Newcastle University

School of Computing Science

Real-Time Performance Diagnosis and Evaluation of Big Data Systems in Cloud Datacenters

Umit Demirbaga

Submitted for the degree of Doctor of Philosophy in the School of Computing Science, Newcastle University

September 2021

© 2021, Umit Demirbaga

Abstract

Modern big data processing systems are becoming very complex in terms of large-scale, high-concurrency and multiple talents. Thus, many failures and performance reductions only happen at run-time and are very difficult to capture. Moreover, some issues may only be triggered when some components are executed. To analyze the root cause of these types of issues, we have to capture the dependencies of each component in real-time.

Big data processing systems, such as Hadoop and Spark, usually work in large-scale, highly-concurrent, and multi-tenant environments that can easily cause hardware and software malfunctions or failures, thereby leading to performance degradation. Several systems and methods exist to detect big data processing systems' performance degradation, perform root-cause analysis, and even overcome the issues causing such degradation. However, these solutions focus on specific problems such as stragglers and inefficient resource utilization. There is a lack of a generic and extensible framework to support the real-time diagnosis of big data systems.

Performance diagnosis and prediction of big data systems are highly complex as these frameworks are typically deployed in cloud data centers that are large-scale, highly concurrent, and follows a multi-tenant model. Several factors, including hardware heterogeneity, stochastic networks and application workloads may impact the performance of big data systems. The current state-of-the-art does not sufficiently address the challenge of determining complex, usually stochastic and hidden relationships between these factors.

To handle performance diagnosis and evaluation of big data systems in cloud environments, this thesis proposes multilateral research towards monitoring and performance diagnosis and prediction in cloud-based large-scale distributed systems by involving a novel combination of an effective and efficient deployment pipeline.

The key contributions of this dissertation are listed below:

- Designing a real-time big data monitoring system called *SmartMonit* that efficiently collects the runtime system information including computing resource utilization and job execution information and then interacts the collected information with the Execution Graph modeled as directed acyclic graphs (DAGs).
- Developing *AutoDiagn*, an automated real-time diagnosis framework for big data systems, that automatically detects performance degradation and inefficient resource utilization problems, while providing an online detection and semi-online root-cause analysis for a big data system.
- Designing a novel root-cause analysis technique/system called *BigPerf* for big data systems that analyzes and characterizes the performance of big data applications by incorporating Bayesian networks to determine uncertain and complex relationships between performance related factors.

Declaration

I declare that this thesis is my own work unless otherwise stated. No part of this thesis has previously been submitted for a degree or any other qualification at Newcastle University or any other institution.

Umit Demirbaga

September 2021

Publications

Published

1. **U. Demirbaga**, A. Noor, Z. Wen, P. James, K. Mitra and R. Ranjan, *"Smart-Monit: Real-time Big Data Monitoring System,"* The 38th International Symposium on Reliable Distributed Systems (SRDS 2019) Lyon, France, OCT 1-4, 2019, 10.1109/SRDS47363.2019.00049. **[Core A Ranking]**

2. **U. Demirbaga**, Z. Wen, A. Noor, K. Mitra, K. Alwasel, S. Garg, A. Zomaya and R. Ranjan, *"AutoDiagn: An Automated Real-time Diagnosis Framework for Big Data Systems,"* IEEE Transactions on Computers, Apr 02 2021, 10.1109/TC.2021.3070639. **[Q1, Core A* Ranking, ISI impact factor 3.131]**

3. **U. Demirbaga**, *"HTwitt: A Hadoop-based Platform for Analysis and Visualization of Streaming Twitter Data,"* Neural Computing and Applications, Springer, Apr 14 2021, 10.1007/s00521-021-06046-y. **[Q1, Core B Ranking, ISI impact factor: 5.606]**

4. K. Alwasel, D. Jha, F. Habeeb, **U. Demirbaga**, O. Rana, T. Baker, S. Dustdar, M. Villari, P. James, and R. Ranjan, *"IoTSim-Osmosis: A Framework for Modelling & Simulating IoT Applications Over an Edge-Cloud Continuum,"* Journal of Systems Architecture, Elsevier, 28 Nov 2020, 10.1016/j.sysarc.2020.101956. **[Q2, Core B Ranking, ISI impact factor: 2.55]**

5. A. Noor, K. Mitra, A. Souza, D. N. Jha, P. P. Jayaraman, **Umit Demirbaga**, Ellis Solaiman, Nelio Cacho, and R. Ranjan, *"Cyber-Physical Application Monitoring across Multiple Clouds,"* Journal of Computer and Electrical Engineering, Elsevier, July 2019, 10.1016/j.compeleceng.2019.06.007. **[Q1, Core B Ranking, ISI impact factor: 2.6]**

6. **U. Demirbaga** and D. N. Jha, *"Social Media Data Analysis using MapReduce*

Programming Model and Training a Tweet Classifier using Apache Mahout," in Proceedings - 8th IEEE International Symposium on Cloud and Services Computing, SC2 2018, 2018, 10.1109/SC2.2018.00024.

7. **U. Demirbaga**, D. N. Jha, N. Booth, T. Roberts, T. Shah, and R. Ranjan, *"A Batch and Real-time Data Analytics Framework for Healthcare Applications,"* Newsletter, IEEE Technical Committee on Cybernetics for Cyber-Physical Systems, Volume 3, Issue 2, August 01, 2018.

Dedication

To those who have committed and to those who will commit their lives to science and to all people who believe in the power of knowledge.

Acknowledgements

It has been a great pleasure working with the faculty, staff, and an amazing group of talented people at Newcastle University, during my tenure as a Doctoral student.

I am very thankful to my Ph.D. advisor, Distinguished Professor Rajiv Ranjan, for his exceptional support, encouragement, and immense knowledge throughout my PhD study. I am very grateful to have been the student of such a knowledgeable, enthusiastic, and ambitious visionary and to be a part of his team.

I would also like to express sincere gratitude to Dr. Karan Mitra (Luleå University of Technology) who facilitated my learning and spent countless hours reviewing my drafts and brainstorming new research ideas.

I would like to thank my fellow lab-mates: Dr. GaganGeet Singh Aujla, Ayman Noor, Bin Qian, Devki Nandan Jha, Khaled Alwasel, Nipun Balan, Top Phengsuwan, Yinhao Li, and Zhenyu Wen, for their friendship and influencing my work with immeasurable discussions.

I am highly indebted to the State of the Republic of Turkey and the Turkish Ministry of National Education for giving me this opportunity to complete my PhD, and thanks for their financial and emotional support.

Furthermore, I would also like to express my deepest gratitude to my parents, Rüstem Demirbaga and Dilber Demirbaga, and my siblings, Pınar Dönmez, Duygu Sefertaş, Sevgi Arısoy, and Eren Demirbaga, who have supported me throughout this journey and have always believed in me.

I also would like to thank the lovely couple, Meryem & Erhan Batmaca for believing in my success and supporting me from the beginning of my abroad journey.

A special thanks to my loving wife, Kübra Kırca-Demirbaga, who was always with me to share my happiness, success and sorrow. I am grateful for her understanding, her patience, and mostly for her love. She is not just a coauthor of this thesis, but the coauthor of my life.

Contents

List of Figures

List of Tables

1

Introduction

Contents

Introduction

Big data systems are used to efficiently process very large amounts of data that cannot be processed with a single computer. Such systems, which use large scale distributed cluster architecture, implement parallel programming models to process data and distributed file systems to store data [2]. The distributed parallel architecture distributes data among multiple servers and creates turbulence, increasing data processing speeds. Big Data systems enable to handle the collection of large and complex data classified as structured, semi-structured, and unstructured data, generated from various data sources. There are two types of data processing systems in use today: (1) batch processing systems like Apache Hadoop[1], Apache Spark[2], Dryad[3]; (2) stream processing systems like Apache Storm[4], Apache Spark Streaming[5], Apache Flink[6], Apache Kafka[7], Apache Samza[8] [3].

A cloud datacenter (CDC) is a physical facility comprised of numerous physical machine hosts (PMs), each of which can create multiple virtual machines (VMs) to execute cloud-based tasks. It is expected that the datacenter is virtualized, and a high-speed network with a fat-tree topology is used to connect clustered PM hosts [4]. Cloud datacenters are an established Infrastructure-as-a-Service (IaaS) offering. Computing hardware including computing unit and data storage devices, racks, routers, switches, power, cabling system, environment system (e.g., air conditioning, humidification), firewalls, and application-delivery controllers are the common and important components of a datacenter [5]. Cloud datacenters are a mature IaaS solution that combines and virtualizes resources such as compute, storage, network, and so on. Cloud datacenters enables cloud applications to be accessed over the internet. Each VM hosted by virtualized PMs can have its own characteristics such as computing power, storage and memory capacities, network speed and cost per hour based on these features.

[1] https://hadoop.apache.org/
[2] https://spark.apache.org/
[3] https://www.microsoft.com/en-us/research/project/dryad/
[4] https://storm.apache.org/
[5] https://spark.apache.org/streaming/
[6] https://flink.apache.org/
[7] https://kafka.apache.org/
[8] https://samza.apache.org/

The tenants are provided satisfying services by cloud datacenters as such computing resources can be easily isolated for each VM. QoS parameters e.g. processing power, performance, availability, etc. have a decisive role in the selection and cost of the virtual machine. Although cloud providers are located in different geographical locations, they are accessible from most parts of the world providing almost unlimited storage and processing power, on-demand.

Quality of Service (QoS) refers to the ability of a service to fulfil certain standards for various aspects of the service such as performance, availability, reliability, efficiency, cost, response time, throughput, makespan, energy consumption and so forth [6]. The levels of such parameters provided by cloud applications as well as the infrastructure that hosts such applications are referred to as QoS. QoS for cloud-based big data systems includes system performance (e.g., availability, response time, throughput, scalability), system data security, system reliability, and system robustness [7]. These parameters are considered for QoS evaluation for a service. In cloud computing, QoS is a critical component of a successful Service Level Agreement (SLA), which ensures the creation of a trustworthy provider-consumer relationship. SLA is basically a contract file that specifies the terms of the agreement between the cloud user and the cloud service provider [8]. This contract narrates the terms and conditions for both parties. Cloud vendors and customers negotiate an SLA to state the QoS specifications based on the key performance indicators of customers and agree on the requirements. Cloud vendors manage their resources easily using SLAs to guarantee the QoS requirements.

1.1 Research Motivation

Big data systems have emerged as a result of the fast increase of data created by sectors such as e-commerce businesses, social networks, healthcare, banking sector, and media and entertainment industries. Big data systems allow for the processing of enormous volumes of data in a relatively short length of time. For example, Facebook runs hundreds of real-time data pipelines in productions and processes more than 500 TB of data daily [9]. Similarly, Netflix big data pipeline processes 11 million events and 24 gigabytes (GB) of data on a per-second basis over 500 billion user actions every day

to discover customer behaviour and buying patterns [10]. However, the enormousness and complexity of the big data system runs in heterogeneous computing resources, multiple tenant environments, as well as has many concurrent execution of big data processing tasks, which makes it a challenge to utilize the big data systems efficiently and reliably. For example, Fig. 1.1 shows that the performance degrades at least 10% when the resources are not utilized efficiently with Setting 2. To manage this issue, it is critical to continually monitor and evaluate all available system resources at all times in a systematic, comprehensive, and automated manner. These resources include CPU, memory, network, I/O, and software components for big data processing.

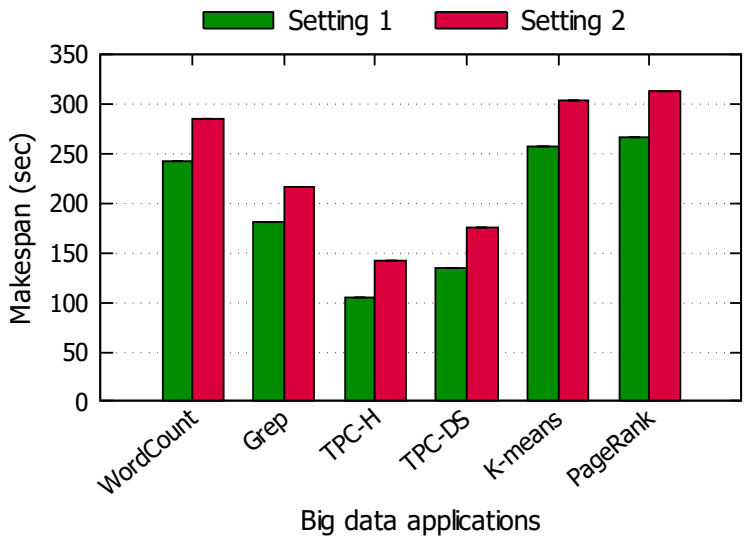


Figure 1.1: Six big data applications are executed in a cloud-based Hadoop cluster with two settings: 1) the input data and jobs are allocated in the same node; 2) the input data and jobs are allocated in different nodes. In Setting 2, the execution time of each application is delayed by transmitting data across nodes.

Big data systems, especially cloud-based applications, run in large-scale computer clusters consisting of thousands of distributed computing nodes with complex interactions including a wide variety of subsets configurations and hundreds of adjustable parameters, which has brought lots of challenges to performance diagnosis in big data systems.

Fig. 1.2 the performance diagnosis of a high-level architecture of a large-scale data processing system. Big data analytics architectures consist of three different layers: data ingestion, analytics, and storage.

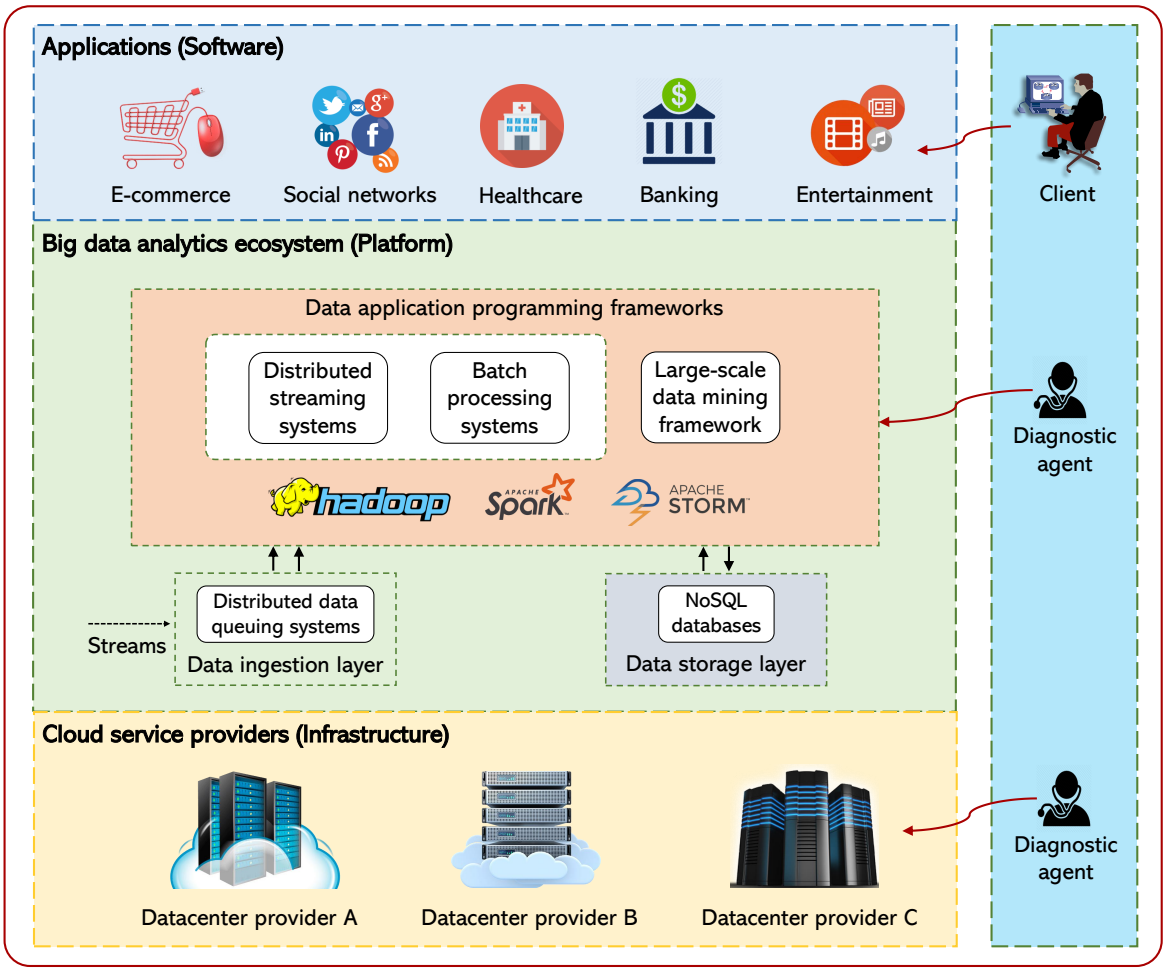


Figure 1.2: Performance diagnosis of big data systems in cloud datacenters

There are already numerous performance diagnosis frameworks, such as Datadog [11], SequenceIQ [12], Sematext [13], TACC Stats [14], Mantri [15], DCDB Wintermute [16], Nagios [17], Ganglia [18], Apache Chukwa [19], DMon[20], available to diagnose cloud-based computer systems. However, some of them focus on either anomaly detection or root-cause analysis. These diagnostic systems are not able to propose a generic and comprehensive solution for the detection of a wide variety of anomalies and root cause analysis of performance issues in cloud-based big data systems.

Performance diagnosis in such complex environments is very challenging due to the following reasons:

- **Capturing the failures or performance reductions** is very hard as they only happen at run-time that needs capturing the dependencies of each component in real-time [21]. In distributed systems, components work together interactively,

not individually, to make a progress or action, and as a result, the system is perceived as a whole, not as a collection of individual components [22]. In order to detect the root cause of a problem, the tasks that trigger each other must be identified, so all these interactive components must be monitored at run-time [23].

- **Monitoring big data systems generates a large volume of logs with varying formats** [24] that makes the analysis of monitoring data for performance diagnosis more difficult as it needs fine-grained data processing as well as fast and accurate preprocessing to remove unnecessary information [3]. Log data is not always homogeneous and systematic, making them difficult to analyze as it requires good capabilities of dealing with noisy and missing data and preserving log sequence information, such as the running state of a task at any given time [25].
- **Big data systems have many simultaneous and interactive components**, making it difficult to find the root cause of the problem [26]. Big data systems have many concurrent tasks, namely multiple tasks running at the same time but not necessarily simultaneously and parallel tasks, namely the tasks executed by different worker nodes at the same time [27]. In such complex systems, due to interdependence and synchronicity of tasks, it is very challenging to find the reasons why: tasks are slowing down, resource utilization is low, the response time is high, or when the network latency is high [28].
- **Big data systems have a large number of configuration settings**, including hundreds of adjustable parameters, where a change in any parameter can affect many processes and operations, making problem localization difficult [29]. For example, Hadoop configuration is driven by some configuration files consisting of many parameters [30] that significantly affect the performance of MapReduce applications, making it difficult to find the root cause of performance degradation of such systems [31].
- **The fact that big data systems are highly scalable and heterogeneous environments** with diverse computing resources [32], such as different CPU, memory, storage, and network capacity, results in a range of different completion times, making it difficult to evaluate the performance of the whole system [33]. For in-

stance, the reasons for performance degradation, such as inconsistencies between tasks, machine failures, machine overloaded, etc. are stemming from big data clusters consisting of nodes with different compute resources and capacities [15].

- **Big data applications running simultaneously on a cluster share computing resources** but may require different resource capacities [34], resulting in performance degradation for some applications. In such a case, the diagnosis of the exact reason for the performance issue becomes more complex as it is difficult to determine whether the performance degradation is due to insufficient resource allocation [35].

Taking into account these aforementioned challenges and concerns, we formulated the following three research questions (RQ):

- **(RQ1)** How to collect monitoring information effectively from big data systems in cloud environments in real-time while processing these streaming data and interact the processed data with the Execution Graph of each task while visualizing the interaction in real-time?
- **(RQ2)** How to detect errors, faults, performance degradation, and inefficient resource utilization problems in big data systems while providing an online detection and root-cause analysis (RCA)?
- **(RQ3)** How to model complex dependencies between several factors for undertaking RCA for diagnosing and predicting reasons for performance degradation while validating the RCA technique over a realistic, large scale test setup that can produce usable and non-biased performance degradation data sets related to Hadoop Applications?

1.2 Research Contributions

There are numerous tools detecting performance reduction problems in cloud-based big data systems. These tools, however, are either for root-cause analysis or anomaly detection and debugging, focusing on specific issues such as stragglers, or inefficient

use of resources. These solutions, however, are incapable of meeting the requirements for an automated performance diagnosis platform for cloud-based big data systems. In addition, existing tools and techniques suffer from significant technological constraints, such as being a generic and extensible framework supporting the real-time diagnosis of big data systems while providing holistic monitoring and detecting performance degradation and enabling root-cause analysis. Furthermore, there is some work regarding probabilistic performance diagnosis and prediction for big data systems. However, they only consider either task status or network issues. There is some work regarding performance diagnosis and prediction for big data systems considering only either task status or network issues. However, there is no prior work on end-to-end performance diagnosis and prediction (considering the impact of both task execution time and network delay on job completion time) for such systems. This is because it is extremely challenging to consider both issues in big data systems at the same time while evaluating the performance of such systems as they consist of numerous factors with many complex hidden dependencies between each other. That is why there is an urgent need for end-to-end performance diagnosis and prediction of big data systems, taking into consideration the execution time of each specific task and network transmission time between each step simultaneously while uncovering the stochastic relationships among those factors. The main contributions of this thesis are as given below:

- We design a real-time big data monitoring system called *SmartMonit* that efficiently collects the run-time system information including computing resource utilization and job execution information and then interacts the collected information with the Execution Graph modeled as directed acyclic graphs (DAGs) [36].
- We develop *AutoDiagn*, an automated real-time diagnosis framework for big data systems, that automatically detects performance degradation and inefficient resource utilization problems, while providing an online detection and semi-online root-cause analysis for a big data system [37].
- We propose, develop, and validate a novel root-cause analysis technique/system called *BigPerf* for big data systems, incorporating Bayesian networks to model

uncertain and complex relationships between relevant factors, such as execution time of each specific task (mapper and reducer), network transmission time between these tasks, data block split time (HDFS to mapper, reducer to HDFS).

1.3 Thesis Structure

This thesis is comprised of six chapters; the organisation of the thesis chapters is presented in Fig 1.3 and the arrows here represent the flow of contents.

- **Chapter 1, Introduction** – This chapter explains the general background of diagnosis and evaluation of cloud-based big data systems. The background includes (i) an overview of big data systems and the types of data processing systems; (ii) a brief information about cloud datacenters; (iii) a general information about Quality of Service (QoS). It also reveals challenges and research questions, along with the thesis contributions.
- **Chapter 2, Literature Review** – This chapter is devoted to the literature review that reviews the literature on virtualization, monitoring, performance diagnosis, big data, Apache Hadoop ecosystem, deployment environment, and commercial and open source tools for big data systems.
- **Chapter 3, SmartMonit: Real-time Big Data Monitoring System** – This chapter proposes a real-time monitoring system that efficiently collects the runtime system information including computing resource utilization and job execution information and then interacts the collected information with the Execution Graph modeled as directed acyclic graphs (DAGs).
- **Chapter 4, AutoDiagn: An Automated Real-time Diagnosis Framework for Big Data Systems** – This chapter presents an automated system that detects performance degradation and inefficient resource utilization problems, while providing an online detection and semi-online root-cause analysis for a big data system.
- **Chapter 5, BigPerf: Probabilistic Performance Diagnosis and Prediction for Cloud-based Big Data Systems** – This chapter proposes a system for probabilistic big data performance diagnosis and prediction, incorporating Bayesian

networks to model uncertain and complex relationships between performance factors.

- **Chapter 6, Conclusion** – This chapter summarises the main findings of the research and contains a discussion, and presents future works.

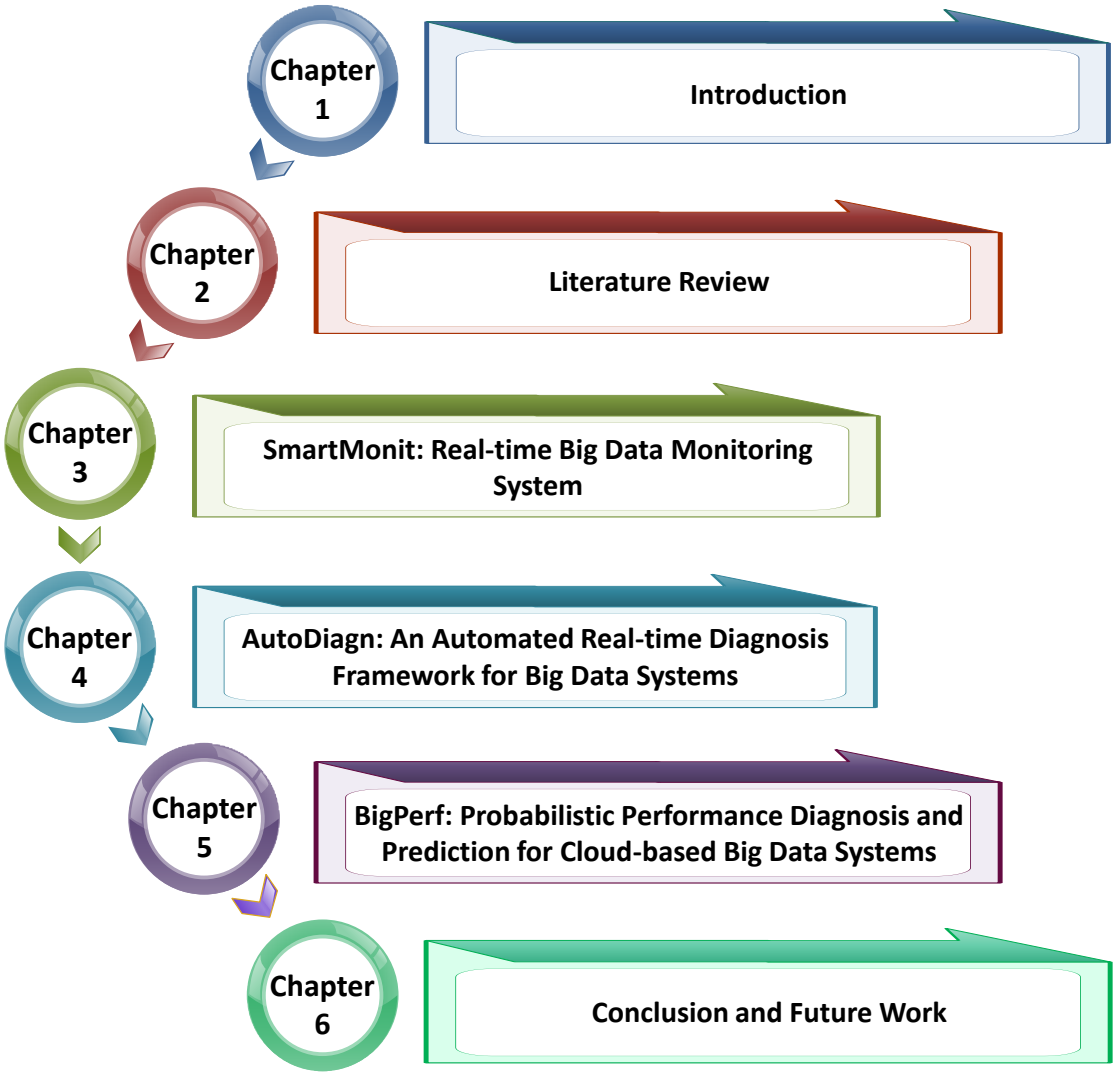


Figure 1.3: Thesis outline

2

Literature Review

Contents

Summary

This chapter presents some background information regarding the overall topic, including a brief description on big data, Apache Hadoop and its main components, big data applications based on MapReduce framework, performance diagnosis of big data systems, cloud computing and its relationship with big data, and commercial and open source tools for big data systems. A major focus of this thesis is to address the challenges of performance diagnosis and evaluation of big data systems.

2.1 Big Data

Big data is a concept that first emerged in the field of astronomy and genetics, which started to be used for the internet in time and thus became a part of our daily life without being aware of and continuously contributing to it [38]. As a result of the computer being so effective in every aspect of our lives, many data has been stored, processed and managed. With the widespread use of the Internet by companies, corporations and people, the circulation, processing, and proliferation of these data in electronic media have produced another result. The data we have mentioned includes the data entered and stored as a requirement of service, as well as a lot of data that seem to be extremely unnecessary and useless, and they grow at an avalanche. With the 2019 figures, over 2.5 quintillion bytes of data are produced in a day in the world, and in 10 years, the total data size is estimated to reach 45 times the current time [39]. So we see a garbage dump consisting of unstructured data. It was not long before it was understood that this phenomenon, which was called as information dump, was originally a large treasure since it could not be used because it was not structural. As a matter of fact, this dump consisting of data such as logs of web servers, social media sharing and publications, blogs, microblogs, internet statistics could actually be made very functional. If interpreted with the right analysis methods, these data should have been able to contribute to the making of important decisions correctly, managing risks correctly and signing new discoveries and discoveries. For these crucial reasons, big data analytics is getting more and more important in some special areas, such as government sector [40], healthcare industry [41], entertainment industry [42], weather

patterns [43], cyber physical systems [44], banking sector [45], IoT technologies [46], natural disaster management [47].

Fig 2.1 shows the growth in the last 10 years for both structured and unstructured stored data.

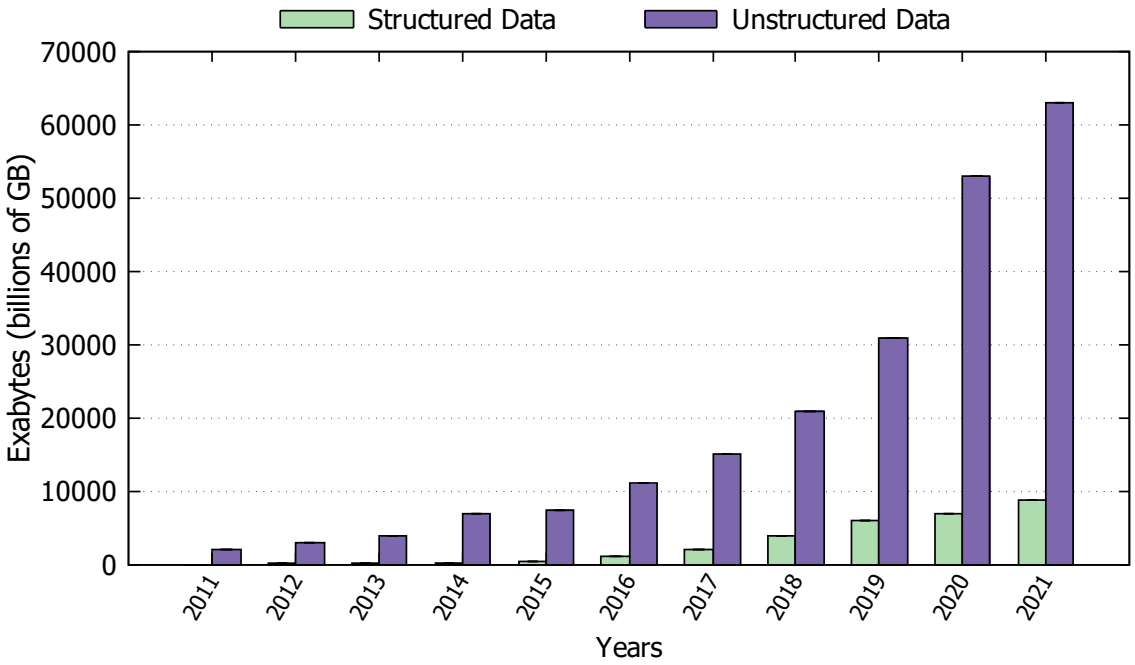


Figure 2.1: Growth in worldwide stored data [1]

2.2 Apache Hadoop Architecture

Hadoop was created by Doug Cutting and Mike Cafarella who are inspired by Google File System and MapReduce programming model in 2005. Hadoop is an open-source library (under Apache Hadoop license) written in Java that allows the users to process the petabyte size of structured, semi-structured and unstructured data sets in parallel by distributing the data across servers and clusters [48]. It has three essential parts, namely YARN which is the cluster resource management, HDFS a distributed file system to store large data sets, and MapReduce which is the processing component.

Fig 2.2 demonstrates Hadoop cluster architecture.

