large-scale data to identify users. How to provide effective personalized recommendations while moving users across cross-devices, became one of the most important research topics in the past few decades. The accurate predictive modeling for identifying users on cross-devices are required. This identification will help in recommendation engine, online advertising and will improve user experiences.

*The task of ICDM 2015:* Drawbridge cross-device connections competition is to identify the users using different devices. Factorization Machines (FM) is used as the base model to predict the users. FM is a generic approach that allows to mimic most factorization models by feature engineering. The factorization machines combine the generality of the feature engineering with the superiority of factorization models in estimating interactions between categorical variables of a large domain.

**Figure 1:** User connected to multiple cross-devices.

*Problem Statement:* Now a day's consumers are using many devices to complete the online task or browse the internet. Consider the case, where a user wants to plan the holiday trip: he reads a travel blog on his smartphone on the way to work, search for hotels on his laptop after work, search for flight tickets on his tablet during dinner, search for trip advisor rated best restaurant in the area on his PC, and also download a travel book on his Kindle. As the users move across the devices to complete their online tasks, their identity becomes fragmented. The ads, recommendations, and messages are not always able to separate whether the activity on the different devices is tied to one user or many users. The model is required that can predict the users as they switch between devices (websites/mobile apps) as shown in figure 1.

## 1.1 Our Contribution

We propose an automated and scalable model for identifying the users across the cross-device connection. Our approach provides a novel way of feature engineering in FFM model. The learning approach used in our model provides an efficient learning rate compared to classical FFM. During the learning phase, the learning rate is divided by the number of features in the opposite field to balance out how much each weight is updated.

## 1.2 Paper Structure

Section II gives an overview of the background. Section III introduces the design and approach of our model. Section IV evaluates our algorithm and presents the results. Section V concludes the paper.

## 1.3 Related Work

In 2010, Rendle [8] introduced Factorization Machines (FM), which combine the advantages of Support Vector Machines (SVM) [9] with factorization models. However, with raw data the performance of FM is culpable. Feature engineering is required beforehand to use FM. Field-aware factorization machines (FFM) [2] have been used to win two Kaggle clickâĂŞthrough rate prediction competitions hosted by Criteo[22] and Avazu[23]. The performance of the model degrades with large matrices.

[22]https://www.kaggle.com/c/criteo-display-ad-challenge
[23]https://www.kaggle.com/c/avazu-ctr-prediction

Moreover, FFM requires pre-feature engineering on the raw data before predicting. Factorization Machines are available in many open source libraries such as GraphLab [4], Spark-libFM[24], and Bidmach [6]. The reason we adopted FFM because it provides an efficient way to solve the larger problems with better accuracy.

The challenges of track 2 of the KDD cup 2012 competition was to predict the click-through rate (CTR) of advertisements [7]. They also used featured engineering with several models ANN, probability models and collaborative filters. They used an interesting method of featured engineering to design their factor model. Another research on predicting clicks on ads on Facebook provides an interesting approach for featured engineering [5]. In the paper, it shows how the number of parameters has an impact on the overall system performance. However, it uses decision tree for prediction over featured engineering.

The approach mentioned in this paper provides an efficient way for the users to perform machine learning without caring much about feature engineering.

# 2 Background

## 2.1 FM (Factorization Machines):

Factorization machines are a new model class that combines the advantages of Support Vector Machines (SVM). Steffen Rendle [8] introduced this model in 2010. This model exhibits similar properties of SVM, but FM is used as general prediction tasks. Like other models, non-linear SVMs or decision trees, FM includes interactions between predictor variables. For example, FMs can learn that young users like to access from different devices, whereas preferences of old users are opposite.

## 2.2 Feature Engineering:

Feature engineering is the process of creating features that make machine learning more accurate and efficient using domain knowledge of the data.

[24] http://spark-packages.org/package/zhengruifeng/spark-libFM

# 3 Approach

As introduced in Section 1, FFM is used as our based model for predicting users across cross-devices. Factorization Machines (FM) are defined as mentioned in equation 4.1. A second-order FM for i-th feature vector $x_i$ of n x n matrix X is defined as in equation 4.1:

$$\widehat{y}(x) := w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < v_i, v_j > x_i x_j \tag{4.1}$$

where, $< v_i, v_j >$ is the dot product of two vectors of size k:

$$< v_i, v_j >:= \sum_{f=1}^{k} < v_{i,f}, v_{j,f} > \tag{4.2}$$

The field-aware factorization machines (FFM) is explained in equation 4.3:

$$\widehat{y}(x) := w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < v_{i,f1}, v_{j,f2} > x_i x_j \tag{4.3}$$

where, $< v_{i,f1}, v_{j,f2} >$ are two vectors with length k. $f_1$ and $f_2$ are respectively the fields of i and j.

The two vectors are compared, and their features are loaded into the model without any feature engineering. The datasets from the device, cookie and IPs are used for feature engineering. The datasets device and cookie are merged using common field *drawbridgeHandle*. As our model handles features, we can just load raw features in the model. The learning model reads the features of the matrices and updates the learning rate. There are 42 different types of features used out of which seven features are shown as an example in the figure 2. Equation 4.4 defines the mapping of two vectors field and feature that is derived from FFM definition:

$$\widehat{y}(x) := \sum_{i=1}^{n} \sum_{j=i+1}^{n} < v_{i,f1}, v_{j,f2} > x_i x_j \tag{4.4}$$

The datasets are fed to the model. Before sending the data, the cookies and devices are merged. On the training phase, it calculates the factors in the feature vector and field vector. In the update, the learning rate is calculated, and learning of the model with each field present in the cell is calculated. The update algorithm performance is quite efficient. The

| Field name | Field Index | Feature Name | Feature Index |
|---|---|---|---|
| Device ID | 1 | id_1000002 | 1 |
| Cookie ID | 2 | Id_10 | 2 |
| Comp OS/type | 3 | Devos_7 | 3 |
| | 3 | Devos_72 | 4 |
| Device type | 4 | Devtype_2 | 5 |
| | 4 | Devtype_3 | 6 |
| IP | 5 | ip8352948 | 7 |
| Is cell IP | 6 | 0 | 8 |
| Property ID | 8 | property_66021 | 9 |

