Transfer Learning Based Performance Modeling and Effective Storage Management in Big Data Ecosystems

by

Fatimah Alsayoud

B.Sc. in Computer and Information Sciences, King Saud University,

Riyadh, Saudi Arabia, 2008

M.Sc. in Computer Science, Ryerson University, Toronto, Canada, 2014

A dissertation

presented to Ryerson University

in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

in the program of

Computer Science

Toronto, Ontario, Canada, 2020

Author's Declaration For Electronic Submission Of A Dissertation

I hereby declare that I am the sole author of this dissertation. This is a true copy of the dissertation, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this dissertation to other institutions or individuals for the purpose of scholarly research

I further authorize Ryerson University to reproduce this dissertation by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my dissertation may be made electronically available to the public.

TRANSFER LEARNING BASED PERFORMANCE MODELING AND EFFECTIVE STORAGE MANAGEMENT IN BIG DATA ECOSYSTEMS

Fatimah Alsayoud

Doctor of Philosophy

Computer Science

Ryerson University, 2020

Abstract

Big data ecosystems contain a mix of sophisticated hardware storage components to support heterogeneous workloads. Storage components and the workloads interact and affect each other; therefore, their relationship has to consider when modeling workloads or managing storage. Efficient workload modeling guides optimal storage management decisions, and the right decisions help guarantee the workload's needs. The first part of this thesis focuses on workload modeling efficiency, and the second part focuses on cost-effective storage management.

Workload performance modeling is an essential step in management decisions. The standard modeling approach constructs the model based on a historical dataset collected from one set of setups (scenario). The standard modeling approach requires the model to be reconstructed from scratch with every time the setups changes. To address this issue, we propose a cross-scenario modeling approach that improves the workload's performance classification accuracy by up to 78% through adopting the Transfer Learning (TL).

The storage system is the most crucial component of the big data ecosystem, where the workload's execution process starts by fetching data from it and ends by storing data into it. Thus, the workload's performance is directly affected by storage capability. To provide a high I/O performance in the ecosystems, Solid State Drive (SSD) are utilized as a tier or as a cache on big data distributed ecosystems. SSDs have a short lifespan that is affected by data size and the number of writing operations. Balancing performance requirements and SSD's lifespan consumption is never easy, and it's even harder when interacting with a huge amount of data and with heterogeneous I/O patterns. In this thesis, we analysis big data workloads I/O pattern impacts on SSD's lifespan when SSD is used as a tier or as a cache. Then, we design a Hidden Markov Model (HMM) based I/O pattern controller that manages workload placement and guarantees cost-effective storage that enhances the workload performance by up to 60% , and improves SSD's lifespan by up to 40%.

The designed transfer learning modeling approach and the storage management solutions improve workload modeling accuracy, and the quality of the storage management policies while the testing setup changes.

Acknowledgments

(In the name of Allah, the Most Gracious, the Most Merciful)

(If you are grateful I will surely increase you [in favor]) Quran 14:7

First and foremost, I thank God Almighty, who bestowed upon us all the incredible blessings and the ability to learn. All that I achieve is ultimately because of Him.

Ph.D. is an unsurpassed journey that includes lots of unique experiences and challenges. It is my great pleasure to thank those who this dissertation would not be a real fulfillment without them.

I am sincerely thankful to my supervisor, Dr. Ali Miri, for the encouragement, support, collaboration and superb guidance. I will never forget his supportive attitude and his respect for everyone's views. He is a knowledgeable supervisor whom I learned a lot from. I am really fortunate to have worked under his supervision. I also would like to thank the supervisory committee, Dr. Alireza Sadeghian, Dr. Chen Ding and Dr. Anthony Bonato , for their time and effort in providing valuable feedback and questions.

Most importantly, I must express my gratitude to the love, encouragement, and continued support that I have received from my family: parents, siblings, husband and kids. Thanks to my parents for being the limitless source of strength and love. Your words always make me feel better and calm me down, even with the long distance between us. I would like to express my sincere appreciation to my siblings for their love and prayers. My husband, Zuhair ALdally. Words cannot express how I feel and none of this would have been possible without your love, support, patience and encouragement. I owe my deepest gratitude to you for your sacrifice and faith in me. Finally, thanks to my little angels, Zahra and Jawad, for bringing joy to me and being patient when I spend time working while you want to play with me. My daughter Zahra, thanks for the notes you leave on my desk such as "Mom- do not give up." They always make my day and inspire me.

Dedication

To my parents

Contents

1 Introduction **1**

1.1 Challenges in Modeling Multi Testing Setups 2

1.2 Modeling I/O pattern for cost-effective storage controller 3

1.3 Workload-aware SSDs usage approaches 4

1.4 Research Contributions 5

1.5 Thesis Organization 9

2 Background and Related Work **10**

2.1 Big Data Ecosystems 10

2.1.1 MapReduce-based ecosystem : Hadoop 11

2.2 Performance Modeling 13

2.2.1 Multi and Single Scenario Under Test Modeling 14

2.2.2 Machine Learning-based Modeling 17

2.3 Distributed Storage Systems Management 21

2.3.1 Storage Capability and Workload Characteristic 23

2.3.2 Policy-based Efficient Storage Systems 26

3 Cross-Scenario Performance Modeling for Big Data Ecosystems **30**

3.1 Introduction 30

3.1.1 Problem Statement and Motivation 32

3.2 Workload Modeling 33

3.3 Big Data Performance Modeling Challenges 35

3.3.1 Scenario Under Test (SUT) Modeling 36
3.4 Proposed Approach Overview 37
3.4.1 Methodology 38
3.5 Case Studies and Experimental Result 41
3.5.1 Software Versions 41
3.5.2 Benchmark 43
3.5.3 Cloud Services 48
3.6 Chapter Summary 52

4 Transfer Learning for Adaptive Policy-based Storage Management 53

4.1 Policy-based Storage Management 56
4.2 Adaptive Storage Policy Management Framework 57
4.2.1 Transfer Learning Engine: 57
4.2.2 Policy-based Storage Controller 59
4.3 Experimental Evaluation and Results 60
4.3.1 Storage Level Policies 61
4.3.2 Workload Level Policies: 64
4.4 Chapter Summary 67

5 Cache or Tier : An Evaluation of SSD Cost With MapReduce Workloads 69

5.1 Introduction 69
5.2 SSDs Usage Approaches 71
5.2.1 Tiering 72
5.2.2 Caching 73
5.2.3 Compression and SSD 74
5.3 Hadoop performance and SSD efficiency 75
5.4 Experiments 75
5.4.1 Setups 75

5.4.2 Benchmark 76
5.5 Evaluation 77
5.5.1 Performance-efficiency 79
5.5.2 Cost-effective 80
5.6 Chapter Summary 84

6 HMM-Based I/O Modeling to Extend The Lifespans of SSDs 86
6.1 Introduction 86
6.2 Flash-based SSDs and Workloads 89
6.2.1 I/O Patterns 90
6.2.2 Workload Placement Policies based on I/O Patterns 91
6.3 HMM-based MapReduce Workload's I/O Patterns Model 92
6.3.1 The Interaction between Big Data ecosystems and SSD 93
6.4 The Proposed Ecosystem-level SSD Cost-effective Controller 94
6.4.1 Construction HMM for MapReduce Workloads 95
6.5 SSD Controller 99
6.5.1 Workload Placement Policies 100
6.6 Chapter Summary 103

7 Conclusions and Future Work 104
7.1 Future Directions 106

Bibliography 109

Acronyms 124

List of Tables

3.1 Experimental Results: Hadoop Versions Hypothesis 41
3.2 Experimental Results: Benchmarks Hypothesis 45
3.3 Experimental Results: Cloud Service Type Hypothesis 49
3.4 Experimental Results: Cloud provider Hypothesis 50

5.1 Hardware Components Description 76
5.2 Summary of the data sets 77
5.3 Workloads Description [1] 78

6.1 I/O patterns and storage policies 101

List of Figures

1.1 Scenario Under Test changing impacts on management decisions. 4

2.1 Hadoop Ecosystem Components 12

2.2 Traditional Workload Performance Modeling Process [2]. 15

2.3 Model Construction Components. 16

2.4 Traditional ML vs Transfer Learning. 18

3.1 Cross-Scenarios Transfer Performance Modeling 37

3.2 ALOJA dataset example [3]. 38

3.3 Cross-Hadoop.Versions Transfer Hypothesis 44

3.4 Cross-Benchmark Transfer Hypothesis 47

3.5 Cross-Cloud Services Type Transfer Hypothesis 49

3.6 Cross-Cloud Provider Transfer Hypothesis 51

4.1 Adaptive Storage Policy Management Framework 58

4.2 Workload Remote Placement Policy 62

4.3 Workload Local Placement Policy 63

4.4 Compression Policy 65

4.5 Replication Policy 66

4.6 Block Size Policy 67

5.1 Workloads' Throughput 80

5.2 Workloads' average aggregate throughput per workload type 81

5.3 SSD normalized lifespan with SSD approcehes. Taller means longer lifespan. 81

5.4 SSD normalized lifespan per workloads. Bigger means longer lifespan. . . . 84

6.1 SSD Cost-effective Controller 95
6.2 HMM-based MapReduce Workload's I/O patterns 99
6.3 Local and remote storage's SSD lifespan of MapReduce workload's where higher represents longer lifespan. 101
6.4 Placement policies normalized SSD lifespan where a long bar represents a longer lifespan. 102

Chapter 1

Introduction

Today's enterprises employ different types of analytics software and techniques to extract the needed knowledge form a massive amount of data. Typically, the analysis process contains several phases where each phase may execute and interact with different software and techniques. For example, the computing phase may communicate with the computing framework and the resource management agent, and the storing phase may communicate with the file system and the computing framework. Big data ecosystems is an environment that aggregate and control a set of software, technique, and hardware to store and process massive data than visualize and deliver the process results efficiently [4].