The following basic features have been taken into consideration while creating the system architecture of Hadoop:

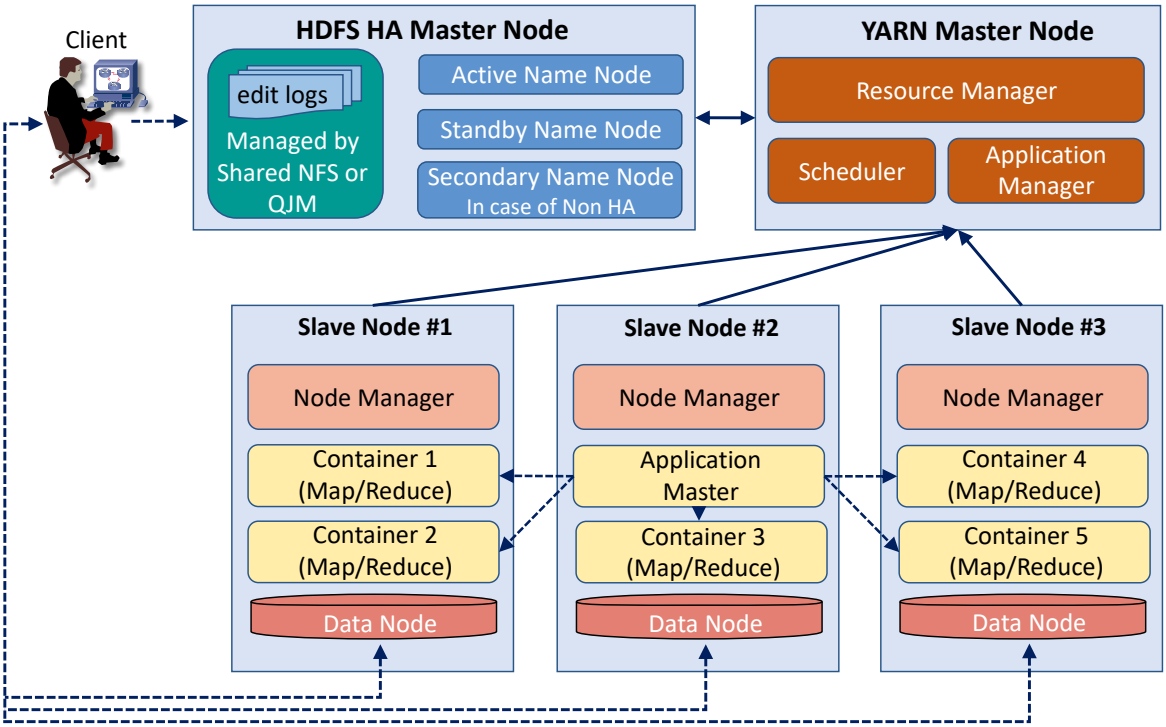


Figure 2.2: A typical Hadoop architecture

Scalable. Unlike traditional data processing and storage methods, Hadoop is a highly scalable data processing and storage platform as it consists of thousands of independent nodes interconnected through fast networks, which enables the administrators to add new nodes as many as they require.

Cost effective. As the amount of data to be stored and processed increases, the need for a more powerful hardware structure increases to handle such data, which is extremely costly in traditional methods. Hadoop provides a distributed computing and distributed storage system that can compose of different types of machines to save infrastructure costs.

Flexible. Hadoop comes with a set of tools that allow businesses to efficiently collect all kinds of data types such as structured, semi-structured and unstructured, from different sources as well as process and store them. By this means, companies derive valuable business insights from data sources such as social networking services, blogs, discussion forums, and customer review sites.

Fast. Hadoop's data processing technique called MapReduce that processes data in parallel chunks rather than in a single queue and its unique storage method bring

significant advantages in terms of speed. The combination of these novelties allows terabytes of data to be processed efficiently within minutes.

Resilient to failure. It refers to fault tolerance. While executing a job consisting of millions of tasks, if any one of the tasks fails due to any unpredictable reasons, such as hardware or network failures, Hadoop then reschedules that task in another node. Moreover, with the replication feature of HDFS, Hadoop provides reliable storage in case of any computer in the cluster fails or disconnects from the network.

The details of three main components of Hadoop are explained below:

2.2.1 YARN

YARN stands for Yet Another Resource Negotiator, responsible for resource management and job scheduling in a Hadoop cluster. It is one of the core component of Hadoop, used for utilization of the resource efficiently and scheduling tasks to be executed on different worker nodes. YARN was implemented in Hadoop 2.0. that consist of three main components, namely Resource Manager, Node Manager, Application Master. Resource Manager is the daemon that runs on the master node of the cluster that gets the submitted job by the client and schedules it over the cluster. The resources of worker nodes are allocated to the running job by the Resource Manager. Node Manager runs on each slave node and schedules the MapReduce tasks and tracks the resource utilization of the node by using Application Master and Containers. It communicates with Resource Manager to report the resource utilization of the node. The Application Master conducts the execution of a job. It executes a single mapper or reducer task in the obtained container from the Resource Manager. YARN spreads the metadata regarding the running jobs over the worker nodes using the Application Masters, which makes Hadoop fault-tolerant in the event of an error in any worker node.

The main components of YARN and the coordination between the components are depicted in Fig 2.3.

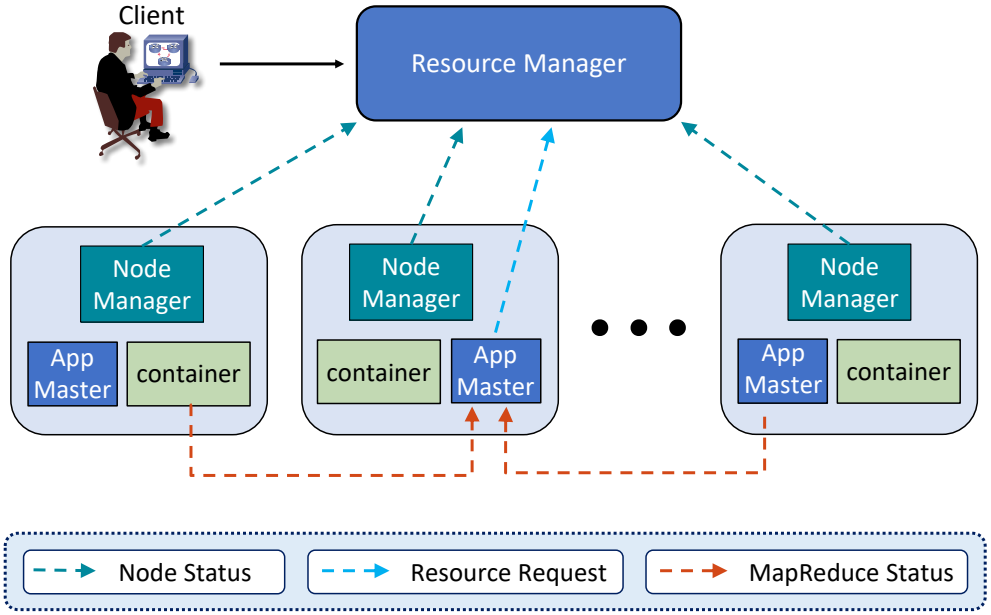


Figure 2.3: YARN architecture and its components

2.2.2 HDFS

HDFS stands for Hadoop Distributed File System which is designed for storing huge amount of data, providing high aggregate data bandwidth and and scaling to hundreds of node [48]. Large data sets stored are easily transformed and executed using MapReduce algorithm. The data is stored in blocks in HDFS and sent to the DataNode machines. The default value is 128 MB and can be changed by the users. In case of any failures in a DataNode, each block is copied (default replication number is three) on different machines to avoid data loss in machine problems, called as Replication Factor. Fig. 2.4 demonstrates the architecture of HDFS.

2.2.3 MapReduce

Apache Hadoop framework implements this pattern and allows many machines in a cluster to work concurrently in processing Big Data. It is used for processing heavy data-sets including a petabyte data-sets and is deployed on numerous nodes (usually commodity hardware machines) that process data in parallel. Most commonly the data storage nodes are the nodes that do the computing as well (moving the computing to the data rather than the other way around). This allows for a more effective data

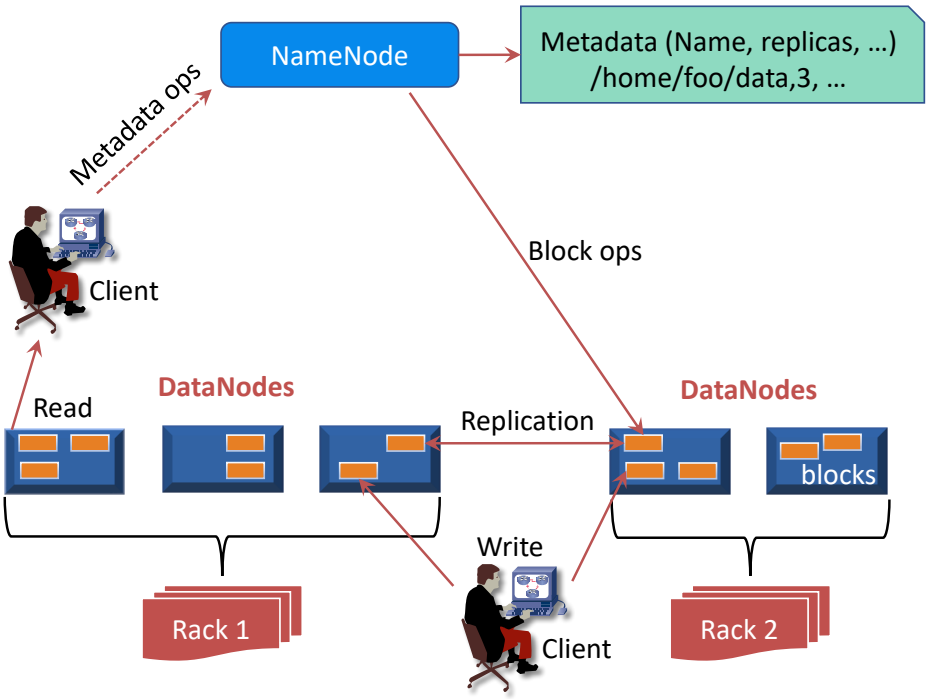


Figure 2.4: HDFS architecture

processing as the overhead of transferring data to the computing nodes is diminished and provides isolation (application cannot impact the progress on the other nodes). MapReduce is a processing technique inspired from Google, which is implemented as a Java framework in Hadoop for processing huge amount of data in parallel across multiple machines called Hadoop cluster [49]. MapReduce splits the data into several small part of data then converts each part into a set of tuples called key-value pairs, then finally reduces these sets of tuples. The figure below explains the working principle of MapReduce paradigm. MapReduce mainly consists of two main functions, Map and Reduce and is executed in three stages, Map, Shuffle and Reduce. The input data is partitioned by Map function and several small chunks of data is created. The workers execute these chucks and produces key-value pairs. Shuffle phase groups these key-value pairs by key and sends them to the responsible Reducer. Then, Reducer takes these grouped key-value pairs and assembles these data tumples into a smaller set of tuples. Finally, it generates a new set of output and stores them in HDFS. Based on the data size, many of map and reduce jobs are distributed and processed concurrently. The master-slave concept is used in MapReduce paradigm that the input file is split into smaller pieces and then each one is fed to worker nodes which process the data,

called map task. Then the outputs of the map tasks are collected by the master node, called reduce task and finally output data is generated. Fig. 2.5 demonstrates simply MapReduce working principle.

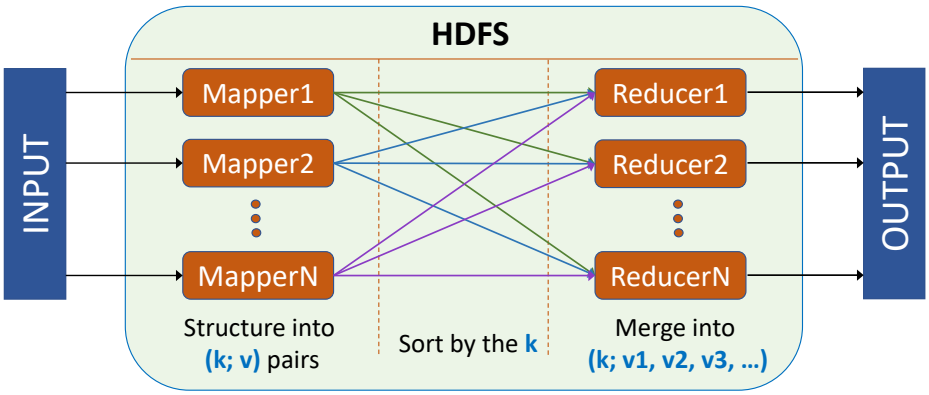


Figure 2.5: MapReduce working principle

Fig. 2.6 depicts the high-level MapReduce processing pattern. The client requests the Master node of the Hadoop cluster to execute its MapReduce program over the data, each 128 MB, stored in a distributed manner across a cluster of worker machines. Once the master node receives the client program, and then it schedules and assigns the program to idle workers. Each map task is allocated to one of the data blocks hosted in the workers by the Application Master in the master node. After the worker nodes receive the task, namely the client program, map tasks start to read the data and extract key-value pairs from the input data that is called intermediate data. This data, then, is written to the local disk to be processed by reduce tasks. Once map tasks are completed, Application Master allocates Reduce tasks in worker nodes. Reduce tasks start to read the intermediate data. First, reduce tasks sort the data based on the key-value pairs of the same key to cluster them together. Reduce tasks check all the sorted key-value pairs and finally, output data belongs to unique keys is generated by the reduce function and the partition of the output file is written on the disk.

2.3 Big Data Applications based on the MapReduce Technology

MapReduce is traditionally a batch analysis system used in applications that need to process lots of data in parallel in an offline mode. In many big data analysis

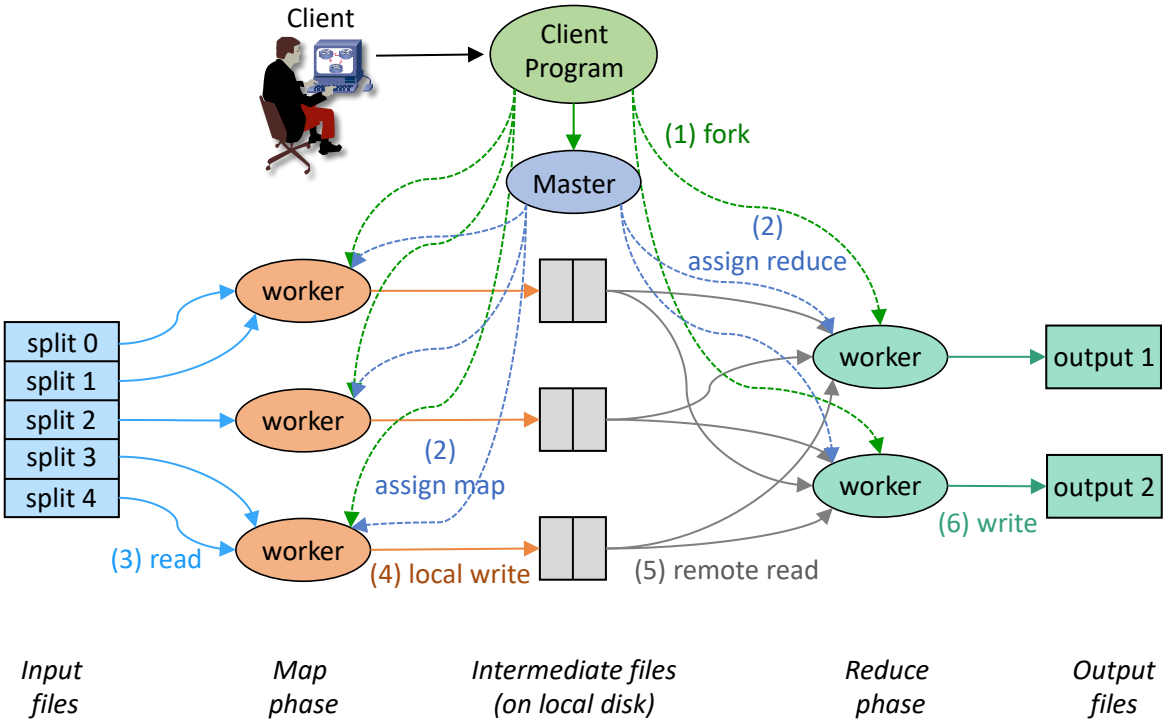


Figure 2.6: High-level MapReduce processing pattern

systems, the streaming data is continuously imported and then processed in batch at set intervals. For instance, the streaming data that belongs to the customers is collected in storage, but the MapReduce job is only executed in a given time interval, such as at the end of the day. This kind of data processing methods is very convenient for a few reasons. First, processing one minute's worth of data at once is not efficient to predict the customer's behaviours for marketing strategy. Second, processing such data causes an ineffective use of computer resources. MapReduce systems reduce the overhead of distributed computation as they typically use reasonably large block sizes.

The applications that use MapReduce technology are listed and explained in detail below.

Social Networks. In recent years, social media platforms have generated tons of data every second as it becomes universally accepted. Millions of people use such platforms for different purposes, such as generating memorable experiences, accessing and providing information, communicating with each other, sharing media, researching products and businesses, etc. This rapid growth of social media has caused some serious issues regarding data storage and management, which has led to the emergence of big data

framework technologies as the classic data process methods are insufficient to handle such data. MapReduce technique, one of Hadoop's main components, provides an appropriate method to store and analyze big social media data and is convenient to apply machine learning techniques on such valuable data as well [50]. That is why, the most popular social networks worldwide, such as Facebook, Twitter, LinkedIn, Pinterest, Instagram, are deployed on Hadoop cluster in data centers and use MapReduce technology to handle their large-scale datasets.

Entertainment. The entertainment industries are more likely to adopt new technologies so that looking at big data capabilities, reducing the cost of providing services, and generating revenue from delivering content. The biggest international video streaming services, such as Netflix, YouTube, Amazon, have to deal with a huge amount of data for achieving those goals. They use Hadoop MapReduce for building big data applications through the use of Cloud datacenters that consist of thousands of interconnected nodes to ensure consumer Quality of Service (QoS) demands. For example, Netflix processes approximately 500 billion events and 1.3 petabytes (PB) of data per day, further, during peak hours, it processes approximately 11 million events and 24 gigabytes (GB) of data on a per-second basis [51] for gaining valuable information, such as discovering the most popular movies based on the customer's watch history, providing suggestions to customers by taking into consideration their interests. Netflix has built and used its own PaaS-like layer for Amazon Elastic MapReduce, called Genie to analyze users' logs and clicks.

Healthcare. Healthcare big data refers to heterogeneous, multi-spectral, missing and indefinite observations (e.g., diagnosis, illness, injury, treatment, physical and mental impairments, demographics, and prevention of disease) data in structured, semi-structured and unstructured formats acquired from primary sources. The structured data consists of ICD codes, phenotype, genotype, genomics information while unstructured data includes medical imaging, memos, environmental, clinical notes, lifestyle, prescriptions, and health economics data [52]. The biggest challenge of big healthcare data analytics is dealing with heterogeneous data to deliver insights for providing better healthcare for millions of patients. MapReduce for such big and complex healthcare data analytics helps in solving the problems of healthcare delivery systems.

Electronic Commerce. E-commerce companies are a big part of the big data source by accumulating customers' information, such as geolocation, age, gender, interest, buying behaviour, demand. Many e-commerce providers take advantage of the power of big data analytics to elevate the shopping experience, increase personalization, optimise pricing, increase sales, predict trends, and forecast the demand. Therefore, they have to deal with the enormous volume of data generated by internet activities. Many popular e-commerce sites, such as Amazon, Alibaba, Walmart, and eBay, use MapReduce to analyze their big data including site records, purchase history, user interaction logs. In addition to that, product recommendation mechanisms are used for personalised suggestion by them. For example, Amazon uses Apache Mahout, a MapReduce-based machine learning library for building a recommender system for its customer for a particular product by analyzing comments or reviews [53].

Fraud Detection. According to Association of Certified Fraud Examiners (ACFE) report, companies lose their revenues approximately 5% to fraud in every year [54]. Fraud detection is a technique to identify fraud as soon as possible once it has been perpetrated. However, as the amount of logs that need to be analyzed increases with the large amount of data, the speed and precision of fraud detection decreases. The analysis of a wide range of data points, such as location of user, details of the device on which the account is managed, IP address, is required for fraud detection in the financial industries, such as banks, insurance companies, investment houses, payment locations, real estate brokers. Hadoop-MapReduce ecosystem is commonly used by these kind of companies to analyze the customer information to be able to identify fraud cases immediately and respond quickly.

2.4 Real-Time Performance Diagnosis of Big Data Systems

2.4.1 What is performance diagnosis?

A general definition of diagnosis: "*an investigation or analysis of the cause or nature of a condition, situation, or problem*", while performance is defined as "*the ability to perform or the manner in which a mechanism performs*" by Merriam Webster [55].

Performance diagnostic is the process of identifying and explaining performance issues that is a crucial part of parallel programming. It is a labour-intensive activity that requires a skilled workforce, especially for big data systems in cloud environments as such systems are commonly deployed in large scale distributed systems consisting of thousands of machines using parallel programming [56].

Definition: Performance diagnosis is an investigation or analysis of the performing ability or situation of the mechanism of each layer in big data systems.

2.4.1.1 The components of performance diagnosis:

Performance diagnosis consists of two main components; monitoring and root-cause analysis (RCA) [57].

- **Monitoring:** Monitoring is the action of measuring the key parameter of the components of the applications, observing the outputs and behaviour of a system, and providing infrastructure information that influences the system performance in each layer in big data systems [58]. Monitoring of big data systems is essential to keep these systems highly available and performing as well as mitigating performance issues [59, 60]. Monitoring is fundamentally an operational concept for i) management of system and hardware resources, such as computing power and storage, and ii) collecting the run-time system information including computing resource utilization and job execution information for troubleshooting the reasons for failures and performance reduction, especially the ones propagated from other causes [61]. Monitoring systems are commonly used to collect data from big data systems to analyse their operation and performance to verify the system health [62].
- **Root-cause analysis:** A commonly accepted definition of root-cause analysis is given by Rooney *et al.* [63] as: "*a process designed for use in investigating and categorizing the root causes of events with safety, health, environmental, quality, reliability and production impacts.*" The initial use of this term in big data systems was to improve the efficiency of speculative execution of stragglers [15]. In today's our modern world, the RCA in big data processing systems aims to iden-

tify not only what and how errors, faults or performance degradation occurred, but also why they happened, to prevent the recurrence of such undesirable and unpredictable events. The RCA enables faster and efficient processing while getting more accurate results as understanding the reasons why an error or performance degradation happens is the key to developing effective systems. There are three major steps towards building an RCA system consisting of the following [63]:

a) **Problem definition:** Problem definition entails determining what is exactly going on and detection of the specific symptoms of the problem. It specifies the gap between the current state and desired state of an operation.

b) **Causal factor charting:** Causal factor charting allows users to create a structure for organizing and analysis of the monitoring information and identify the gaps and incomplete or incorrect information as the investigation progresses. It is basically a kind of flow chart with logic tests that explains the circumstances before an occurrence as well as the circumstances surrounding these events.

c) **Root-cause identification:** In this step, a root-cause map, a kind of decision diagram, is created to determine the reasoning for each casual factor's underlying causes. In this way, users can easily identify the underlying reasons for any errors, faults or performance degradation.

2.4.1.2 The methods for performance diagnosis:

There are three types of performance diagnosis approaches for big data systems: statistical, data mining-based and machine learning-based as explained below:

- **Statistical performance diagnosis:** In this technique, a score representing the system performance is generated by observing the system performance metrics to diagnose the performance anomalies. A minimum threshold is set using this score. Values below this threshold are considered anomaly scores which indicates poor performance [64]. After describing the tasks showing poor performance, user-defined functions (UDF) are used to perform the customized functions over

the dataset obtained from the system for troubleshooting the cause of performance degradation.

- **Data mining-based performance diagnosis:** Data mining techniques, such as classification, clustering, regression, prediction, association rule mining, are used to diagnose the performance reductions. This technique defines some boundaries for valid activities for any system (i.e., big data systems) to discriminate between normal and abnormal behaviour [65]. To this end, the main focus of this technique is to generate descriptions and predictions about the system through the performance metrics.
- **Machine learning-based performance diagnosis:** This approach uses machine learning to diagnose the performance of the systems by using the learning ability of software over time. This technique creates a model on the basis of previous results (training data) directly without relying on a predetermined equation [66]. The accuracy of machine learning-based performance diagnosis systems depends on the training data as the number of samples available for learning increases the pattern recognition as long as the model is overfitting [67].

Statistical performance diagnosis technique is more beneficial than other techniques. Instead of detecting performance anomalies in individual values of metrics, the statistical technique checks all metrics simultaneously and it does not require any prior information or parameter estimation about the dataset. Moreover, this technique is capable of detecting previously unidentified errors or anomalies without any preliminary test after being identified by the user using UDF. Furthermore, the statistical performance diagnosis technique ensures high accuracy of the detection of the performance anomalies as compared to the other two techniques [68].

2.4.2 Why performance diagnosis?

The exponential technological advancement plays an important technological innovation in hardware and software technology, which causes an increase in system size and complexity growth of big data systems. As system size and complexity grows, performance degradation highly occurs in such systems due to software and hardware

related problems, and resource contraction. Therefore, big data systems increasingly suffer from performance degradation. However, it is not possible to diagnose such cases by examination manually as manual intervention is considerable tedious, time-consuming and error-prone [69]. To this end, an automated real-time performance diagnosis system is a prerequisite for efficient and accurate data analysis while reducing the cost of services and service-level agreement (SLA) violation rate of big data systems [70].

2.4.3 Requirements for an automated performance diagnosis platform

The analysis of the characteristics related to the realization of an efficient and reliable service that meets the requirements under the specified requirements and within the maximum limits of the system parameters demonstrates the performance of a big data system [71]. Although there is a large volume of published studies proposing solutions to improving big data systems performance, the detection of a wide range of anomalies and performance of root-cause analysis in big data systems remains a major challenge that needs a generic and comprehensive solution.

The following technical challenges must be addressed for building an automated performance diagnosis platform for big data systems:

- **Fine-grained monitoring:** A diagnosis framework needs a holistic monitoring system for continuous information collection [72] including computing resource utilization and job execution information from large-scale computer clusters in real-time [73].
- **Elasticity:** Diagnostic systems can operate continuously without interruption or manual reconfiguration to minimize resource consumption in complex environments [74], such as the cloud, where resources are unpredictable and architecture is subject to rapid changes [75].
- **Scalability:** A diagnostic system is considered scalable when it works independently of workload while maintaining performance efficiency [76]. The scalability of the system indicates its ability to grow with the demands of the system [77].

- **Multi-tenancy:** Modern big data processing systems are typically deployed in cloud environments in which many applications share the same infrastructure [78]. An ideal diagnosis system should be able to accurately identify how much of the shared cloud resources and services are used by the application in order to properly evaluate the performance of the application [79].
- **Comprehensiveness:** A performance diagnosis system should be capable of diagnosing different physical and virtual infrastructure, various platforms, and a wide range of cloud services [80], [81].
- **Generalizability:** A diagnostic system should be an independent framework that can be deployed to existing big data systems and should have the capability of adding new APIs to gain the functionality to root cause analysis [82], [83].
- **Resilience:** A system is robust to the extent that it protects its essential capabilities against interruption caused by adverse events and situations in a timely and effective manner [84], [85].
- **Reliability:** A reliant diagnosis system can deliver its service for a certain amount of time under specified conditions. In other words, the reliability of a system shows the quality of performing consistently well over time [86], [87].

2.5 Deployment Environment

2.5.1 Cloud computing

Cloud computing is a novel computing term in which dynamically scalable and often virtualized resources are delivered as services to many users over the internet. Users can easily access a large number of devices, including programs, applications and storage over the internet, via services offered by cloud computing providers. Cost reductions, high availability, and simple scalability are some of the benefits of cloud computing technologies [88]. Cloud computing is a collection of services that can be presented as a layered architecture that consists of three main services. Software as a Service (SaaS), which is the top layer of the system, allows users to run cloud-based applications remotely, such as Google apps, Facebook, YouTube. Platform as a Service (PaaS)

is an application development and deployment platform that allows users to develop, run and manage applications easily, such as AWS Elastic Beanstalk, Windows Azure, Google App Engine, OpenShift. Infrastructure as a service (IaaS) is an online service that provides virtualized computers with guaranteed resources for both processing and storage over the internet without management responsibilities, such as Amazon Web Services (AWS), Google Compute Engine (GCE), Microsoft Azure, Cisco Metacloud. There are three types of deployment models in Cloud computing [89]. First, a private cloud is a deployment approach that provides some benefits, such as virtualization, scalability, elasticity, and self-service but through a more restricted and proprietary architecture. Access to private cloud resources is extremely limited and is only available to workers of a single company. The second one is public cloud that is a deployment paradigm that allows anyone to utilize and install its services for an ongoing fee. It delivers similar advantages to the private cloud but through a less control and customizable architecture. The last one is hybrid clouds that is the combination of public and private clouds. It is one of the most robust techniques for implementing Cloud application architecture since it incorporates the functions and features of both the public and private cloud deployment models. Amazon AWS is one of the prominent providers.

2.5.2 Why cloud computing for big data?

The rapid growth of data, transfer speed, diverse data in electronic environments has led to two major big data challenges, processing and storage. Big data clusters are the best choice to overcome the limitation imposed by the limited processing and storage capacity of a single machine or computer. One of the most important feature of cloud computing is offering an internet-based remote network of interconnected machines along with elasticity associated with scale-out solutions. Big data clusters interact with web services, such as AWS, Google Cloud, Microsoft Azure, for both data processing and storage considerations in a distributed environment offered by cloud computing technologies which aim to meet the Quality of service (QoS) requirements of customers and ensure service level agreements (SLAs). The web services provide thousands of virtual computing machines providing secure, resizable compute capacity

(i.e. Amazon's EC2 vs Google Compute Engine (GCE)) for processing huge amount of data stored in a scalable, high-speed, web-based cloud storage service (i.e. Amazon S3, Google Cloud Storage (GCS)). Another important and foundational element of cloud computing is virtualization which separates the operating system from the hardware. Users can virtualize their networks, storage, servers, data, and applications via virtualization techniques [90]. Hadoop is the most popular big data framework that can scale up to thousands of nodes per cluster [91]. That is why it is essential to deploy Hadoop on a distributed cloud computing environment for getting more accurate result within a short period of time as well as ensuring restoration of data in case of data loss while increasing the resource utilization by allowing several workloads to run on each physical host server via virtualization. Moreover, in cloud computing deployment, the response time decreases as the number of instances increases, as the data is distributed and processed over all the nodes via the Hadoop cluster [92].

2.6 Commercial and Open Source Tools for Big Data Systems

There are a variety of tools developed by both academia and companies for big data systems described below:

Datadog [11] is cloud-scale monitoring and analytics solution that aggregates the data from databases, servers and different tools to monitor Hadoop clusters in a unified environment via the dashboard. It provides real-time tracking for the system and alerts when a certain threshold is exceeded. Hadoop cluster can be monitored by installing the Datadog Agent on the worker nodes in the cluster. Its agents collect the specific metrics from each node as well as the status of each job and task. Furthermore, Datadog is used for network and security monitoring for cloud-based systems [93]. However, Datadog shows anomalies but cannot do root-cause analysis for such cases [94].

SequenceIQ [12], developed by Hortonworks computer software company to accelerate Hadoop deployment, and is now a project under Hortonworks/Cloudera licence [95], is yet another framework for Hadoop that its architecture is built on the ELK

stack, which includes Elasticsearch, Logstash, and Kibana. It aims to achieve a clear separation between monitoring tools and the Hadoop implementation by leveraging a Docker-based architecture. SequenceIQ monitors applications based on resource utilization (i.e., CPU, memory) by applying auto-scaling to Hadoop YARN via Service Level Agreement (SLA) policy. SequenceIQ evaluates customers costs with the detailed measurement of their MapReduce jobs. However, it is not able to do root-cause analysis and investigation of any failures.

Sematext [13] is a tool for big data systems that allows users to collect metrics, logs, and events from the frontend to the backend from the whole system. It provides both real-time metrics collection and anomalies detection, such as slow transactions and communication between systems and applications. It can be easily integrated with existing software and tools, such as Solr, Apache Hadoop, Docker, Apache Kafka, Apache Spark, Apache Storm, MongoDB, Nginx, Cassandra, Elasticsearch. Moreover, users can have predefined conditions in their metrics to be notified when the conditions are met, for example, if CPU usage reaches a certain threshold. Sematext also provides visualization for big data tasks by a user-friendly graphical interface. However, it does not conduct root-cause analysis for any of the systems.

TACC Stats [14] is a continuous monitoring tool that collects the infrastructure information and progress status of each individual task from HPC systems. It has been in production use for about five years on numerous different systems and currently is used by many HPC systems. It enables analysis and reporting through the collected data, such as resource utilization, energy consumption, network, and I/O activity. It is able to give insights about the performance reduction regarding job errors, system faults, and resource needs based on resource utilization. However, TACC Stats does not have the ability for visualization of the collected data and does not provide root-cause analysis. Furthermore, it developed for HPC clusters and are not suitable for big data systems.

Mantri [15] presents a real-time monitoring and systematic method that categorizes the main reasons causing outliers in a big data system. It characterizes the prevalence of stragglers in Hadoop systems as well as troubleshooting the cause of stragglers. Moreover, it utilized the outputs of the root cause analysis to improve the resource

allocation in Hadoop clusters. However, Mantri focuses on only the MapReduce programming framework in the Hadoop system; does not discuss the other big processing frameworks, such as Apache Spark[1], Apache Flink[2].

DCDB Wintermute [16] provides not only the feature of real-time monitoring but also are able to identify the performance issues and troubleshoot the cause of the issues. It provides a wide range of configuration options to meet the different needs of Operational Data Analytics (ODA) applications on HPC systems. However, it is not possible to monitor and debug big data systems using DCDB Wintermute as it is suitable only for HPC systems.