**Figure 2:** Interactions between fields and their features.

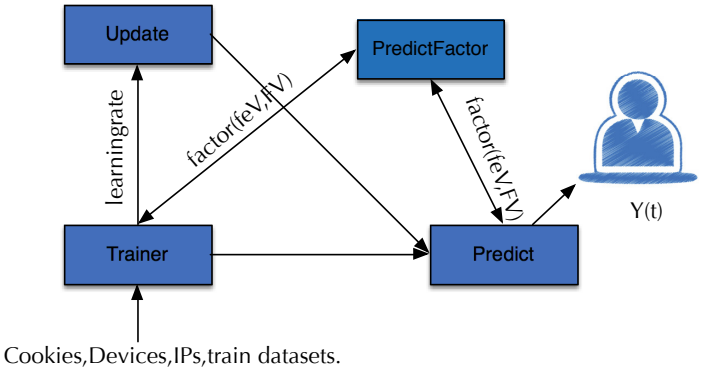



**Figure 3:** Predictive learning Data Model.

learning rate is divided by the number of features in the opposite field. In fact, in this process, it can save time to traverse the entire matrix. And finally, users are predicted in the prediction block as shown in figure 3

Figure 4 depicts the interaction pairs of the model between field vector and feature vector. Within every cell of the matrix in Figure 4, the model had its own set of features for the factorization. For example, there are 8 unique device types, which are used as a feature for the field name device type. For unique 8 types of device, the mapping between fields and features is shown in figure 5. There are 89 unique device operating systems, 368566 unique properties, 443 unique categories, etc. The training file consists of 142770 devices and the test file consists of 61156. There are almost 2.1 million cookies records. The traditional FFM provides an efficient method

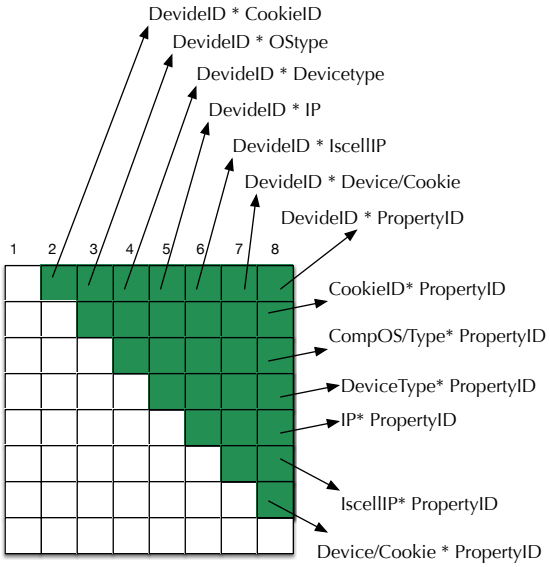



**Figure 4:** Factors in the feature vector and field vector

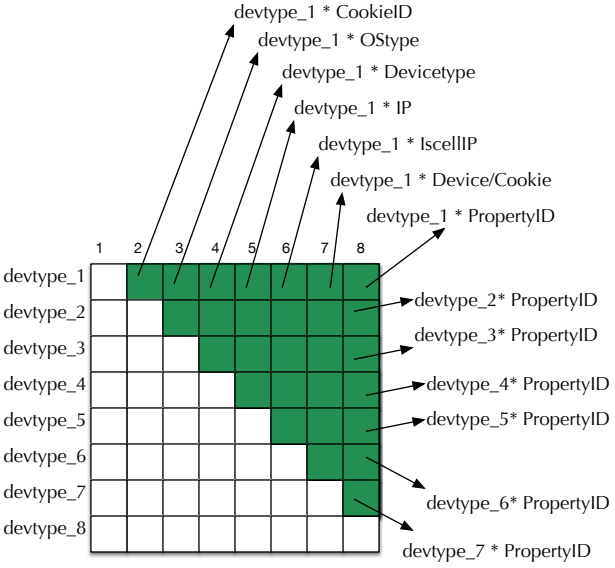



**Figure 5:** Factors multiplication between fields and features of devicetype field.

of factor field multiplication. Our model reads the entire field and divides the feature using the number of features in the opposite field, which makes

it efficient by jumping the cells in the matrix.

The model deals with the properties directly without the need for any feature engineering. The model can adapt the properties directly from the raw data. Before feeding data to the model, the number of potential cookie/device pairs is reduced down to about 900k and then treats as a regression problem. There exist several properties for devices and cookies. This makes learning challenging for FFM in terms of time and memory complexity.

### 3.1 FFM Learning rate:

The issue with learning rate is that the other features interacting with the property feature will get updated once for every property and since they are all updates towards the same field this means we update the same weight again for every property. With the large-scale datasets where the properties of fields are massive, this tends to be an issue. For example, we could have over 200 properties for one sample and maybe just 1 for another. This can be compared to one of the samples getting 200 times higher learning rate towards that field. So it would either end up with a too high or too low learning rate âĂŞ neither works very well.

Our approach solves this problem by dividing the learning rate by the number of features in the opposite field. For example, if there are 200 properties in one sample and 1 for another. The learning rate is divided by 200, so the total amount of learning rate applied is the same as if there was 1 property only. This approach fits the training data to the point of extreme precision, but it could easily over-fit. Another key challenge is to overwhelm the over-fit if there are the small weights adding up. We need to add Regularization [3] to reduce the impact of the smaller weights making a more robust solution, which does not over-fit.

### 3.2 Feature Engineering:

The model can automatically handle features from the original data, which makes the model scalable and efficient. This provides advantages over Support Vector Machines (SVMs) and Gradient Boost Machines (GBMs). We can achieve similar accuracy, using other machine learning algorithms like SVMs, GBMs, Decision trees, etc. But feature engineering is necessary to be performed before processing data to the model, which consumes unnecessary latency and performance degradation of the model. In fact, users need to understand the properties of the matrix to perform feature

engineering. 42 different features are extracted that is listed in table 4.1.