Big data ecosystems have become a popular choice for many enterprises, not only because of their ability to process big data but also their capability to serve a variety of workload types with lower time and resource consumption. Different types of workloads have different characteristics, behaviors, Input/Output (I/O) patterns, and resource consumption. They also have different requirements, such as performance level, cost range, and resource needs. To support different workload types and to meet various requirements, the ecosystems contain and interact with complex and heterogeneous software stacks and hardware elements.

As a result of different software and hardware complications, several challenges have emerged in workload modeling and storage management. Typically, the workload model is constructed in accordance with the workload's execution history, mainly the software and the hardware test setups. To keep serving the workload with its need, big data ecosystems need to change Software (SW) and Hardware (HW) setups frequently, as required. Consequently, workload models have to be also reconstructed frequently, which takes time and resources. At the same time, it is important to keep the model up to date as it helps define optimal storage management decisions.

Managing a big data storage system is an urgent task due to the data size and the storage heterogeneity. Usually, the ecosystems interact with multiple storage drive types such as Hard Disk Drive (HDD) and flash-based SSD to store intermediate and final data. In particular, there has been a growing interest in embedding SSDs into big data ecosystems due to their high performance. However, SSDs have a short lifespan and high price tag. Therefore, agile management decisions have to be defined to balance the storage cost and the performance of the workloads.

1.1 Challenges in Modeling Multi Testing Setups

Workload modeling simplifies the connection between the workload factors such as the I/O pattern and the test objective like performance. The traditional modeling approach structures the model based on the data coming from particular test setups including software and hardware setups; we refer to that approach as *Single Scenario Under Test (Single-SUT)*. This approach's model constructs based on the Machine Learning (ML) algorithm; therefore, the approach is inheriting the assumptions of the algorithm, such as requiring training and evaluating datasets to have the same distribution.

A single SUT modeling approach is not the ideal approach for modeling big data ecosystems due to data distribution diversity that generate from modifying one or more values in the ecosystem setups such as changing benchmarks, software versions, or cloud service types. In practice, users typically change the setups to meet individual or application needs. For example, a big data ecosystem may be moved from on-premise to the cloud when more storage capacity is needed. Another example is changing the benchmark measurement tool to analyze different SW elements.

Modeling design is an essential aspect of several critical management decisions such as workload placement and resource planning, and changing the model's design impacts sequences in the management decisions as is represented in Figure 1.1. The model has to stay accurate even when it is design changes to guarantee the management's decision quality. Mostly the model's design changes when the setups change; we refer to setups change as *scenario*. So, when the scenario changes, the model's design changes. Therefore the model accuracy and the decision quality. The reason behind changing the model's design whenever changing the scenario is the relationship between the model construction and scenario associated data distribution. The commonly used approach to address the modeling issue with scenario changing is reconstructing the model from scratch whenever the scenario is changed. This reconstructing can result in a waste of time and resources. Although some work considers studying the impacts of scenario change on generating different data distribution such as [5] and [6], there is no work to consider that impact on designing an accurate model. To address this issue, we propose a *cross-scenario workload modeling approach* that can improve the workloads' performance classification accuracy. The proposed approach adopts the transfer learning concept for reusing models across different but related scenarios.

1.2 Modeling I/O pattern for cost-effective storage controller

Workloads have various I/O access patterns that represent from the workloads I/O characteristics such as the Read/Write (R/W) ratio and sequentiality [7].

While traditional workloads produce only one I/O access pattern, distributed workloads produce at least two different I/O access patterns, one on the local storage and one

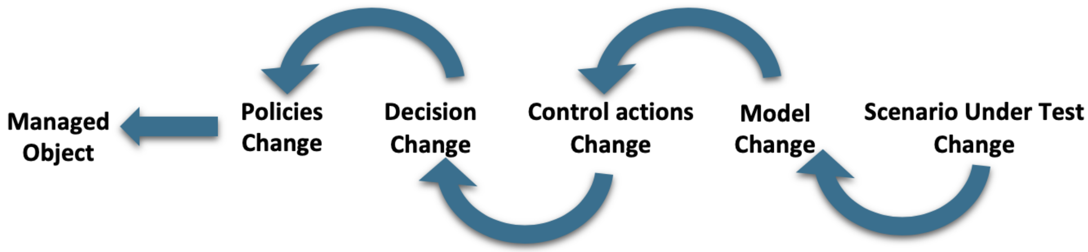

Figure 1.1: Scenario Under Test changing impacts on management decisions.

or more on the distributed (remote) storage. For example, MapReduce based ecosystems such as Hadoop execute workloads on two stages, Map and Reduce. Each of them interacts with the local and the remote storage differently [8] .

There is a robust relationship between the workload I/O access pattern and storage consumption [9]. Therefore, modeling workloads I/O patterns provide valuable information to define optimal storage and cost-efficient policies. Modeling the I/O pattern is an urgent task on the I/O sensitive drives such as SSDs [10]. The sensitivity to the I/O pattern comes from the SSD's physical design that has components with a limited lifespan decreasing after every write operation [11].

In this thesis, we proposed a SSD, cost-effective controller for big data ecosystems. The controller is responsible for maximizing the SSD lifespan on the Hadoop ecosystem through two phases. First, we model MapReduce workload's I/O patterns by employing the HMM. The model's result is used to define workload placement policies that can reduce SSD utilization and improve the SSD lifespan on the Hadoop ecosystem.

1.3 Workload-aware SSDs usage approaches

The SSD's outperforming ability compared with other drive types like HDDs, increases the need to utilize them in the storage system either as a tier [12] [13] or as a cache [14] [15]. As previously discussed, in distributed storage systems, SSDs can be attached to the local or remote storage. Mostly, remote storage uses SSDs as a tier to permanently store the final data produced by the computing phase, and local storage uses SSDs as a cache to temporarily store the intermediate data produced and processed by the computing phase.

The tiering cost is different from caching cost since the first one requires moving data and the other one requires copying data. An analysis of the workload's performance and the storage cost in each approach helps reduce SSD's overall cost.

The cost is not the only difference between the usage approach. Storage reduction techniques such as compression are performed differently in each approach. Therefore, in this thesis, we study the effect of the workloads on SSD's lifespan when SSDs are used as a tier or as a cache in the big data ecosystem. We also analyze the relationship between SSD usage approaches and workload performance. Furthermore, we study the compression impacts on the lifespans and the performance on the tiering and caching SSDs usage approach.

1.4 Research Contributions

Workload performance modeling uses to control many management decisions, such as storage policies. Design an accurate model in a dynamic environment such as a big data ecosystem is a challenging task due to the need for updating the model whenever the environment setups change. The new setups have different data distribution than the old one and thereof require a different model. Several types of research have been conducted to solve the dynamic environment performance modeling issue, but most for them, focus on reconstructing the model, while some define a correlation between models with different setups and data distribution [16]. In this thesis, we will define a new modeling approach that reduces the previously built model to improve a new one.

We have four contributions in two directions: modeling and storage management. In the modeling direction, we have one contribution: transfer learning modeling approach, and in the storage management direction, we have three contributions: adaptive policy-based storage framework, SSDs usage approaches, and HMM-based I/O modeling. The four contributions and their related research questions are listed below.

1. How can we effectively predict workloads performance in big data ecosystems when test setups Scenario Under Test (SUT) are changing? More specifically:

    - Is a single-SUT modeling method satisfying the big data ecosystem's needs?
    - How could we modify the testing setups without constructing a new performance model from scratch?
    - How useful is a previously built performance model for building a new one?
        - Typical testing setups are frequently updated in the big data ecosystem. For example, a big data ecosystem may be moved from on-premise to the cloud when there is a need for more storage. Another example is changing the benchmark measurement tool to analyze different SW elements. Therefore, a single SUT modeling approach does not satisfy the needs of big data ecosystems, and a new model has to be constructed whenever the SUT is changed.

          In this thesis, we will introduce a new modeling approach called cross-scenario transfer that improves performance model accuracy when testing setups are changed. The proposed method design is based on a transfer learning concept that provides knowledge transfer mechanisms between two models to avoid the fresh start overhead. To demonstrate the usefulness of the method, we will examine it in four SUT case studies: benchmark types, software versions, cloud service types, and cloud providers , each with a couple of hypotheses. This contribution has been submitted to appear in [17]. The designed approach helps reduce modeling time and improve model accuracy, which can be used to design reliable storage decisions.