Nagios [17] is an open-source platform that provides multi-layer monitoring including systems, networks, and infrastructure for cloud-based systems, such as a number of protocols (HTTP, ICMP, SNMP, FTP, SSH), nodes resources (CPU, memory, I/O), as well as the information regarding host hardware. Although it requires a centralized server to collect the metrics, this allows users to plugin their own scripts. Nagios also provides a user-friendly dashboard based on the web that shows the system metrics as well as the current status of the system. Nagios can work both agent-based and agentless. It consists of many different agents performing different tasks, such as collecting and transferring the data, when it uses the agent-based model. On the other hand, it uses WMI and SNMP technologies for the agentless model [96]. However, it requires a centralized server to collect the systems metrics.

Ganglia [18] is a scalable distributed monitoring system that provides a time-series perspective regarding machine resource usage metrics, such as CPU, memory, storage, and network use to provide insight into how high-performance computer systems are used. Since it is easily scalable and distributed software, it works well with high-performance computing (HPC), clusters of thousands of computers, and as well as grid computing systems. Ganglia makes extensive use of technologies such as XML, XDR and RRDtool for data representation, data transfer, and data storage and visualization, respectively. As it provides built-in support for HBase (a non-relational NoSQL database created as part of the Apache Software Foundation's Hadoop project), it is a

[1]https://spark.apache.org/
[2]https://flink.apache.org/

popular tool widely used to monitor a Hadoop cluster [97]. Ganglia's decentralized architecture provides high concurrency and a very low per-node overhead. The computer system metrics are monitored and forwarded via the monitoring daemons running on each compute node. In parallel with this process, this monitored data is pulled by meta daemons and stored in a database. It also provides a PHP web frontend that displays all the collected metrics properly in the browser, which helps users to understand how the system works. However, Ganglia does not have an alert feature that can alert or inform the user in case of an error or a slowdown in the system.

Apache Chukwa [19], released under Apache 2.0 license, is a scalable, distributed, and open-source system for monitoring and data collecting from large-scale distributed systems. Chukwa is built on top of the HDFS and MapReduce framework, which makes it scalable and robust. It gives a better understanding of tasks status, resource usage, and the system's health by the generated results from the collected metrics [98]. Apache Chukwa consists of three main components, namely collector, agent, and adaptor. Agents, which has many adaptors, are deployed on each worker node. The monitoring metrics are collected by adaptors and reported to the relevant agent. And agents, then, transfer the collected data to the collector that sinks the data in a Data Sink File in HDFS. Although Apache Chukwa is one of the most commonly used monitoring frameworks for big data systems, it is not suitable for supporting a root-cause analysis.

DMon [20] is a monitoring project designed as a Web service and developed in the framework of the DICE project. It collects performance and quality-related metrics from each of the sub-components in big data frameworks (i.e., Hadoop) by deploying several sub-components. Thanks to its distributed and high availability architecture, it can be easily integrated into the other big data systems. Similar to SequenceIQ, DMon also uses the ELK stack as its core service, which provides a robust base for itself, to process and ingest the unstructured logs. In addition to all these, DMon has the ability to detect data locality issues and to repair such problems using targeted optimizations [99]. However, it does not have alert and visualization options.

Apache Ambari [100] is an open-source tool developed by the Apache Software Foundation that allows monitoring and managing a Hadoop cluster easily with the help of

REST APIs. It also provides a web user interface to show the collected metrics as well as to manage the Hadoop cluster. System administrators can select metrics and define thresholds for them to create alerts. System administrators can easily manage the operations in YARN (a core component of Hadoop) that comes along with the Hadoop 2.0 as Ambari is designed for the Hadoop cluster to make its management simpler. Apache Ambari deploys a master/slave architecture, in which the master node manages the slave nodes to carry out specific tasks [101]. Besides, it has the ability to notify the users when something goes wrong. However, like the other discussed monitoring tools, it cannot pinpoint the reasons why it happens. Although Ambari is a good solution for monitoring and managing the cluster, it is not enough for optimizing and increasing the performance of big data systems.

Table 2.1 presents a brief overview of various tools for big data frameworks with the help of [102]. The poor/good/excellent representations for *Big data frameworks support* are the indications of how many different systems the tools can run on. Poor (*) shows that the tool can only work on Hadoop/MapReduce framework, while Good (**) ones support Apache Spark framework as well as Hadoop/MapReduce. In the context of *Real-time monitoring*, Good (**) specifies that the tool performs near real-time tracking while Excellent (***) tracks big data systems in real-time. All these results are obtained by comparing the experimental results in the related articles with each other.

Table 2.1: A comparison of the literature review to the major issues addressed in this thesis.

Feature	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]
Big data frameworks support	**	*	**	×	*	×	*	*	*	**
Real-time monitoring	***	***	***	***	***	***	***	**	***	**
Root-cause analysis	×	×	×	×	✓	✓	×	×	×	✓
Underlying resource monitoring	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Real-time monitoring for big data tasks	✓	✓	✓	×	✓	×	×	×	✓	✓
Auto-scaling functionality	✓	✓	✓	✓	✓	✓	×	×	✓	✓
User-defined alerts	✓	×	✓	×	×	×	✓	×	×	×
Visualization of big data tasks	✓	×	✓	×	×	×	×	✓	×	×
User-customized root-cause analysis	×	×	×	×	×	×	×	×	×	×
Hidden/implicit relations analysis	×	×	×	×	×	×	×	×	×	×

Abbreviations: ×, No; ✓, Yes; *, Poor; **, Good; ***, Excellent ;

3

SmartMonit: Real-time Big Data Monitoring System

Contents

Summary

This chapter presents SmartMonit, a real-time big data monitoring system, which collects infrastructure information and process status of each task. At the same, we develop a real-time stream process framework to analyze the coordination among tasks to tasks and infrastructures to tasks. This coordination information is essential for troubleshooting the reasons for failures and performance reduction, especially the ones propagated from other causes.

3.1 Introduction

Monitoring is the key factor for cloud-based big data systems to give the idea regarding the system health and status. The big data systems, such as Hadoop and Spark are running in the large-scale computer cluster. For example, Fuxi [103] is an extended implementation of YARN, which is deployed in a cluster with over 5000 nodes and serves hundreds of millions of customers at Alibaba. For these large-scale systems, there are two key issues that cause performance reduction and inefficiently resource utilization. First is the task failures caused by diverse sources of software and hardware faults, and the second is the unsuitable scheduling policies.

The fault detection in big data systems, however, is very hard due to considerable scale, distributed environment and a large number of concurrency jobs. State-of-the-art research is not focused on detecting *emergent failures*. The *emergent failures* happen when the errors exceed the propagation boundaries during the interaction among hardware and software components, and can only be identified at run-time [21]. For example, the *stragglers* or *tailing behavior* in Hadoop system; the slower execution of a job may cause the late-time failures for many other tasks which have strict time constraints related to service-level agreements (SLAs). Moreover, the cluster scheduler uses some heuristics that prioritizes the important jobs and fairly allocates the resource among the jobs [104]. However, these methods ignore the information of the job structure (or dependencies) and schedule the jobs to the available resources without considering the job structure in run-time [105].

In order to detect the *emergent failures* or root the reasons for the performance reduction, we need to have a comprehensive and consistent monitoring plan to collect the information from each individual process job, while storing, maintaining and analyzing very large volumes of the monitoring data[44]. The monitoring tools such as Google Cloud Monitoring[1], Sparklint[2], and Datadog [3] aim to collect various information such as CPU, memory, disk, available bandwidths and execution status of each job. However, they are not able to do a real-time multiresolution analysis that narrows the scope and increases the resolution, thereby pruning the non-important information.

A monitoring system basically consists of two main components, namely agent manager and agents. Agents are plugged into each VM/container without considering the cluster heterogeneity to collect performance metrics and send the metrics to the agent manager placed into a separate machine. The manager is responsible for gathering, preprocessing and filtering the data from the agents. Finally, it sends this data to the database via a message broker (i.e., RabbitMQ[4] vs Apache Kafka) to ensure data transmission reliability [94]. Fig. 3.1 shows an effective conceptual architecture for implementing a monitoring system for big data systems.

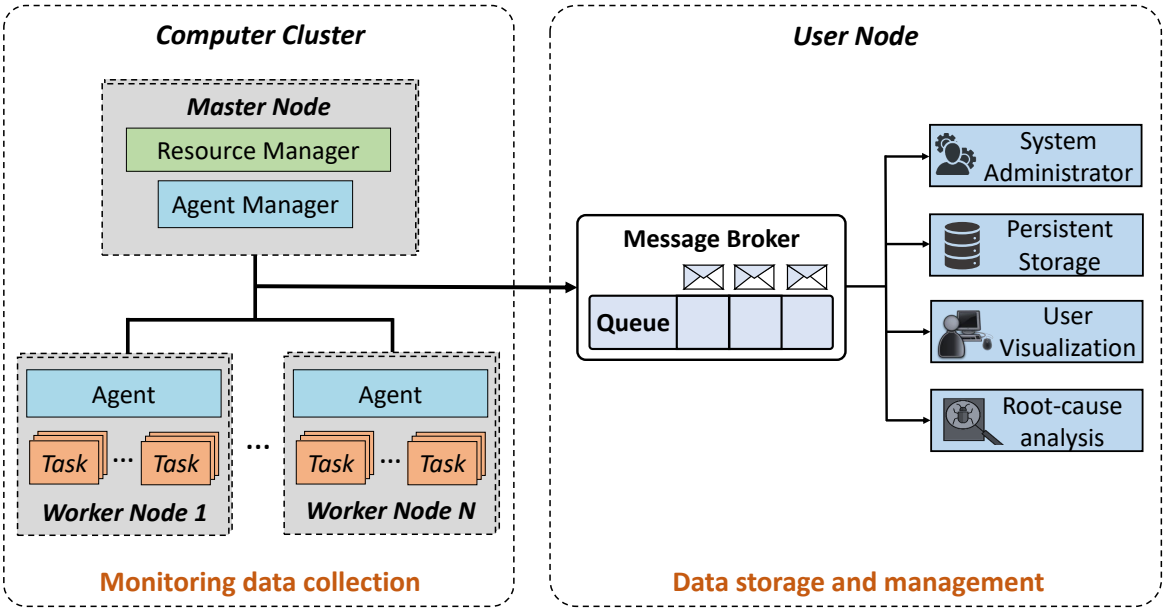


Figure 3.1: Example scenario for monitoring big data systems

[1]https://kubernetes.io/docs/tasks/debug-application-cluster/resource-usage-monitoring/
[2]https://github.com/groupon/sparklint
[3]https://docs.datadoghq.com/
[4]https://www.rabbitmq.com/

In this chapter, we propose a real-time monitoring system that efficiently collects the run-time system information including computing resource utilization and job execution information and then interacts the collected information with the *Execution Graph* modeled as directed acyclic graphs (DAGs). For example, our system is able to capture the job execution stages and the dependencies of each job in real-time; at the same, the resource utilization of each job and its underlying host are monitored as well. The main contributions are summarized as follows.

- We develop a big data monitoring system which can efficiently collect the comprehensive monitoring information from large-scale computer cluster in real-time.
- At the same, we process these streaming data and interact the processed data with the *Execution Graph* of each task while visualizing the interaction in real-time.

To demonstrate the effectiveness of our system, we plugged our system into a Hadoop cluster deployed on AWS. The above mentioned monitoring information is collected in real-time and visualized in a user-friendly interface. Additionally, we again plugged our system in a small Hadoop cluster deployed on AWS for the visualization that is available as a screen-cast video on YouTube [106].

The chapter is organised as follows. §3.2 discusses the related work. §3.3 presents the system overview while §3.4 discusses the experiment and results analysis. §3.5 explains the visualization while §3.6 discusses the limitations of this chapter and highlights our further work. Finally, §3.7 concludes the chapter discussing the conclusion.

3.2 Related Work

There are a number of big data systems tools developed by both academia [14], [19], companies [11], [13] as well as community tools in open repositories [12], [18].

Datadog [11] provides real-time monitoring by collecting data from databases, servers and different tools to monitor Hadoop clusters in a unified environment via the dashboard and alerts when a certain threshold is exceeded. Datadog agents are installed on

each running node in the cluster, providing monitoring of the Hadoop cluster. Datadog is also used for network and security monitoring for cloud-based systems [93]. However, Datadog shows only anomalies, it cannot perform root cause analysis for such cases [94]. SequenceIQ [12] is built on the ELK stack, which includes Elasticsearch, Logstash, and Kibana [95]. It uses a docker-based architecture that monitors resource utilization (i.e., CPU, memory). Although it is very efficient in monitoring big data clusters by means of auto-scaling facilities, it is not able to do root-cause analysis and investigation of any failures. Sematext [13] is capable of collecting metrics, logs, and events from the frontend to the backend of the whole system in real-time. It detects anomalies, such as slow transactions and communication between systems and applications. Furthermore, customers may have specified conditions in their metrics and be informed when the circumstances are satisfied, such as when CPU consumption hits a specific threshold. Sematext's user-friendly graphical interface also allows for the visualisation of massive data jobs. It does not, however, do a root-cause analysis on any of the systems as its system is not convenient to perform such tasks. Nagios [17] collects logs from different layers of big data systems including systems, networks, and infrastructure in real-time. It allows users to plugin their own scripts to monitor centralized servers. It also has a user-friendly web dashboard to show the system metrics as well as the health status of the system [96]. However, it is not capable of monitoring the big data tasks such as mapper, reducer, and is not able to provide an infrastructure to do root-cause analysis for performance degradation for big data systems. Ganglia [18] is able to collect metrics regarding resource usage, such as CPU, memory, storage, and network use to provide information about how the system works. It is a popular tool for monitoring a Hadoop cluster since it has built-in support for HBase (a non-relational NoSQL database produced as part of the Apache Software Foundation's Hadoop project) [97]. It also includes a PHP web interface that appropriately shows all gathered metrics in the browser, allowing users to understand how the system works. However, Ganglia lacks an alert function that can notify or warn the user in the event of a system fault or slowness. Apache Chukwa [19] provides a better knowledge of job status, resource utilization, and system health [98] that is scalable and resilient since it is built on top of the HDFS and MapReduce frameworks. Despite being one of the

most widely used monitoring frameworks for large data systems, Apache Chukwa is not ideal for root-cause analysis.

Although there already exist some monitoring tools for big data systems, there is a lack of a comprehensive and consistent monitoring tool to enable performing root-cause analysis of the problems for those who want to optimise their big data systems.

3.3 System Overview

Monitoring systems are commonly used to collect data from any systems to analyse their operation and performance to verify the system health [107]. The proposed SmartMonit is an agent-based model for monitoring big data systems, which facilitates the implementation of new data collection APIs. It consists of two agents, namely *SmartAgent* and *Agent*. The metrics are collected via these two agents and stored in a time-series database. *SmartAgent* collects the status of each task along with the cluster information processed data while *Agent* collects the infrastructure information of the system to check the health status of each worker node in the cluster. Fig. 3.2 shows the methodology of the integration of SmartMonit in a Hadoop cluster. The details of the system architecture will be discussed in §3.3.1.

3.3.1 System Architecture

This section explains the architecture and implementation details of SmartMonit. Fig. 3.3 shows three main components of our SmartMonit including *Information Collection*, *Computation and Storing* and *Visualization*.

3.3.1.1 Information Collection

The *Information Collection* is used to collect the job and task metrics and the resource utilization of the nodes in a large-scale computer cluster in real time via *SmartAgent* and *Agent*. The *SmartAgent* is deployed on the master node collects the specific information of tasks (mappers and reducers), application (job details) and the cluster information from worker nodes through the NodeManagers using *ResourceManager*

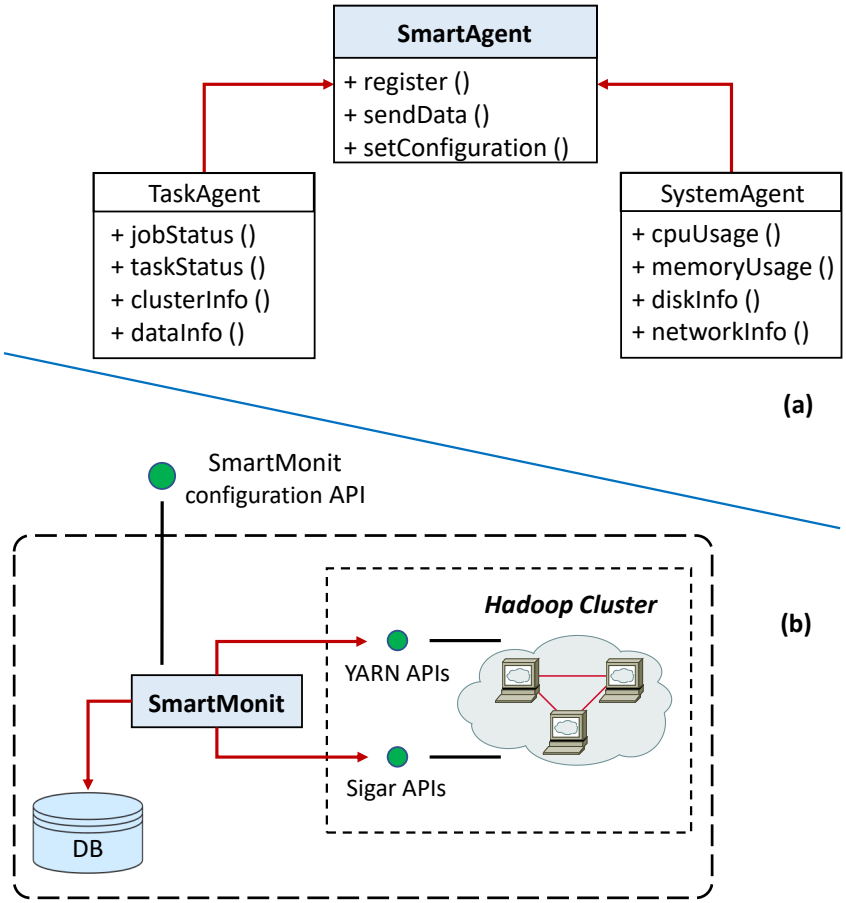


Figure 3.2: Monitoring agents model (a); Implementation of SmartMonit mechanism in a Hadoop cluster (b).

REST API's[5]. Also, the *SIGAR* library[6] is plugged into the *SmartAgent* to monitor the utilization of the resources in the Master node, including CPU, memory and network bandwidths. The collected all the data is filtered by using GSon Library[7] to remove the redundant information. The *SmartCollector* obtains the process information that can be used to build the *Execution Graph* (the details is discussed in §3.5.2). All monitoring information is sending as streaming data to the *Computation and Storing*. Moreover, in the slave nodes, the *Agent* collects the process information by using *SmartCollector* and the utilization of the resources in the host node via *SIGAR* library. The obtained information is directly sent to *Computation and Storing*.

SmartMonit exposes a set of simple interfaces for system monitoring. While Table 3.1 shows the APIs, a set of real-time stream processing functions, used to collect the jobs-

[5]https://hadoop.apache.org/docs/r3.2.1/hadoop-yarn
[6]https://github.com/hyperic/sigar
[7]https://github.com/google/gson

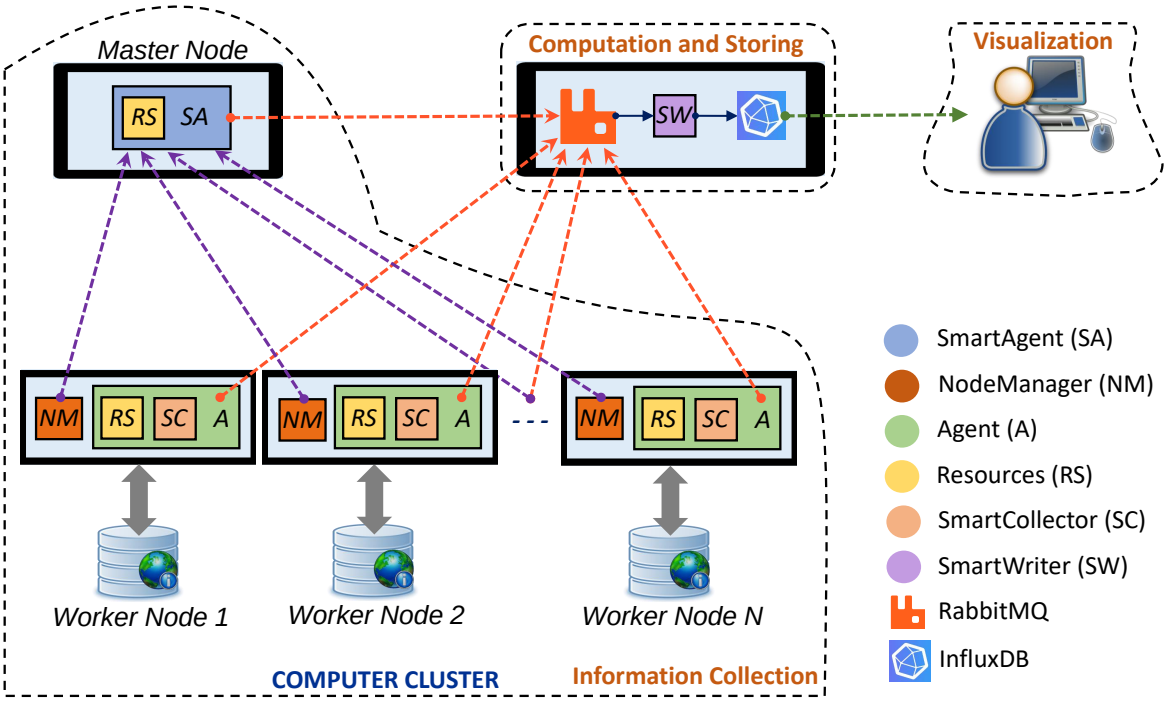


Figure 3.3: The framework of SmartMonit.

related information, Table 3.2 demonstrates the APIs that collect the system-related information from large-scale big data systems.

3.3.1.2 Computation and Storing

The *RabbitMQ Server*[8] is used to collect monitoring information sending from the cluster; the *RabbitMQ Server* is an open source message broker system which provides high throughput, low latency and reliable communication among the applications to applications. Then, the *SmartWriter* analyzes the collected information pulling from *RabbitMQ Server* in real-time and writes the processed results into *InfluxDB*[9] which is an open-source time series database. This database provides high-availability storage and the retrieval of time series data, such as operations monitoring data, sensor data and application metrics.

[8]https://www.rabbitmq.com/
[9]https://www.influxdata.com/

Table 3.1: SmartMonit monitoring interface for jobs.

Information Collection	Description
Job information	
`jobSubmitTime()`	Return the `time in which job submitted`.
`jobResponseTime()`	Return the `response time of the job` in sec.
`jobStartTime()`	Return the `time the job started`.
`jobMakespan()`	Return the `elapsed time` since the job started in sec.
`jobClusterUsage()`	Return the `percentage of resources` of the cluster that the job is using.
`jobProgress()`	Return the `progress` of the job in percentage.
`jobMapProgress()`	Return the `progress of the map tasks` in percentage.
`jobReduceProgress()`	Return the `progress of the reduce tasks` in percentage.
`jobTaskPending()`	Return the `number of the tasks` still to be run.
`jobTaskRunning()`	Return the `number of the running tasks`.
`jobTaskCompleted()`	Return the `number of the completed tasks`.
`jobState()`	Return the `job state` according to the Resource Manager.
`jobScheduler()`	Return the `type of scheduler` algorithm valid in the cluster.
`jobReplicaNum()`	Return the `number of times` Hadoop framework replicate each data block.
Task (Mapper/Reducer) information	
`taskStartTime()`	Return the `time in which the task started`.
`taskProgress()`	Return the `the progress of the task` in percentage.
`taskExecTime()`	Return the `execution time` since the task started in sec.
`taskHost()`	Return the `name of the node` this map runs on.
`taskBlockId()`	Return the `id of the block` processed by the task.
`taskState()`	Return the `the state` of the task.
`taskInputData()`	Return the `data size read by task` in mb.
`taskOutputData()`	Return the `data size written by task` in mb.
`taskFinishTime()`	Return the `time in which the task finished`.
Cluster information	
`clsActiveNodes()`	Return the `number of active nodes`.
`clsAvailableMem()`	Return the `amount of memory available` in mb.
`clsAvailableVCores()`	Return the `number of available virtual cores`.
`clsTotalMemory()`	Return the `amount of total memory` in mb.
`clsTotalVCores()`	Return the `total number of virtual cores`.
`clsTotalNodes()`	Return the `total number of nodes`.
`clsUnhealthyNodes()`	Return the `number of unhealthy nodes`.
Container information	
`containerMem()`	Return the `total of memory allocated` to the job's running containers in mb.
`containerCores()`	Return the `number of virtual cores` allocated to the job's running containers.
Data information	
`inputDataSize()`	Return the `size of input data` in mb.
`outputDataSize()`	Return the `size of output data` in mb.
Input Data information	
`blockLocations()`	Return the `location of the blocks` the job used.
`inputReplicaNum()`	Return the `number of replication` of the input file.

Table 3.2: SmartMonit monitoring interface for system.

Information Collection	Description
Resource information	
`nodeVCoreNum()`	Return the `number of cores` in the node.
`nodeCpuUsage()`	Return the `cpu usage` of the node in percentage.
`nodeProcCpu()`	Return the `cpu usage by each process` in the node in percentage.
`nodeProcMem()`	Return the `memory usage by each process` in the node in percentage.
`nodeMemUsage()`	Return the `memory usage` of the node in percentage.
`nodeFreeMem()`	Return the `size of free memory` of the node in mb.
`nodeUsedMem()`	Return the `size of used memory` of the node in mb.
`nodeTotalMem()`	Return the `size of total memory` of the node in mb.
`nodeUpload()`	Return the `upload speed` of the node as MB/s.
`nodeDownload()`	Return the `download speed` of the node as MB/s.
`diskReadSpeed()`	Return the `disk read speed of the node` as GB/s.
`diskWriteSpeed()`	Return the `disk write speed of the node` as GB/s.

3.3.1.3 Visualization

The *Visualization* includes two parts: *query engine* and *user interface*. The *query engine* queries the database in a pre-defined time interval to build the *Execution Graph*. The *Execution Graph* and other collected monitoring information is presented in a user friendly interface. The details will be discussed in §3.5.

3.4 Experimental Evaluation

In this section, we present the experimental results of SmartMonit to evaluate its efficiency and applicability in large-scale data processing systems in cloud data center environments.

3.4.1 Experimental setup

Environments. A comprehensive evaluation is conducted to evaluate our proposed monitoring system. To this end, we deployed Hadoop 3.2.0 on a VM-based infrastructure over 31 AWS nodes with 1 master and 30 workers, each of which has a Ubuntu 18.04 LTS (HVM) operating system. All nodes have 4 CPU cores and 16 GB memory.

Similar to the *micro-benchmark*, we deployed a VM with the same instance type outside the Hadoop cluster the *computation and storing* model. The methodology behind collecting the monitoring information is the same as experiment setup one.

Benchmarks and workload. We used six well-known Hadoop benchmarks in our evaluations namely: WordCount[10], Grep[11], TPC-H[12], TPC-DS[13], K-means clustering[14], and PageRank[15]. We used 11 different datasets varying in size from 6 to 128 gigabytes. In particular, we have used the datasets, containing only text, obtained from the PUMA[16] for WordCount and Grep benchmarks. For TPC-H and TPC-DS, the datasets are generated by executing the JAR provided at the addresses listed in the related footnotes. Similarly, the dataset for K-means clustering is generated using the JAR, available here[17], with the specified number of dimensions and clusters. Finally, the dataset, which consists of reviews from Amazon, for the PageRank benchmark is generated by executing the JAR provided by Stanford University[18]. Table 3.3 shows the details of the experiments conducted.

Table 3.3: The experiment environments regarding workload and system specifications

Benchmark	**Task number range**	**Data size range (GB)**	**CPU (core)**	**Memory (GB)**
WordCount	48 to 1024	6 to 128	4	16
Grep	48 to 1024	6 to 128	4	16
TPC-H	20 to 435	6 to 128	4	16
TPC-DS	25 to 535	6 to 128	4	16
K-means	46 to 1022	6 to 128	4	16
PageRank	47 to 1023	6 to 128	4	16

[10]http://wiki.apache.org/hadoop/WordCount
[11]http://wiki.apache.org/hadoop/Grep
[12]http://www.tpc.org/tpch/
[13]http://www.tpc.org/tpcds/
[14]https://en.wikipedia.org/wiki/K-means_clustering
[15]https://en.wikipedia.org/wiki/PageRank
[16]https://engineering.purdue.edu/p̃uma/pumabenchmarks.htm
[17]https://github.com/mameli/k-means-hadoop
[18]http://snap.stanford.edu/data/web-Amazon-links.html

3.4.2 Performance and overheads

Performance evaluation. SmartMonit collects the task metrics via *SmartAgent* every three seconds as the Hadoop APIs release new information every 2-3 seconds while the infrastructure information of the system is collected via *Agent* every second as the resource usage of the system changes every second. *SmartAgent* needs to complete the metrics collection from all the running tasks (e.g. mapper or reducer) every three seconds. So, we evaluate the performance of *SmartAgent* if it catches up with the deadline. To this end, we evaluate the scalability and performance of our presented SmartMonit monitoring tool by measuring the end-to-end delivery time of *SmartAgent* with six different benchmark applications with different-sized datasets.

Fig. 3.4 shows the completion time of the metric collection based on the number of tasks running in parallel in milliseconds.

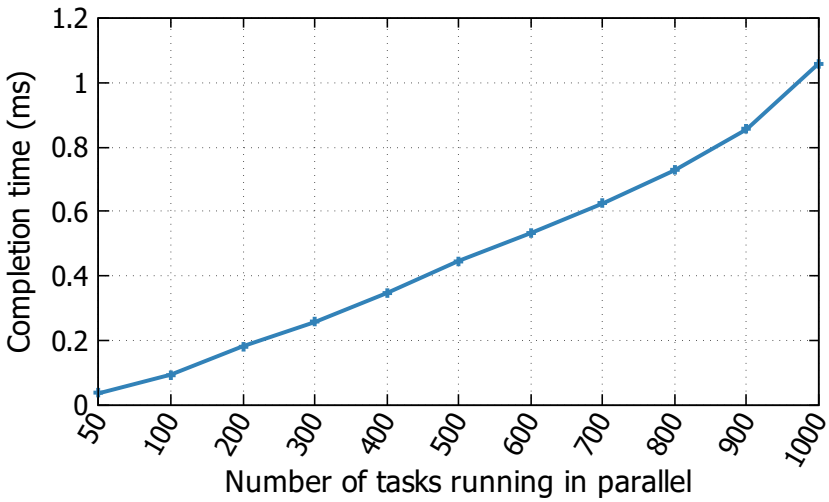
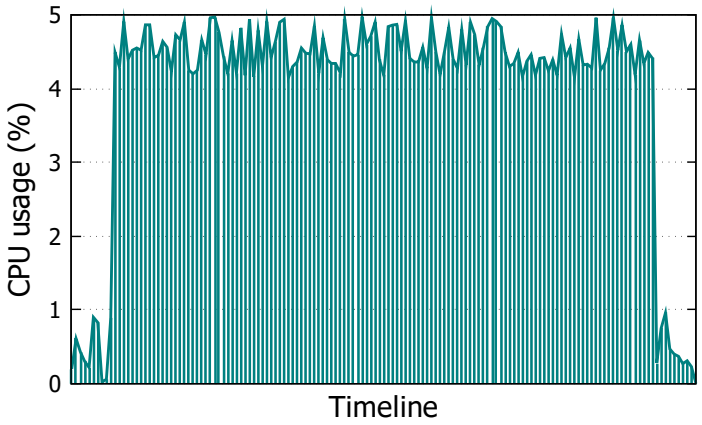


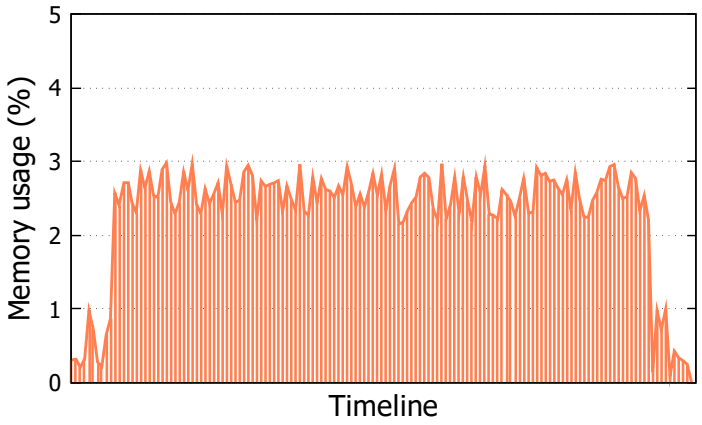
Figure 3.4: Metrics collection completion time

System overheads. We measure the CPU and memory usage of SmartMonit to evaluate the system overhead introduced by it. Fig. 3.5(a) shows the CPU utilization of SmartMonit while Fig. 3.5(b) demonstrates the memory utilization of SmartMonit. In summary, SmartMonit consumes approximately 2.58% memory and 4.53% CPU.