**Table 4.1:** List of features used in the model.

| **S.N.** | **Fieldname** | **Type** |
|---|---|---|
| 1 | Device type | Device |
| 2 | Device OS version | |
| 3 | Device Country Info | |
| 4 | *DeviceAnonymous_c*0 | |
| 5 | *DeviceAnonymous_c*1 | |
| 6 | *DeviceAnonymous_c*2 | |
| 7 | *DeviceAnonymous_*5 | |
| 8 | *DeviceAnonymous_*6 | |
| 9 | *DeviceAnonymous_*7 | |
| 10 | *deviceProperties* | |
| 11 | computer OS type | Cookie |
| 12 | Browser version | |
| 13 | Cookie country info | |
| 14 | *CookieAnonymous_c*0 | |
| 15 | *CookieAnonymous_c*1 | |
| 16 | *CookieAnonymous_c*2 | |
| 17 | *CookieAnonymous_*5 | |
| 18 | *CookieAnonymous_*6 | |
| 19 | *CookieAnonymous_*7 | |
| 20 | cookieProperties | |
| 21 | DeviceIPFreq count | Device IP |
| 22 | DeviceIPAnonymous Count 1 | |
| 23 | DeviceIPAnonymous Count 2 | |
| 24 | DeviceIPanonymous Count 3 | |
| 25 | DeviceIPAnonymous Count 4 | |
| 26 | DeviceIPAnonymous Count 5 | |
| 27 | CountIPsForDevice | |
| 28 | CookieIPFreq count | Cookie IP |
| 29 | CookiesLeastFrequentIP | |
| 30 | CookiesSecondLeastFrequentIP | |
| 31 | CookiesThirdLeastFrequentIP | |
| 32 | CookieIPAnonymous Count 1 | |
| 33 | CookieIPAnonymous Count 2 | |
| 34 | CookieIPanonymous Count 3 | |
| 35 | CookieIPAnonymous Count 4 | |
| 36 | CookieIPAnonymous Count 5 | |
| 37 | CountIPsForCookie | |
| 38 | IPaggIs cell IP | IP aggregation |
| 39 | IPaggTotal Freq | |
| 40 | IPaggAnonymous count c0 | |
| 41 | IPaggAnonymous count c1 | |
| 42 | IPaggAnonymous count c2 | |

# 4 Result

*Setup:* Our system is comprised of 6 core, 64 GB of ECC DDR-2 RAM, 1 TeraBytes of storage. The operating system used is Windows 8. Java 1.7 is used to build the model.

*Datasets* used here are provided by Drawbridge during Kaggle competition. The datasets consist of relational information about users, devices, cookies, information on IP addresses and behavior. It consists of different sets of data. First device table provides basic information of the device. It provides high-level summary information regarding the device. Drawbridge Handle field uniquely identify a person behind the device and cookie. Device and cookie of the same person will have the same handle. Similarly, cookie table provides similar information about the cookie. Second, IP table describes the joint behavior of device or cookie on IP address. One device or cookie can appear on multiple IPs, so we put all the IPs into a bag. Third, IP aggregation table provides IP address information. Fourth, property and property category table provides the information regarding website (cookie) and mobile app (device) that user has visited before. It also provides the list of hash values for specific name of the website and mobile app and the list of categorical information of the websites and mobile apps. Detailed information for the datasets is provided in the actual Kaggle competition [1].

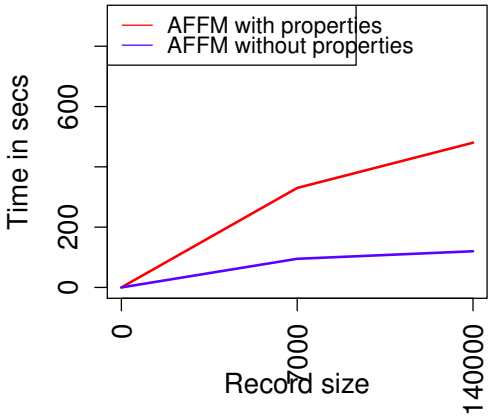



**Figure 6:** Performance graph comparison of FFM with our model.

AFFM performance is compared with features and without features in figure 6. The lack of features saves lots of computation memory and time. Varying the size of training data to evaluate the performance of the model. The training data consists of 142770 devices and the test file consists of 61156. There is almost 2.1 million cookies record. Performing features selection on devices, cookie and IP tables will be time and memory consuming. One of the team member Gilberto [25] used Decision tree's but for this paper, we will focus on the results got from FFM. AFFM provides better compromise accuracy with good performance. There is always a trade-off between accuracy and performance.

Regularization is added to our model so that it reduces overfitting. The AFFM produces a score of 0.864845 in the private Leaderboard. The actual leaderboard score for the competition is 0.789924, but after the competition had ended we realized everyone removed -1 value handles, so trying that approach lead scores to 0.864845. Our model is tested without properties and the scores in leaderboard is 0.852774.

## 5 Conclusion

In this paper, we presented our FFM model with auto feature engineering capability. Our model can perform on the raw datasets. It has inbuilt ability to calculate the feature in the field and drops the learning rate of the model. Our model does not read all the unique values of cell present for that features. The model has a steady learning rate which allows it to use raw properties of the devices and the cookies. The model provides the score of 0.864845 in the leaderboard. We are able to improve the model performance by performing the auto feature engineering and balanced learning rate. This emphasizes the importance of feature engineering and learning rate.

## References

[1] Kaggle:ICDM 2015. *ICDM 2015: Drawbridge Cross-Device Connections Data Description*. URL: `https://www.kaggle.com/c/icdm-2015-drawbridge-cross-device-connections/data`.

[25] https://www.kaggle.com/titericz

[2] YuChin Juan Yong Zhuang and Wei-Sheng Chin. *Field-aware Factorization Machines*. URL: http://www.csie.ntu.edu.tw/~r01922136/slides/ffm.pdf.

[3] Yves Kodratoff. *Introduction to machine learning*. Morgan Kaufmann, 2014.

[4] Yucheng Low, Joseph E Gonzalez, Aapo Kyrola, Danny Bickson, Carlos E Guestrin, and Joseph Hellerstein. "Graphlab: A new framework for parallel machine learning." In: *arXiv preprint arXiv:1408.2041* (2014).

[5] He Xinran, Pan Junfeng, Jin Ou, Xu Tianbing, Liu Bo, Xu Tao, Shi Yanxin, Atallah Antoine, Herbrich Ralf, Bowers Stuart, et al. "Practical lessons from predicting clicks on ads at facebook." In: *Proceedings of 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM. 2014, pp. 1–9.

[6] John Canny and Huasha Zhao. "Bidmach: Large-scale learning with zero memory allocation." In: *BigLearning, NIPS Workshop*. 2013.

[7] Michael Jahrer, A Toscher, JY Lee, J Deng, H Zhang, and J Spoelstra. "Ensemble of collaborative filtering and feature engineered models for click through rate prediction." In: *KDDCup Workshop*. 2012.