2. How can storage policies, such as the replication factor, help guarantee workload performance?
    - Do we need to redesign storage policies when changing environment setups?
        - We will present an optimal storage policy management framework that can guarantee the required performance even when the scenarios change. The framework is a policy-based controller that transfers knowledge between test cases to improve the policy's quality by adopting the TL method. We study four policies on two levels: storage and workload. At the storage level, we will define local and remote workload placement. At the workload level, we define compression, replication, and block size policies. The designed framework improves policy quality comparing with the stand-alone model. While other work in the literature focus on updating policies as the environment change, our work update policies but with considering of the old policies.
3. What is most cost-efficient SSD usage approach of big data workloads? Is it tiering or caching?
    - How do different workload types affect SSD usage approach efficiency ?
    - How do data reduction techniques such as compression affect the SSD's lifespan with various types of workloads?
        - We will investigate and analyze SSD usage approaches efficiency with nine different MapReduce workloads that contain three different intensities, I/O, Central processing unit (CPU) and Hybrid. We calculate the SSD lifespan in each SSD usage approach with all workload types. We also employ compression on all workloads to see which SSD usage approaches benefit more from space reduction techniques. This contribution has been published in [18].

4. How can the workload's behavior such as I/O pattern be used to control storage costs such as SSD's lifespan?

   – We will design and implement an HMM-based I/O patterns model for MapReduce workloads. The model studies the workload's I/O patterns in both local and remote storage. We then employ policy-based management to define a cost-effective SSD controller. The proposed controller provides workload placement policies to balance workload performance requirements and reduce the SSD's lifespan usage when it is used in the local or remote storage. The idea behind our model is controlling data that is going to store in the SSD before it has arrived into the drive.. This has been published in [19].

We have examined the four contributions in the Hadoop ecosystem that integrated Hadoop File System (HDFS) as a distributed file system and interacted with SSDs as a local and distributed hardware storage. We evaluated the contributions with multiple types of big data workloads.

The result in this thesis have appeared in part in the following publications:

1. "HMM Optimized Modeling of SSD Storage for I/O MapReduce Workloads", In proceedings of the 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON). IEEE, 2019 [19].

2. " SSD: Cache or Tier -An Evaluation of SSD Cost and Efficiency Using MapReduce." In proceedings of the 16th International Conference on Computer Systems and Applications (AICCSA). IEEE , 2019 [18].

3. "Cross-Scenario Performance Modeling for Big Data Ecosystems", to appear in the 22nd International Conference on Human-Computer Interaction (HCI). 19-24 July 2020 [17].

1.5 Thesis Organization

The remainder of this thesis is structured as follows:

Chapter 2 provides the necessary background in big data ecosystems, performance modeling challenges, and distributed storage management. **Chapter 3** presents our transfer learning modeling approach called: cross-scenario transfer to improve performance prediction in big data ecosystems. **Chapter 4** presents a innovative adaptive policy-based storage management fremwork. **Chapter 5** discusses workload performance and SSD lifespan with two SSD usage approaches: tiering and caching. **Chapter 6** presents a policy-based workload placement solution is defined according to the HMM-based workload I/O model. Finally, **Chapter 7** details the conclusions and future work.

Chapter 2

Background and Related Work

This chapter provides the necessary background and related work about big data analytics ecosystems modeling and storage management that are used in this thesis. We review the big data analytics ecosystem's overall characteristics in Section 2.1. Then, in Section 2.2, we discuss the requirements of modeling performance in big data ecosystems when there is a need to deal with a Multi-SUT. We also discuss the abilities of state of the art ML algorithms and methods to meet performance modeling requirements, specifically focusing on the TL method that is used in this thesis. Finally, in Section 2.3, we discuss big data storage management issues and the benefit of applying the modeling approach in the storage management process. The model's results are used to define optimal storage management decisions through adopting policy-based management technology as explained in Subsections 2.3.1 and 2.3.2.

2.1 Big Data Ecosystems

Big Data Ecosystem is an environment that contains various software and hardware to store and process massive amounts of data, serve mixed workloads, and support different requirements. Ecosystem architecture can include infrastructure, analytics tools, data structures, and management [4].

Most of today's organizations have no other option but to deploy big data analytics ecosystems to extract useful information from big data sets in their storage system. These ecosystems bring many benefits into organizations, but also introduce many system management complexities that are a burden to the organization and cause high costs. The management complexity comes from the ecosystem's characteristics. Every ecosystem has many elements that interact with each other in different ways. These ecosystems contain heterogeneous hardware and software, deal with a variety of data structures and serve multiple types of workloads with various requirements. Furthermore, it is common that an SW or HW is to be used by multiple ecosystems. For example, the Hadoop resource management element Yet Another Resource Negotiator (YARN) [20] is used by many other ecosystems such as Spark [21] and Storm [22]. The ecosystems have a multi exchangeable components like processing fremwork and storage file system. In this thesis, we use Hadoop as a big data ecosystem case study. The Hadoop components represent in Figure 2.1, where the components can include, but not limited to, workloads, applications, processing framework, and storage files system. The workloads are traveling across different I/O stack stages, where stages can include application, processing framework, file system, and back-end storage, and therefore the workloads have various I/O pattern on each stage.

The challenge with managing big data ecosystems is that their elements, data and workloads are changing frequently. Consequently, an effective management design requires active monitoring and intelligent modeling. The model is designed to test a particular objective such as performance where the model output is used to define management's decisions.

2.1.1 MapReduce-based ecosystem : Hadoop

Apache Hadoop [23] is a well-known MapReduce-based ecosystem. It has been used widely for big data analytics by many users. Hadoop contains three layers: storage layers HDFS, resource management layer YARN and processing layer MapReduce framework.

HDFS is a Hadoop middle file system layer between the back-end storage layer (storage media) and the other Hadoop layers: YARN, and MapReduce. While YARN manages

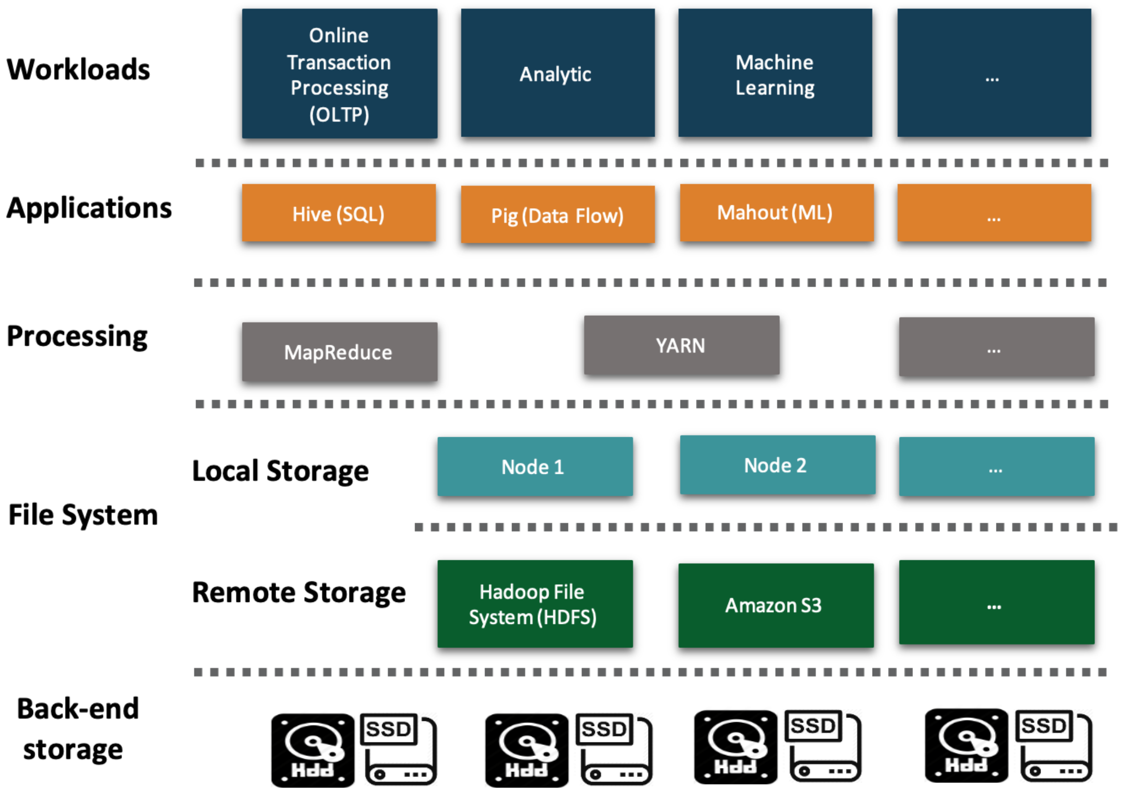

Figure 2.1: Hadoop Ecosystem Components

MapReduce and HDFS workload placement and scheduling policies, HDFS administers data placement policies to ensure a high locality where data is stored close to the process node to avoid network traffic. HDFS has two types of nodes: *Name Node (Master)* and *Data Node (Slave)*. The master node contains information about the slave's meta-data and about the file system tree to define data location on the slaves. Slaves contain the actual storage media where the data is distributed.