Moreover, Fig. 3.6 demonstrates the network and storage overheads of our tool. The extra network load and and total storage introduced by SmartMonit are very low, but they increase as the number of tasks running in parallel increases. For example, when



((a)) CPU utilization



((b)) Memory utilization

Figure 3.5: Resource utilization of SmartMonit.

the number of parallel task is 100, there are about 22.5 KB/s data sent from agents to RabbitMQ cluster. In addition, when the jobs is completed, the total size of this data in disk is 2.85 MB/s. The network and storage overheads increase to 223 KB/s data and 7.2 MB/s, respectively, when the number of parallel tasks is 1000.

3.4.3 Execution time evaluation of the benchmarks

In this section, we evaluate the execution time for six well-known Hadoop benchmarks as indicated in Table 3.3. 11 different datasets varying in size from 6 to 128 gigabytes are the input of each benchmark application.

Fig. 3.7, Fig. 3.8, Fig. 3.9, Fig. 3.10, Fig. 3.11, Fig. 3.12, show the impact of data set size on execution time in seconds for Hadoop benchmarks namely: WordCount, Grep, TPC-H, TPC-DS, K-means, and PageRank, respectively. It is clearly seen

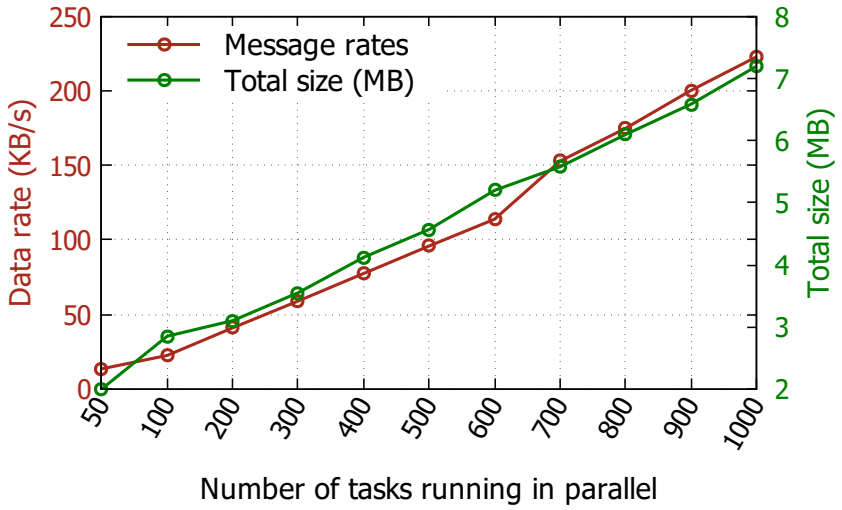


Figure 3.6: The network and storage overheads of SmartMonit

that although the sizes of the datasets are the same for each benchmark, a different numbers of tasks are executed, resulting in different execution times. Moreover, some benchmarks have the same number of tasks (i.e., WordCount and Grep), they have different execution time.

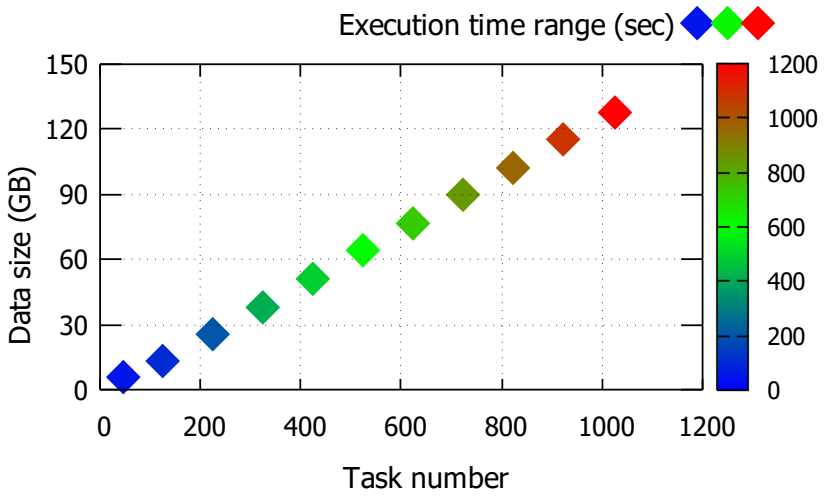


Figure 3.7: WordCount execution time on different data size

3.5 Visualization

This section demonstrates the execution workflow of our SmartMonit by interacting it with a micro-benchmark.

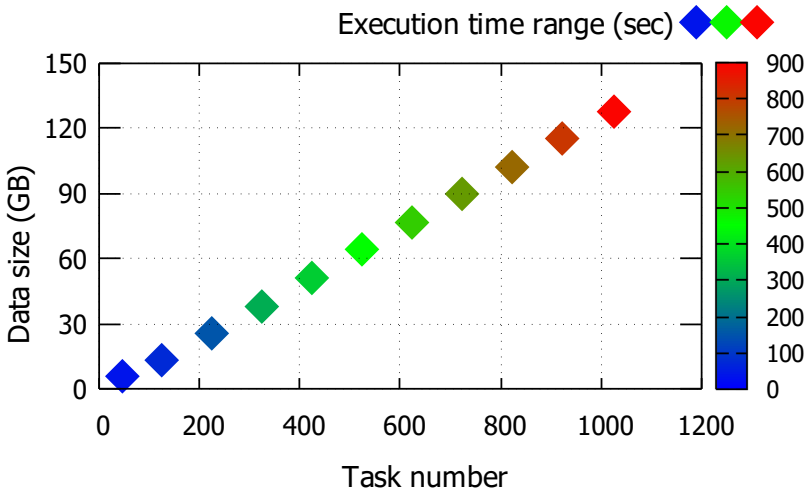


Figure 3.8: Grep execution time on different data size

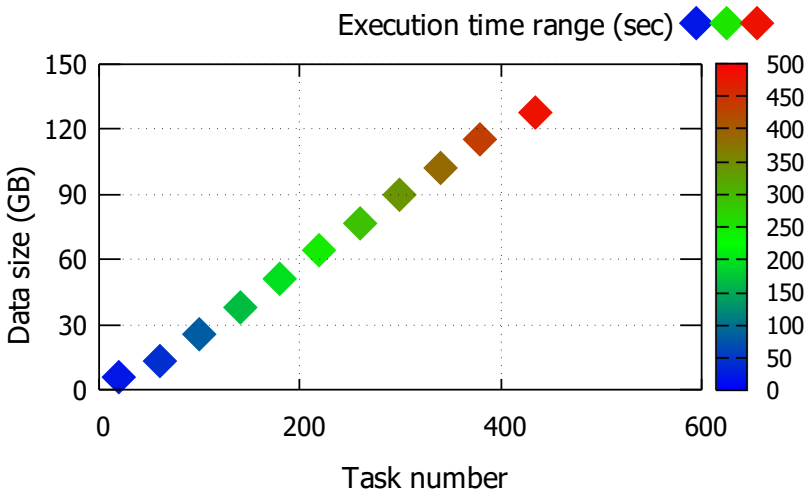


Figure 3.9: TPC-H execution time on different data size

3.5.1 Micro-benchmark

We used Hadoop 3.2.0 and deployed it over 3 AWS virtual machines (VMs). All nodes have 2 CPU cores and 8 GB memory. Moreover, we deployed the *computation and storing* model on a VM with the same instance type outside the Hadoop cluster. The *Agent* and *SmartAgent* are deployed inside the cluster to collect the monitoring information in real-time and the high-level deployment structure is shown in Fig. 3.3.

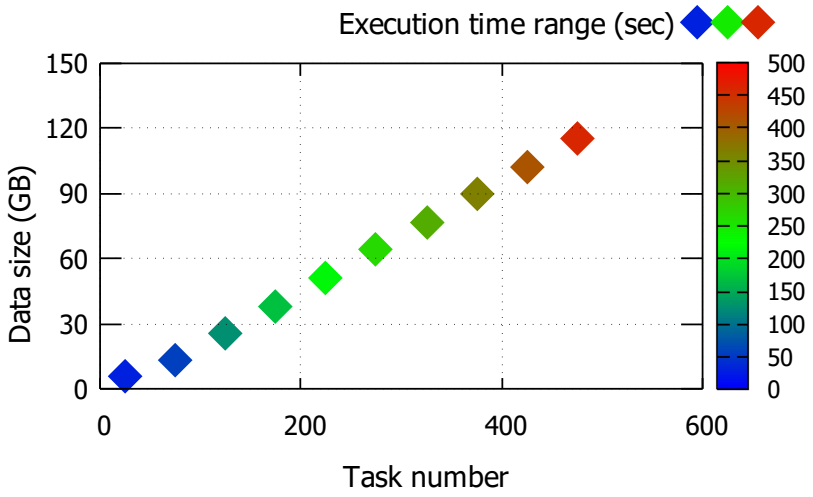


Figure 3.10: TPC-DS execution time on different data size

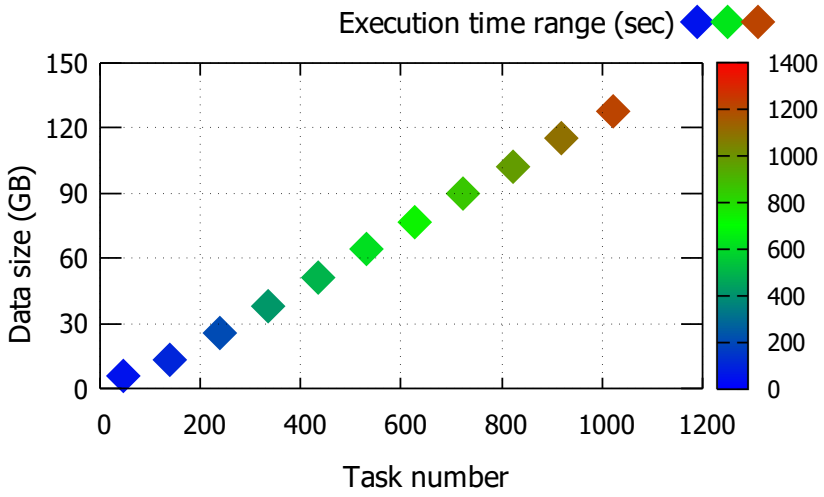


Figure 3.11: K-means execution time on different data size

3.5.2 Building Execution Graph

We develop a real-time stream process module to capture the *Execution Graph* of an application while it is running. This module consists of *SmartCollector* and *SmartWriter*. The *SmartCollector* collects the size of each key-value pair generated from each node and sends the collected information to RabbitMQ Server. Then, the *SmartWriter* analyzes the streaming data and computes data transferring size among the mappers and reducers. The following describes the implementation details of this module and Fig. 3.13 illustrates the logic of the algorithm via a WordCount application.

In the map phase, each mapper is assigned more than one keys and each key has one

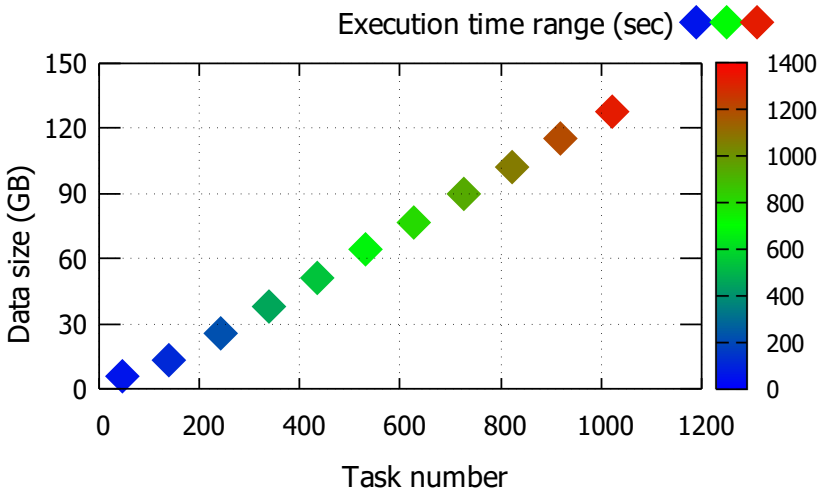


Figure 3.12: PageRank execution time on different data size

value which is equal to 1. Therefore, we use a *4-tuple* to record the information of each *key-value* pair, i.e., *Map_id, key, key-value size, App_id* as shown in Fig. 3.13, **Step 1** and the recorded tuples is forwarded to RabbitMQ Server when it is obtained. In the reduce phase (see **Step 2**), we apply a similar method but using *3-tuple* to record the information of each reducer, i.e., *Reduce_id, key, App_id*.

Finally, in **Step 3** we use *Key* and *App_id* to match the dependencies among the mappers and the reducers from the same application in *SmartWriter*. For example, The second and third tuple in **Step 1** have the same key ("Science"), and the key ("Science") is shuffled to Reduce2 (see the second tuple in **Step 2**). As a result, we are able to compute the size of the data that is shuffled from mappers to reducers according to the table shown in **Step 3**.

The notations used in this section are summarized in Table 3.4.

Algorithm 1 shows the implementation details of the algorithm of SmartWriter. Once map phase starts, each map id and each key of the key-value pairs generated by mapper tasks are stored into related MultiValueMap (see Algorithm 1, Line 7). Similarly, when reduce phase starts, each reduce id and each key of the key-value pairs generated by reducer tasks are stored into related MultiValueMap (see Algorithm 1, Line 10). Once outputs from reducers is received, SmartWriter starts calculating the size of the key that matches in both MultiValueMap collections (see Algorithm 1, Line 15) and then is presented in the result (see Algorithm 1, Line 17).

Algorithm 1: SmartWriter key matcher

Input: M_p - overall map progress in percentage,
R_p - overall reduce progress in percentage,
M_{i_d} - id of map task,
R_{i_d} - id of reduce task,
M_k - key value that map task generates,
R_k - key value that reduce task generates,
$\mathcal{K}$ - size of the key value that map task generates.

Output: $\mathcal{R}$ - output.

```
1  // Create a MultiValueMap M_m to store the M_id
2   M_m ← M_m[0]
3  // Create a MultiValueMap R_m to store the R_id
4   R_m ← R_m[0]
5  while M_p > 0.0 ∨ R_p < 100.0 do
6      // Put M_k and M_id into M_m
7       M_m.put( M_k, M_id)
8      while R_p > 0.0 ∨ R_p < 100.0 do
9          // Put R_k and R_id into R_m
10          R_m.put( R_k, R_id)
11         for each key of M_m do
12             for each key of R_m do
13                 if M_m.key contains R_m.key then
14                     //Get the size of M_k
15                     K ← M_k.getBytes().length
16                     R ← M_k.key is going to R_m.key size= K
17                     print R
18                 end
19             end
20         end
21     end
22 end
```

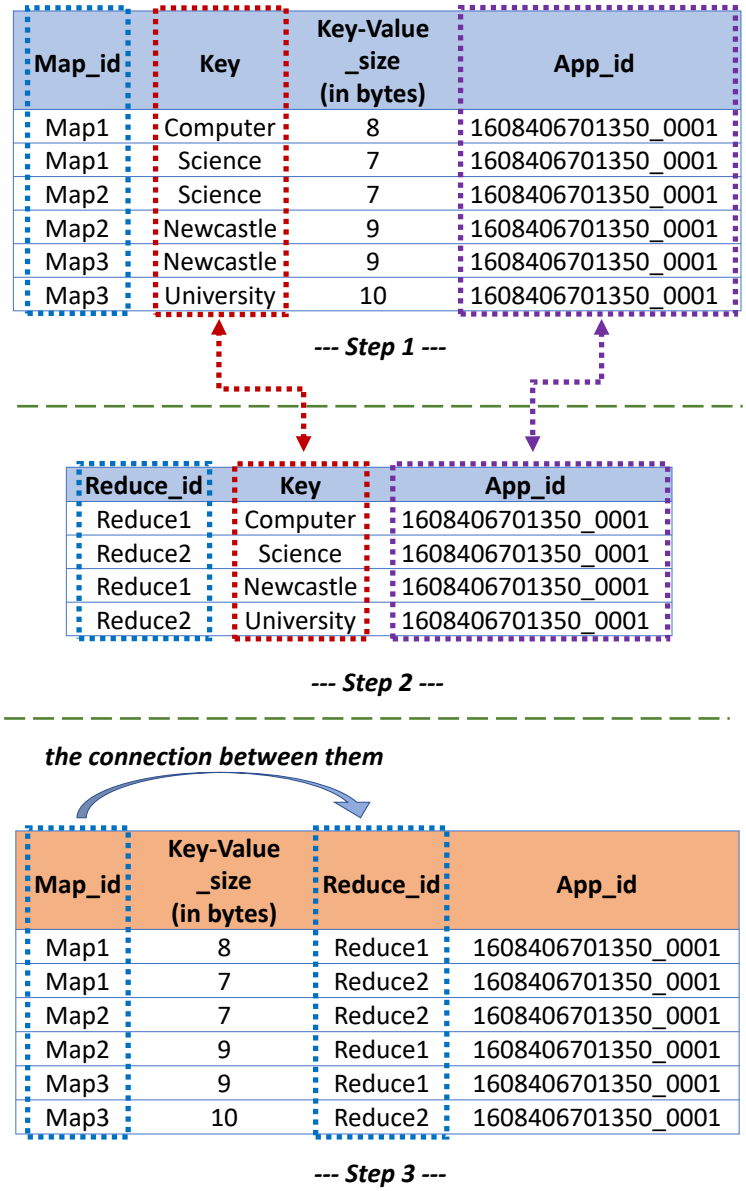


Figure 3.13: The algorithm of SmartWriter.

Implementation Technologies. We build the execution graph using different technologies to fetch the gathered information from the database and visualize them in real-time. These technologies are HTML, CSS, and PHP that help to improve the functionality of efficiency of the graph. HTML stands for HyperText Markup Language that is the standard markup language used to structure a page and its content to be displayed in a web browser [108]. The sections of web pages, namely such as sections, paragraphs, headings, links, could be structured within bullet points, images, tables or paragraphs easily by the user. The basic structure of our execution graph is created using HTML, such as the locations of the circles symbolizing mappers and

Table 3.4: A summary of symbols used in this section

Symbols	Description
M_p	Overall map progress in percentage
R_p	Overall reduce progress in percentage
M_{i_d}	Id of map task
R_{i_d}	Id of reduce task
M_k	Key value that map task generates
R_k	Key value that reduce task generates
$\mathcal{K}$	Size of the key value that map task generates
$\mathcal{R}$	Output of the algorithm

reducers. CSS, Cascading Style Sheets, is an essential technology for the web. It is a style sheet language designed for differentiation of presentation and content by changing the characteristics of the content, including layout, colors, and fonts [109]. For example, in our execution graph, CSS is primarily used to colour the fields to highlight the data belonging to each individual mapper and reducer as well as the arrows that indicate the communication between the mappers and reducers. PHP[19] is a recursive acronym for "PHP: Hypertext Preprocessor". PHP is an open-source, general-purpose and server-side scripting language widely used in web development. It is easily embedded in HTML that performs very important functions, such as creating, opening, reading, writing, and closing for files and adding, deleting, and modifying the elements within a database [110]. PHP has the critical role of fetching all the metrics stored in the InfluxDB database in a given time interval for visualization them in our execution graph.

3.5.3 Real-time demonstration

After the experimental environment is set up based on the micro-benchmark, we ran various configurations of *WordCount* application in term of input data size, number of mappers and number of reducers to evaluate our system.

Fig. 3.14 is a screenshot that shows the real-time execution status and resource uti-

[19]https://www.php.net/

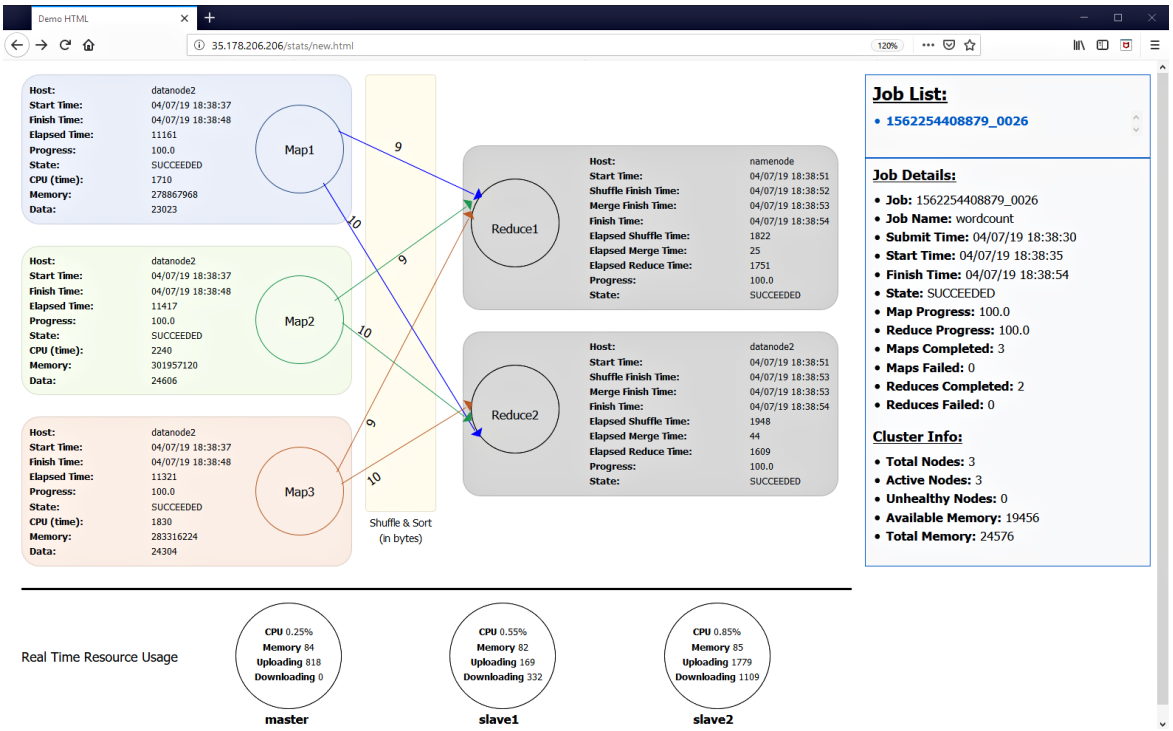


Figure 3.14: Execution graph in a real-time monitoring system.

lization of running a configuration of the *WordCount* application. In order to show the full picture of our design, the screenshot was taken when all map jobs and reduce jobs are completed. More details, please see our screen-cast video on [106].

In the map phase, Figure 3.14 indicates that all mappers are scheduled to *slave1* and their execution status and the resource usage of the entire cluster (see cycles in the button) are monitored by the *Agents* and *SmartAgent*, and displayed in the user interface in real-time.

In the shuffle phase, the dependencies between mappers and reducers are obtained by analyzing the collected information through our real-time stream process algorithm discussed on §3.5.2. Notably, the algorithm also computes the input data size of each reducer (see the numbers above the dependencies) in real-time in the shuffle phase. With this information, we can diagnose the non-salient reasons that cause the performance reduction in a Hadoop cluster. For example, if most of the reducers are not running on the nodes contained their input data, the data shuffling will reduce the performance significantly.

The reduce phase is very similar to the map phase, we collect the execution status

of reducers and resource usage of their hosts (see Fig. 3.14). The right-hand side summarizes the collected monitoring information of the entire Hadoop cluster.

3.6 Discussion and Future Work

SmartMonit proposes an efficient data collection system and a visualization graph for big data systems. It is able to collect all the systems metrics as well as task-related information. However, it is not able to monitor the reasons for performance degradation. To this end, in the next chapter, we propose a performance diagnosis system built on top of SmartMonit that enables root cause analysis.

3.7 Conclusion

In this chapter, we propose and validate SmartMonit, a novel tool that efficiently monitors the big data system. The proposed system collects the run-time system information including computing resource utilization, such as CPU, memory, disk, and network operation as well as job-related information, such as start/finish time, makespan, number of tasks, task/data locations. Importantly, it is able to process the collected information to build a dynamic *Execution Graph* for each application while visualizing the graph in real-time. This allows users to track whole the system and the execution status of the jobs. Moreover, it provides a robust system to build a debugging system for developers. SmartMonit uses agent-based architecture which ensures high availability and scalability. Besides, it enables plug-in new APIs to develop the system or monitoring different big data systems for developers who wants to use their own APIs. The proposed system was evaluated by using different benchmarks along with 11 different datasets varying in size from small to large that deployed on a large-scale cluster in AWS. Moreover, a micro-benchmark was used to evaluate the efficiency and performance of the execution graph on a Hadoop cluster consisting of 3 AWS VMs. The experimental results show that our proposed system, SmartMonit, can be used to efficiently monitor big data systems consisting of thousands of tasks and nodes with very little CPU and memory usage.

4

AutoDiagn: An Automated Real-time Diagnosis Framework for Big Data Systems

Contents

Summary

This chapter presents AutoDiagn, a generic and flexible framework that provides holistic monitoring of a big data system, while detecting the symptom of performance reduction and enabling root-cause analysis. An implementation of the proposed framework interacts with a Hadoop cluster and is evaluated with real-world benchmark applications. All experiments are conducted on AWS. Experimental results show that our implementation has a small resource footprint, high throughput and low latency.

4.1 Introduction

The rapid surge of data generated through sectors like social media, financial services and industries has led to the emergence of big data systems. Big data systems enable the processing of massive amounts of data in relatively short time frames. For instance, the Netflix big data pipeline processes approximately 500 billion events and 1.3 petabytes (PB) of data per day, further, during peak hours, it processes approximately 11 million events and 24 gigabytes (GB) of data on a per-second basis. Facebook has one of the largest data warehouses in the world, capable of executing more than 30,000 queries over 300 PB data every day. However, the enormousness and complexity of the big data system runs in heterogeneous computing resources, multiple tenant environments, as well as has many concurrent execution of big data processing tasks, which makes it a challenge to utilize the big data systems efficiently and reliably[44]. To overcome this, it is imperative to continuously monitor and analyze all available system resources at all times in a systematic, holistic and automated manner. These resources include CPU, memory, network, I/O and the big data processing software components.

Most of the commercial [11][12][13] and academic big data monitoring systems mainly focus on visualizing task progress, and the system's resource utilization [14]. However, they do not focus on the interaction between multiple factors and performing root-cause analysis for performance degradation [102][99]. Moreover, works such as [111, 112] aim to find the best parameters to optimize the performance of big data processing systems, they do not focus on the root-cause analysis that may indicate the viable

reasons behind performance degradation and may provide intuitions for parameter tweaking.

Mantri [15] presents a systematic method that categorizes the main reasons causing outliers in a big data system. The authors' work was focused on the MapReduce programming framework in the Hadoop system; they do not discuss how Mantri can be applied to other big processing frameworks (e.g., Apache Spark[1], and Apache Flink[2]). Garraghan *et al.* [113] proposed an online solution to detect long-tail issues in a distributed system. However, these solutions were built for specific scenarios with much scope left for analyzing a variety of problems that can exist in a large scale big data processing system.

To the best of our knowledge, there is a lack of a generic and comprehensive solution for the detection of a wide range of anomalies and performance of root-cause analysis in big data systems. Developing a general and extensible framework for diagnosing a big data system is not trivial. It requires well-defined requirements which could enable the broader adoption of root-cause analysis for the big data systems, flexible APIs to interact with an underlying monitoring system and integration of multiple solutions for detecting performance reduction problems while enabling the automatic root-cause analysis. In this chapter, we tackle this research gap, and design and develop AutoDiagn to automatically detect performance degradation and inefficient resource utilization problems, while providing an online detection and semi-online root-cause analysis for a big data system. Further, it is designed as a microservice architecture that offers the flexibility to plug a new *detection and root-cause analysis module* for various types of big data systems.

The contributions of this chapter are as follows:

- *An online and generic framework:* We develop a general framework called AutoDiagn which can be adapted for the detection of a wide range of performance degradation problems while pinpointing their root-causes in big data systems.
- *A case study:* We develop a novel real-time stream processing method to detect

[1]https://spark.apache.org/
[2]https://flink.apache.org/

symptoms regarding outliers in a big data system. After that, we develop a set of query APIs to analyze the reasons that cause the outlier regarding a task.

- *A comprehensive evaluation:* We evaluate the feasibility, scalability and accuracy of AutoDiagn through a set of real-world benchmarks over a real-world cloud cluster.

The chapter is organized as follows. We discuss the related work in §4.2. The design requirements and idea are outlined in §4.3. In §4.4, we illustrate the high-level system architecture. §4.5 presents a case study that we implemented and the case study is evaluated in §4.6. §4.7 discusses the limitations of this chapter and highlights our further work. Before drawing a conclusion in §4.8.

4.2 Related Work

Much recent work in big data systems focuses on improving workflows [114–116], programming framework [117–119], task scheduling [120–122].

Root-cause analysis. There is a large volume of published studies describing the role of root-cause analysis. The authors of [15, 123, 124] take the next step of understanding the reasons for performance reduction. Mantri [15] characterizes the prevalence of stragglers in Hadoop systems as well as troubleshooting the cause of stragglers. Dean and Barroso [123] analyze the issues causing tail latency in big data systems. Garraghan *et al.* [113, 125] proposed a new method to identify long tail behavior in big data systems and evaluated in google data trace. The authors in [126] use offline log analysis methods to identify the root cause of outliers in a large-scale cluster consisting of thousands of nodes by tracking the resource utilization. Similarly, Zhou *et al.* [127] use a simple but efficient rule based method to identify the root cause of stragglers.

Along with these similar works, there are some researchers using statistical and machine learning methods for root-cause analysis. The authors of [28] introduce a Regression Neural Network (RNN) based algorithm to trouble-shoot the causes of stragglers by processing Spark logs. More algorithms such as the associated tree and fuzzy data envelopment analysis [128] and Reinforcement Learning [129] are applied for finding the reasons of stragglers in Hadoop and Spark.

In [130], a Pearson coefficient of correlation is used for root cause analysis to measure linear correlation between system metrics, workload and latency. However, these works lack a systematic solution for root cause analysis for big data processing systems and the proposed methods are not applicable for real-time systems.

Different to other work, the authors of [131] propose a new algorithm that aims to reduce the proportion of straggler tasks in machine learning systems that use gradient-descent-like algorithms. This work offers an idea to develop new Diagnosers for machine learning systems using our framework.

Anomaly detection and debugging. The authors in [132] propose a rule-based approach to identify anomalous behaviors in Hadoop ecosystems by analyzing the task logs. This work only analyzes the task logs, which fails to capture the performance reduction issues caused by inefficient utilizing the underlying resources. Next, Khoussainova *et al.* [133] build a historical log analysis system to study and track the MapReduce jobs which cause performance reduction based on their relevance, precision and generality principles. However, this cannot be performed for real-time anomaly detection. Du *et al.* [134] train a machine learning model from the normal condition data by using Long Short-Term Memory (LSTM) and this trained model is used for detecting in Hadoop and OpenStack environments. Our AutoDiagn provides infrastructure into which the trained models can be plugged to enrich the applications.

Real-time operational data analytic system. Agelastos *et al.* [135] propose a monitoring system for HPC systems, which can capture the cases of applications competing for shared resources. However, this system does not consider root-cause analysis of the performance reduction. The authors of [14, 16] do not only provide the feature of real-time monitoring, but are also able to identify the performance issues and troubleshoot the cause of the issues. In addition to them, [136] uses a type of artificial neural network called autoencoder for anomaly detection. They first monitor the system in real-time and collect the normal data for training the model used to discern between normal and abnormal conditions in an online fashion. However, these systems are developed for HPC clusters and are not suitable for big data systems.

4.3 Requirements and design idea

In this section, we analyze the key requirements of the real-time big data diagnosis system, extracting the essential features from the literature. Next, we present the key idea of the framework design.

4.3.1 Fundamental prerequisite for diagnosing big data processing systems

In order to design a generic framework for diagnosing big data processing systems, we classified the fundamental requirements of building a diagnosis system on such systems as follows:

- **Infrastructure monitoring:** Collecting the information about the underlying system, such as network conditions, CPU utilization, memory utilization, and disk I/O status.

- **Task execution monitoring:** Collecting the task information, including execution time, progress, location, location of its input data, input data size, output data size, CPU/memory usage, and process state (running, waiting, succeeded, failed, killed).

- **Abnormal behavior or fault detection:** Detecting abnormal behaviors in big data processing systems, such as slowing tasks, failed tasks, very high/low resource usage, and experiencing very high response time for the requests.

- **Root-cause analysis:** Finding the root cause of performance reduction in big data processing systems, such as the reasons why: tasks are slowing down, resource utilization is low, the response time is high, or when the network latency is high.

- **Visualization:** Visualizing the collected metrics and the results of root-cause analysis of any failures causing performance reduction in the cluster with a user-friendly interface in real-time.

4.3.2 Key design idea

Motivated by the above-mentioned requirements and inspired by medical diagnosis, we highlight the design idea of root-cause analysis for big data processing systems as shown Fig. 4.1, which aims to provide holistic monitoring and root cause analysis for big data processing systems. First, a set of *Symptom Detectors* is defined and developed in **Symptom Detection** to detect the abnormalities of the big system by processing collected system information stream in real-time. Once a symptom (abnormality) is detected, the **Diagnosis Management** may launch the corresponding *Diagnosers* to troubleshoot the cause of the symptom. One symptom may correspond to root causes. Finally, the decisions are made based on the root-cause analysis results.