[8] Steffen Rendle. "Factorization Machines with libFM." In: *ACM Trans. Intell. Syst. Technol.* 3.3 (May 2012), 57:1–57:22. ISSN: 2157-6904. DOI: 10.1145/2168752.2168771. URL: http://doi.acm.org/10.1145/2168752.2168771.

[9] Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

# Paper VI: Enrichment of Machine Learning based Activity Classification in Smart Homes using Ensemble Learning

# Enrichment of Machine Learning based Activity Classification in Smart Homes using Ensemble Learning

**B. Agrawal[1], A. Chakravorty[1], T. Wiktorski[1], C. Rong[1]**

[1] Department of Electrical Engineering and Computer Science, University of Stavanger

**Abstract:**
Data streams from various Internet-Of-Things (IOT) enabled sensors in smart homes provide an opportunity to develop predictive models to offer actionable insights in form of preventive care to its residence. This becomes particularly relevant for Aging-In-Place (AIP) solutions for the care of the elderly. Over the last decade, diverse stakeholders from practice, industry, education, research, and professional organizations have collaborated to furnish homes with a variety of IOT enabled sensors to record daily activities of individuals. Machine Learning on such streams allows for detection of patterns and prediction of activities which enables preventive care. Behavior patterns that lead to preventive care constitute a series of activities. Accurate labeling of activities is an extremely time-consuming process and the resulting labels are often noisy and error prone. In this paper, we analyze the classification accuracy of various activities within a home using machine learning models. We present that the use of an ensemble model that combines multiple learning models allows to obtain better classification of activities than any of the constituent learning algorithms.

# 1 Introduction

The core concept of smart home environments is to collect information about their residents through an array of sensors in order to provide actionable insights about their behavior. The driving factor in this direction is the growing number of the aging population [5]. There has been a growth in interest in research on sensor networks, techologies, ICT, data science and services that would enable monitoring daily activities of indivisuals, provide real time feedback, enable proactive & preventing care and detect behaviour patterns that could lead to chronic conditions, such as Alzheimer. In addition, these technologies also aim to reduce the cost and allow the elderly to live longer in the comfort of their own homes. Collectively, they can be classified as Aging-In-Place [39, 32] (AIP) solutions.

The primary component of such AIP solutions are homes furnished with various internet enabled sensors, such as movement sensors, cameras, environment sensors and body network sensors (accelerometer, etc.) that enable the collection of data. The interaction of residents with these sensors can allow for determination of various activities of daily living. Activities collected over a time period describe their living behaviour which, in turn, can open a wide array of possibilities in the form of care provided to the elderly [21, 19, 36]. Research in activity recognition based on the interaction with home sensors has emerged as one of the main areas of investigation for AIP solutions [11, 35]. Activity recognition enables the identification of various actions performed by indivisuals in their homes, for example, bathing, using the toilet, sitting on a chair, sleeping and cooking.

Machine Learning [8] is a study of pattern recognition, computational intelligence, and computational learning theory that allows to automatically learn and make accurate predictions based on past observations. It plays a crucial role in activity recognition in smart homes as past true observations of activity labels for a series of sensor interactions can be used for classifying future interactions in the same or simillar homes. In fact, much of the current reseach in activity recognition uses machine learning to classify sensor interactions into activity labels. However, just machine learning itself is limited in terms of their classification accuracy as they are dependent on a specific learning model that is being used. Comparing multiple machine learning models and combining multiple models that contribute to the accuracy and gives better results than using those models

individually [31]. This form of learning is called Ensemble Learning. In this paper, we demonstrate that ensemble learning performs significantly better than the majority of smart home projects that use some form of machine learning for activity recognition. We also compare the accuracy of different individual models with the ensemble model.

The rest of the paper is structured as follows. Section 2 provides an introduction to various machine learning models used in our evaluation. The methodology that is used for the evaluations in terms of ensemble learning is described in section 3. Section 4 introduces the data used for the evaluation and presents the results. We discuss the related work in section 5 and conclude in section 6.

## 2 Background

Distinct advantages can be derived from different learning strategies. These strategies are dependent on the complexity of the learned concept, the structure of the data, the frequency or granularity of the data, the ability to deal with noisy data and the ability to process continuous or discrete features. In the following subsections, we provide an overview of machine learning techniques and progression of such techniques into compounded methods used for classification of different activities in a smart home.

- **Random Forest:** Random Forest is an ensemble learning method for classification and regression which operates by constructing multitude of decision trees at training time [40]. Random Forests grow many classification trees. To classify a new target from an input vector, each vector is added down the trees in the forest. Each tree gives a classification and all trees vote for that particular class. The forest chooses the classification with the most votes.
- **TF-IDF:** Tf-idf is term which means frequency-inverse document frequency, and the tf-idf weight is a weight often used in information retrieval and text mining [42]. This weight specifies how important a word is to a given document in the overall collection. The importance increases proportionally to the number of times a word appears in the document. TF-IDF are often used by search engines for page ranking.
- **Naive Bayes:** A Naive Bayes classifier is a classifier that is based on the popular Bayes' probability theorem. They are known for creating simple yet well performing models, especially in the area of

document classification [41].

- **k-nearest neighbors:** k nearest neighbors is one of the simplest algorithms that classifies new classes based on distance functions (e.g.: Manhattan, Euclidean, Minkowski, and Hamming) [20]. KNN has been widely used in statistical estimation and pattern recognition since the 1970's as a non-parametric technique.
- **XGBoost:** XGBoost is "Extreme Gradient Boosting" [2]. Usually, a single tree is not strong enough to be used in practice. What is actually used is the so-called tree ensemble model that sums the production of multiple trees together. In Graident Boosting, the error of one tree is adjusted to another, using parameter gradient in order to offer better prediction for the new tree. XGBoost is primarily a tree ensemble model.
- **Neural Network:** A neural network [37], is a network of interconnected processing elements (neurons) that work in unison to solve a specific problem. They have been inpired by the way a biological nervous system works and processes information.
- **Ensemble:** Ensemble learning is the art of combining diverse set of individual models together to improvise on the stability and predictive power [18].