The MapReduce framework is a Hadoop processing layer and it has two phases: *Map* and *Reduce*. When a workload is assigned to the framework by YARN, the (input data) load from HDFS splits into *chunks*. The chunks feed the Mappers, and the processing results (intermediate data) write locally or cache into YARN for shuffling. The shuffle phase result process by the Reduce phase to produce the final output and store it in HDFS. The MapReduce workload contains sequence and dependent phases where the output of a phase is the input of another phase. It is mostly executed in a distributed environment and it works based on the idea of moving code, not data. The MapReduce workload execution mechanism is different from the Online Transaction Processing (OLTP) workload's execution mechanism. Subsequently, workload placement policies are different. Performance and storage utilization are also different.

2.2 Performance Modeling

The model-driven [24] approach is one of the most commonly used approaches in software testing. It has been applied successfully to test many objectives such as performance [25], resource utilization [26] and power consumption [27]. The modeling approach provides a foundational methodology to abstract and represents the relationship between a particular actor and the target testing objective. For example, workload performance modeling abstract the relationship between the workloads as an actor and the performance as a target testing objective. In general, the workload performance model is constructed from a sequence process: monitoring the workload's behavior on a participial object , collecting the workload's trace data, applying an analysis technique on the historical data, and finally defining the relationship between workloads and performance using a participial metric [2]. The traditional workload performance modeling process is seen in Figure 2.2.

Performance modeling plays a critical role in many essential management decisions such as workload scheduling and resource planning. Therefore, developing a more accurate performance model is a challenging task in big data ecosystems due to the ecosystem's stack complexity and environmental heterogeneity. *White-box* and *black-box* modeling approaches are typically used to simplify the modeling process in the big data ecosystem. The white-box approach is commonly used when the internal details are essential factors for decision making like considering configuration values for software configuration tuning [28] or configuration optimization [29]. On the other hand, the black-box approach applies when the internal details are not important. What is important is the relationship between the input values and the testing objective. It is the most common approach to modeling a big data ecosystem as it provides the needed information without dealing with complicated details. For example, the black-box approach is used in the Hadoop ecosystem to gain insight information, establish features correlation [30], define the correlation between workload profile and workload execution cost [31], and enhance the ecosystem's deployment in cloud services [32].

2.2.1 Multi and Single Scenario Under Test Modeling

Although there are a number of work in the literature that attempt to address performance modeling in the big data ecosystem, most of these work focuses on single SUT modeling. Single SUT means that the tested and the evaluated data sets are collected from the same testing scenario and therefore have the same distribution. The testing scenario can include but is not limited to the software version, the hardware capability, the deployment setups or the testing tool. Single SUT modeling refers to building a model based on the data

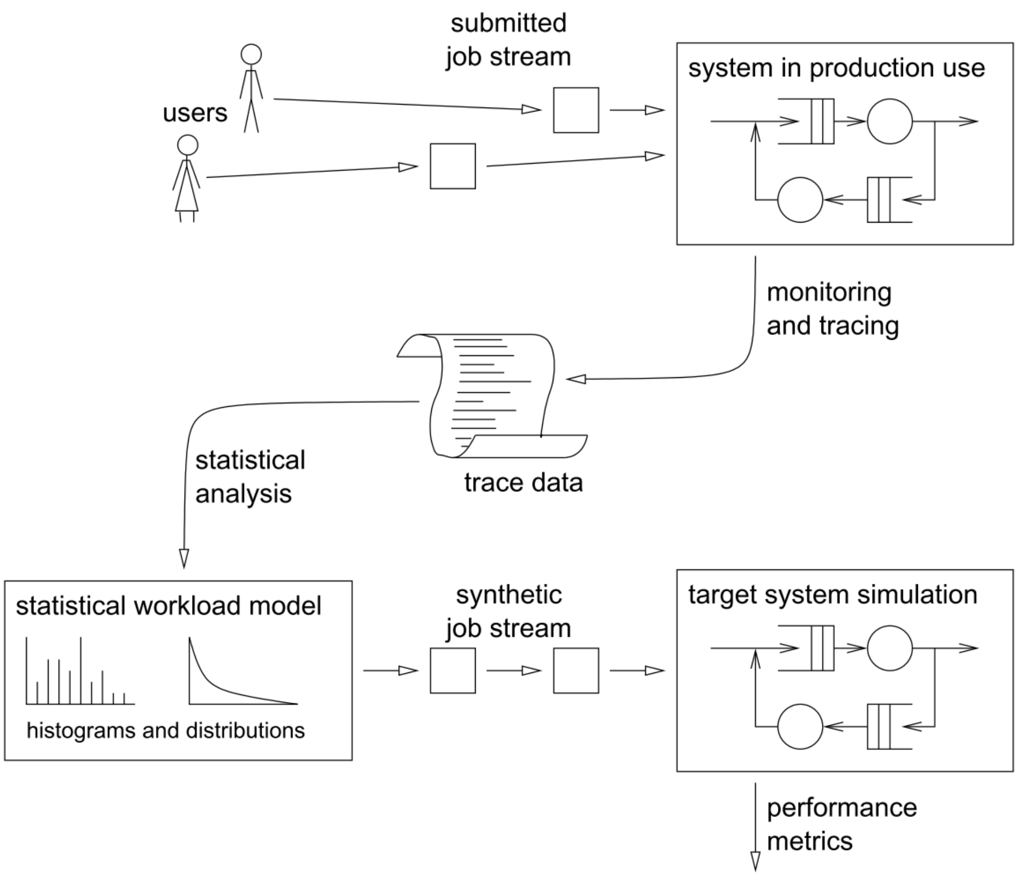

Figure 2.2: Traditional Workload Performance Modeling Process [2].

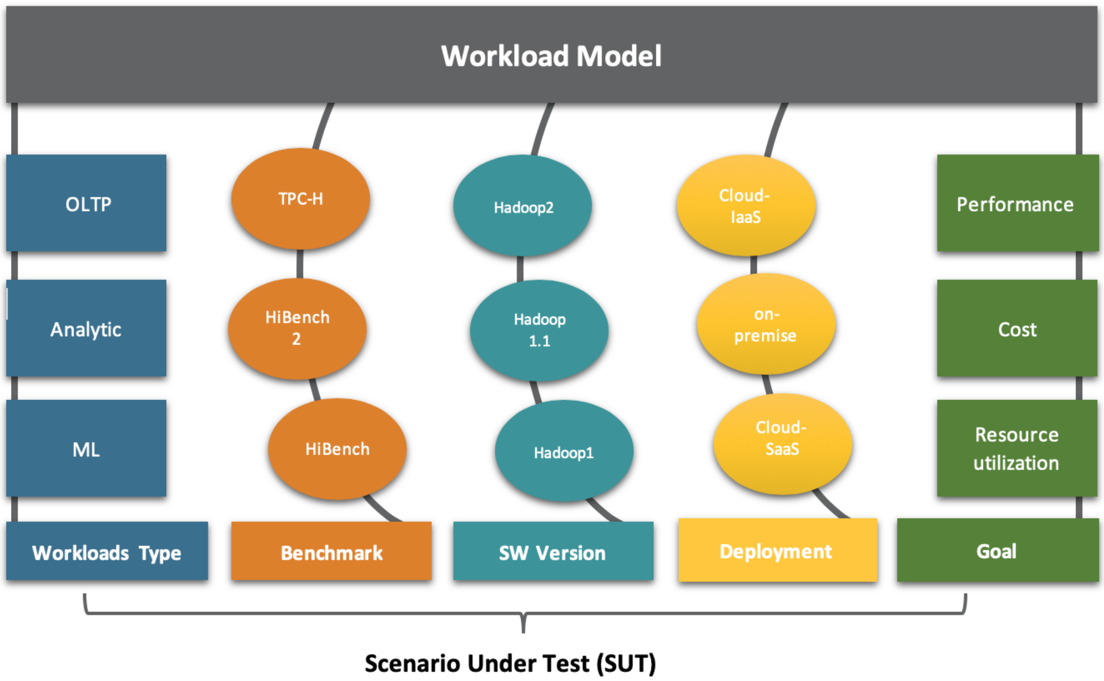

Figure 2.3: Model Construction Components.

collected from a single test setup. SUT is not the ideal method for modeling the big data ecosystem. This is because big data ecosystems have a lot of possible testing scenarios that are generated from changing one or more elements in the testing environment such as changing benchmarks, software versions or cloud deployment solutions. For example, we can construct a performance model in the Hadoop ecosystem based on monitoring the workload's execution behavior while using Hadoop version1.3.0, deploying Hadoop on Microsoft Azure cloud computing with Infrastructure as a Service (IaaS), and examining the HiBench benchmark. An overview of single SUT modeling's construction components is represented in Figure 2.3.