4.3.3 The generalizability of AutoDiagn

Modern big data processing systems consists of two main types: Big data analytics (e.g., Hadoop, Spark) and Stream processing (e.g., Flink, Spark Stream). Based on our design idea, our AutoDiagn is an independent framework that can be deployed alongside existing big data cluster management systems (e.g., Apache YARN), and ideally it is suitable for root-cause analysis of any big data processing system. However, for the scope of this chapter and practical certainty, the implementation of AutoDiagn focuses on debugging root causes of performance degradation (e.g., slow task execution time) in Hadoop due to faults such as data locality, cluster hardware heterogeneity, and network problems (e.g., disconnection). Although we have validated the functionality of AutoDiagn in the context of Hadoop and considering different classes of workload (e.g., WordCount, Grep, TPC-H, TPC-DC, K-means clustering, PageRank), it is generalizable to other big data processing systems executing similar classes of workload.

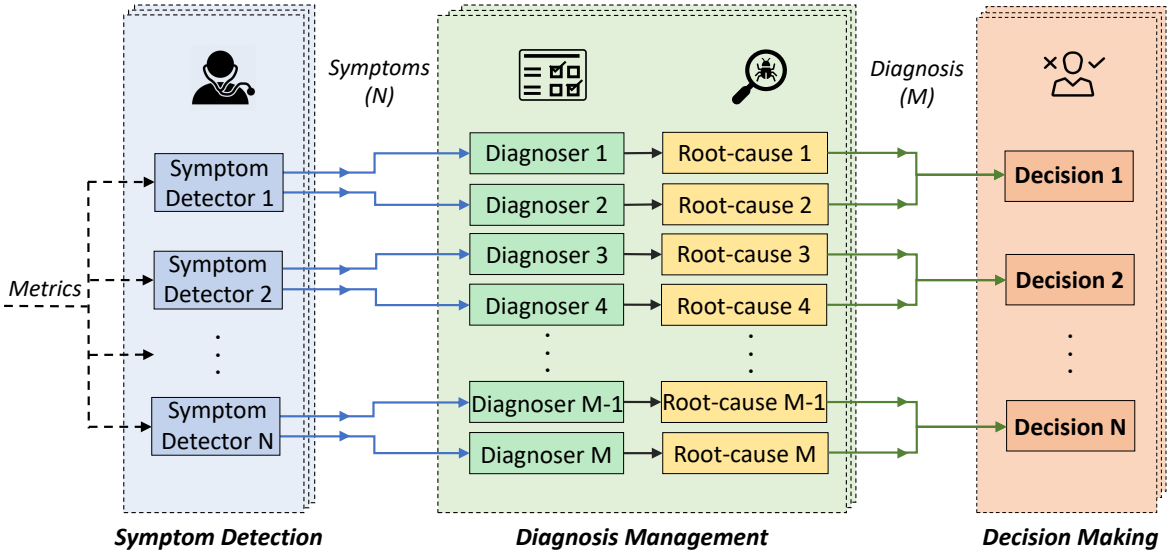


Figure 4.1: The key design idea of root-cause analysis for big data processing systems

4.4 AutoDiagn Architecture

Following the design idea laid out in §4.3, we introduce AutoDiagn, a novel big data diagnosing system. We first illustrate the high-level system architecture and then describe the details of each component. AutoDiagn is implemented in Java and all source code is open-source on GitHub[3].

4.4.1 Architecture overview

AutoDiagn provides a systematic solution that automatically monitors the performance of big data systems while troubleshooting the issues that cause performance reduction. Fig. 4.2 shows its *two* main components: *AutoDiagn Monitoring* and *AutoDiagn Diagnosing*. *AutoDiagn Monitoring* collects the defined metrics (logs) and feeds *AutoDiagn Diagnosing* with them in real-time. Once the abnormal symptoms are detected by analyzing the collected metrics, a deeper analysis is conducted to troubleshoot the cause of abnormal symptoms.

AutoDiagn Monitoring. AutoDiagn Monitoring is a decentralized real-time stream processing system that collects comprehensive system information from the big data system (e.g., Hadoop Cluster). The *Collected Metrics* is a set of pre-defined monitoring

[3]https://github.com/umitdemirbaga/AutoDiagn

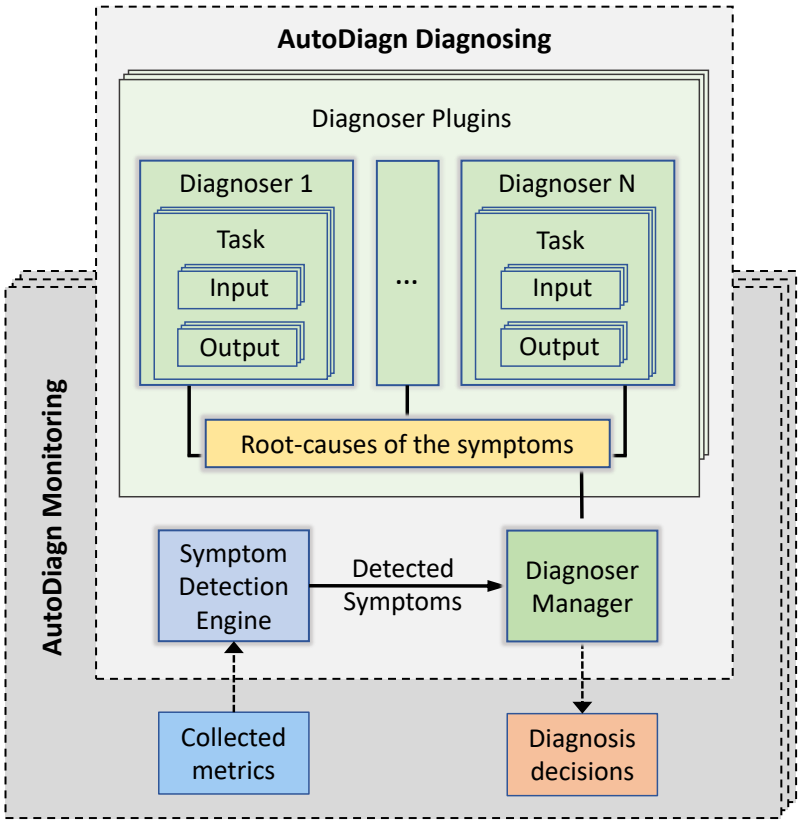


Figure 4.2: The high-level architecture of the AutoDiagn system

entities (e.g., CPU usage, memory usage, task location, task status) used to detect the abnormal symptoms. Moreover, the system information, required for understanding the cause of detected abnormal symptoms, is collected in this modular.

AutoDiagn Diagnosing. AutoDiagn Diagnosing is an event based diagnosing system. First, the carefully crafted metrics are injected into the *Symptom Detection Engine*, a real-time stream processing module, to detect the abnormal symptoms in a big data system. In this chapter, we use the outlier, a common symptom for performance reduction in a Hadoop cluster, as a case study to demonstrate the proposed framework. §4.5.1 illustrates the details of technology that we developed for symptom detection. Moreover, our system follows the principle of modular programming; the new symptom detection method can be easily plugged in. *Diagnoser Plugins* is a component for trouble-shooting the reasons behind the detected symptom. A set of *Diagnosers* is instantiated by the *Diagnoser Manager* when their corresponding symptoms are detected. Then the instantiated *Diagnosers* query a time series database to obtain the required input and their outputs illustrate the cause of the detected symptoms.

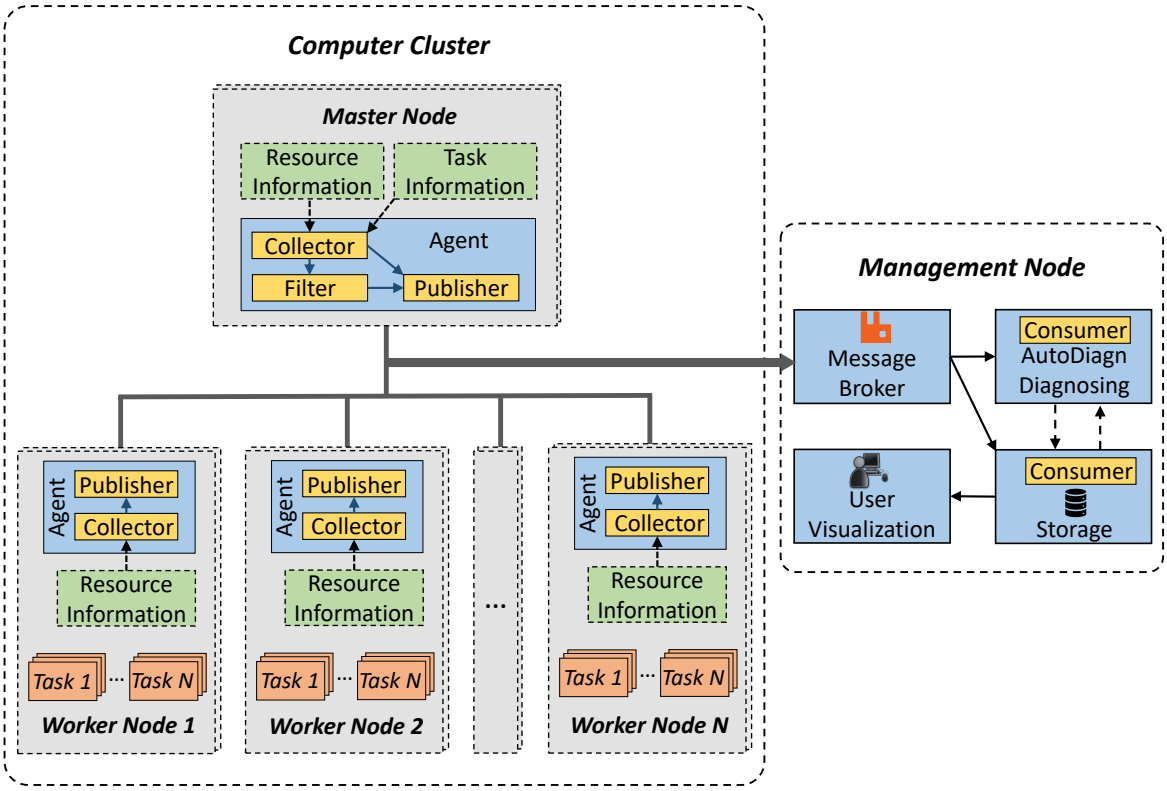


Figure 4.3: The high-level architecture of the monitoring framework

4.4.2 *AutoDiagn monitoring framework*

AutoDiagn monitoring framework is a holistic solution for continuous information collection in a big data cluster. The framework needs to have a fast, flexible and dynamic pipeline to transfer the collected data as well as a high performance, large scale storage system. We now describe an implementation of the framework for a big data computer cluster, and the high-level system architecture is shown in Fig. 4.3.

Information Collection. In each compute node, we develop and deploy an *Agent* to collect real-time system information. For the worker node, the *Agent* collects the usage of computing resource via SIGAR APIs[4], including CPU, memory, network bandwidth, and disk read/write speeds. Moreover, the *Agent* in the master node collects the usage of computing resource as well as the job and tasks information. The *Filter* is developed by using Gson Library[5] to remove the less important information obtained

[4]https://github.com/hyperic/sigar
[5]https://github.com/google/gson

from ResourceManager REST API's[6], thereby reducing the size of data transmission. The collected information is sent to RabbitMQ[7] cluster which is a lightweight and easy-to-deploy messaging system in each time interval via *Publisher*.

Storage. The acquired information is time series data, we therefore choose InfluxDB[8] for data storage. InfluxDB is a high performance, scalable and open source time series database which provides a set of flexible open APIs for real-time analytics. The *Consumer* subscribes the related stream topics from RabbitMQ and interacts with InfluxDB APIs to inject the information to the database.

Interacting with AutoDiagn Diagnosing. The information required for symptom detection is directly forwarded and processed in AutoDiagn diagnosing via a *consumer*. If a symptom is detected, InfluxDB will be queried by AutoDiagn diagnosing for root-cause analysis. Finally, the analysis results are sent back to the database to be stored.

User Visualization. The user visualization allows the users to have a visible way to monitor their big data system. We utilize InfluxDB's client libraries and develop a set of RESTful APIs to allow the users to query various information, including resource utilization, job and task status, as well as root cause of performance reduction.

4.4.3 AutoDiagn diagnosing framework

In this section, we discuss the core components of the AutoDiagn Diagnosing framework (see Fig. 4.2), as well as the interactions with each other and the AutoDiagn Monitoring framework.

Symptom Detection Engine. The symptom detection engine subscribes a set of metrics from the real-time streaming system. §4.5.1 illustrates the technique that we developed for outlier detection. This component follows microservices architecture to which new symptom detection techniques can be directly attached to our AutoDiagn, interacting with other existing techniques to detect new symptoms.

Diagnoser Manager. The diagnoser manager is the core entity responsible for selecting the right diagnosers to find the reasons that cause the detected symptoms.

[6]https://hadoop.apache.org/docs/r3.2.1/hadoop-yarn
[7]https://www.rabbitmq.com/
[8]https://www.influxdata.com/

Additionally, the diagnoser manager is developed as a front-end component, triggered by various detected symptoms (events) via a RESTful API, exposing all diagnosing actions within our framework. The API includes general actions such as starting, stopping or loading a diagnoser dynamically, and specific actions such as retrieving some metrics. Importantly, the diagnoser manager is able to compose a set of diagnosers to complete the diagnosing jobs that may require the cooperation of different diagnosers.

Diagnoser Plugins. The diagnoser plugin contains a set of diagnosers; and a diagnoser is the implementation of the specific logic to perform root-cause analysis of a symptom. Each diagnoser refers to a set of metrics stored in a time series database as the input of its analysis logic. Whenever it is activated by the diagnoser manager, it will perform an analysis, querying the respective metrics, executing the analytic algorithm, and storing the results. §4.5.2 discusses the algorithms to detect the outlier problems, for example, in a Hadoop cluster. The diagnoser plugin is also designed as a microservice architecture which has two advantages: i) a new diagnoser can be conveniently plugged or unplugged on-the-fly without affecting other components; ii) new root-cause analysis tasks can be composed by a set of diagnosers via RESTful APIs.

4.4.4 AutoDiagn diagnosing interfaces for Hadoop

AutoDiagn exposes a set of simple interfaces for system monitoring, symptom detection and root-cause analysis. Table 4.1 shows that two types of APIs are defined: high-level APIs and low-level APIs. The high-level APIs consist of **Symptom Detection**, **Diagnoser** and **Decision Making**. The **Symptom Detection APIs** are a set of real-time stream processing functions used to detect the defined symptoms causing the performance reduction in the Hadoop system. Each **Diagnoser** is a query or a set of queries, which aim to find one of the causes of a symptom. For example, `QueryNonLocal()` tries to find all non-local tasks within a time interval, which is one of the reasons that causes an outlier. Finally, the **Decision Making** APIs are used to analyze the results from each **Diagnoser** and make the conclusion. These high-level APIs have to interact with the low-level APIs (**Information Collection**) to obtain system information including resource usage, and the execution information of the big data system (e.g., ask and job status in a Hadoop system). Based on this flexible design,

users can define and develop their own Symptom Detection, Diagnoser and Decision Making APIs and plug them into AutoDiagn.

4.4.5 Example applications

We now discuss several examples for big data system root cause applications using AutoDiagn API.

Outliers. Outliers are the tasks that take longer to finish than other similar tasks, which may prevent the subsequent tasks from making progress. To detect these tasks, the real-time stream query `QueryOutlier()` is enabled in the *Symptom Detection Engine*. This function consumes each task's completion rate (i.e., progress) and the executed time to identify the outlier tasks (detailed in §4.5.1). Next, three APIs `QueryNonlocal()`, `QueryLessResource()` and `QueryNodeHealth()`, corresponding to three *Diagnosers* that are used to analyze the reasons causing the detected symptom, are executed. `QueryNonlocal()` queries whether the input data is allocated on the node on which an outlier task is processed. In addition, `QueryLessResource()` investigates whether outlier tasks are running on the nodes that have less available resource. Moreover, `QueryNodeHealth()` examines if an outlier task had to be restarted due to the disconnected nodes from the network. Finally, `RootcauseOutlier()` is used to process the results from the three Diagnosers and make the conclusion. All the APIs are shown in Table 4.1 and the technical details are illustrated in §4.5.

Inefficient resource utilization. In our case this means that some tasks are pending (or waiting) to be on worker nodes; at the same time, some worker nodes are idle, e.g., low CPU and memory usage. There are many reasons that cause this issue, but here we consider two key causes: *task heterogeneity* and *resource heterogeneity*. The type of tasks in a big data system are various, including CPU intensive tasks, IO intensive tasks and memory intensive tasks. However, the underlying computing resources are typically equally distributed to these tasks, thereby causing inefficient resource utilization. The latter is caused by the heterogeneous underlying computing resources due to the multiple concurrent processing task environments and the queues are built on the saturated nodes.

Table 4.1: AutoDiagn diagnosing interface. See §4.4.4 for definitions and examples

Symptom Detection (High-level APIs)	**Description**
`QueryOutlier()`	Execute a `Query` that returns the list of `outliers` if any.
`QueryResourceUtil()`	Execute a `Query` that returns the list of the `worker nodes in which the computing resources are not utilized effectively` if any.
Diagnoser (High-level APIs)	**Description**
`QueryNonLocal()`	Execute a `Query` that return the list of `non-local tasks` if any.
`QueryLessResource()`	Execute a `Query` that returns `false` if the cluster is not homogeneous in terms of having resource capacity (CPU/memory).
`QueryNodeHealth()`	Execute a `Query` that returns the list of `disconnected worker nodes` in the cluster if any.
`QueryOversubscribed()`	Execute a `Query` that returns the list of the `oversubscribed tasks` if any.
`QueryDiskIOboundTasks()`	Execute a `Query` that returns the list of the `disk- or IO-bound tasks` if any.
Decision Making (High-level APIs)	**Description**
`RootcauseOutlier()`	Execute a `Query` that illustrate the main reason of the cause of the outlier.
`RootcauseResInef()`	Execute a `Query` that illustrate the main reason of the cause of inefficient resource utilization.
Information Collection (Low-level APIs)	**Description**
`taskExecTime()`	Return the `execution time` since the task started in sec.
`taskProgress()`	Return the `progress` of the running task as a percentage.
`taskInput()`	Return the `input data size` of the running task in mb.
`taskBlock()`	Return the `block id` this task process.
`taskHost()`	Return the `name of the node` this `task ran on`.
`taskCPUusage()`	Return the `CPU usage of the task`.
`taskMemoryUsage()`	Return the `memory usage of the task`.
`taskContainerCPU()`	Return the `allocated CPU` to the container this `task ran on`.
`taskContainerMemory()`	Return the `allocated memory` to the container this `task ran on`.
`blockHost()`	Return the `names of the nodes` that host the `block`.
`pendingTasks()`	Return the `number of the tasks waiting to be run`.
`nodeTotalCoreNum()`	Return the `number of the CPU core number` of the node.
`nodeCPUUsage()`	Return the `CPU utilization` of the node.
`nodeTotalMem()`	Return the `total memory capacity` of the node.
`restartedTasks()`	Return the `name of the restarted tasks` due to nodes that got disconnected from the network.
`nodeMemUsage()`	Return the `memory utilization` of the node.
`nodeDiskReadSpeed()`	Return the `disk read speed` of the node.
`nodeDiskWriteSpeed()`	Return the `disk write speed` of the node.
`nodeUploadSpeed()`	Return the `network upload speed` of the node.
`nodeDownloadSpeed()`	Return the `network download speed` of the node.

To detect the *inefficient resource utilization* in a big data system, the real-time stream query `QueryResourceUtil()` is used within a defined time interval. We compute the mean and standard deviation of the usage resources of the whole cluster. If the standard deviation is far from the mean, we will further query whether the tasks are queued on the nodes which have high resource usage rates. If inefficient resource utilization is detected, two *Diagnosers*, `QueryOversubscribed()` and `QueryDiskIOboundTasks()`, which are the root-cause analysis APIs shown in Table 4.1, are executed to perform root-cause analysis. `QueryOversubscribed()` checks the type of tasks queuing on the saturated nodes. The `QueryDiskIOboundTasks()` checks whether the saturated nodes have less available computing resource, while processing the allocated tasks. The conclusion of the cause of inefficient resource utilization is made in `RootcauseResInef()`.

4.4.6 Parallel Execution

Following the key design idea, the diagnosers are triggered by the corresponding detected symptom. However, we are able to parallelize the execution of each symptom detector and its diagnosers by partitioning the input data. For example, if one symptom detector needs to process too many data streams, we can use two of the same instances of the symptom detector to process the data streams and aggregate the results from two symptom detectors. The diagnoser can follow the same strategy for parallel execution.

4.4.7 Reliability analysis

AutoDiagn follows the centralized design for data collection, which simplifies the implementation of the *Symptom Detection*, *Diagnosis Management* and *Decision Making*. They can easily obtain the required information from one place, instead of interacting with the entire big data system. Moreover, the centralized design does not mean unreliability, due to the high-availability of RabbitMQ. The RabbitMQ cluster can overcome the node fail in the message queuing system while ensuring scalability.

4.5 Case Study

In the previous section, we have discussed that our framework supports detection of multiple types of symptoms (e.g., outliers, inefficient resource utilization). However, detecting these symptoms is non-trivial; and each symptom can be detected by using different algorithms with different input metrics. In this section, we present a case study that details the technology of detecting outliers and the root-causes analysis for the detected outliers. The notations used in this section are summarized in Table 5.2.

4.5.1 Symptom detection for outliers

Ananthanarayanan *et al.* [15] defined the outlier tasks' run-time to be 1.5 times higher than that of the median task execution time; their method is based on the assumption that all tasks are started at the same time and are the same type (i.e., the same input data and the same processing code), which is not suitable for real-time symptom detection, because in a time interval the tasks may be submitted at different times; the input data size of the tasks and the code for tasks are not always the same. In this section, we use *Performance* ($\mathcal{P}$) to measure the outlier as shown in Eq 4.1. $\mathcal{O}$ represents the normalized value of the *task progress* in terms of percent work complete, and $\mathcal{T}$ is the normalized value of the task execution time.

$$\mathcal{P} = \frac{\mathcal{O}}{\mathcal{T}} \tag{4.1}$$

Eq 4.2 is used to normalize the $\mathcal{O}$ and $\mathcal{T}$, where x_{min} and x_{max} are the minimal and maximal values of the given metrics (e.g., task progress and execution time) in a time interval. We set $b = 1$ and $a = 0.1$ to restrict the normalized values within the range from 0.1 to 1 [137].

$$x_{norm} = a + \frac{(x - x_{min})(b - a)}{x_{max} - x_{min}} \tag{4.2}$$

Moreover, we define the outlier tasks which have 1.5 times less *performance* value than the median *performance* value in each time interval. Fig. 4.4 shows a snapshot of a

Table 4.2: A summary of symbols used in this section

Symbols	**Description**
J_p	Job progress
$\mathcal{N}$	Name of the task
N_l	List of $\mathcal{N}$
$\mathcal{P}$	Performance of the $\mathcal{N}$
P_l	List of $\mathcal{P}$
$\mathcal{O}$	Progress of the $\mathcal{N}$
O_l	List of $\mathcal{O}$
$\mathcal{T}$	Execution time of the $\mathcal{N}$
T_l	List of $\mathcal{T}$
m_{ed}	The performance of median task
$\mathcal{D}$	Non-local tasks
D_l	List of Non-local task
$\mathcal{R}$	Task running on the node with less resources
R_l	List of $\mathcal{R}$
$\mathcal{W}$	Restarted tasks due to the nodes' network failure
W_l	List of $\mathcal{W}$
S_l	List of outlier task
Sd	Non-local outlier
Sd_l	List of Sd
Sr	Outlier stemming from the resource variation
Sr_l	List of Sr
Sw	Outlier stemming from disconnected nodes
Sw_l	List of Sw
$\mathcal{F}$	Factor value of 1.5 used to find the $\mathcal{S}$

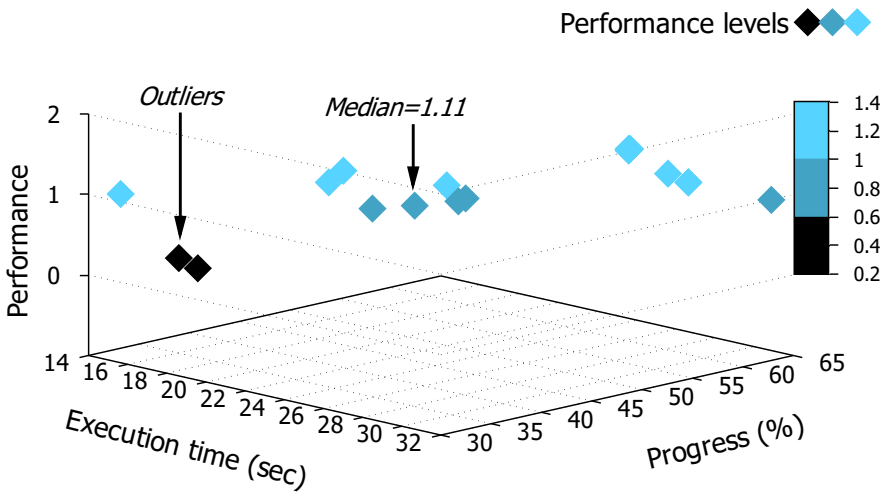


Figure 4.4: Performance evaluation of the tasks

time interval (e.g., three seconds), and two mappers are identified as outliers. More evaluations will be discussed in §4.6.

Algorithm 2 demonstrates the proposed *ASD* (automated symptom detection) algorithm in the AutoDiagn system. It is fed by the streaming data provided by the AutoDiagn Monitoring system during job execution. First, the performance of each running task is calculated (see Algorithm 2, Line 11) using Eq 4.1. Next, the *median* value of the performance of all tasks is taken to be used to detect outliers (see Algorithm 2, Line 16). Then, the tasks whose performance is 1.5 times less than the performance of the *median* task are selected as outliers (see Algorithm 2, Line 20). As a final step, these tasks detected as outliers are sent to the *Diagnosis Generation* component for root-cause analysis (see Algorithm 2, Line 24).

4.5.2 Root cause analysis for outliers

When the detected symptoms are passed to the *Diagnoser Manager*, the corresponding *Diagnosers* are executed for trouble-shooting. The following subsection illustrates the technologies that we have developed for analyzing the causes of outliers in a Hadoop cluster.

Algorithm 2: Automated symptom detection for outliers

Input: J_p - job progress in percentage,
$\mathcal{F}$ - factor,
$\mathcal{N}$ - name of the running task,
N_l - list of $\mathcal{N}$,
$\mathcal{O}$ - progress of the task,
O_l - list of $\mathcal{O}$,
$\mathcal{T}$ - execution time of the task,
T_l - list of $\mathcal{T}$.

Output: S_l - list of outliers $\mathcal{S}$.

```
// Create a list S_l to store the S
S_l ← S_l[0]
// Initialize the m_ed
m_ed ← m_ed[0]
while J_p < 100.0 do
    //Clear the S_l and P_l
    S_l ← Clear (S_l^new, S_l)
    P_l ← Clear (P_l^new, P_l)
    for each N in N_l do
        //Compute P
        P = O / T
        //Insert the P into the P_l
        P_l.add(P)
    end
    //Get the m_ed from the P_l
    m_ed ← Median value of P_l
    for each value of P_l do
        if (P * F) < m_ed then
            //Insert the N into the S_l
            S_l.add(N)
        end
    end
    //Update the S_l in Diagnosis Generation component
    S_l ← Update (S_l^new, S_l)
    //Update the N_l, O_l, T_l, J_p
    N_l ← Replace (N_l^new, N_l)
    O_l ← Replace (O_l^new, O_l)
    T_l ← Replace (T_l^new, T_l)
    J_p ← Replace (J_p^new, J_p)
end
```

4.5.2.1 Root cause of outliers

In this section, we follow the three main reasons that cause outliers, discussed in [15], i.e., Data locality, Resource heterogeneity, and Network failures.

Data locality. Hadoop Distributed File System (HDFS) stores the data in a set of machines. If a task is scheduled to a machine which does not store its input data, moving data over the network may introduce some overheads to cause the outliers issue.

Resource heterogeneity. The machines in a Hadoop cluster may be homogeneous with the same hardware configuration, but the run-time computing resources are very heterogeneous due to the multiple talents environment, multiple concurrent processing task environment, machine failures, machine overloaded etc. If a task is scheduled to a bad machine (e.g., has less computing resource) it may cause an outlier issue. Moreover, resource management systems for a large-scale cluster like YARN split the tasks over the nodes equally without considering the resource capacities of the nodes in the cluster, but only takes into account sharing the node's resources among the tasks running on the node equally by default [138]. That is more likely to raise an outlier problem in the cluster.

Network failure.

In Hadoop clusters, the network disconnection can cause the running tasks allocated on a disconnected node to be restarted on other nodes, which may lead to the task becoming an outlier, and increase the completion time. The following illustrates the three algorithms that we developed to identify the outliers caused by the three reasons.

The following illustrates the three algorithms that we developed to identify the outliers caused by the three reasons.

4.5.2.2 Detecting data locality issues

We assume that a *non-local* task ($\mathcal{D}$) (e.g., mapper) is executed on a node where its input data is not stored (In the following, we use ***Sd*** to represent *non-local outliers*). To detect these tasks, we develop Algorithm 3 to check whether a set of outliers is

Algorithm 3: Root-cause analysis of outliers

Input: S_l - list of outliers in time interval from t to $t+1$

Output: Sd_l - list of non-local outliers Sd,
Sr_l - list of outliers stemming from resource variation Sr,
Sw_l - list of outliers stemming from disconnected nodes Sw.

```
1  // Find all 𝒟 within the given time interval
2  D_l ← QueryNonLocal(t, t+1)
3  //Find the common elements in the D_l and S_l, and add them into the Sd_l
4  Sd_l ← RetainAll (D_l, S_l)
5  // Find all ℛ within the given time interval
6  R_l ← QueryLessResource(t, t+1)
7  //Find the common elements in the R_l and S_l, and add them into the Sl_l
8  Sr_l ← RetainAll (R_l, S_l)
9  // Find all 𝒲 within the given time interval
10 W_l ← QueryNodeHealth(t, t+1)
11 //Find the common elements in the W_l and S_l, and add them into the Sw_l
12 Sw_l ← RetainAll (W_l, S_l)
```

caused by a data locality issue. The input of our algorithm is a list of detected outliers during the time interval from t to $t+1$ and one of its outputs is a list of outliers which also belongs to the *non-local* tasks. First, we query our time series database to obtain all *non-local* tasks within the given time interval (see Algorithm 3, Line 2).

Here, `QueryNonLocal()`, a root-cause analysis API, is used to find the non-local ones among the running tasks in that period of time. It compares the location where the task is running (host node of the task) with the nodes where the data block is replicated for fault tolerance via information collection APIs shown in Table 4.1, `taskHost()` and `blockHost()`. If the task is not running on any of these nodes (nodes hosting a copy of the block), this task is marked as a non-local task. In the second step (Algorithm 3, Line 4), we obtain the common elements of list D_l and S_l. These elements symbolize the non-local outliers stemming from a data locality issue.

4.5.2.3 Detecting resource heterogeneity issues

Algorithm 3 is designed to identify the outliers caused by the resource heterogeneity. The tasks running on the nodes which have less computing resource ($\mathcal{R}$) tend to be outliers [33] (in the following, we use ***Sr*** to represent *outliers running on the nodes which have less computing resource*). In Algorithm 3, the list of detected outliers during the time interval from t to $t+1$ is used as input and one of the outputs of the algorithm is a list of outliers which also belongs to the tasks running on the node with less computing resource. The time series database is queried to obtain all *the tasks running on the node with less computing resource* within the given time interval (see Algorithm 3, Line 6).

Here, `QueryLessResource()`, a root-cause analysis API, is used to check the heterogeneity of the nodes that host only the running tasks based on the resource specifications of them in that period of time. It detects the nodes with less resource capacity in terms of CPU core numbers and the total amount of memory among the nodes hosting the running tasks. The resource specifications of the nodes (i.e., CPU core numbers, total amount of memory) are obtained from each node via information collection APIs shown in Table 4.1, `nodeTotalCoreNum()` and `nodeTotalMem()` APIs. As a second step (Algorithm 3, Line 8), we obtain the common elements of list R_l and S_l. These elements symbolize the outliers stemming from a cluster heterogeneity issue.

4.5.2.4 Detecting network failure issues

Since S_l is obtained from Algorithm 2, a Diagnoser is executed via `QueryNodeHealth()` to find all restarted tasks due to the nodes disconnected by network failure within the given time interval (see Algorithm 3, Line 10). The low-level API `restartedTasks()` is called which distinguishes the restarted tasks due to network failure from the speculation of straggler tasks by analyzing the information of the tasks that is provided by the monitoring agent. Thereafter, we compute the list Sw_l that contains the outlier tasks caused by the network failure (see Algorithm 3, Line 12).

4.5.2.5 Decision making

In this case study, we use a simple decision make method that compares the lists Sd_l, Sr_l and Sw_l and the probability of the reasons causing the outliers by using the number of the elements of a list divided the total number of outlier tasks. For instance, the probability of the performance reduction caused by data locality is $\frac{|Sd_l|}{|S_l|}$. More advanced methods such as deep learning models can be used for processing more complicated decision making tasks in future work.

4.6 Evaluation

In this section, we present a comprehensive evaluation showing the capacity and the accuracy rate of AutoDiagn, as well as a analysis of its resource consumption and overheads.