# 3 Methodology

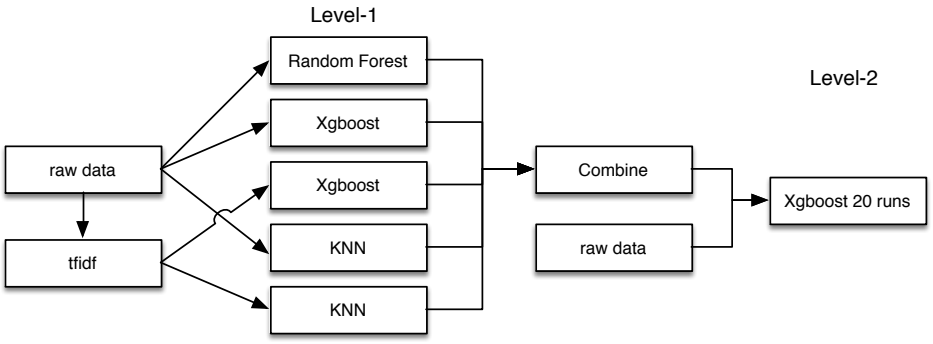



**Figure 1:** An overall architecture of our learning ensemble model.

In this section, we present our idea of building an ensemble model by combining different machine learning models. Additionally, there are other techniques such as bagging, stacking, and boosting schemes that are

similar to ensemble [38]. However, evaluating all of them is beyond the scope of this paper.

In ensemble methods, several individual models are combined to form a single model. The most common approach is taking the average or weighted average of each model. However, in practice. much advanced and complex combination techniques are used [34] [9]. Further, an ensemble model has also been demonstrated to perform extremely well for real world scenarios [24]. The most popular examples are that of NetFlix Prize Challenge[26] and KDD cup 2015 [1] where more than 100 models were combined together to achieve better accuracy. The number of machine learning models to ensemble can be determined by taking the best combination of $n$ models that are linear [28].

Ensemble methods should have multiple good models with sufficiently uncorrelated errors. The individual models are normally combined into a single model as in equation 4.1:

$$Y_{ens}(t) = \frac{1}{n} \sum_{i=1}^{n} M(t) \tag{4.1}$$

where $Y_{ens}(t)$ is the output of the ensemble model, $M(t)$ are the outputs of the individual models and $n$ is the number of models.

In order to evaluate the classification accuracy of activity recognition in smart homes, an ensemble model with 2 level ensemble was created, as shown in figure 1. On level-1, we combine 3 different models (XGBoost, Random Forest, KNN) on raw datasets and 2 different models (KNN and XGBoost) on feature extracted datasets. At first, we calculated TF-IDF of the raw data and then fed it into these models. We combined the output prediction of all 5 models and used it as a feature for the next level. Level-2 combined the outputs of all 5 models and used it as input along with the raw datasets. We used multiple XGBoost runs to achieve better accuracy.

## 3.1 Feature Extraction

Feature extraction is an important component for getting better accuracy. The raw datasets used in our experiments have a set of missing feature data which is handled by inputting it as the mean column values. Other features such as mean, standard deviation, minimum, median, and maximum values of all of the sensors are also extracted. Moreover, the activity labels are

[26] http://netflixprize.com/

probabilistic, these labels are averaged over multiple annotators, excluding 0 and 1. The normalization of data between 0 and 1 is done with the use of a standardization technique. Standardization is a feature that removes the mean and scales the variable to unit variance [22]. Standarization and normalization are very common requirements for many machine learning algorithms which assume that all features are centered around 0 and have variance in the same order. If a feature has higher variance, it might dominate the objective function and make the classifier unable to learn from other features.

### 3.2 Learning Models

In our ensemble model, we used three types of models: Random Forest, XGBoost and KNN. The selection of the model is defined by the accuracy of the individual model and is explained in section 4.2. A combination of additional models could lead to overfitting [44]. In order to avoid overfitting, we choose to use to top three models. The input data (section 4.1) consists of 100 different features which is the combination of 3 sensors: accelerometer, video and environmental data. Additionally, we added extra features which vary for each sensor. The output of these models is combined and fed as feature to another level.

## 4 Emperical Evaluation

In this section, we describe the raw datasets and present the results which evaluate and validate our model. It evaluates the accuracy level of activity classification and investigates the benefits of the ensemble model in activity identification.

### 4.1 Data Description:

The activities are identified with the data from three types of sensors; accelerometer, video (RGB-D), and environmental sensor. The accelerometer is sampled at 20 Hz and is in a raw format. Video data is based on features extracted from the center of mass and bounding box of the identified persons. Environmental data consists of Passive Infra-Red (PIR) sensors, and is provided in a raw format. Detailed data is provided in [4]. Twenty posture activity labels are annotated in the dataset, and described in table 4.1:

**Table 4.1:** Description of Data

| Label | Description |
|---|---|
| a_ascend | ascent stairs |
| a_descend | descent stairs |
| a_jump | jump |
| a_loadwalk | walk with load |
| a_walk | walk |
| p_bent | bending |
| p_kneel | kneeling |
| p_sit | sitting |
| p_lie | lying |
| p_squat | squatting |
| p_stand | standing |
| t_bend | stand to bend |
| t_kneel stand | kneel to stand |
| t_lie sit | lie to sit |
| t_sit lie | sit to lie |
| t_sit stand | sit to stand |
| t_stand kneel | stand to kneel |
| t_stand sit | stand to sit |
| t_straighten | bend to stand |
| t_turn | turn |

The prefixes 'a','p', and 't' are used as a label to indicate an ambulation activity, static postures, and posture-to-posture transitions. These labels are the target variables that need to be classified with the use of train and test datasets.

Further [4] describes the weights and distribution of these activities and is demonstrated in figure 2. The *Prior Class Distribution* displays prior distribution of the training data for the 20 different activities. It shows that *p_sit* occurs nearly 20% of the time and some of the other activities such as *p_squat*, *a_jump*, and *a_descend* occur about less than 2% of the time. The *Class Weight* graph shows the activity names and their associated weights. Activities that are performed more frequently than other activities are weighted less than those performed rarely. The

activity *a_jump* has the highest weight when compared to activities, such as *p_sit* and *p_kneel* which are given weights of 0.2 and 1.04, respectively. This further alludes to the fact that the prediction of rare activities, such as a jump is 5 times more likely than a frequent activity like sit.