The single SUT method requires building many models from scratch to design an accurate model for a big data ecosystem. Each SUT has a unique testing setup corresponding to a different data distribution. Typically, big data workloads, such as Hadoop Map Reduce workloads, have diverse behaviors and patterns. Therefore, they generate different data distributions [5]. There is a need to design a multi SUTs modeling method to support the diverse distribution in the big data ecosystem.

In general, the multi tests are either constructed from the same dataset or from different datasets. Each of the construction approaches has its own challenges and methods. However, dealing with multi tests that are constructed from different datasets is more challenging due to the data and test diversity.

Multi tests from the same dataset mostly refer to multiple testing or multiplicity, which is a well-known issue since 1979 [33]. Multiplicity refers to any simultaneous testing issue that deals with more than one hypothesis for the same dataset and therefore the same distribution. It is addressed explicitly in clinical trials [34] where there is a need to build more than one hypothesis around the same data to define different relationships. Multiple testing is an active research area and a good number of methods are defined to solve it such as hierarchical test procedures [35].

Multiplicity-based methods are not suitable for multi test constructs from different datasets or multi test constructs having different distributions such as on multi SUTs, which is the focus of this thesis. For this reason, we explore the current research and methods that can address multi tests with dissimilar data or distributions in the next section.

2.2.2 Machine Learning-based Modeling

There are several ML algorithms available in the literature that address modeling in a big data ecosystem. For example, [36] employ *Supported Vector Machine (SVM)* to design a resource utilization model in the Hadoop ecosystem and [37] use tree-based regression to model Hadoop configuration parameters.

Most common ML algorithms such as SVM and regression trees are not sufficient for modeling multi SUTs in a big data ecosystem. This is because, as discussed previously, SUTs generate different distribution, and most of the ML algorithms assume trained and validated data to have the same distribution, which means that the algorithms are not working as accurate when dealing with a different distribution [38]. Consequently, there

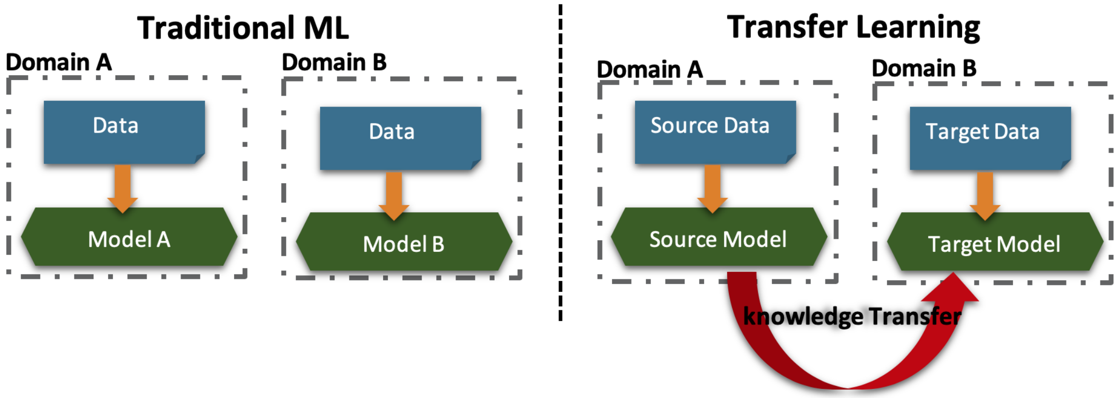

Figure 2.4: Traditional ML vs Transfer Learning.

is a need for a performance modeling method that guarantees model accuracy even when the setups and data distribution are changing. Typically, a ML algorithm constructs one model for each data distribution; the challenge is on connecting the different models to improve the accuracy of one of them.

To address the above modeling issue, in this thesis, we introduce a cross-SUTs concept that improves big data ecosystem modeling when the dataset distribution is changed. We discuss and then utilize the TL method in the Subsections below.

2.2.2.1 Transfer Learning for Cross-scenarios Modeling

The main issue with applying ML on dynamic environment workload modeling is the need to reconstruct the model whenever the data distribution is changing. It is common to have multiple models for the same environment but with different setups. Much of the literature on multiple workload modeling methods are focusing on the models that are constructed based on similar data distribution. For example, [39] design a cooperation method that aggregates the historical data model with the simulated data model to improve the overall model accuracy, where the data on both models have the same distribution. Another example is the selection method [40] that choice one model among others based on their accuracy where are models use the same data but different learning models like classification and clustering or a different ML algorithm. Both cooperation and selection modeling methods are not suitable for our goal of solving the modeling issue when setups and data distribution are changing.

To date, a number of studies have attempted to evaluate multiple workload models with different data distribution. Performance-anomaly detection [41] dynamically adjusting the systems parameter changes between the models to reduce resource utilization in cloud computing. Applying the anomaly detection method to improve modeling accuracy in a big data ecosystem is debatable since it typical to have noisy data is common in the ecosystem. Another multiple model studies have been proposed to explain the relationship between models; for example, [42] interstage the similarity between models and [16] study the correlation between feature selection and model accuracy. This work has a different goal of improving model accuracy when having multiple models; therefore, we adopt TL.

The TL method relaxes the traditional ML algorithm's assumption that restrains the trained and validated data to have the same distribution [38]. TL assumes that the trained dataset and the validated dataset have different distributions or different features space but have related knowledge to be shared. A well known TL real-life example is how people use the knowledge of driving a car to learn how to use a bicycle. Although the car's physical components and driving rules are different from a bicycle, there is some common knowledge between riding a bicycle and driving a car.

The TL method can be applied to almost all learning model types like classification, regression and clustering. It allows the learning model to be constructed accurately, even with the dataset (domain) from different features or distributions, or with another leaning model (task). There are several types of TL methods that support learning models with various components and contexts.

The two main components in TL methods are task and domain, and the two players are source and target. Typically, the TL method has a source task, target task, source domain and target domain, where Knowledge transfers from the sources to targets. Three TL method types can be defined based on the components and players [38]:

- *Inductive Transfer Learning:* specify when Source and Target Domains are the **same** and Source and Target Tasks are **different** but related.
- *Unsupervised Transfer Learning:* specify when Source and Target Domains are **different** but related and Source and Target Tasks are **different** but related.
- *Transductive Transfer Learning:* specify when Source and Target Domains are **different** but related and Source and Target Tasks are the **same**.

TL method types can also be categorized in respect of the contexts [38] into :

- *Instances Transfer:* specify when Source Domain label knowledge is used to define Target Domain label.
- *Feature-representations Transfer:* specify when Source Domain feature representation is discovered to increase similarity with the Target Domain.
- *Parameters Transfer:* specify when Source Domain parameters share with the Target Domain and are used to improve the Target Domain learning start-up performance.
- *Relational knowledge Transfer:* specify when Source Domain generated knowledge maps into the Target Domain for performance improvement.

In this thesis, we adopt transductive and parameters transfer learning methods. We select the transductive method because we are examining different domains (SUTs), but there are related as we are examining domains from the same cases study. For example, in the software version case study, the Source domain can be a particular version, and the Target domain is another version. They are related, but different. Another reason for using the transductive method is because we are focusing only on one task type (classification) that we are using for both Source and Target tasks. TL methods have different goals, such as improving model accuracy and reducing training data size. The goal of this thesis is to improve the modeling accuracy; thereof, we use parameter transfer that helps improve the learning task start-up performance.

Although the concept of transfer learning has gained attention since 1995 [38], it is most commonly used by only particular research areas like image recognition, and text classification more than other areas. Over time, considerable literature has developed around applying TL for other research topics like dynamic resource provisioning [43], cloud configuration [44] , software defect prediction [45], and software effort estimation [46]. More recently, a few works of literature have renewed interest in involving TL on improving performance modeling, but design TL method based on defining source and target domain when configuration parameters are changing [47]. To the best of our knowledge. TL methods have not been applied to improve performance modeling when testing setups are changing.

Regardless of the power of TL techniques, the outcome of methods is not always positive. The **negative transfer** issue raises when the result of the TL techniques gives lower performance than the stand-alone ML model. Some factors need to be studied to find out how math a model can benefit from TL methods, and if the TL does not add any value to the model. Some research has been studying the reason for the negative transfer from a different perspective where it could cause by data, features, or task. A high-quality data helps TL techniques to extract and transfer meaningful information and, therefore, [48] designed a framework using conditional Kolmogorov complexity to measure data relatedness between source and target tasks. Besides, understand the relationships of the parameters can help make the right decision about applying TL [49].

2.3 Distributed Storage Systems Management

An accurate modeling design is the first step to define reliable storage management decisions and policies. The model that is designed for storage management has to consider different factors for performance testing. In addition to the workload's diversity and the ecosystem's characteristics in performance modeling, the storage management model has to include storage related information such as back-end storage capability and the storage distribution mechanism. In this section, we will discuss storage related modeling factors in big data ecosystems.