4.6.1 Experimental setup

Environments. We set up the Hadoop YARN clusters over 31 AWS nodes with 1 master and 30 slaves with the Operating system of each node being Ubuntu Server 18.04 LTS (HVM). The Hadoop version is 3.2.1 and the Hive version is 3.1.1. To meet our experimental requirements, we built two types of cluster. In **Type I** each node has the same configuration (i.e., 4 cores and 16 GB memory). In **Type II**, 25 nodes have 4 cores and 16 GB memory and 6 nodes have 2 cores and 4 GB memory.

Benchmarks and workload. We used four well-known Hadoop benchmarks in our evaluations namely WordCount[9], Grep[10], TPC-H[11], TPC-DS[12], K-means clustering[13], and PageRank[14]. The input of each benchmark application is 30GB.

Methodology. Our experiments aim to evaluate the effectiveness of AutoDiagn. To this end, we manually inject the above-mentioned three main reasons to cause the outliers,

[9]http://wiki.apache.org/hadoop/WordCount
[10]http://wiki.apache.org/hadoop/Grep
[11]http://www.tpc.org/tpch/
[12]http://www.tpc.org/tpcds/
[13]https://en.wikipedia.org/wiki/K-means_clustering
[14]https://en.wikipedia.org/wiki/PageRank

which can be summarized as four types of execution environment. **Env** $\mathcal{A}$: we perform all benchmark experiments in the cluster **Type I**. **Env** $\mathcal{B}$: we perform all benchmark experiments in the cluster **Type I**, but skew the input size stored on different nodes. **Env** $\mathcal{C}$: we perform all benchmark experiments in the cluster **Type II** (a heterogeneous cluster). **Env** $\mathcal{H}$: we perform all benchmark experiments in the cluster **Type I**, and disconnect some nodes' network during execution. Each benchmarking is repeated 5 times and results are reported as the average and standard deviation. In total, there are 90 experiments conducted in our evaluation.

4.6.2 Diagnosis detection evaluation

In this section, we evaluate the accuracy of our symptom detection method. To this end, we execute our benchmarks in **Env** $\mathcal{B}$ to increase number of ***Sd*** tasks (see §4.5.2.2). Next, to increase the issue of resource heterogeneity (***Sr*** referring to §4.5.2.3), we run the benchmarks in **Env** $\mathcal{C}$. Thereafter, we run the benchmarks in **Env** $\mathcal{H}$ to emulate the network failure (***Sw*** referring to §4.5.2.4). Finally, we compare the detected Outlier tasks with the ground truths that are the data locality, resource heterogeneity, and network failure issues observed by the AutoDiagn diagnosing system.

Table 4.3, Table 4.4, and Table 4.5 summarize all the results. All benchmarks achieve high accuracy by using our proposal symptom detection method. The highest accuracy for both ***Sd*** and ***Sr*** are 92.3%, and for ***Sw*** is 94.7% and the overall accuracy for outlier detection is 91.3%, where the *Error* represents the variation of the accuracy depending on the repeated experiments.

We compute the accuracy of our symptom detection method by using the number of detected outlier tasks divided by the *actual* number of the tasks that can cause the outlier issue. Table 4.3, for example, $\mathcal{D}$ is the total number of non-local tasks and Outliers (Sd) is the number of detected outlier tasks that belong to non-local task. Therefore, the accuracy is $\frac{Sd}{\mathcal{D}}$. Table 4.4 and Table 4.5 follow the same approach to compute the accuracy.

Outlier verification. To further verify the ***Sd***, ***Sr***, and ***Sw*** are the main reasons causing the outliers, we conduct the following comparison experiments: 1) comparing

Table 4.3: The accuracy of symptom detection for non-local outliers in a homogeneous cluster

Benchmark	**Total tasks**	$\mathcal{D}$	**Outliers (detected as *Sd*)**	**Accuracy (%)**	**Error (σ)**
WordCount	234	32	29	90.63	3.9
Grep	236	37	33	89.19	4.8
TPC-H	102	13	12	92.31	6.72
TPC-DS	126	13	12	92.31	6.1
K-means	234	34	29	85.29	1.25
PageRank	235	28	25	89.29	6.2

Table 4.4: The accuracy of symptom detection for the outliers stemming from resource variation in a heterogeneous cluster

Benchmark	**Total tasks**	$\mathcal{R}$	**Outliers (detected as *Sr*)**	**Accuracy (%)**	**Error (σ)**
WordCount	234	37	33	89.19	2.77
Grep	236	26	24	92.31	4.77
TPC-H	102	9	8	88.89	5.47
TPC-DS	126	13	12	92.31	6.9
K-means	234	36	33	91.67	2.88
PageRank	235	30	28	93.33	5.35

the execution time of local tasks and non-local tasks; 2) comparing the execution time of the tasks running in **Env** $\mathcal{A}$ and **Env** $\mathcal{C}$; and 3) comparing the execution time of normal tasks and restarted tasks due to network failure. Fig. 4.5(a) proves that non-local tasks consume more time than local tasks due to the overload introduced by data shuffling. Additionally, we compare the throughput of the local tasks and non-local tasks in terms of how much data can be processed in each second. Fig. 4.6 reveals that the throughput of non-local tasks is only 70% that of local tasks.

Moreover, Fig. 4.5(b) shows that the execution time of the tasks running on **Env** $\mathcal{A}$ is less than that on **Env** $\mathcal{C}$. This is because the tasks are equally distributed to all computing nodes and the less powerful nodes are saturated. Furthermore, Fig. 4.8(a) shows that the CPU usage of less powerful hosts reaches 100%, thereby building a task queue in these hosts, increasing the overall execution time. However, Fig. 4.8(b)

Table 4.5: The accuracy of symptom detection for the outliers stemming from network failures

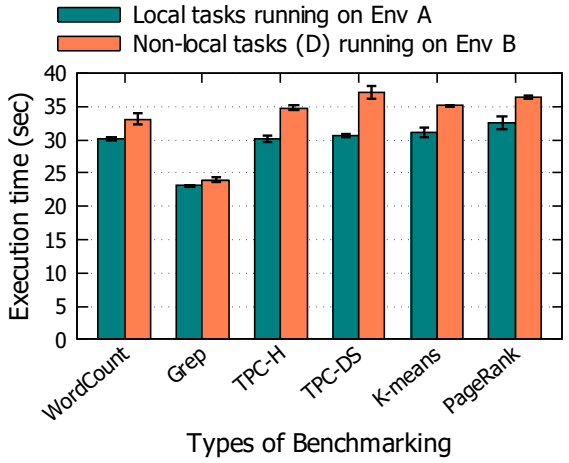
Benchmark	**Total tasks**	$\mathcal{W}$	**Outliers (detected as *Sw*)**	**Accuracy (%)**	**Error (σ)**
WordCount	234	11	10	90.91	1.83
Grep	236	13	12	92.31	6.73
TPC-H	102	13	12	92.31	6.54
TPC-DS	126	15	14	93.33	5.43
K-means	234	17	16	94.12	4.33
PageRank	235	19	18	94.74	4.23

reveals that the powerful hosts have sufficient computing resources for processing the allocated tasks.

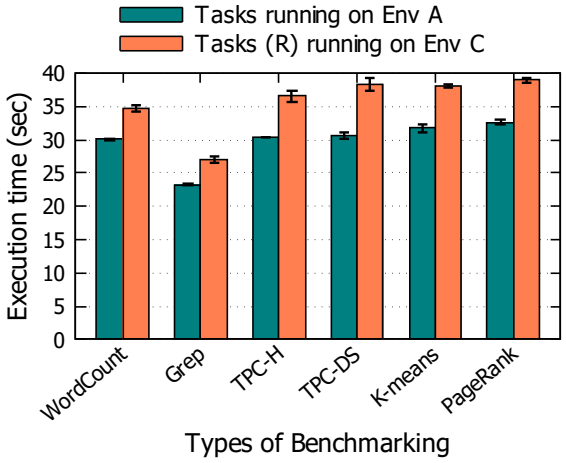
Furthermore, Fig. 4.5(c) shows that the execution time of the restarted tasks are longer than the normal tasks. As Fig. 4.7 illustrates, we compute the execution time of the restarted task by adding the execution time of the task in the disconnected node and that in the rescheduled node.

4.6.3 Performance and overheads

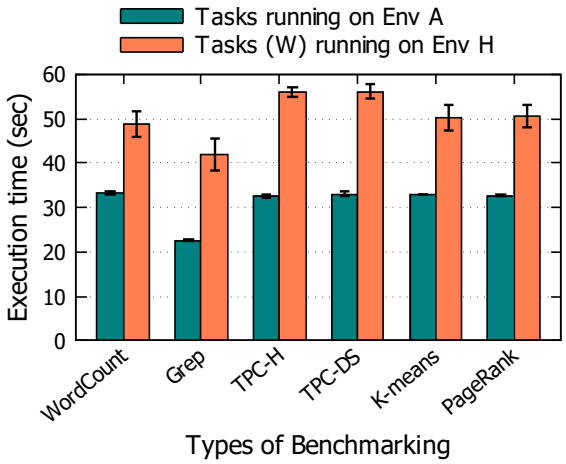
Performance evaluation. We evaluate the performance of AutoDiagn by measuring the end-to-end response time of symptom detection and root-cause analysis. Since they are not affected by the types of benchmark, we report the average of the response time. Fig. 4.9(a) shows that the real-time symptom detection can achieve a low response time, which only has 96 milliseconds and 1059 milliseconds with 100 tasks and 1000 tasks, respectively. Although the response time increases linearly, the parallel execution method discussed in §4.4.6 can be applied to reduce the latency. The response time for root cause analysis is higher than that of symptom detection. For 100 tasks and 1000 tasks, their response times are 0.354 seconds and 5.974 seconds, respectively. Unlike the symptom detection which is very sensitive to latency because of the follow-up processes, triggering the further root-cause analysis or alerting the system managers, Root-cause analysis aims to provide a holistic diagnosing of a big system and the analysis results may help to improve the system performance in future.



((a)) Local tasks vs Non-local tasks



((b)) Homogeneous cluster vs Heterogeneous cluster



((c)) Normal tasks vs Restarted tasks caused by network failure

Figure 4.5: Comparison of execution time of the tasks

As a result, the real-time root-cause analysis is not compulsory.

System overheads. To evaluate the system overhead introduced by AutoDiagn, we measure the CPU and memory usage of AutoDiagn Monitoring (agent) and AutoDiagn Diagnosing. Table 4.6 shows that -*AutoDiagn Monitoring* only consumes approximately 2.52% memory and 4.69% CPU; while -*AutoDiagn Diagnosis* uses 2.08% memory and 3.49% CPU.

Fig 4.9(b) shows the network overhead of AutoDiagn. The extra communication cost introduced by our tool is small but it increases when the number of parallel tasks increases. For example, when the number of parallel task is 100, there are about 45 messages per second sent from agents to RabbitMQ cluster and the total size of these

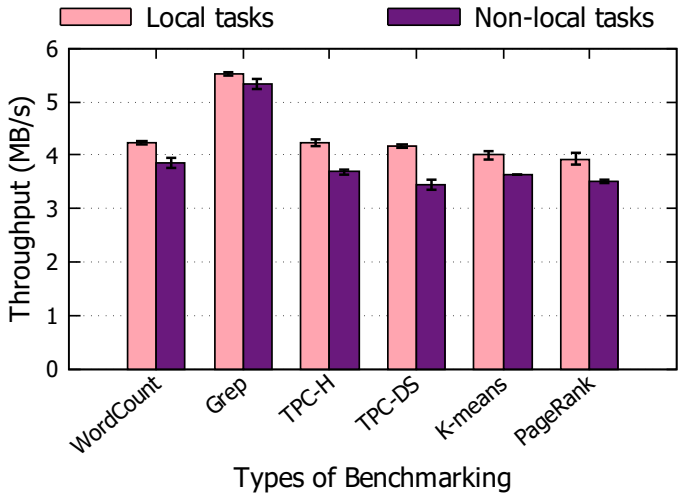


Figure 4.6: The throughput of AutoDiagn

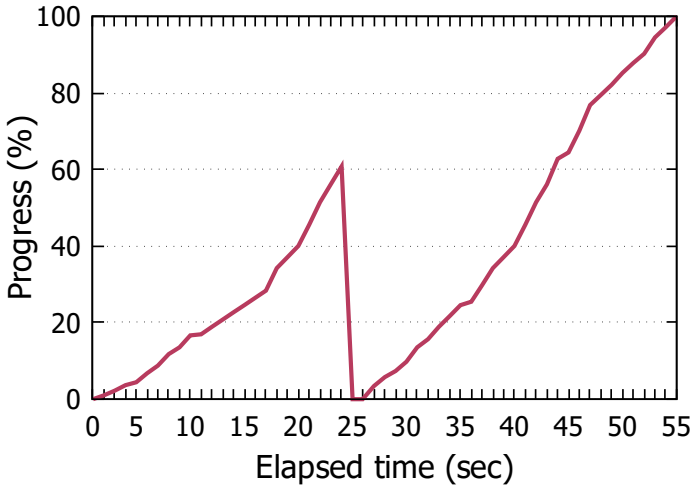


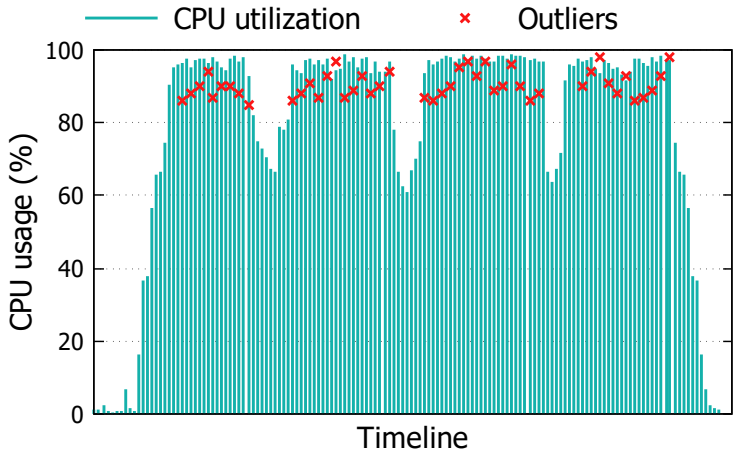
Figure 4.7: The life cycle of the restarted task

messages is 13.5 KB/s. The message rate and network overhead increase to 615 per second and 223 KB/s, respectively, when the number of parallel tasks is 1000.

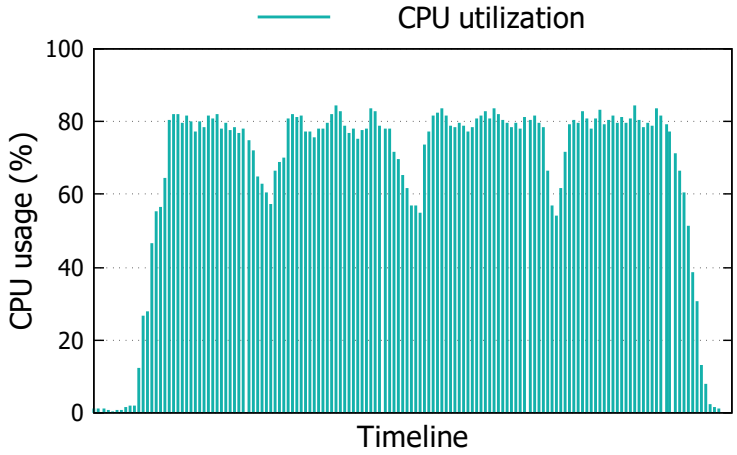
Table 4.6: Resource overhead caused by AutoDiagn components

Components	Mem (%)	CPU (%)
AutoDiagn Monitoring	2.52	4.69
AutoDiagn Diagnosing	2.08	3.49

Storage overheads. AutoDiagn needs to dump the system information to a database which may consume extra storage resource. **In our evaluation experiments, it only cost 3.75 MB disk space in total**. Obviously, increasing the types of symptom detection and root cause analysis will also consume more storage resources.



((a)) CPU utilization of less powerful hosts and outliers

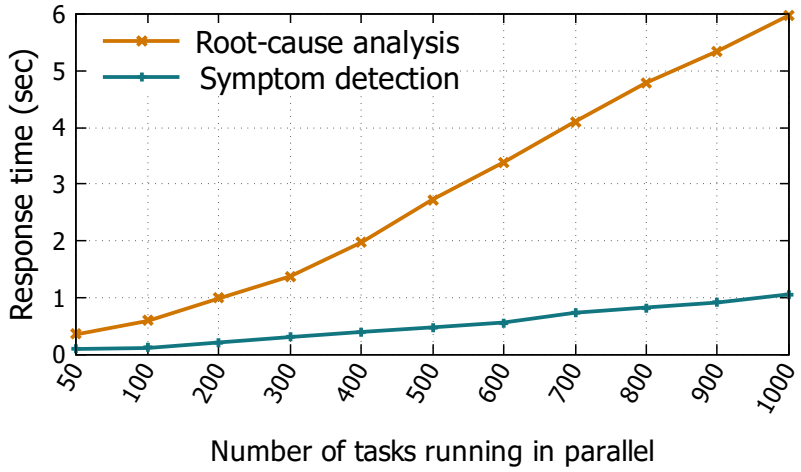


((b)) CPU utilization of high power hosts

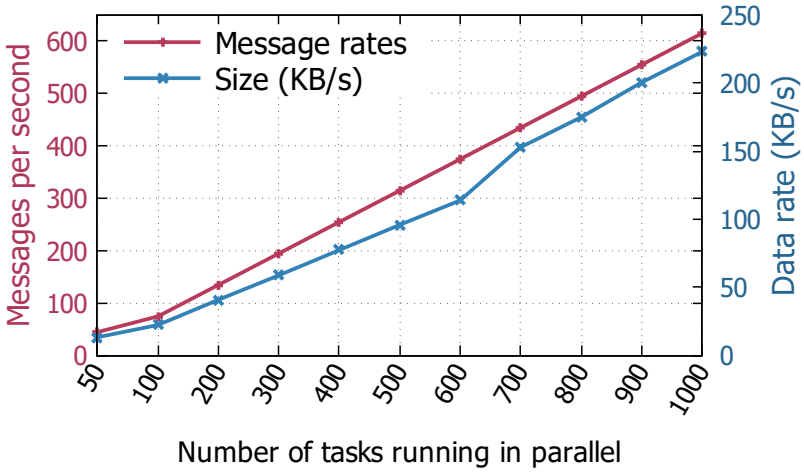
Figure 4.8: CPU utilization of two nodes running simultaneously. Outliers are most likely to occur in the nodes which have less computing resource.

4.7 Discussion and Future Work

In this chapter, we propose a general and flexible framework to uncover the performance reduction issues in a big data system. In particular, we develop and evaluate big data applications for outliers. AutoDiagn is able to detect the problems arising from only network failures. The effects of bandwidth delays between nodes on job completion time cannot be calculated as it requires an SDN-based monitoring system. For this purpose, we propose a novel root-cause analysis techniques for cloud-based big data systems using a well-validated simulator BigDataSDNSim [139].



((a)) The end-to-end response time of AutoDiagn diagnosis system



((b)) The message rates and network overhead

Figure 4.9: Performance evaluation and network overhead of AutoDiagn

4.8 Conclusion

In this chapter, we have presented AutoDiagn, a framework for enabling diagnosing of large-scale distributed systems to ascertain the root cause of outliers, with the core purpose of unravelling the concretization of complicated models for system management. After making a comprehensive literature review and identifying the requirements for real-world problems, we conceived its design. The combination of user-defined functions powered by APIs and the agent-based monitoring system along with the findings obtained from an empirical analysis of the experiments we conducted play a fundamental role in the development of the system. AutoDiagn can be applied to most big data systems along with the monitoring systems. We have also presented the imple-

mentation and integration of the AutoDiagn system to the SmartMonit [36], real-time big data monitoring system, combined in our production environment. In our implementation on a large cluster, we find AutoDiagn very effective and efficient.

Outliers are one of the main problems in big data systems that overwhelm the whole system and reduce performance considerably. AutoDiagn embraces this problem to reveal the bottlenecks alongside their root causes.

5

BigPerf: Probabilistic Performance Diagnosis and Prediction for Cloud-based Big Data Systems

Contents

Summary

This chapter presents BigPerf, a novel Bayesian system for diagnosing and predicting root causes of performance degradation and bottlenecks of Hadoop Applications. BigPerf analyzes and characterizes the performance of Hadoop applications by incorporating Bayesian networks to determine uncertain and complex relationships while dealing with noisy and missing data. Our extensive simulation studies validate BigPerf and show that it can efficiently diagnose the causes regarding the performance of Hadoop Applications. We also show that BigPerf predicts systems performance with high accuracy of approximately 94.04%.

5.1 Introduction

Big data systems such as Hadoop[1] are typically deployed through modern virtualized cloud computing platforms that process large-scale data in a parallel manner. Such systems, composed of a network of hundreds of nodes, provide high reliability and high fault tolerance under highly concurrent and stochastic application workloads using big data programming model such as MapReduce [125].

MapReduce is a distributed programming model for processing large-scale datasets on a cluster of commodity hardware [140]. It uses a master/slave architecture and distributes multiple tasks across the slave nodes to execute them in parallel. MapReduce consists of two main steps, namely map and reduce. In the map step, the mappers read the data, usually stored in Hadoop Distributed File System (HDFS), and processes it based on a user-provided code; it then generates a set of intermediate key-value pairs and temporarily transfers them to HDFS [141]. Once all the mappers finish processing, the reduce phase starts to execute the reducers which take the above mentioned key-value pairs as an input and produces the final output. The communication and data traffic among these interacting software elements (mappers, reducers and HDFS), input data volume, and infrastructure factors (hardware resource types and their processing/storage power and dynamic network conditions) causes a complex challenges,

[1] https://hadoop.apache.org/

such as task failures and/or application performance degradation [15], [142], [143]. Consequently, the task-structure complexity in such systems makes the problem of diagnosing and predicting root causes of performance degradation very challenging and difficult.

To understand the problem in little more depth, let us consider an example to illustrate what root-cause analysis (RCA) [144] means in the context of Hadoop applications. For example, total job completion time in MapReduce-based Hadoop application depends on several stochastic factors, such as network delay, processing power of the underlying hardware resources (configuration of VM/container) (see Fig. 5.1). Therefore, the core challenge is how to accurately model the relationship between these stochastic factors and their complex interdependencies to clearly understand the issues related to performance degradation (e.g., slow mappers/reducers and/or slow data transmission between HDFS and mapper/reducer tasks and/or overall slow application response time). While several past efforts have focus in developing optimal scheduling algorithms for improving the overall makespan (execution times) of Hadoop applications, limited approaches [145] [146] have considered the issue of understanding and analyzing sub-optimal Hadoop application performance due to complex inter-play between stochastic behaviours of underlying factors (e.g., hardware and network heterogeneity, run-time network conditions).

Hence, in this chapter we want to investigate following research questions (RQ):

- (**RQ1**) How do we model complex dependencies between several factors for undertaking RCA for diagnosing and predicting reasons for performance degradation?
- (**RQ2**) How do we validate the RCA technique over a realistic, large scale test setup that can produce usable and non-biased performance degradation data sets related to Hadoop Applications?

To the best of our knowledge, no prior work exists to answers the questions mentioned above regarding building a comprehensive RCA technique for diagnosis and prediction of performance issues by taking into consideration several, inter-dependent stochastic factors. Motivated from this fact, and to solve the research questions mentioned above, the contributions of this chapter are as follows:

- To counter RQ1, we propose and develop a novel RCA technique/system called BigPerf for big data systems[2], incorporating Bayesian networks to model uncertain and complex relationships between relevant factors, such as execution time of each specific task (mapper and reducer), network transmission time between these tasks, data block split time (HDFS to mapper, reducer to HDFS).
- To counter RQ2, we validate our RCA technique (BigPerf) using extensive and well-validated simulator BigDataSDNSim [139]. Our experiments show that the BigPerf can be used to perform a fine-grained RCA for the issues causing performance degradation. We found that the accuracy of the proposed BigPerf system to be approximately 94.04%. BigDataSDNSim has been rigorously validated and evaluated against the performance of real-world Hadoop cluster test bed. The validation against real test-bed assured us that BigDataSDNSim[3] is capable of accurately modeling and simulating Hadoop application execution environments.

The chapter is organized as follows. §5.2 discusses the related work. §5.3 presents BigPerf. §5.4 discusses the experiment and results analysis. Finally, §5.5 presents the conclusion.

5.2 Related Work

Since the Apache Hadoop ecosystem is the most comprehensive big data framework and MapReduce is one of the most popular programming models, many authors have worked on modelling their performance for efficient and optimal task scheduling. For instance, Wang *et al.* [147] implement the locally weighted linear regression (LWLR) and linear regression (LR) algorithms to create three types of prediction models based on different characteristics to predict the execution time of large-scale data-driven applications deployed on the Hadoop see whether the job can meet a deadline. Khan *et al.* [145] propose a performance model to define the amount of resources needed for Hadoop application completion within a certain deadline. The authors in [148] design deep Bayesian networks called TraceAnomaly that captures the sequential behaviour

[2]This chapter considers the Apache Hadoop framework, an open source implementation of MapReduce programming model.

[3]https://github.com/kalwasel/BigDataSDNSim

of an application by leveraging the call paths information. However, it only focuses on outlier detection for microservice environments and is not able to evaluate the relationship between complex factors probabilistically. Lin *et al.* [146] analyze the relationships among the Map and Reduce tasks to estimate the complexity of the task execution. Furthermore, they estimate the task completion time based on the monetary costs with the help of a performance model. The authors in [149] propose an analytical model for predicting the response time of MapReduce applications that focus on predicting delays in synchronization between map and reduce tasks. However, their approach ignore the impact of configuration of computing resources (e.g., CPU speed, memory size) and bandwidth capabilities on task completion time. Nonetheless, these approaches can not take root cause analysis of performance degradation (e.g., slow mapper execution due to HDFS network issues).

Wang *et al.* [150] use a simulator called MRPerf to assess the impact of software and hardware failures as well as inter-connect topologies and data locality issues on the performance of MapReduce-based Hadoop Applications. However, they do not analyze and model the relationship between the factors considered and their mutual impact. Kambatla *et al.* [151] use the historical data of MapReduce jobs to predict the performance while minimizing monetary deployment cost in public cloud computing systems. They do so by only analyzing the Hadoop Application's resource consumption while ignoring inter-play between the hardware performance of computing resources, hosting the Map and Reduce tasks, and network status. Hence, this approach is also not suitable for undertaking RCA.

Kavulya *et al.* [152] use an instance-based learning technique to characterize patterns for resource utilization and Map/Reduce tasks. They do so by analyzing the historical traces of Hadoop Application execution. Ganapathi [153] predicts the performance of Hadoop applications using machine learning techniques by correlating the pre-execution properties of the workload with the performance metrics measured after execution, such as response time, resource usage. The authors in [154] propose a machine learning-based approach to predict the performance of MapReduce tasks. However, this work has some limitations on job performance estimation due to the lack of formal mathematical models. Morton *et al.* [155] use the critical path method

Table 5.1: The features supported by existing work and BigPerf

Related work	Features				
	Task performance analysis	**System performance analysis**	**Network performance analysis**	**Hidden/implicit relations analysis**	**Root-cause analysis**
[145]	✓	×	×	×	×
[146]	✓	✓	✓	×	×
[149]	✓	×	×	×	×
[150]	✓	✓	✓	×	×
[151]	×	✓	×	×	×
[152]	✓	✓	✓	×	×
[153]	✓	✓	×	×	×
[155]	✓	×	×	×	×
[37]	✓	✓	✓	×	✓
[154]	✓	✓	✓	×	×
BigPerf	✓	✓	✓	✓	✓

to predict the remaining time of a Map or Reduce task based on the execution time of mappers. However, these approaches are not tailored towards undertaking root cause analysis.

In our recent work, we [37] propose an automated fault diagnosis framework for Hadoop application that can highlight the factors leading to slow execution time. However, our approach is not capable of modeling the relationship between those stochastic factors.

Table 5.1 illustrates the differences and similarities of existing works as compared with BigPerf.

5.3 BigPerf: Bayesian Performance Diagnosis and Prediction for Cloud-based Big Data Systems

This section presents BigPerf – a Bayesian system for big data QoS diagnosis and prediction. Bayesian networks (BNs) are probabilistic graphical models used to model uncertain (often hidden) complex inter-dependencies between random variables for a

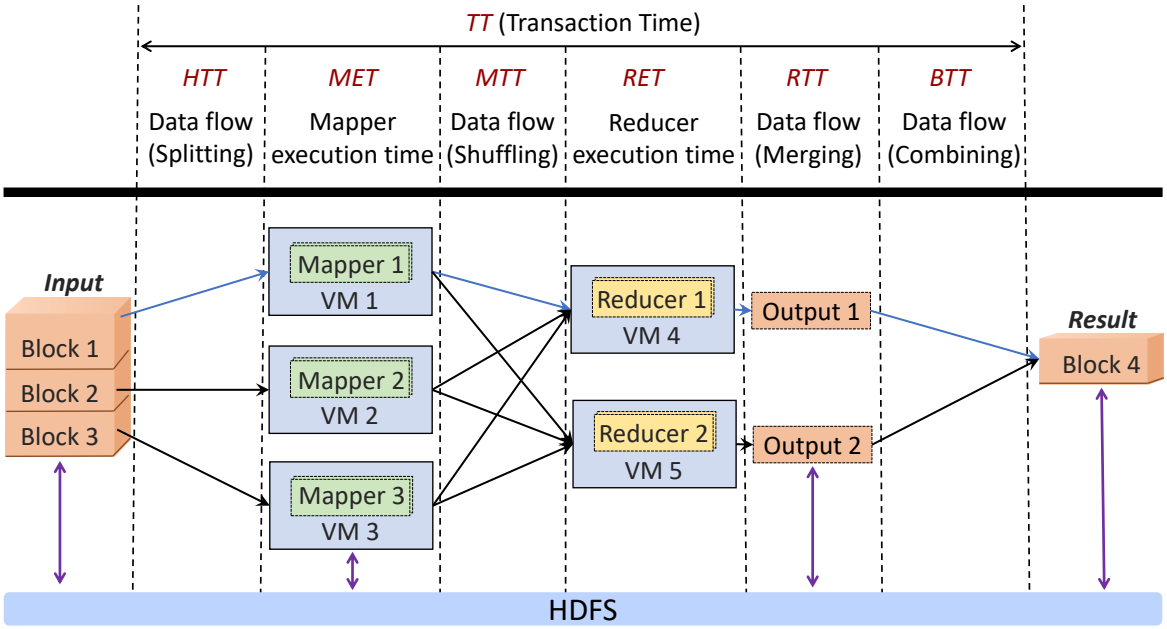


Figure 5.1: End-to-end Transaction time of a task

system under consideration. BNs use a directed acyclic graph (DAG) to describe a set of variables and their conditional relationships. BNs are ideal for taking an observed event and estimating the likelihood that any of multiple known causes had a role. BNs have been successfully used in a wide range of application domains ranging from computer networks to medicine [156].

Fig. 5.1 depicts the life cycle of a MapReduce-based Hadoop Application workflow. Although a MapReduce program basically executes in two stages, namely mapper stage and reducer stage consisting of shuffle, sort and reduce, there are six different stages from submitting the tasks to getting the final result. First, input data stored in HDFS is split into a set of mapper tasks and processed by parsing the key/value pair. The generated intermediate results are stored in HDFS. After that, the sorted data is passed through a user-defined reduce function. Each reducer generates its own results. Finally, all the results are combined and written to HDFS [157]. This whole process consists of the time to distribute the input data to the mapper tasks, the time for the mapper tasks to process the data to generate the intermediate data, the time to distribute the intermediate data to the reducer tasks, the time for the reducer tasks to generate the result, and the time to combine these produced results into a single output.

5.3.1 BNs for Big Data QoS Diagnosis and Prediction

We describe the steps to develop BNs for big data diagnostics and prediction systems in this chapter. Fig. 5.2 illustrates our overall approach. First, the stakeholders gather big data systems performance results via benchmarking studies or via simulators such as *BigDataSDNSim* simulator [139]. After that, the benchmarked results are pre-processed and stored in databases. Third, a BN is learned using the structural learning algorithms, pre-processed data[4], or the domain expert manually creates it using his/her knowledge and experience. Fourth, the modelled BN is used for probabilistic diagnosis by inserting evidence into the BN for determining the probability of a random variable (or factor) taking a particular value. Lastly, this BN can be used for both diagnosis and prediction by the stakeholders if diagnostic outcomes are deemed sufficient; otherwise, the first three steps are repeated till the best BN is found.

We now discuss these steps in detail.