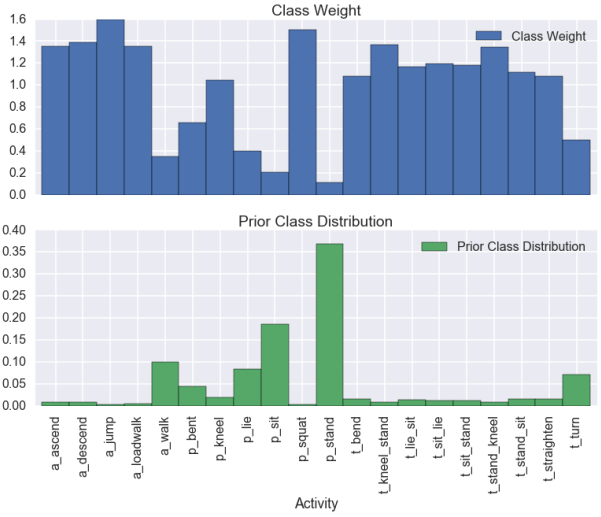



**Figure 2:** Class weights.

## 4.2 Evaluation:

The activity classification is performed by combining different models. We performed feature extraction with the use of TFIDF on the raw datasets to enchance the accuracy level. The accuracy of 5 different models was evaluated by executing them over the raw and feature extracted datasets as shown in table 4.2. The best three models were then choosen to form an ensemble model. The evaluation of ensemble model is done with the use of the brier score since the outcome of activities is in a probabilistic forecast. The brier score is an average of the sum of the squared errors of a probabilistic prediction [45].The brier score is shown in equation 4.2.

$$BS = \frac{1}{N}\sum_{t=1}^{N}\sum_{c=1}^{C} w_c(p_{t,c} - a_{t,c})^2 \tag{4.2}$$

where N is the number of test instances, C is the number of classes, $w_c$ is the weight of each class, $p_{t,c}$ is the probability of the predicted output and $a_{t,c}$ is the actual outcome of the event. Therefore, the smaller the brier score, the higher the accuracy.

**Table 4.2:** Accuracy of a single model on raw and feature extracted datasets

| Model | datasets | brier score |
|---|---|---|
| Random Forest | raw | 0.24 |
| K-NN classifiers | raw | 0.217 |
| K-NN classifiers | tfidf | 0.227 |
| Xgboost | raw | 0.23 |
| Xgboost | tfidf | 0.232 |
| neural network | raw | 0.31 |
| neural network | tfidf | 0.299 |
| naive bayes | raw | 0.273 |
| naive bayes | tfidf | 0.266 |

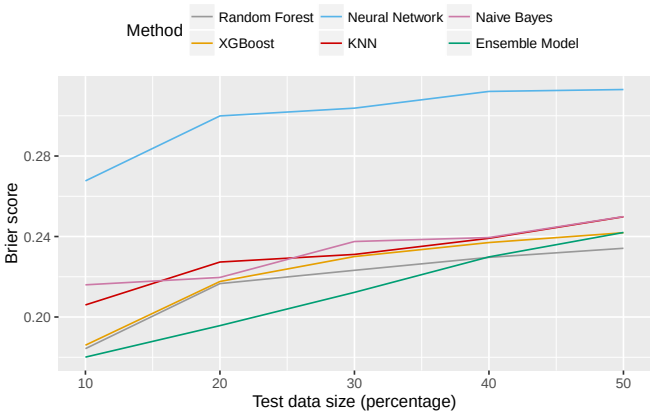



**Figure 3:** Brier score of different model with variation of test dataset size from 10% to 50%.

In figure 3, we compare our model to other machine learning models (KNN, XGBoost, RandomForest, Naive Bayes, Neural Network). The average brier score for all activies was measured for each model. The ensemble model achieves better accuracy accross different training and test dataset sizes. Its performance deteriorates much slower with the reduction in the training sizes and has a linear nature.

In order to demonstate the individual activity classification accuracy, the dataset was split into 80% for training and 20% for testing the ensemble model. Figure 4 shows the accuracy of the model for classifying the activities. The model is compared with the actual label present in the datasets. The overall brier score of our ensemble model is 0.164. It is

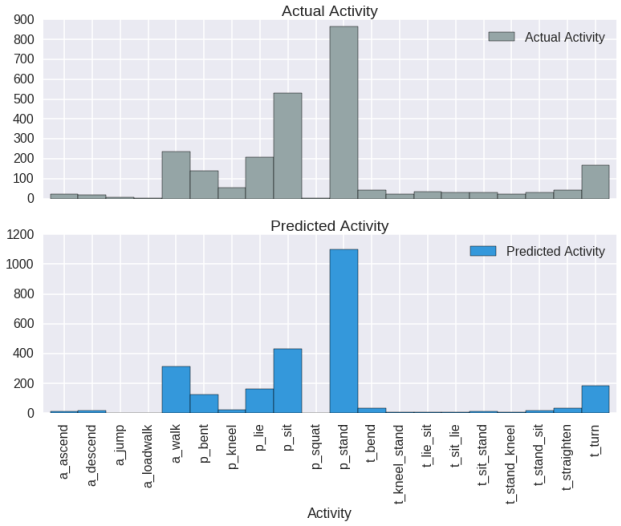



**Figure 4:** Evaluation of ensemble model on different activities.

able to closely classify rare activties, such as *p_squat*, *a_jump* and others. However, frequent activities are slightly over- or underestimated.

Figure 5 shows the area under a receiver operating characteristic (ROC) curve. ROC curve represents the performance of an ensemble model [43]. It shows the estimated sensitivity and specificity for different cutoffs. ROC curve is ploted against the false positive rate (FPR) on the x-axis and the true positive rate (TPR) on the y-axis. The grey (45-degree) diagonal line represents the average performance of a Uniform (0, 1) random variable. The further away a line is from the diagonal line (in terms of sensitivity), the better the accuracy. Overall, gain in sensitivity (true positive rate) is $> 86\%$.

The ROC curve can be combined into a single value by calculating the convex shape in ROC curve to get AUC (Area Under Curve). The AUC of our model is 0.869. The optimal point in ROC curve is (FPR,TPR) = (0, 1) which means that a curve has no false positive. In the figure, activities, such as $p_{squat}$ and $a_{jump}$ have low AUC value because these activities are rare and the training datasets have very limited patterns for such activities.