In a big data ecosystem, it is essential to give the modeling process a lot of attention when designing a reliable storage system such as defining the ecosystem storage-related configuration values [50], the scheduling model [51], the workloads placement rules [52] and the storage services applying policies. Examples of storage services are deduplication and compression [53], replication [54] and encryption [55].

Modeling and managing are very related concepts, where changing the model effect changing the management decisions as shown in Figure 1.1, changing the model impacts management decisions. For example, if the target object is a storage drive and the desired objective is storage cost, a model defines based on a participial historical dataset, the control actions such as cost evolution metrics are change, and the decisions such as capacity usage threshold is change the storage-related policies change such as X type of workloads place in drive type X.

The distributed storage systems could be managed at hardware or software levels. The software level can include the I/O stack stages such as application, host, and hypervisor. Each of those I/O stage has a unique modeling process and different management policies due to the stage components interacting and the stage requirements. For example, to support availability, replication factor policies have to be defined at the software level while Redundant Array of Independent Disks (RAID) has to be defined at the hardware level. Most of the storage management aspects such as monitoring, scheduling, protection, resource allocation and workload placement are managed differently based on the controlling location. In this thesis, we focus on controlling workload placement in big data ecosystems. We specifically study the Hadoop ecosystem.

Big data workload behavior interacts differently with the storage system than the simple operation workloads like read and write that straightforwardly store or retrieve data. Big data workloads are affected by the ecosystem computing processing procedure that is typically done in two stages locally and remotely. The data is fetched from the remote storage to feed the computing phases where the first phase divide the work and store an intermediate data temperately in local storage. The locally stored set of data is retrieved and enters the computing phase again for last processing phase. The final result data of the last processing phase is stored in the remote storage. The workloads have different behaviors and I/O patterns in both local and remote stages [7] and therefore have different performance [25].

The workload performance is not only affected by the workload's behavior, it is also affected by the back-end storage capability. Different types of storage drives provide different capabilities. For example, in general, SSD drives outperform HDD drives. Most of the big data ecosystems contain heterogeneous back-end storage to serve different needs. However, it is a challenge to manage heterogeneous drives that have a different hardware architecture design and software structural design. It is important to consider the back-end storage capability when managing workload performance.

In the next subsection, we discuss the two direct associations between storage capability and workload characteristic in big data ecosystems. We specifically focus on SSD drive storage. A detailed big data workloads and SSDs storage associations investigation is provided in Chapter 5. The following subsections explore how to apply policy-based management technology to balance between maximizing workload performance and minimizing storage cost. Our proposed policy-based model is explained in Chapter 6.

2.3.1 Storage Capability and Workload Characteristic

In this subsection, we explore the bidirectional communication between big data workloads and back-end storage. We examine how workload characteristic affects storage cost and how storage efficiency affects the workload's performance. Specifically, we are focusing on SSDs when they are added into big data ecosystems as a local or remote storage. We also discuss the SSD's placement strategies to achieve a participial objective from a big data ecosystem perspective.

Adding SSDs into the storage system is one of the most efficient ways to improve I/O intensive workload performance and reduce energy consumption in big data ecosystems [56]. Usually, SSD drives outperform HDD drives due to the SSD and HDD hardware architecture design differentiation [13].

The most durable component in most of the SSD drives is the solid-state memory with NAND chips. NAND chips provide SSD drives with a high Input/Output Operations Per Second (IOPS) compared with the spinning platter HW design in HDD drives that require extra time for executing read and write operations during the rotation and seeking process. Also, NAND chips have lower disk failure rates than HDDs because they don't have moving physical parts that get damaged over time. On the other hand, SSDs have a high cost coming from their price and the drive's short lifespan, which are represent the SSDs fundamental cost metrics, dollar-per-GB and lifespan consumption respectively. The limited lifespan of SSDs comes from the NAND flash memory. The NAND flash has a fixed number of pages on each data block which are typically 64 or 128 pages. The flash page helps SSD to speed up random access read and write operations by up to 100 times than HDD. However, SSD writing operations cannot commit until there are enough clean pages; therefore, SSDs are very sensitive to the number of writes. The cleaning process on the page that holds called Program/Erase (P/E) cycle. Due to the P/E process, typically, whenever the number of write operations goes over the suggested limit by the vendor, the SSD reliability drops, and the error rate increases. Managing the SSD lifespan is important for getting the most out of SSD capability and reducing storage costs.

Managing the storage system's cost is a critical issue in the big data ecosystem since the amount of data generated worldwide and processing by big data ecosystems is doubling almost every two years. Storage represents a small portion of the overall for most organization budget [57]. Consequently, there is an urgent need to balance between maximizing performance and minimizing storage expenditure. However, controlling storage spending and workloads performing in big data ecosystems is not a straightforward task. The ecosystems generate and process a lot of data, utilize heterogeneous back-end storage and serve multiple types of workloads.

A single SSD drive can provide a different performance when it serves workloads with distinct I/O access patterns. For example, workloads with random I/O access or high read ratios perform better than high write ratio workloads [58]. The SSD consumption cost is also different from one workload to another, where the cost is associated with the workload's I/O pattern. For example, high write ratio workloads require a high number of P/E cycles, which exhausts the SSD drive and reduces the drive's lifespan [11].

Embedding SSD into distributed storage systems has been actively study by both academia and industry. The SSD drives can be appended into a system on various locations such as host, hypervisor, application or ecosystem and can be used in different approaches such as caching or tiering.

Big data workloads interact with the storage system in two main locations, locally and remotely. There is a trend to use it for the temporarily intermediate data in the local storage before moving it into the remote storage. Commonly, SSD drives use as cache on local storage and as a tier on the remote storage.

In case of big data workload, the local I/O pattern is different than the remote I/O pattern. For example, typically, local data has a higher write ratio than remote data as it is accessed frequently by the computing phases until completing the computing process.

Previous research have shown that utilizing SSD drives can improve big data workloads' performance but improvement depending on the workload differently based on the workload type and SSD usage approach. In the Hadoop ecosystem, several studies investigated the performance improvement of a variety of MapReduce workloads when introducing

SSD into the ecosystem as a tier [59], [60], [61] [12] or as a cache [8]. To examine the difference in SSD utilization usage approaches impact on MapReduce workloads, [62] and [61] compared the performance improvement when applying SSD as a tier and as a cache.

There are a number of work in the literature on utlization of SSDs in big data ecosystems that deal with the question of how SSD usage impacts workload performance. For example, [62] and [10] consider evaluating the SSD's cost. The impact of the big data workload on the SSD lifespan cost remains unanswered at present. In this thesis, we investigate the impact of utilizing SSD's tiering and caching approaches on MapReduce performance, and we also examine how each type of workload utilizes the SSD lifespan in each usage approach.

2.3.2 Policy-based Efficient Storage Systems

In this subsection, we discuss the impact of bidirectional relationship between the storage and workload on defining optimal storage management decisions in big data ecosystems. The relationship is used on designing a policy-based management framework.

Storage system management is an active research area especially in data-intensive environments such as big data ecosystems which try to address questions about workload scheduling, configuration optimizing, resource provisioning, data and workload placement, tier controlling and storage services applying. Our focus is on workload placement strategies, tiering control and storage services such as compression and replication.

Management is a broad concept that includes a variety of methods and techniques that trying to meet the desired objective such as performance, capacity, cost, availability and reliability. Policy-based technology is one of the most well-known management technologies that is used widely in many areas including storage system management [63]. The policy-based structure typical use the If-This-Then-That (IFTTT) service. Although the structure seems simple, the policy-based technology has a lot of complicated policy forms such as domain, goal, conflict, decision point, mapping and activation forms [64]. The activation formula can be defined as *proactive* or *reactive*. The policy actions are activated in the reactive form only when the predefined thresholds are reached such as CPU usage or storage available capacity. These type of policy action are popular in the cloud environment where the user selects a preferable service class, and the user all corresponding workloads are executing on predefine infrastructure setups like storage derive type and for a limited amount of usage like participial capacity. Reactive policies do not consider the workload's characteristic and pattern. On the other hand, proactive policies define based on a collected monitoring data and therefore, the policies action activated according to the workload's characteristic.

Service Level Agreement (SLA) cannot be guaranteed on reactive policies since the actions are only activated after recording an observation, while SLA are more likely to be achieved in proactive policies which predict workload needs in advance [65]. The robust of proactive policies are associated with the quality of the historic data and the accuracy of the corresponding model. Therefore, in this thesis, we focus on designing an accurate model as addressed in Section 2.2.

Although several studies have used policy-based technology successfully in storage management such as for configuring Storage Area Network (SAN)s [66], automating storage allocation [52] and controlling a distributed object [67], most of these work consider a single knowledge abstraction level. For example, if a workload required X resource than placement rule is Y. In this example, the workload's needs are abstract but not the storage capabilities. The Software-Defined Storage (SDS) [68] concept solves the knowledge abstraction isolation issue. SDS provides a framework to the abstract workload's I/O characteristic and requirements, and storage capabilities to define dynamic storage related policies. SDS shifts storage management from the hardware level to software level with a comprehensive overview of workloads and storage.