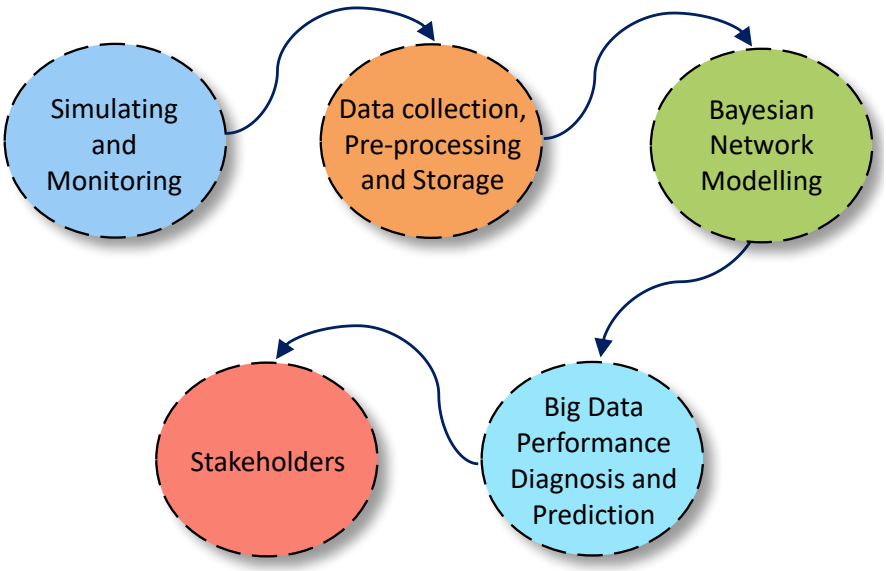


Figure 5.2: Approach for Big Data QoS diagnosis and prediction.

We consider BNs for big data performance analysis and prediction. We chose BNs as a tool based on their numerous advantages over reasoning, neural networks and decision trees, as indicated below [158]:

[4]https://github.com/umitdemirbaga/BigPerf

- BNs can learn with scarce and sparse data.
- BNs can deal with several data types, for example, numerical, non-numerical, binary, categorical, and ordinal, to name a few.
- BNs can incorporate domain or expert knowledge compared to neural networks, decision trees, and linear and non-linear regression.
- Using conditional reasoning and hidden variables, BNs can uncover hidden inter-dependencies that are impossible in other methods.
- BNs can be learned efficiently from data using structural learning algorithms. If a BN structure is already developed, they can be learned through a well-known expectation-maximization (EM) algorithm.
- BNs can be used efficiently in both real and non-real-time systems for prediction.
- BNs can easily be extended to Dynamic Bayesian networks (DBN) to respond over time.
- BNs can be used with utility theory to make decisions under uncertainty.

We now illustrate how BNs can be used in modelling many variables to diagnose and forecast a big data system output effectively. A BN can be defined as follows:

Definition: *A Bayesian network (BN) is a directed acyclic graph (DAG) where random variables form the nodes of a network. The directed links between nodes form causal relationships. The direction of a link from X to Y means that X is the parent of Y. Any entry in the network can be calculated using the joint probability distribution (JPD) denoted as:*

$$P(x_1, ..., x_m) = \prod_{i=1}^{m} P(x_i | Parents(X_i)) \tag{5.1}$$

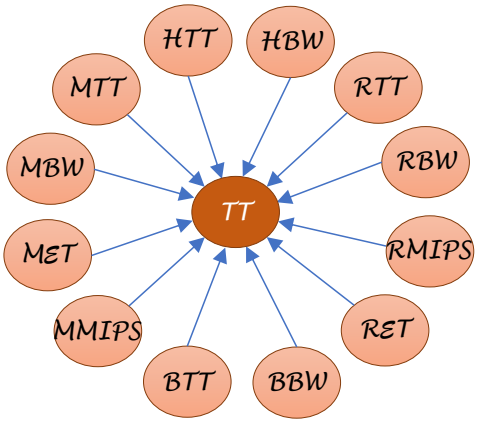
where parent nodes X_i, is the parent of the node x_i.

Table 5.2: A summary of symbols used in this section

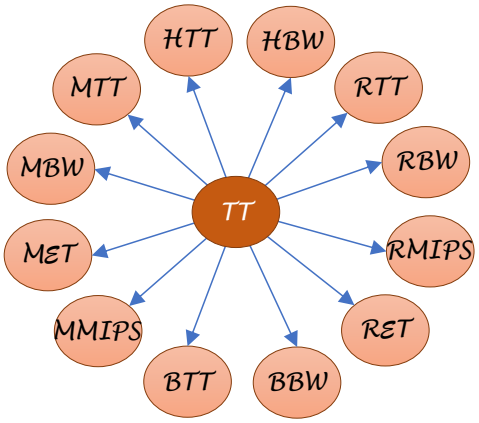
Symbols	**Description**
MIPS	Million instructions per second
HBW	Bandwidth between HDFS and the VM which hosts mapper in Mbps
HTT	Data transmission time between HDFS and the VM which hosts mapper in milliseconds
MMIPS	*MIPS* of the VM which hosts mapper
MET	Mapper execution time in milliseconds
MBW	Bandwidth between the VM which hosts mapper and the VM which hosts reducer in Mbps
MTT	Data transmission time between the VM which hosts mapper and the VM which hosts reducer in milliseconds
RMIPS	*MIPS* of the VM which hosts reducer
RET	Reducer execution time in milliseconds
RBW	Bandwidth between the VM which hosts reducer and the VM which hosts output in Mbps
RTT	Data transmission time between the VM which hosts reducer and the VM which hosts output in milliseconds
BBW	Bandwidth between the VM which hosts output and the VM which hosts block in Mbps
BTT	Data transmission time between the VM which hosts output and the VM which hosts block in milliseconds
TT	Transaction time in milliseconds ($HTT + MET + MTT + RET + RTT + BTT$)

BNs include a clear and thorough definition of the problem domain and a detailed explanation of the causal ties between multiple nodes (random variables) [158]. Example BNs for diagnosis and prediction of the performance of big data systems are shown in Fig. 5.3. The notations given in the chapter are summarized in Table 5.2.

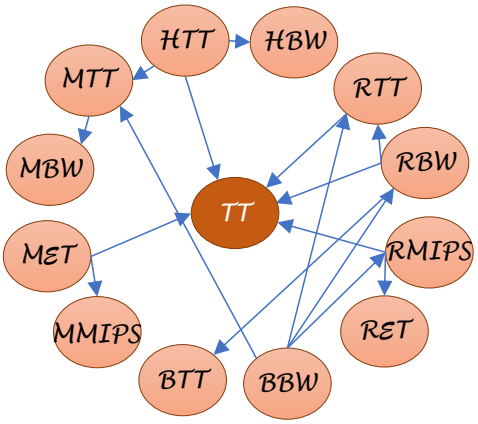
In these BNs, the oval nodes represent the random variables, which are modelled together in order to probabilistically determine their impact on each other. In a BN, an arc's path from one node(s) to another node(s) is a parent-child relationship, where the parent node is probabilistically explicitly influenced by the child node. In Fig. 5.3(c), for instance, the arcs from the nodes *RBW* and *BBW* to *RTT* indicate that these nodes are parents to the child node *RTT*; and are used to determine the effect of the *RBW* and the *BBW* on the *RTT* and vice versa.



((a)) Noisy-OR Network



((b)) Naïve Bayes Network (NBN)



((c)) Complex Bayesian Network

Figure 5.3: Bayesian Networks for Big Data QoS diagnosis and prediction

In several ways a BN can be created (see Fig. 5.3(a) to Fig. 5.3(c)) such as the Noisy-Or Network, the Naïve Bayes Network, or a more complicated model (such as in Fig. 5.3(c)) Complex Bayesian Network where most nodes are interconnected based on conditional relationships. Manual BN development can be challenging as it can be difficult for the stakeholders/domain experts to identify causal dependencies between the random variables. Consider nodes A and B to test the relationship between two random variables. Suppose that the domain expert wants to determine the effect of node A on node B, i.e., if the domain expert fixes the state (s ∈ S where S is a set of states) of a node A, and if that does not change the belief of node B then, A is not a cause of B [159].

A random variable (node) in a BN can be both continuous and discrete. Conditional probability tables (CPTs) are defined for discrete nodes. Continuous probability dis-

tributions (CPDs) are defined for continuous nodes. The CPTs are learned from data, or the domain expert sets them manually based on their experience. When data is available such as in our case, the stakeholders use structural learning algorithms such as structural expectation maximization [159]. In this chapter, we show that even simpler BNs can be efficiently applied to the modelling, diagnosis and prediction of the performance of big data systems.

After a BN is created by learning algorithms or domain experts, it must be validated. In order to verify accuracy and precision of a BN [159], cross validation is routinely carried out. A portion of the testing data is to prepare the BN throughout the cross-validation. The remaining data or test data are used for the predictive accuracy of the model. The most commonly used expectation-maximization (EM) algorithm [159] is considered for BN model parameter testing. After the BNs prediction accuracy has been met, the stakeholders or domain experts can use these BNs in real world applications.

5.4 Experiment and Results Analysis

In this section, we present the experimental results related to BigPerf. We validate BigPerf using GeNIe Bayesian Network development environment[5]. We obtained MapReduce datasets that include map and reduce processing information and the data regarding networks among MapReduce elements (e.g., HDFS, mappers, and reducers) using the BigDataSDNSim simulator[139].

5.4.1 Experiments

As mentioned previously, we used BigDataSDNSim [139] to simulate MapReduce applications that run in cloud datacenters. Through the simulator, we gathered fine-grained performance-related data for the Hadoop system. BigDataSDNSim is rigorously validated and evaluated against real big data MapReduce applications that run in cloud environments. To this end, a WordCount application was deployed on a Hadoop cluster. The experiment was executed six times and the average processing and network

[5]https://www.bayesfusion.com/

Table 5.3: Configuration for validating BigDataSDNSim

Environment	MIPS	Configuration for each VM			
		Number of cores	Total number of MIPS	Memory size	Network bandwidth
Real experiment	3592	4	14,368	4 GB	850 Mbps
BigDataSDNSim with α	3563	4	14,252	4 GB	1000 Mbps
BigDataSDNSim without α	3592	4	14,368	4 GB	1000 Mbps

Table 5.4: Configuration for validating MapReduce application

Environment	Total executed MIPS per mapper	Total executed MIPS per reducer	Number of mappers	Number of reducers	File size (HDFS to mappers)
Real experiment	296,939	100,576	2	1	272.7 MB
Simulated experiments	296,939	100,576	2	1	272.7 MB

transmission times were recorded. For the simulation, two different experiments were carried out: *BigDataSDNSim with* α *overhead* and *BigDataSDNSim without* α *overhead*. All the results obtained from the experiments were compared with each other. Table 5.3 shows the configuration validation of the real and simulated experiments while Table 5.4 shows the configuration parameters used in the validation for MapReduce application in detail. As seen in Table 5.3, the bandwidth of the real experiment is 850 Mbps while the bandwidth of the simulated experiments is 1000 Mbps. This minor variance is acceptable as there are several constraints that prevent actual VMs from reaching maximum network capacity, such as CPU speed, hard drive (I/O) speed, and the size of RAM allotted to the MapReduce application.

The validation results assure that BigDataSDNSim efficiently mimics real environments, where the accuracy and correctness is validated against the real Hadoop system combining MapReduce and different network types (including software-defined networks) with an equivalent simulated environment. Additionally, BigDataSDNSim obtains different large-scale datasets in a configurable fashion while avoiding the limitations posed by real environments, such as the difficulties to obtain large-scale MapReduce data under varying network and host conditions.

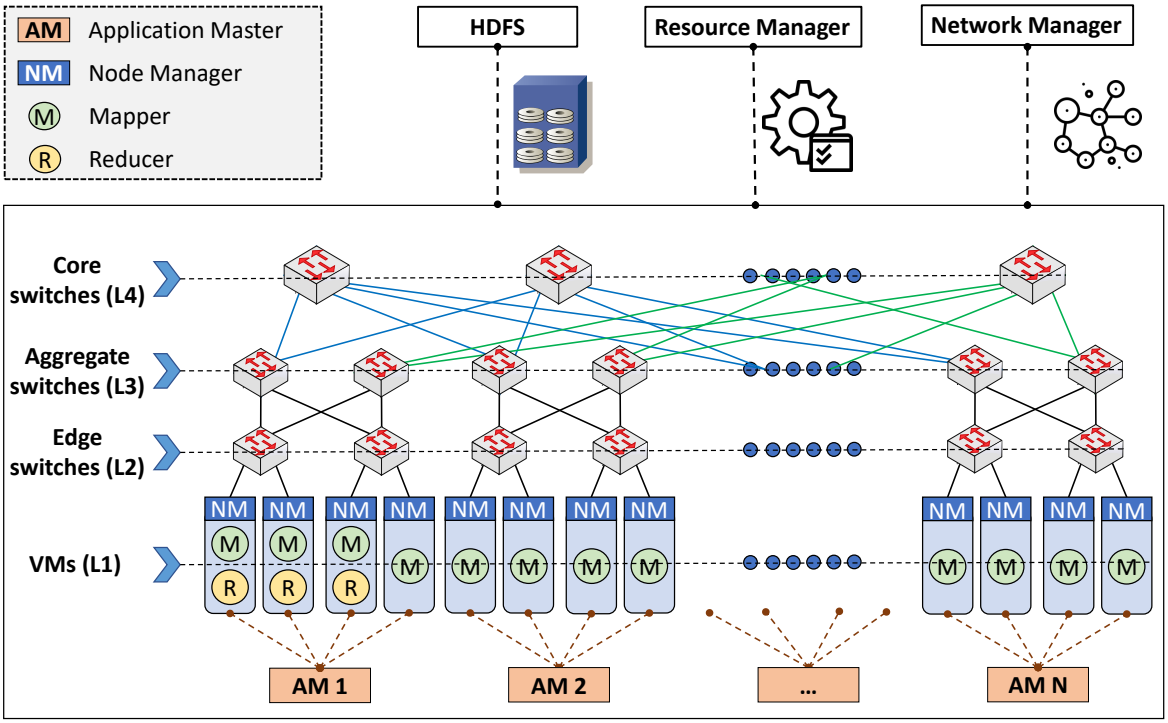


Figure 5.4: Fat-tree topology used in the simulated use-case experiments.

Using BigDataSDNSim for running our experiments offered following clear benefits over real testbed, that can be costly in terms of time and money:

- We were able to capture the performance data (e.g., HDFS network usage, HDFS to mapper virtual machine data transfer delay, HDFS to reducer virtual machine data transfer delay) at much finer granularity as compared to a real test-bed.
- We were able to model highly heterogeneous hardware and network configurations.
- We were able to model highly complex MapReduce-based Hadoop application workflows.
- The simulator offered us a simple, controlled and repeatable experimental environment. The source code of the simulator can be found here[10].

Simulation setup We simulated a large scale big data system, where our simulation setup comprised 128 hosts, 256 virtual machines (VMs), and 20 network switches.

[10]https://github.com/kalwasel/BigDataSDNSim

Table 5.5: Cloud Datacenter Configuration

HOST		VM		NETWORK	
Specs	**Quantity**	**Specs**	**Quantity**	**Link**	**Bandwidth**
CPUs	8	CPUs	4	HDFS <-> Edge switch	4 Gbps
RAM	30 GB	RAM	8 GB	Edge <-> Aggregate switches	1 Gbps
MIPS	10K	MIPS	1250	Aggregate <-> Core switches	1 Gbps
Total Hosts	128	Total VMs	256	-	-

Abbreviations: CPU, Central Processing Unit; VM, Virtual Machine.

Table 5.6: Applications Configuration for Cloud-based Big Data System

WORKLOAD		TASK	
Specs	**Quantity**	**Specs**	**Quantity**
MIPS	450K-150K	Mapper	256
NETWORK	455 GB	Reducer	10
-	-	Block size	950 MB

While Table 5.5 presents the configuration used for simulations, Table 5.6 specifies the configurations of the big data application. Our network design was based on a fat-tree topology, as depicted in Fig. 5.4. The topology consisted of three layers of switches (core, aggregate, and edge) and one layer of hosts where each host hosted several VMs that run map and/or reduce tasks. As shown in the figure, a resource manager was used to configure several MapReduce applications with different setup configurations as required. The application master (AM) was used to control and manage every requested MapReduce application separately. The node manager (NM) was used to control a VM that resides on a host. An SDN controller controlled and managed the network in real-time dynamically. Note that, BigDataSDNSim [139] uses a *MIPS* to represent the processing speed of VMs.

Dataset. Using simulations, we gathered our big data benchmark dataset, which contains a total of 2540 records. This study includes information for 13 benchmarks shown in Table 5.7. Bandwidth refers to the amount of data that can be transferred from source to destination within a given timeframe [160]. The *HBW* benchmark is used to measure the bandwidth value, which affects the *HTT*. Similarly, the *MBW* benchmark is used to measure the bandwidth value which affects the *MTT* while

the *RBW* benchmark is used to measure the bandwidth value, which affects the *RTT*. Moreover, the *BBW* benchmark is used to measure the bandwidth value, which affects the *BTT*. The *MIPS* (million instructions per second) number is a measure used to measure computer performance and indicates the amount of work processed per unit of time, which is used for the *MMIPS* and the *RMIPS* values that affect the *MET* and the *RET* respectively. Finally, the *TT* refers to the end-to-end completion time required for each task to be completed. For example, the blue line in Fig. 5.1 shows the (*TT*) for a specific task in the whole system. The dataset is publicly available on GitHub[6].

Table 5.7: Statistics related to all values present in the dataset

Benchmark	Min.	Max.	Mean	Std. Dev.	Count
HBW *(Mbps)*	0	11.81	7.75799	4.56164	2540
HTT *(ms)*	0	355.03	199.259	136.779	2540
MMIPS	652289	949556	799991	91188.4	2540
MET *(ms)*	765.49	1478.93	119.98	201.854	2540
MBW *(Mbps)*	1.49	30	5.58806	7.79096	2540
MTT *(ms)*	2.49	50.52	31.4074	15.1864	2540
RMIPS	150000	563904	292648	152369	2540
RET *(ms)*	120	480	300	180.035	2540
RBW *(Mbps)*	3.8	150	38.462	57.3594	2540
RTT *(ms)*	6.33	250.14	145.363	96.2698	2540
BBW *(Mbps)*	0	50	7.96	14.0595	2540
BTT *(ms)*	0	270.02	218.016	104.125	2540
TT *(ms)*	1007.64	2864.22	2014.03	379.803	2540

5.4.2 Performance Diagnosis

We now discuss how BigPerf using BNs is used to probabilistically diagnose the performance of a big data system. Fig. 5.5 shows the screenshot of proposed system implemented in GeNIe platform.

[6]https://github.com/umitdemirbaga/BigPerf

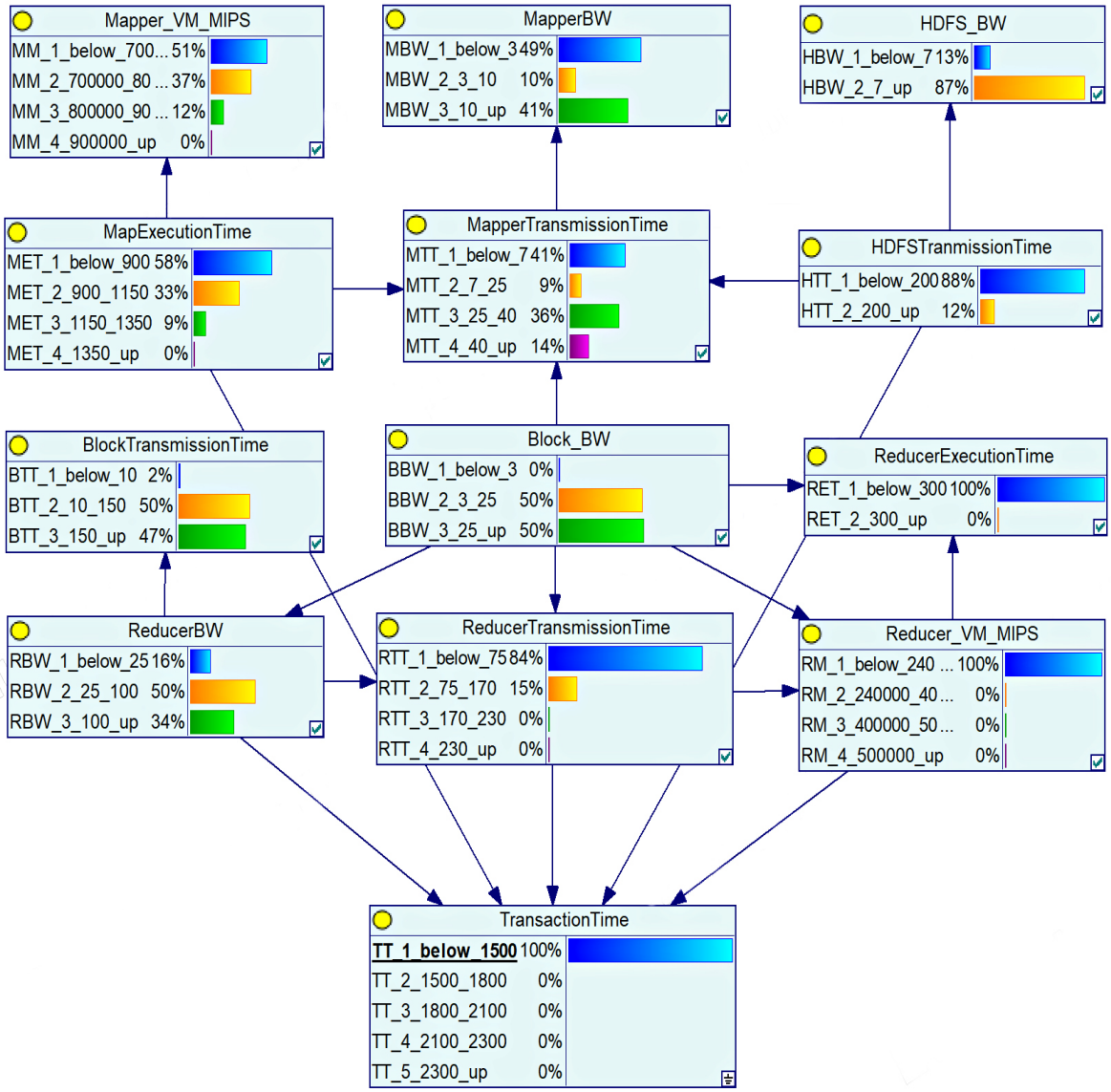


Figure 5.5: Screenshot of Bayesian Network implementation in GeNIe platform.

5.4.2.1 Transaction Time

Table 5.2, the transaction time (TT) describes the total job completion time. In this case, we would like to diagnose what are the most important factors that influence TT. We created a BN using a structural learning algorithm based on the data gathered from simulation studies. Firstly, we discretized all the random variables listed in Table 5.2 based on hierarchical clustering and manual fine-tuning to learn a BN. Upon discretization, we discovered several BN structures and selected the one that provided the best results in terms of diagnosis and prediction capabilities. Fig. 5.3 shows three such BNs.

Fig. 5.5 shows a complex BN (shown in Fig. 5.3(c))implemented in the GeNIe platform [161]. As can be observed, several random variables affect each other in a complex and stochastic manner and may have hidden relationships that may be hard to capture. Our BN alleviates these challenges. In particular, our BN learned using structural learning algorithm shows that *HTT*, *MET*, *RTT*, *RMIPS* and *RBW* directly affect *TT*. Here *HTT*, *MET*, *RTT*, *RMIPS*, and *RBW* are the parents to a child node *TT*. These nodes are linked to each other probabilistically, where the conditional probabilities are learned directly through simulation data.

Table 5.8: QoS value states representation using hierarchal discretization for transaction time *(TT)* in milliseconds

State	Range	Counts
1	0 to 1500	241
2	1500 to 1800	527
3	1800 to 2100	668
4	2100 to 2300	504
5	greater than 2300	600

To diagnose how these factors affect each other and in turn how they affect *TT*, we performed *reasoning* using our BN. Reasoning is performed by providing evidence by the stakeholders in the form of degree of belief (by varying probabilities) regarding a particular state of a random variable. For instance, if we would like to diagnose the conditions for the best-case scenario where the *TT* is less than 1500 ms (see Table 5.8), we can enter the evidence of 100% for state *TT* below 1500 ms to determine the states or other random variables (see Table 5.8). For instance, we diagnosed that for the *TT* to be below 100 ms, *RMIPS* should be below 240000 with 100% probability; further, we should have a very good *HTT* (i.e., it should be below 200 ms as shown by 88% probability). So from this analysis, we determine that *RMIPS* and *HTT* have significant impacts on the performance of *TT* such that we need both high capacity CPUs and very good network conditions to have the best *TT*. As can be noted, our BN gives the ability to provide evidence to any state of any random variable in the BN to draw a large number of conclusions intuitively.

Let us consider the worst-case scenario where we enter the evidence as "*TT* above 2300 ms" with 100% probability. We then perform the inference; upon which we note that

RMIPS should be between 400000 and 500000 with 41% probability; further, we should have poor *RBW* and it should be below 25 Mbps as showed by 100% probability. In addition, we should have high *RTT* (i.e., it should above 230 ms with 55% probability). So from this analysis, we determine that we need an excellent network conditions for excellent *TT* performance of the MapReduce applications.

5.4.2.2 Mapper Performance Diagnosis

To further validate BigPerf, we studied the most important factor(s) that affect the mapper performance. We used hierarchical discretisation method with manual fine tuning to discretize the MET values as shown in Table 5.9.

Table 5.9: QoS value states representation using hierarchal discretization for mapper execution time *(MET)* in milliseconds

State	Range	Counts
1	0 to 900	516
2	900 to 1150	855
3	1150 to 1350	604
4	greater than 1350	565

For diagnosing *MET*, we studied the impact of all the random variables on it. In particular, we entered evidences in our BN using various combinations of random variables. However, we found out that the CPU is the most dominant factor i.e., when we provided the evidence in *MMIPS* for the range below 700000 with 100% probability, we found that *MET* would be below 900 ms with 100% probability which is the best case scenario. The rest of the factors did not influence *MET* evidenced by uniform probability distribution of the states belonging to these random variable.

5.4.2.3 Reducer Performance Diagnosis

We used hierarchical discretisation method with manual fine tuning to discretize the QoS values. In all, we created two states for this dataset as shown in Table 5.10.

We studied the most important factor(s) that affect the reducer performance. In particular, we studied the impact of all the random variables on *RET*. For diagnosing *RET*, we entered evidences in our BN using various combinations of random variables.

Table 5.10: QoS value states representation using hierarchal discretization for reducer execution time *(RET)* in milliseconds

State	Range	Counts
1	0 to 300	1270
2	greater than 300	1270

We started by selecting the *RBW* below 25 and we found the performance of *RBW* to be reasonably predictable where there was 60% chance that the values will lie in the range between 230 and up. We, then, selected *RBW* between 25 and 100 and we found that most of the *RBW* lie in the range of below 75 Mbps. (state 1) with the probability of 65%. Finally, we studied the performance of *RBW* between 100 Mbps and up and we found that most of the *RBW* lie in the range of below 75 Mbps (46% probability) lie in the range of 35 and 65 (state 1).

We selected *RET* below 300 ms and we found that all of the *RMIPS* (100% probability) lie in the range of below 24000 (state 1, see Table 5.11). From this analysis, we diagnose that most important factor(s) that affect the reducer performance is CPU.

Table 5.11: QoS value states representation using hierarchal discretization for reducer VM MIPS *(RMIPS)*

State	Range	Counts
1	0 to 24000	1270
2	24000 to 400000	508
3	400000 to 500000	508
4	greater than 500000	254

Based on the results presented in this sub-section we gather that BigPerf is able to identify the most relevant factors that can be used to efficiently diagnose the big data system.

5.4.3 Big Data QoS Prediction

We developed three BNs: Noisy-Or (NOR), Naïve Bayes Network (NBN), and Complex Bayesian Network (CBN) (see Fig. 5.3 and Fig. 5.5). For training the model, we used the expectation-maximization algorithm (EM algorithm) [159]. The EM algorithm also deals efficiently with scarce, sparse or missing data to learn the most efficient

BN model [158]. Table 5.12 shows the prediction accuracy of all our BNs. As can be observed, the CBN performs the best, followed by NOR and NBN. CBN makes the most of conditional relationships between all the random variables and provides the best prediction accuracy. Finally, the NOR model is the hardest to train as all the probabilities of random variables are computed to predict the state of a single random variable. However, it performs reasonably well in our case. In summary, our results demonstrate that BigPerf predicts QoS efficiently with an overall prediction accuracy score of nearly 94.04% and meet the requirement of a BN-based methodology for the diagnosis and prediction of the performance of big data frameworks.

Table 5.12: Big data performance prediction accuracy (%) for different type of Bayesian Networks

Type	***MMIPS***	***RMIPS***	***BBW***	***HBW***	***MET***	***RET***	***TT***
NOR	66.25%	81.41%	83.66%	92.08%	81.29%	97.71%	80.71%
NBN	40.23%	59.17%	81.14%	62.99%	42.16%	80.23%	68.26%
CBN	92.63%	88.46%	100%	99.29%	93.54%	100%	84.40%

5.5 Conclusion

This chapter proposes, develops and validates BigPerf – a Bayesian system for big data performance diagnosis and prediction. The results presented in the chapter clearly show that BigPerf can be used to diagnose and predict the performance of big data systems efficiently. The major highlight of BigPerf is that it can take into consideration several factors, modelled as random variables together, such as execution time of each specific task (mapper and reducer), network transmission time between these tasks as well as data block split time (HDFS to mapper, reducer to HDFS) for efficient diagnosis of the performance of big data systems. The complex and uncertain relationships between specified factors can be modelled probabilistically by BigPerf to predict several hypotheses regarding the performance of big data systems. We also validated the BigPerf prediction capability and show that it predicts the performance of big data systems with high accuracy of approximately 94.04%.

6

Conclusion

Contents

Summary

In this chapter, we summarize the research work presented in this dissertation. Then, we outline the contributions and propose future research directions for addressing the existing challenges in the current state-of-the-art.

6.1 Thesis Summary

Big data systems have many simultaneous and interactive components, making it difficult to find the root cause of the problem. Big data systems have many concurrent tasks, namely multiple tasks running at the same time but not necessarily simultaneously and parallel tasks, namely the tasks executed by different worker nodes at the same time. In such complex systems, due to interdependence and synchronicity of tasks, it is very challenging to find the reasons why: tasks are slowing down, resource utilization is low, the response time is high, or when the network latency is high. Moreover, performance diagnosis and prediction of big data systems are highly complex as these frameworks are typically deployed in cloud data centers that are large-scale, highly concurrent, and follows a multi-tenant model. Several factors, including hardware heterogeneity, stochastic networks and application workloads may impact the performance of big data systems.

This thesis explored numerous challenges for the diagnosis of big data systems in cloud datacenters and proposed solutions that ease the diagnosis process. In particular, this thesis contributes as:

Chapter 1 presents the general background of diagnosis and evaluation of cloud-based big data systems including an overview of big data systems and the types of data processing systems, a brief information about cloud datacenters, and a general information about Quality of Service (QoS). It also reveals challenges and research questions, along with the thesis contributions.

Chapter 2 presents some background information regarding the overall topic, including a brief description on big data, Apache Hadoop and its main components, big data applications based on MapReduce framework, performance diagnosis of big data systems,

cloud computing and its relationship with big data, and commercial and open source tools for big data systems. A major focus of this thesis is to address the challenges of performance diagnosis and evaluation of big data systems.

Chapter 3 presents SmartMonit, a real-time Big data monitoring system, which collects infrastructure information, process status of each task. At the same, we develop a real-time stream process framework to analyze the coordination among tasks to tasks and infrastructures to tasks. This coordination information is essential for troubleshooting the reasons for failures and performance reduction, especially the ones propagated from other causes.

Chapter 4 presents AutoDiagn, a generic and flexible framework that provides holistic monitoring of a big data system, while detecting the symptom of performance reduction and enabling root-cause analysis. An implementation of the proposed framework interacts with a Hadoop cluster and is evaluated with real-world benchmark applications. All experiments are conducted on AWS. Experimental results show that our implementation has a small resource footprint, high throughput and low latency.

Chapter 5 presents BigPerf, a novel Bayesian system for diagnosing and predicting root causes of performance degradation and bottlenecks of Hadoop Applications. BigPerf analyzes and characterizes the performance of Hadoop applications by incorporating Bayesian networks to determine uncertain and complex relationships while dealing with noisy and missing data. Our extensive simulation studies validate BigPerf and show that it can efficiently diagnose the causes regarding the performance of Hadoop Applications. We also show that BigPerf predicts systems performance with high accuracy of approximately 94.04%.

6.1.1 Limitations

The proposed monitoring system, SmartMonit, provides a holistic performance data collection for big data systems while visualising the collected information with the Execution Graph modeled as directed acyclic graphs (DAGs). However, the developed graph is an undynamic form that works only with a specified number of nodes. Thus, the visualization system is tested and demonstrated with a cluster consisting of three

nodes. In addition, although the proposed diagnosis system called AutoDiagn is designed as a microservice architecture that offers the flexibility to plug a new detection and root-cause analysis module for various types of big data systems, it needs to be implemented new applications including symptom detection and root-cause analysis to populate our system. Additionally, the storage overhead increases with the number of applications increasing. So, new techniques are required to reduce overheads.

Finally, the developed big data performance diagnosis and prediction system, BigPerf, predicts the performance of big data systems with high accuracy of approximately 94.04% while uncovering the complex and uncertain relationships between performance features clearly. However, the benchmark dataset is gathered using a well-validated simulator. A real Hadoop cluster and real applications need to be used to conduct the performance analysis with the BigPerf system.

6.2 Future Research Directions

This section provides motivation for a variety of potential areas of future research, which can be inspired by the work done in this PhD thesis.