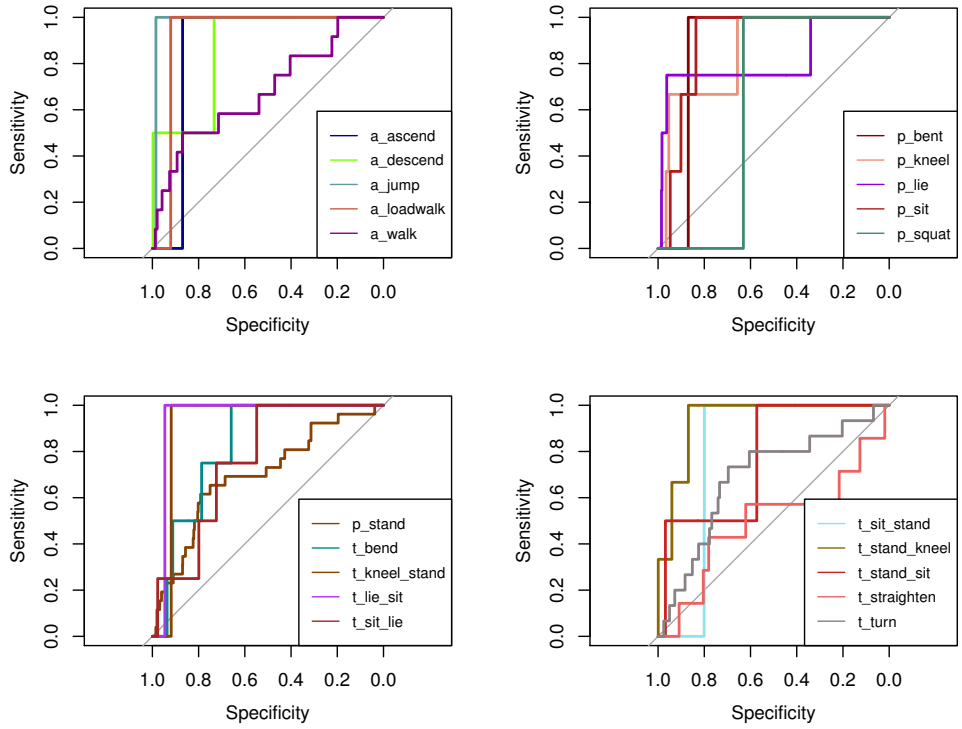



**Figure 5:** Area under a receiver operating characteristic (ROC) curve for evaluating the ensemble model.

# 5 Related Work

There has been ample research on machine learning models used for activity recognition in smart homes. Recognition and detection of human activities have been addressed with neural network models [3, 35, 23, 12], decision trees [6, 35, 16], bayes classifiers [6, 7, 35], random forests [13, 6, 29], support vector machines [6, 15, 25, 10], adaptive boosting [6, 29] and K-NN classifiers [30, 27].

However, using a single model is still the prime challenge for machine learning in smart homes. It can be attributed to the variance in the deployment context in which learning takes place. Alternatively, ensemble learning addresses these challenges as it combines multiple models. It uses the output of one or multiple model(s) depending on their accuracy as feature(s) to another [33]. The research in the direction of combining multiple machine learning models into an ensemble learning model for

activity recognition in smart homes is quite novel. It is limited with some contirbutions from [26, 14, 17].

# 6 Conclusion

In this paper, we discussed the importance of activity recognition for smart homes. Different machine learning techniques have been described in literature to classify activities of residents, based on their interaction with their home sensors. However, a number of challenges and limitations were identified in these approaches. In practice, no single machine learning model can achieve an acceptable level of accuracy on a given dataset. To overcome the limiations of such existing work, we proposed an alternative state-of-the-art ensemble model. The output of multiple classifiers was fed into an ensemble model as input. It achieved a better classification performance as compared with individual classifiers. We demonstrated that our ensemble model outperformed other techniques. Further, we evaluated the result with a ROC curve which indicated that all classified activities were more closer to true positives.

# References

[1] R blogger. *KDD Cup 2015: The story of how I built hundreds of predictive models.And got so close, yet so far away from 1st place.* URL: `https://www.r-bloggers.com/kdd-cup-2015-the-story-of-how-i-built-hundreds-of-predictive-models-and-got-so-close-yet-so-far-away-from-1st-place/`.

[2] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system." In: *arXiv preprint arXiv:1603.02754* (2016).

[3] Homay Danaei Mehr, Huseyin Polat, and Aydin Cetin. "Resident activity recognition in smart homes by using artificial neural networks." In: *2016 4th International Istanbul Smart Grid Congress and Fair (ICSG)*. IEEE. 2016, pp. 1–5.

[4] Niall Twomey, Tom Diethe, Meelis Kull, Hao Song, Massimo Camplani, Sion Hannuna, Xenofon Fafoutis, Ni Zhu, Pete Woznowski, Peter Flach, et al. "The SPHERE Challenge: Activity Recognition with Multimodal Sensor Data." In: *arXiv preprint arXiv:1603.00797* (2016).

[5] David E Bloom, David Canning, and Alyssa Lubet. "Global population aging: Facts, challenges, solutions & perspectives." In: *Daedalus* 144.2 (2015), pp. 80–92.

[6] Diane J Cook, Maureen Schmitter-Edgecombe, and Prafulla Dawadi. "Analyzing Activity Behavior and Movement in a Naturalistic Environment using Smart Home Techniques." In: *IEEE journal of biomedical and health informatics* 19.6 (2015), pp. 1882–1892.

[7] Tom Diethe, Niall Twomey, and Peter Flach. "Bayesian active transfer learning in smart homes." In: *ICML Active Learning Workshop*. Vol. 2015. 2015.

[8] Rob Schapire. "Machine learning algorithms for classification." In: *Princeton University* 10 (2015).

[9] Manuel Woniak Michałand Graña and Emilio Corchado. "A Survey of Multiple Classifier Systems As Hybrid Systems." In: *Inf. Fusion* 16 (Mar. 2014), pp. 3–17. ISSN: 1566-2535. DOI: 10.1016/j.inffus.2013.04.006. URL: http://dx.doi.org/10.1016/j.inffus.2013.04.006.

[10] Diane J Cook, Aaron S Crandall, Brian L Thomas, and Narayanan C Krishnan. "CASAS: A smart home in a box." In: *Computer* 46.7 (2013).

[11] Liming Chen, Chris D Nugent, and Hui Wang. "A knowledge-driven approach to activity recognition in smart homes." In: *IEEE Transactions on Knowledge and Data Engineering* 24.6 (2012), pp. 961–974.