In this thesis, we adopt the SDS concept to design storage policies in the big data ecosystem. From workload knowledge abstraction, we examine the workload's I/O pattern and from storage knowledge abstraction, we consider drive types and usage approaches. We study the workload placement and compression service in two storage usage approaches: *caching* and *tiering*. The policies define for two objective workload performance or storage cost which is measured by SSD's lifespan consumption.

A heterogeneous storage system contains drives with various capabilities. Typically, storage drives are used as a tier but some drives that have a high capability such as SSDs are a good candidate to use as a cache storage. Therefore, SSDs can be managed, when they use as a cache, on hardware [15] and software [14] levels. The SSD's lifespan is utilized differently in each SSDs usage approach. Tier storage usually contains a large data size for a long time while the cache storage deals with quite a smaller data size and holds it for a shorter time. In addition, the data has to replicate in the tiering approach for the fault tolerance requirement which causes high storage consumption compared with the caching approach. In a big data ecosystem, the workload has two types of data: *intermediate* and *final*. Although intermediate data seems to utilize storage lighter than the final result, studies show that some workloads have heavy intermediate data [8].

There are some other factors that need to be considered when measuring storage usage in addition to data size such as the I/O pattern. Intermediate data usually has a high write ratio which shortens the SSD's life. It is hard to define the best SSD usage approach for big data workloads. Understanding the benefits and drawbacks of SSD's usage approach for a practical workload helps design the right workload placement policies, which can help in maximizing SSD's lifespan. Although there is some of literatures on managing SSD's lifespan through reducing writing traffic [69] [70] and applying data reduction techniques like deduplication [71], [72] or compression [73], most of the SSD lifespan controlling solutions are defined at the hardware level. Managing SSD's lifespan at the hardware level allows these policies to control what the storage receives, but software level management can define policies to control what sent into the storage in first place. There is relatively a small body of literature that is concerned with managing SSD from a software perspective such as [10] which defines I/O placement policies into SSDs based on SSDs cost, and [8] that define SSD cache allocation in the Hadoop ecosystem. However, none of them consider SSD's lifespan consumption and SSD's usage approaches which we cover in this thesis.

The SSD's lifespan and the corresponding workload performance is affected by storage services such as compression and replication. Compression technology is usually applied to reduce data size but at the same time, it can have a positive or negative impact on the workload's performance. This compression technology is particularly useful in minimizing lifespan usage at a hardware level [73] and software level when using SSDs as a cache [14]. Studies have also shown that there is a clear association between the workload's I/O pattern and SSD's lifespan [74]. In this thesis, we examine the compression impact on both SSDs usage approach when executing various I/O patterns in a big data ecosystem.

In this chapter, we discussed the related background and related work, where In Section 2.1, we explained why big data ecosystem management is a challenging task and how defining an accurate modeling design can improve the quality of management decisions. In Section 2.2, we reviewed the limitation of current ML driven modeling that relies on a single SUT method and we showed the need for a new modeling method that fits the big data ecosystem's requirements. In Section 2.3, we discussed the management challenges for one of the most critical components in the big data ecosystem, the storage system. Then, we explored the relationship between the workload's performance and storage capability, and between storage cost and workload characteristics in Subsection 2.3.1. This is used as a guide to define policy-based storage management decisions as discussed in Subsection 2.3.2.

Chapter 3

Cross-Scenario Performance Modeling for Big Data Ecosystems

3.1 Introduction

Big data ecosystems have become one of the main element in today's information technology environments. These ecosystems support big data sets and provide a variety of execution methods to meet system workload requirements. Big data ecosystems contain heterogeneous hardware and software, and they support a variety of data and workloads.

Designing optimal management policies and actions for big data ecosystems requires active monitoring and intelligent modeling. The model deign to test a particular objective like performance. Modeling for performance testing is one of the most successful management analyzing approaches. It can be used to measure the performance of a specific system object or a specific executing workload. In both cases, the performance testing design is impacted by the characteristics of the running workloads. For example, HDD delivers its best performance when it serves sequential access workloads and not random access workloads. Another example is that the Hadoop ecosystem performs better with analytic workloads than OLTP workloads.

Workload performance modeling provides an approach to examine performance on a particular SUT, where scenario can include cloud service, software version, and benchmark type. In general, the model result is a significant input element on many system decisions such as resource allocation. Therefore, it is crucial to design an accurate workload model as the performance test results reliability level is in line with the model accuracy.

Designing an accurate workload model for big data ecosystems is a challenging task due to ecosystem complexities and heterogeneity. There are several case studies in big data ecosystems [75] such as software version that has many possible scenarios like version 1.0 and version 2.0. Another example is when considering a cloud service type as a case study, scenarios can include IaaS and Software as a Service (SaaS).

Different scenarios can result dissimilar workload distributions. However, most of ML algorithms assume the train and evaluate data to have a similar distribution [38], which lead most of the current modeling approaches to use the same assumption. This assumption does not fit with big data ecosystem characteristics where the workload's distribution is changed with many possible scenarios. Constructing a model for each scenario from scratch is time-consuming and resource intensive. A similar distribution assumption does not work well in many real-life cases. For example, in computer vision, there is a need to recognize numbers either coming from handwritten data or from a picture where they have dissimilar distributions.

Some of ML related methods such as TL developed to deal with the distribution similarity constraint. TL provides a method to transfer knowledge between domains with a different distribution or dissimilar feature space to avoid building a fresh model every time the scenario is changed and to improve the model's accuracy. It is a well-used method on computer vision and natural language processing literature. More detail about TL is explained in Section 2.2.2. In this work, we will use TL to improve the performance model in a big data ecosystem.

3.1.1 Problem Statement and Motivation

The need for an accurate performance model remains even when the scenario or the executing workload is changed in a big data ecosystem. Designing an accurate model for a big data ecosystem such as Hadoop while considering scenario and workloads changing is a challenging task. Although there are some work in Hadoop dealing with performance modeling such as [30], [31] and [32], they are focus on a single SUT. Only few work have considered multi SUT. For example, [5] provide a comprehensive analysis of how the workload behaviour, characteristic and distribution changes with scenarios change, and [76] designed a map task scheduling model for multi cloud service under test. However, none of these work consider improving the performance model for a particular SUT by utilizing another SUT model.

In practice, users typically change the setups to meet individual or application needs. For example, a big data ecosystem may be moved from on-premise to the cloud when there is a need for more storage. Another example is changing the benchmark measurement tool to analyze different SW elements. Although scenarios usually change frequently on a big data ecosystem, the scenarios modification factors have not been considered on the big data performance modeling yet.

In this chapter, we investigate the accuracy of a big data ecosystem performance model with our proposed *cross-scenario transfer* approach. This approach builds a performance model based on a particular SUT ($Scenario_{src}$) and then transfers the source knowledge into another SUT ($Scenario_{tgt}$) to improve the target model's accuracy. A cross-scenario transfer approach adopts the inclusion method (multi scenarios) instead of the isolation (single scenario) method that is used by most existing performance modeling approaches. The inclusion method relaxes the sensitivity between model accuracy and the SUT characteristic. We demonstrate the approach with four scenarios: benchmarks, cloud service types, cloud providers, and Hadoop versions each with a couple of hypotheses. An example of a hypothesis in the Hadoop versions case study is a cross-major version where we examine how effective is applying our approach between two different major versions. Our experimental results show noticeable model accuracy improvement on the $Scenario_{tgt}$ with the proposed approach.

The chapter is organized as follows. Sections 3.2 and 3.3 give a background of workload performance modeling challenges. The proposed approach overview is presented in Section 3.4. The evaluated case studies and the experimental result are discussed in Section 3.5. Finally, chapter summary are presented in Section 3.6.

3.2 Workload Modeling

In general, modeling provides a foundational methodology to abstract and represent a particular aspect or relationship. Workload modeling establishes a connection between the workload characterization and the desired testing object. It helps to track how the workload and the corresponding testing object are changing. There are several possible algorithms for workload modeling such as prediction, evolution, optimization and simulation. The algorithm is selected based on the model's objective. It is important to select the right design factors and define an accurate workload model. This is because many critical management decisions are using it as one of their fundamental elements.

Today's big data ecosystems serve a variety of workload types such as OLTP, Decision Support System (DSS), analytical and Machine Learning workloads. Each type has unique attributes and characterization. Moreover, the workload's pattern, behaviour and distributions change with the execution environment. Workload behaviours are very sensitive to execution environment components, setups and capability.

Workload modeling provides a method to simplify the relationship between workload characterization and behaviours with the desired testing object for a particular testing environment [25]. The testing object is the workload attributes that the model is designed to test it, such as performance, cost and resource utilization. The object measurement metric defined during the model construction is based on the final objective. For example, performance can be measured based on the workload's execution time or the throughput. Another essential aspect of workload modeling is the testing environment that affects workload behaviour and testing object values. In general, the model design is based on data from an environment with an aggregation of SWs and HWs. However, usually only one of the environmental elements is used to define the testing factors. For instance, in the application performance model, the application represents the testing environment and performance represents the testing object. The workload model for performance testing investigates the relationship between workloads and the corresponding performance.