6.2.1 SDN-based Light-weight Monitoring Framework for Big Data Systems

Our monitoring system, SmartMonit, is designed in a loosely-coupled manner, the monitored information of the components can be easily scaled. However, the storage overhead increases with the number of applications increasing. [162] proposed a caching method to aggregate the information before sending to destination nodes. This direction can be explored in future work to reduce the storage overhead and network overhead. Moreover, SmartMonit is not able to catch the network-level information, such as end-to-end network tracking. SDN-level data collection system is required to populate our SmartMonit for future work.

6.2.2 Diagnosis Framework for Big Data Systems using AI Techniques

The performance diagnosis system presented in Chapter 4 proposes an automated real-time diagnosis framework for big data systems. This system consists of two main parts: Monitoring component to monitor big data systems, and diagnosing component to detect the symptom of performance reduction and enable root-cause analysis. For diagnosis, the system uses user-defined functions (UDFs) technique to analyze the performance metrics from streaming data and from different levels in a storage unit. Future studies can extend our framework with artificial intelligence (AI), such as machine learning and deep learning techniques that are effective at detecting false patterns that are not present in datasets, as they have ability to learn new patterns from data and they increase the accuracy of both symptom detection and root cause analysis and reduce complexity and computation time. For example, if something goes wrong, then the AI model can improve itself and its predictive based on new coming data.

6.2.3 Online Performance Diagnosis and Prediction for Big Data Systems

A Bayesian system for performance diagnosis and prediction for cloud-based big data systems presented in Chapter 5 raises an interesting point for discussion on further work. The complex and uncertain relationships between specified factors are modelled probabilistically by BigPerf to predict many hypotheses regarding the performance of big data systems while taking into consideration various factors together. However, this systems validated offline using the data generated by BigDataSDNSim simulator [139]. A potential thread for future research is to build an online performance diagnosis and prediction model for big data systems using Bayesian networks to identify the most relevant factors to performance issues by uncovering the complex and uncertain relationships between specified factors among the performance metrics in real-time.

6.2.4 Performance Evaluation of Container-based Big Data Applications in Multiple Cloud Environments

Big data analytics includes various complex operations such as storing, cleaning, organizing, modelling, analysis and presentation of data at scale. Installing big data systems and integrating them with each other to ensure a robust big data processing system is a major challenge in cloud computing environments. Besides, although technically correct, all cloud services are prone to failure and come with some problems, such as cost and security risks. A lot of work has been done to handle the problems regarding installation deployment difficulties, cost management, and security risks by proposing Docker-based big data systems across multiple clouds [163], [164], [165]. However, the performance diagnosis and failure managements in such systems still remain unsolved. In this thesis, we propose AutoDiagn, an automated real-time diagnosis framework for big data systems to diagnose performance issues on VM based big data systems in a single cloud environment. Future work can improve our system to perform performance diagnosis on docker container-based big data processing systems in multiple clouds.

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
HDFS	Hadoop Distributed File System
HTTP	Hyper Text Transfer Protocol
IoT	Internet of Things
I/O	Input/output
OS	Operating System
QoS	Quality of Service
REST API	Representational State Transfer API
SLA	Service Level Agreement
TCP	Transmission Control Protocol
VM	Virtual Machine
YARN	Yet Another Resource Negotiator

References

[1] Data management: In-depth guide. Accessed: 2021-06-03. [Online]. Available: https://www.merriam-webster.com/.

[2] A. Oussous, F.-Z. Benjelloun, A. A. Lahcen, and S. Belfkih, "Big data technologies: A survey," *Journal of King Saud University-Computer and Information Sciences*, vol. 30, no. 4, pp. 431–448, 2018.

[3] R. Ren, J. Cheng, X. He, L. Wang, C. Luo, and J. Zhan, "Hybridtune: Spatio-temporal data and model driven performance diagnosis for big data systems," *arXiv preprint arXiv:1711.07639*, 2017.

[4] B. Lebiednik, A. Mangal, and N. Tiwari, "A survey and evaluation of data center network topologies," *arXiv preprint arXiv:1605.01701*, 2016.

[5] S. Patidar, D. Rane, and P. Jain, "A survey paper on cloud computing," in *2012 second international conference on advanced computing & communication technologies*. IEEE, 2012, pp. 394–398.

[6] W. Hussain, F. K. Hussain, O. K. Hussain, E. Damiani, and E. Chang, "Formulating and managing viable slas in cloud computing from a small to medium service provider's viewpoint: A state-of-the-art review," *Information Systems*, vol. 71, pp. 240–259, 2017.

[7] C. Tao and J. Gao, "Quality assurance for big data application-issuses, challenges, and needs." in *SEKE*, vol. 2, no. 3, 2016, pp. 1–7.

[8] I. Ayadi, N. Simoni, and T. Aubonnet, "Sla approach for" cloud as a service"," in *2013 IEEE Sixth International Conference on Cloud Computing*. IEEE, 2013, pp. 966–967.

[9] S. Rong and Z. Bao-wen, "The research of regression model in machine learning field," in *MATEC Web of Conferences*, vol. 176. EDP Sciences, 2018, p. 01033.

[10] A. K. Maheshwari, "Big data applications and architectures for emerging countries," in *Advancing Innovation and Sustainable Outcomes in International Graduate Education*. IGI Global, 2021, pp. 120–142.

[11] Datadog. Accessed: 2020-07-13. [Online]. Available: https://www.datadoghq.com/.

[12] Sequenceiq. Accessed: 2020-07-14. [Online]. Available: https://github.com/sequenceiq.

[13] Sematext. Accessed: 2020-07-13. [Online]. Available: https://sematext.com/.

[14] R. T. Evans, J. C. Browne, and W. L. Barth, "Understanding application and system performance through system-wide monitoring," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2016, pp. 1702–1710.

[15] G. Ananthanarayanan, S. Kandula, A. G. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using mantri." in *Osdi*, vol. 10, no. 1, 2010, p. 24.

[16] A. Netti, M. Müller, C. Guillen, M. Ott, D. Tafani, G. Ozer, and M. Schulz, "Dcdb wintermute: Enabling online and holistic operational data analytics on hpc systems," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, 2020, pp. 101–112.

[17] Nagios. Accessed: 2020-07-15. [Online]. Available: https://www.nagios.org/.

[18] Ganglia. Accessed: 2020-07-15. [Online]. Available: http://ganglia.info/.

[19] Apache chukwa. Accessed: 2020-07-14. [Online]. Available: https://chukwa.apache.org/.

[20] Dmon. Accessed: 2020-07-12. [Online]. Available: https://github.com/Open-Monitor/dmon.

[21] P. Garraghan, R. Yang, Z. Wen, A. Romanovsky, J. Xu, R. Buyya, and R. Ranjan, "Emergent failures: Rethinking cloud reliability at scale," *IEEE Cloud Computing*, vol. 5, no. 5, pp. 12–21, 2018.

[22] P. Verissimo and L. Rodrigues, *Distributed systems for system architects*. Springer Science & Business Media, 2001, vol. 1.

[23] L. Braubach and A. Pokahr, "Addressing challenges of distributed systems using active components," in *Intelligent Distributed Computing V*. Springer, 2011, pp. 141–151.

[24] O. Hummel, H. Eichelberger, A. Giloj, D. Werle, and K. Schmid, "A collection of software engineering challenges for big data system development," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2018, pp. 362–369.

[25] B. Debnath, M. Solaimani, M. A. G. Gulzar, N. Arora, C. Lumezanu, J. Xu, B. Zong, H. Zhang, G. Jiang, and L. Khan, "Loglens: A real-time log analysis system," in *2018 IEEE 38th international conference on distributed computing systems (ICDCS)*. IEEE, 2018, pp. 1052–1062.

[26] A. Miranskyy, A. Hamou-Lhadj, E. Cialini, and A. Larsson, "Operational-log analysis for big data systems: Challenges and solutions," *IEEE Software*, vol. 33, no. 2, pp. 52–59, 2016.

[27] D. P. Acharjya and K. Ahmed, "A survey on big data analytics: challenges, open research issues and tools," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 2, pp. 511–518, 2016.

[28] S. Lu, X. Wei, B. Rao, B. Tak, L. Wang, and L. Wang, "Ladra: Log-based abnormal task detection and root-cause analysis in big data processing with spark," *Future Generation Computer Systems*, vol. 95, pp. 392–403, 2019.

[29] Q. Guo, Y. Li, T. Liu, K. Wang, G. Chen, X. Bao, and W. Tang, "Correlation-based performance analysis for full-system mapreduce optimization," in *2013 IEEE International Conference on Big Data*. IEEE, 2013, pp. 753–761.

[30] A. Jain and M. Choudhary, "Analyzing & optimizing hadoop performance," in *2017 International Conference on Big Data Analytics and Computational Intelligence (ICBDAC)*. IEEE, 2017, pp. 116–121.

[31] B. T. Rao, N. Sridevi, V. K. Reddy, and L. Reddy, "Performance issues of heterogeneous hadoop clusters in cloud computing," *arXiv preprint arXiv:1207.0894*, 2012.

[32] K. Bakshi, "Considerations for big data: Architecture and approach," in *2012 IEEE aerospace conference*. IEEE, 2012, pp. 1–7.

[33] A. Rasooli and D. G. Down, "Guidelines for selecting hadoop schedulers based on system heterogeneity," *Journal of grid computing*, vol. 12, no. 3, pp. 499–519, 2014.

[34] H. Gao, Z. Yang, J. Bhimani, T. Wang, J. Wang, B. Sheng, and N. Mi, "Autopath: harnessing parallel execution paths for efficient resource allocation in multi-stage big data frameworks," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2017, pp. 1–9.

[35] J. Li, Y. Wang, J. Yu, and S. Guo, "An hmm-based performance diagnosis approach for hadoop clusters," in *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE, 2016, pp. 1–4.

[36] U. Demirbaga, A. Noor, Z. Wen, P. James, K. Mitra, and R. Ranjan, "Smartmonit: Real-time big data monitoring system," in *2019 38th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2019, pp. 357–3572.

[37] U. Demirbaga, Z. Wen, A. Noor, K. Mitra, K. Alwasel, S. Garg, A. Zomaya, and R. Ranjan, "Autodiagn: An automated real-time diagnosis framework for big data systems," *IEEE Transactions on Computers*, 2021.

[38] A. Labrinidis and H. V. Jagadish, "Challenges and opportunities with big data," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 2032–2033, 2012.

[39] L. Williams, "Data dna and diamonds," *Engineering & Technology*, vol. 14, no. 3, pp. 62–65, 2019.

[40] G.-H. Kim, S. Trimi, and J.-H. Chung, "Big-data applications in the government sector," *Communications of the ACM*, vol. 57, no. 3, pp. 78–85, 2014.

[41] U. Demirbaga, D. N. Jha, N. Booth, T. Roberts, T. Shah, R. Ranjan, and A. Batch, "Tc-cps newsletter," *TC*, vol. 1, no. 6, 2018.

[42] M. D. Smith and R. Telang, *Streaming, sharing, stealing: big data and the future of entertainment*. Mit Press, 2016.

[43] K. Zhou, C. Fu, and S. Yang, "Big data driven smart energy management: From big data to big insights," *Renewable and Sustainable Energy Reviews*, vol. 56, pp. 215–225, 2016.

[44] A. Noor, K. Mitra, E. Solaiman, A. Souza, D. N. Jha, U. Demirbaga, P. P. Jayaraman, N. Cacho, and R. Ranjan, "Cyber-physical application monitoring across multiple clouds," *Computers & Electrical Engineering*, vol. 77, pp. 314–324, 2019.

[45] H. Hassani, X. Huang, and E. Silva, "Digitalisation and big data mining in banking," *Big Data and Cognitive Computing*, vol. 2, no. 3, p. 18, 2018.

[46] K. Alwasel, D. N. Jha, F. Habeeb, U. Demirbaga, O. Rana, T. Baker, S. Dustdar, M. Villari, P. James, E. Solaiman *et al.*, "Iotsim-osmosis: A framework for modeling and simulating iot applications over an edge-cloud continuum," *Journal of Systems Architecture*, vol. 116, p. 101956, 2021.

[47] J. Phengsuwan, T. Shah, P. James, D. Thakker, S. Barr, and R. Ranjan, "Ontology-based discovery of time-series data sources for landslide early warning system," *Computing*, pp. 1–19, 2019.

[48] K. Shvachko, H. Kuang, S. Radia, R. Chansler *et al.*, "The hadoop distributed file system." in *MSST*, vol. 10, 2010, pp. 1–10.

[49] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[50] U. Demirbaga, "Htwitt: a hadoop-based platform for analysis and visualization of streaming twitter data," *Neural Computing and Applications*, pp. 1–16, 2021.

[51] M. Rodrigues, M. Y. Santos, and J. Bernardino, "Big data processing tools: An experimental performance evaluation," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 9, no. 2, p. e1297, 2019.

[52] B. Cyganek, M. Graña, B. Krawczyk, A. Kasprzak, P. Porwik, K. Walkowiak, and M. Woźniak, "A survey of big data issues in electronic health record analysis," *Applied Artificial Intelligence*, vol. 30, no. 6, pp. 497–520, 2016.

[53] S. G. Walunj and K. Sadafale, "An online recommendation system for e-commerce based on apache mahout framework," in *Proceedings of the 2013 annual conference on Computers and people research*, 2013, pp. 153–158.

[54] S. Mardani, M. K. Akbari, and S. Sharifian, "Fraud detection in process aware information systems using mapreduce," in *2014 6th Conference on Information and Knowledge Technology (IKT)*. IEEE, 2014, pp. 88–91.

[55] Dictionary | merriam-webster. Accessed: 2021-06-30. [Online]. Available: https://www.merriam-webster.com/.

[56] B. R. Helm, P. Malony, and S. Fickas, "Capturing and automating performance diagnosis: the poirot approach," in *Proceedings of 9th International Parallel Processing Symposium*. IEEE, 1995, pp. 606–613.

[57] H. Jayathilaka, C. Krintz, and R. Wolski, "Performance monitoring and root cause analysis for cloud-hosted web applications," in *Proceedings of the 26th International Conference on World Wide Web*, 2017, pp. 469–478.

[58] M. Julian, *Practical Monitoring: Effective Strategies for the Real World*. " O'Reilly Media, Inc.", 2017.

[59] K. Zhang, M. Wan, T. Qu, H. Jiang, P. Li, Z. Chen, J. Xiang, X. He, C. Li, and G. Q. Huang, "Production service system enabled by cloud-based smart resource hierarchy for a highly dynamic synchronized production process," *Advanced Engineering Informatics*, vol. 42, p. 100995, 2019.

[60] C. Qian, Y. Zhang, Y. Liu, and Z. Wang, "A cloud service platform integrating additive and subtractive manufacturing with high resource efficiency," *Journal of Cleaner Production*, vol. 241, p. 118379, 2019.

[61] M. Andreolini, M. Colajanni, M. Pietri, and S. Tosi, "Adaptive, scalable and reliable monitoring of big data on clouds," *Journal of Parallel and Distributed Computing*, vol. 79, pp. 67–79, 2015.

[62] G. S. Aujla and A. Jindal, "A decoupled blockchain approach for edge-envisioned iot-based healthcare monitoring," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 2, pp. 491–499, 2020.

[63] J. J. Rooney and L. N. V. Heuvel, "Root cause analysis for beginners," *Quality progress*, vol. 37, no. 7, pp. 45–56, 2004.

[64] G. Jin, L. Song, X. Shi, J. Scherpelz, and S. Lu, "Understanding and detecting real-world performance bugs," *ACM SIGPLAN Notices*, vol. 47, no. 6, pp. 77–88, 2012.

[65] F. Castro, A. Vellido, A. Nebot, and F. Mugica, "Applying data mining techniques to e-learning problems," in *Evolution of teaching and learning paradigms in intelligent environment*. Springer, 2007, pp. 183–221.

[66] O. I. Obaid, M. A. Mohammed, M. Ghani, A. Mostafa, and F. Taha, "Evaluating the performance of machine learning techniques in the classification of wisconsin breast cancer," *International Journal of Engineering & Technology*, vol. 7, no. 4.36, pp. 160–166, 2018.

[67] T. Dietterich, "Overfitting and undercomputing in machine learning," *ACM computing surveys (CSUR)*, vol. 27, no. 3, pp. 326–327, 1995.

[68] M. K. Ahirwar, M. K. Ahirwar, and U. Chourasia, "Anomaly detection in the services provided by multi cloud architectures: a survey," *Int J Res Eng Technol*, vol. 3, no. 09, pp. 196–200, 2014.

[69] O. Tuncer, E. Ates, Y. Zhang, A. Turk, J. Brandt, V. J. Leung, M. Egele, and A. K. Coskun, "Online diagnosis of performance variation in hpc systems using machine learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 883–896, 2018.

[70] K. Radha, B. Rao, S. M. Babu, K. Rao, V. Reddy, and P. Saikiran, "Service level agreements in cloud computing and big data," *International Journal of Electrical and Computer Engineering*, vol. 5, no. 1, p. 158, 2015.

[71] L. E. B. Villalpando, A. April, and A. Abran, "Performance analysis model for big data applications in cloud computing," *Journal of Cloud Computing*, vol. 3, no. 1, pp. 1–20, 2014.

[72] Y. Zheng, X. Yi, M. Li, R. Li, Z. Shan, E. Chang, and T. Li, "Forecasting fine-grained air quality based on big data," in *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 2015, pp. 2267–2276.

[73] X. Gao, F. Yang, C. Shang, and D. Huang, "A review of control loop monitoring and diagnosis: Prospects of controller maintenance in big data era," *Chinese Journal of Chemical Engineering*, vol. 24, no. 8, pp. 952–962, 2016.

[74] I. Giannakopoulos, N. Papailiou, C. Mantas, I. Konstantinou, D. Tsoumakos, and N. Koziris, "Celar: automated application elasticity platform," in *2014 IEEE International Conference on Big Data (Big Data)*. IEEE, 2014, pp. 23–25.

[75] I. Gorton and J. Klein, "Distribution, data, deployment: Software architecture convergence in big data systems," *IEEE Software*, vol. 32, no. 3, pp. 78–85, 2014.

[76] D. Talia, "Clouds for scalable big data analytics," *Computer*, vol. 46, no. 05, pp. 98–101, 2013.

[77] H. Hu, Y. Wen, T.-S. Chua, and X. Li, "Toward scalable systems for big data analytics: A technology tutorial," *IEEE access*, vol. 2, pp. 652–687, 2014.

[78] C. Yang, Q. Huang, Z. Li, K. Liu, and F. Hu, "Big data and cloud computing: innovation opportunities and challenges," *International Journal of Digital Earth*, vol. 10, no. 1, pp. 13–53, 2017.

[79] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao, "A framework for native multi-tenancy application development and management," in *The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007)*. IEEE, 2007, pp. 551–558.

[80] Y. Li and X. Zhai, "Review and prospect of modern education using big data," *Procedia Computer Science*, vol. 129, pp. 341–347, 2018.

[81] S. Yu, "Data processing and development of big data system: A survey," in *International Conference on Artificial Intelligence and Security*. Springer, 2021, pp. 420–431.

[82] C. S. Mayo, M. M. Matuszak, M. J. Schipper, S. Jolly, J. A. Hayman, and R. K. Ten Haken, "Big data in designing clinical trials: opportunities and challenges," *Frontiers in oncology*, vol. 7, p. 187, 2017.

[83] R. Elshawi, S. Sakr, D. Talia, and P. Trunfio, "Big data systems meet machine learning challenges: towards big data science as a service," *Big data research*, vol. 14, pp. 1–11, 2018.

[84] R. Jhawar and V. Piuri, "Fault tolerance and resilience in cloud computing environments," in *Computer and information security handbook*. Elsevier, 2017, pp. 165–181.

[85] F. H. Gebara, H. P. Hofstee, and K. J. Nowka, "Second-generation big data systems," *Computer*, vol. 48, no. 01, pp. 36–41, 2015.

[86] J. Bao, H. Wu, and Y. Yan, "A fault diagnosis system-plc design for system reliability improvement," *The International Journal of Advanced Manufacturing Technology*, vol. 75, no. 1-4, pp. 523–534, 2014.

[87] I. Noorwali, D. Arruda, and N. H. Madhavji, "Understanding quality requirements in the context of big data systems," in *Proceedings of the 2nd International Workshop on BIG Data Software Engineering*, 2016, pp. 76–79.

[88] B. Furht, "Cloud computing fundamentals," in *Handbook of cloud computing*. Springer, 2010, pp. 3–19.

[89] L. Wang, R. Ranjan, J. Chen, and B. Benatallah, *Cloud computing: methodology, systems, and applications*. CRC Press, 2017.

[90] Y. Xing and Y. Zhan, "Virtualization and cloud computing," in *Future Wireless Networks and Information Systems*. Springer, 2012, pp. 305–312.

[91] Z. Sun, L. Sun, and K. Strang, "Big data analytics services for enhancing business intelligence," *Journal of Computer Information Systems*, vol. 58, no. 2, pp. 162–169, 2018.

[92] X. Dai and B. Bensaou, "Scheduling for response time in hadoop mapreduce," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.

[93] W. Pourmajidi, J. Steinbacher, T. Erwin, and A. Miranskyy, "On challenges of cloud monitoring," *arXiv preprint arXiv:1806.05914*, 2018.

[94] A. Noor, D. N. Jha, K. Mitra, P. P. Jayaraman, A. Souza, R. Ranjan, and S. Dustdar, "A framework for monitoring microservice-oriented cloud applications in heterogeneous virtualization environments," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 156–163.

[95] S. Qanbari, A. Farivarmoheb, P. Fazlali, S. Mahdizadeh, and S. Dustdar, "Telemetry for elastic data (ted): Middleware for mapreduce job metering and rating," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 2. IEEE, 2015, pp. 104–111.

[96] G. Iuhasz and I. Dragan, "An overview of monitoring tools for big data and cloud applications," in *2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE, 2015, pp. 363–366.

[97] M. Massie, B. Li, B. Nicholes, V. Vuksan, R. Alexander, J. Buchbinder, F. Costa, A. Dean, D. Josephsen, P. Phaal *et al.*, *Monitoring with Ganglia: tracking dynamic host and application metrics at scale*. " O'Reilly Media, Inc.", 2012.

[98] A. Rabkin and R. Katz, "Chukwa: A system for reliable large-scale log collection," in *Proceedings of LISA'10: 24th Large Installation System Administration Conference*, 2010, p. 163.

[99] I. Drăgan, G. Iuhasz, and D. Petcu, "A scalable platform for monitoring data intensive applications," *Journal of Grid Computing*, vol. 17, no. 3, pp. 503–528, 2019.

[100] Apache ambari. Accessed: 2019-07-15. [Online]. Available: https://ambari.apache.org/.

[101] S. Wadkar and M. Siddalingaiah, "Apache ambari," in *Pro Apache Hadoop*. Springer, 2014, pp. 399–401.

[102] G. Iuhasz, D. Pop, and I. Dragan, "Architecture of a scalable platform for monitoring multiple big data frameworks," *Scalable Computing: Practice and Experience*, vol. 17, no. 4, pp. 313–321, 2016.

[103] Z. Zhang, C. Li, Y. Tao, R. Yang, H. Tang, and J. Xu, "Fuxi: a fault-tolerant resource management and job scheduling system at internet scale," in *Proceedings of the VLDB Endowment*, vol. 7, no. 13. VLDB Endowment Inc., 2014, pp. 1393–1404.

[104] X. Sun, C. Hu, R. Yang, P. Garraghan, T. Wo, J. Xu, J. Zhu, and C. Li, "Rose: Cluster resource scheduling via speculative over-subscription," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 949–960.

[105] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 270–288.

[106] "Demonstration. [online]. available: https://youtu.be/Ok0iJBbC5zA."

[107] G. S. Aujla, M. Barati, O. Rana, S. Dustdar, A. Noor, J. T. Llanos, M. Carr, D. Marikyan, S. Papagiannidis, and R. Ranjan, "Com-pace: Compliance-aware cloud application engineering using blockchain," *IEEE Internet Computing*, vol. 24, no. 5, pp. 45–53, 2020.

[108] P. Lubbers, B. Albers, F. Salim, and T. Pye, *Pro HTML5 programming*. Springer, 2011.

[109] H. W. Lie and B. Bos, *Cascading style sheets: designing for the Web*. Addison-Wesley Longman Publishing Co., Inc., 1997.

[110] L. Welling and L. Thomson, *PHP and MySQL Web development*. Sams Publishing, 2003.

[111] S. Babu, "Towards automatic optimization of mapreduce programs," in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 137–142.

[112] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: Sql and rich analytics at scale," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of data*, 2013, pp. 13–24.

[113] P. Garraghan, X. Ouyang, P. Townend, and J. Xu, "Timely long tail identification through agent based monitoring and analytics," in *2015 IEEE 18th International Symposium on Real-Time Distributed Computing*. IEEE, 2015, pp. 19–26.

[114] Z. Wen, T. Lin, R. Yang, S. Ji, R. Ranjan, A. Romanovsky, C. Lin, and J. Xu, "Ga-par: Dependable microservice orchestration framework for geo-distributed clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 1, pp. 129–143, 2019.

[115] Z. Wen, J. Cała, P. Watson, and A. Romanovsky, "Cost effective, reliable and secure workflow deployment over federated clouds," *IEEE Transactions on Services Computing*, vol. 10, no. 6, pp. 929–941, 2016.

[116] Z. Wen, R. Qasha, Z. Li, R. Ranjan, P. Watson, and A. Romanovsky, "Dynamically partitioning workflow over federated clouds for optimising the monetary cost and handling run-time failures," *IEEE Transactions on Cloud Computing*, 2016.

[117] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010, pp. 135–146.

[118] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica *et al.*, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[119] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[120] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 261–276.

[121] N. J. Yadwadkar and W. Choi, "Proactive straggler avoidance using machine learning," *White paper, University of Berkeley*, 2012.

[122] A. Badita, P. Parag, and V. Aggarwal, "Optimal server selection for straggler mitigation," *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, pp. 709–721, 2020.

[123] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[124] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, and B.-G. Chun, "Making sense of performance in data analytics frameworks," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, 2015, pp. 293–307.

[125] P. Garraghan, X. Ouyang, R. Yang, D. McKee, and J. Xu, "Straggler root-cause and impact analysis for massive-scale virtualized cloud datacenters," *IEEE Transactions on Services Computing*, vol. 12, no. 1, pp. 91–104, 2016.

[126] X. Ouyang, P. Garraghan, R. Yang, P. Townend, and J. Xu, "Reducing late-timing failure at scale: Straggler root-cause analysis in cloud datacenters," in *Fast Abstracts in the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. DSN, 2016.

[127] H. Zhou, Y. Li, H. Yang, J. Jia, and W. Li, "Bigroots: An effective approach for root-cause analysis of stragglers in big data system," *IEEE Access*, vol. 6, pp. 41 966–41 977, 2018.

[128] Z. He, Y. He, F. Liu, and Y. Zhao, "Big data-oriented product infant failure intelligent root cause identification using associated tree and fuzzy dea," *IEEE Access*, vol. 7, pp. 34 687–34 698, 2019.

[129] H. Du and S. Zhang, "Hawkeye: Adaptive straggler identification on heterogeneous spark cluster with reinforcement learning," *IEEE Access*, vol. 8, pp. 57 822–57 832, 2020.

[130] J. P. Magalhães and L. M. Silva, "Root-cause analysis of performance anomalies in web-based applications," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, 2011, pp. 209–216.

[131] R. Bitar, M. Wootters, and S. El Rouayheb, "Stochastic gradient coding for straggler mitigation in distributed learning," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 277–291, 2020.

[132] A. M. Chacko, J. S. Medicherla, and S. M. Kumar, "Anomaly detection in mapreduce using transformation provenance," in *Advances in Big Data and Cloud Computing*. Springer, 2018, pp. 91–99.

[133] N. Khoussainova, M. Balazinska, and D. Suciu, "Perfxplain: debugging mapreduce job performance," *arXiv preprint arXiv:1203.6400*, 2012.

[134] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1285–1298.

[135] A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fullop, A. Gentile, S. Monk, N. Naksinehaboon, J. Ogden *et al.*, "The lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications," in *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2014, pp. 154–165.

[136] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "Anomaly detection using autoencoders in high performance computing systems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 9428–9433.

[137] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[138] T. Renner, L. Thamsen, and O. Kao, "Coloc: Distributed data and container colocation for data-intensive applications," in *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 2016, pp. 3008–3015.

[139] K. Alwasel, R. N. Calheiros, S. Garg, R. Buyya, M. Pathan, D. Georgakopoulos, and R. Ranjan, "Bigdatasdnsim: A simulator for analyzing big data applications in software-defined cloud data centers," *Software: Practice and Experience*, 2020.

[140] U. Demirbaga and D. N. Jha, "Social media data analysis using mapreduce programming model and training a tweet classifier using apache mahout," in *2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2)*. IEEE, 2018, pp. 116–121.

[141] H. E. Ciritoglu, J. Murphy, and C. Thorpe, "Hard: a heterogeneity-aware replica deletion for hdfs," *Journal of big data*, vol. 6, no. 1, pp. 1–21, 2019.

[142] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments." in *Osdi*, vol. 8, no. 4, 2008, p. 7.

[143] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, 2011, pp. 242–253.

[144] Y. Y. Wee, W. P. Cheah, S. C. Tan, and K. Wee, "A method for root cause analysis with a bayesian belief network and fuzzy cognitive map," *Expert Systems with Applications*, vol. 42, no. 1, pp. 468–487, 2015.

[145] M. Khan, Y. Jin, M. Li, Y. Xiang, and C. Jiang, "Hadoop performance modeling for job estimation and resource provisioning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 441–454, 2015.

[146] X. Lin, Z. Meng, C. Xu, and M. Wang, "A practical performance model for hadoop mapreduce," in *2012 IEEE International Conference on Cluster Computing Workshops*. IEEE, 2012, pp. 231–239.

[147] N. Wang, J. Yang, Z. Lu, X. Li, and J. Wu, "Comparison and improvement of hadoop mapreduce performance prediction models in the private cloud," in *Asia-Pacific Services Computing Conference*. Springer, 2016, pp. 77–91.

[148] P. Liu, H. Xu, Q. Ouyang, R. Jiao, Z. Chen, S. Zhang, J. Yang, L. Mo, J. Zeng, W. Xue *et al.*, "Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 48–58.

[149] E. Vianna, G. Comarela, T. Pontes, J. Almeida, V. Almeida, K. Wilkinson, H. Kuno, and U. Dayal, "Analytical performance models for mapreduce workloads," *International Journal of Parallel Programming*, vol. 41, no. 4, pp. 495–525, 2013.

[150] G. Wang, A. R. Butt, P. Pandey, and K. Gupta, "A simulation approach to evaluating design decisions in mapreduce setups," in *2009 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems*. IEEE, 2009, pp. 1–11.

[151] K. Kambatla, A. Pathak, and H. Pucha, "Towards optimizing hadoop provisioning in the cloud." *HotCloud*, vol. 9, no. 12, pp. 28–30, 2009.

[152] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An analysis of traces from a production mapreduce cluster," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE, 2010, pp. 94–103.

[153] A. Ganapathi, "Predicting and optimizing system utilization and performance via statistical machine learning," Ph.D. dissertation, UC Berkeley, 2009.

[154] S. Kadirvel and J. A. Fortes, "Grey-box approach for performance prediction in map-reduce based platforms," in *2012 21st International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2012, pp. 1–9.

[155] K. Morton, M. Balazinska, and D. Grossman, "Paratimer: a progress indicator for mapreduce dags," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010, pp. 507–518.

[156] E. R. Hruschka and M. do Carmo Nicoletti, "Roles played by bayesian networks in machine learning: an empirical investigation," in *Emerging Paradigms in Machine Learning*. Springer, 2013, pp. 75–116.

[157] C. Tunc, S. Hariri, and A. Battou, "A design methodology for developing resilient cloud services," in *Handbook of System Safety and Security*. Elsevier, 2017, pp. 177–197.

[158] K. Mitra, A. Zaslavsky, and C. Åhlund, "Context-aware qoe modelling, measurement, and prediction in mobile computing systems," *IEEE Transactions on Mobile Computing*, vol. 14, no. 5, pp. 920–936, 2013.

[159] P. R. Norvig and S. A. Intelligence, *A modern approach*. Prentice Hall, 2002.

[160] K. Lai and M. Baker, "Measuring bandwidth," in *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, vol. 1. IEEE, 1999, pp. 235–245.

[161] Genie software package. Accessed: 2020-04-15. [Online]. Available: https://www.bayesfusion.com/.

[162] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman, "Aggregation and degradation in jetstream: Streaming analytics in the wide area," in *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, 2014, pp. 275–288.

[163] N. Naik, "Docker container-based big data processing system in multiple clouds for everyone," in *2017 IEEE International Systems Engineering Symposium (ISSE)*. IEEE, 2017, pp. 1–7.

[164] R. Chaudhary, G. S. Aujla, N. Kumar, and J. J. Rodrigues, "Optimized big data management across multi-cloud data centers: Software-defined-network-based analysis," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 118–126, 2018.

[165] Y. Wang, Q. Wang, X. Chen, D. Chen, X. Fang, M. Yin, and N. Zhang, "Containerguard: A real-time attack detection system in container-based big data platform," *IEEE Transactions on Industrial Informatics*, 2020.