[12] Liyanage C De Silva, Chamin Morikawa, and Iskandar M Petra. "State of the art of smart homes." In: *Engineering Applications of Artificial Intelligence* 25.7 (2012), pp. 1313–1321.

[13] Ahmad Jalal, Jeong Tai Kim, and Tae-Seong Kim. "Human activity recognition using the labeled depth body parts information of depth silhouettes." In: *Proceedings of the 6th International Symposium on Sustainable Healthy Buildings, Seoul, Korea*. 2012, pp. 1–8.

[14] Chao Chen, Barnan Das, and Diane J Cook. "A data mining framework for activity recognition in smart environments." In: *Intelligent Environments (IE), 2010 Sixth International Conference on*. IEEE. 2010, pp. 80–83.

[15] Anthony Fleury, Michel Vacher, and Norbert Noury. "SVM-based multimodal classification of activities of daily living in health smart homes: sensors, algorithms, and first experimental results." In: *IEEE transactions on information technology in biomedicine* 14.2 (2010), pp. 274–283.

[16] Eunju Kim, Sumi Helal, and Diane Cook. "Human activity recognition and pattern discovery." In: *IEEE Pervasive Computing* 9.1 (2010), pp. 48–53.

[17] Parisa Rashidi and Diane J Cook. "Multi home transfer learning for resident activity discovery and recognition." In: *KDD Knowledge Discovery from Sensor Data* (2010), pp. 56–63.

[18] Lior Rokach. "Ensemble-based classifiers." In: *Artificial Intelligence Review* 33.1-2 (2010), pp. 1–39.

[19] Marie Chan, Eric Campo, Daniel Estève, and Jean-Yves Fourniols. "Smart homes—current features and future perspectives." In: *Maturitas* 64.2 (2009), pp. 90–97.

[20] Leif E Peterson. "K-nearest neighbor." In: *Scholarpedia* 4.2 (2009), p. 1883.

[21] Marie Chan, Daniel Estève, Christophe Escriba, and Eric Campo. "A review of smart homes—Present state and future challenges." In: *Computer methods and programs in biomedicine* 91.1 (2008), pp. 55–81.

[22] Andreas Stolcke, Sachin Kajarekar, and Luciana Ferrer. "Nonparametric feature normalization for SVM-based speaker verification." In: *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2008, pp. 1577–1580.

[23] Huiru Zheng, Haiying Wang, and Norman Black. "Human activity detection in smart home environment with self-adaptive neural networks." In: *Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference on*. IEEE. 2008, pp. 1505–1510.

[24] Robert M Bell and Yehuda Koren. "Lessons from the Netflix prize challenge." In: *ACM SIGKDD Explorations Newsletter* 9.2 (2007), pp. 75–79.

[25] Oliver Brdiczka, Patrick Reignier, and James L Crowley. "Detecting individual activities from video in a smart home." In: *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*. Springer. 2007, pp. 363–370.

[26] Donghai Guan, Weiwei Yuan, Young-Koo Lee, Andrey Gavrilov, and Sungyoung Lee. "Activity recognition based on semi-supervised learning." In: *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007)*. IEEE. 2007, pp. 469–475.

[27] Clemens Lombriser, Nagendra B Bharatula, Daniel Roggen, and Gerhard Tröster. "On-body activity recognition in a dynamic sensor network." In: *Proceedings of the ICST 2nd international conference on Body area networks*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 2007, p. 17.

[28] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 0387310738.

[29] Tran The Truyen, Dinh Q Phung, Svetha Venkatesh, and Hung Hai Bui. "Adaboost. mrf: Boosted Markov random forests and application to multilevel activity recognition." In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. IEEE. 2006, pp. 1686–1693.

[30] Ling Bao and Stephen S Intille. "Activity recognition from user-annotated acceleration data." In: *International Conference on Pervasive Computing*. Springer. 2004, pp. 1–17.

[31] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. "Ensemble selection from libraries of models." In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 18.

[32] Eric Dishman. "Inventing wellness systems for aging in place." In: *Computer* 37.5 (2004), pp. 34–41.

[33] Saso Dzeroski and Bernard Ženko. "Is Combining Classifiers with Stacking Better than Selecting the Best One?" In: *Machine Learning* 54.3 (2004), pp. 255–273. ISSN: 1573-0565. DOI: 10.1023/B:MACH.0000015881.36452.6e. URL: http://dx.doi.org/10.1023/B:MACH.0000015881.36452.6e.

[34] Ludmila I Kuncheva. "Classifier ensembles for changing environments." In: *International Workshop on Multiple Classifier Systems*. Springer. 2004, pp. 1–15.

[35] Emmanuel Munguia Tapia, Stephen S Intille, and Kent Larson. "Activity recognition in the home using simple and ubiquitous sensors." In: *International Conference on Pervasive Computing*. Springer. 2004, pp. 158–175.

[36] Frances K Aldrich. "Smart homes: past, present and future." In: *Inside the smart home*. Springer, 2003, pp. 17–39.

[37] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[38] Ron Meir and Gunnar Rätsch. "An introduction to boosting and leveraging." In: *Advanced lectures on machine learning*. Springer, 2003, pp. 118–183.

[39] SL Bernard, S Zimmerman, and JK Eckert. "Aging in place." In: *Assisted living* 224 (2001), p. 241.

[40] Leo Breiman. "Random forests." In: *Machine learning* 45.1 (2001), pp. 5–32.

[41] Irina Rish. "An empirical study of the naive Bayes classifier." In: *IJCAI 2001 workshop on empirical methods in artificial intelligence*. Vol. 3. 22. IBM New York. 2001, pp. 41–46.

[42] Joel Larocca Neto, Alexandre D Santos, Celso AA Kaestner, Neto Alexandre, D Santos, et al. "Document clustering and text summarization." In: (2000).

[43] Andrew P Bradley. "The use of the area under the ROC curve in the evaluation of machine learning algorithms." In: *Pattern recognition* 30.7 (1997), pp. 1145–1159.

[44] Igor V Tetko, David J Livingstone, and Alexander I Luik. "Neural network studies. 1. Comparison of overfitting and overtraining." In: *Journal of chemical information and computer sciences* 35.5 (1995), pp. 826–833.

[45] Allan H Murphy. "A new vector partition of the probability score." In: *Journal of Applied Meteorology* 12.4 (1973), pp. 595–600.