Each aspect of the workload model should be designed and selected carefully since the accuracy of the design affects the accuracy of many management decisions and actions. The model can be used for descriptive, predictive and prescriptive analytics where the analytics output, for example, produces performance insight or predicts resource provisioning. The workload model can also be used for simulating workloads [77] and evaluating a system configuration [78]. Indeed, the workload-aware concept becomes a common aspect of different management architecture.

Workloads have different behaviours and patterns that change based on many factors like workload structure and the testing environment. For example, the behaviour of database workloads is different than the ML workloads. The last one is more complicated, requiring more resources and taking more time than the first one. The challenge occurs when a particular environment serves both types of workloads which is a normal situation in today's applications. The workload-aware concept is adopted on the system to serve each workload with its need, and define the management decision and action differently for each workload.

3.3 Big Data Performance Modeling Challenges

Modeling big data workloads for performance testing or in short performance modeling is a challenging task due to the ecosystem's complexity and the variability of the workload. It is challenging to design an accurate model for a big data ecosystem that has many interacting components and for workloads with very wide distributions. Traditional performance modeling assumes that data comes from a single SUT and has the same distribution. Both assumptions do not meet the need of big data ecosystems. Big data ecosystems have a complex architecture with several stages, multi-configuration parameters and multi SW elements. These ecosystems contain many highly interactive stages such as computing, resource management and a distributed file system which control how the workload is executed, how many resources are allocated to it and where it should be placed, respectively. Each of the controlling decisions impacts the workload's overall performance. Furthermore, the ecosystems have a massive amount of possible configuration parameters. Each of them has multiple possible values and each of the values affects the performance differently.

The SW elements in big data ecosystems are dependent on each other and some of the elements interact with elements from other ecosystems. For example, the Hadoop resource management element YARN [20] is used by many other systems such as Spark [21] and Storm [22]. Also, the HDFS is used by OpenStack Swift and Amazon S3 [23]. The SW characteristics and the interaction have an implication on workload behaviour and therefore workload performance.

Each aspect of the big data ecosystem architecture impacts the performance of the workloads and can cause a change in workload distributions. It is hard to keep track of how each aspect of the ecosystem impacts performance. As written by [75] "we do not know much about real-life use cases of big data systems at all."

Two well-known modeling methods are used for simplifying big data ecosystem complexity: white box and black box approaches. White box applies when the internal details are essential factors for decision making like considering configuration values for configuration tuning [28] or configuration optimization [29]. In contrast, the black box method does not consider the internal ecosystem details, and it is used by most work that focuses on the testing output instead of ecosystem details. Most of the black box methods and many of the white box approaches follow the original modeling assumption of using a single SUT with the same distribution. Such assumptions would require building a considerable number of models from scratch to cover the possible big data scenarios. The proposed approach in this work benefits from the pre-built models on constructing a new one to improve model accuracy, and save model construction time and resources.

3.3.1 Scenario Under Test (SUT) Modeling

Most performance modeling approaches rely on a single SUT where data is collected from the same environment setups. For example, if the desired test object is an application, then the model is built based on collecting or simulating data from a particular application. Usually, the model built for a particular application cannot work as accurately for another application.

The performance modeling single SUT requirement is coming from the algorithm's restriction used on the model. The most used algorithms in performance modeling are analytic and ML algorithms. Both types of algorithms require the trained data and the evaluated data to have the same distributions and feature space. To guarantee those requirements, the performance model expected data needs to come from a single SUT.

The issue is that most of today's case studies deal with changing the original scenario for different reasons. The model's accuracy cannot be guaranteed when any of the SUT factors are changed. For this reason, in most cases, the whole model has to be reconstructed when any change happens. A large number of models are needed to cover all of the possible scenarios.

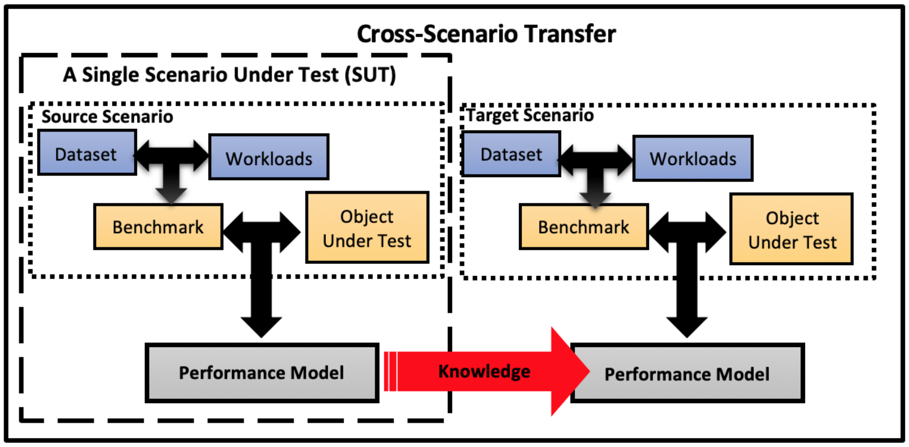

Figure 3.1: Cross-Scenarios Transfer Performance Modeling

Even though a single SUT approach gets great attention from both industrial and academic communities, it has several limitations such as lack of supporting diverse scenarios. It requires contracting many models and isolating the built model from the other related models. It consumes time and resources, and is sensitive to workload distributions. A single SUT limitation motivates us to define the cross-scenario approach that can support multi-scenarios in big data ecosystems and improve performance model accuracy.

3.4 Proposed Approach Overview

The proposed approach overview is illustrated in Figure 3.1 and listed below:

- To provide the cross-scenarios transfer approach with the correct data, both the Source $Scenario_{src}$ and Target $Scenario_{tgt}$ have to follow the same preparation process. For example, the process includes normalizing numeric data, coding categorical data and classifying the target output.

- Once the dataset is prepared, the $Scenario_{src}$ and the $Scenario_{tgt}$ are defined according to the desired hypothesis. For example, in the cloud service type case study, if the hypothesis is a cross-service.type, then the source and target scenario are two different cloud service types. Details about the source and target scenarios definition

id_exec	id_cl	bench	exe_time	start_time	end_time	net	disk	bench_type	maps
2	3	terasort	472.000	2014-08-27 13:43:22	2014-08-27 13:51:14	ETH	HDD	HiBench	8

iosf	replicas	iofilebuf	compression	blk_size	# data nodes	VM_cores	VM_ram	validated	version
10	1	65536	None	64	9	10	128	1	1

Figure 3.2: ALOJA dataset example [3].

in each case study are specified in Section 3.5.

- The Cross-Scenarios transfer approach is applied for each formulated hypothesis. The approach contains three steps: build the source model according to $Scenario_{src}$, build the target model according to $Scenario_{tgt}$, and build the cross-scenarios transfer model according to the built source model and the $Scenario_{tgt}$.

- Source and Target models are constructed with the Multi-Layer Perceptron (MLP) method.

- The built source model knowledge is used to build a cross-scenarios transfer model for the $Scenario_{tgt}$.

- The accuracy of results for the target (stand-alone) model and the target (cross-scenarios transfer) are analyzed for each hypothesis.

- We execute each hypothesis three times to calculate the average result of stand-alone and transfer learning models.

- To study the impact of sample size on the model's accuracy, we examined each hypothesis with six sample size 50, 150, 250, 350, 450, and 500.

3.4.1 Methodology

Transfer learning is defined to relax distribution similarity constraints on trained and the evaluated data. TL assumes that the trained dataset and the validated dataset have different but related distributions. The TL method can be applied to almost all of the learning models such as classification, regression, and clustering. It provides a way to transfer knowledge between different learning tasks or between different domains. There are two types of domains: *Source* and *Target*. The Source domain is where the knowledge transfers from and the Target domain is where the knowledge transfers to.

The Source and the Target domain in TL methods are having different data distribution. In this thesis, we define the domains as scenarios, where the $Scenario_{src}$ have different data distribution than the $Scenario_{tgt}$. The scenario represents a set of setup values, and modifying one or more of those values can cause changing the data distribution [79]. In our experiment we study how TL can help improve the $Scenario_{tgt}$ accuracy when changing only one value such as software version while the other setup values are unchanging, In other words, the $Scenario_{src}$ and the $Scenario_{tgt}$ have the same setup values except for one value which refers as cause study. For example, in the software version case study, the difference between the scenarios is the software version.

Our TL approach is implemented on Python 3.6.9, and the environment setups are Keras 2.2.0, Tensorflow 1.1.0, and Numpy 1.17.3.

We utilize MLP model for the stand-alone model and the $Scenario_{src}$ model in the TL approach. To evaluate our approach by measuring $Scenario_{tgt}$ accuracy with our approach and with the stand-alone model. We select the accuracy metric for classification because we test balance data that have an equal number of samples belonging to each classification class.

$$Accuracy = \frac{number\ of\ correct\ predictions}{the\ total\ number\ of\ predictions\ made} \tag{3.1}$$

Multilayer Perceptron MLP is a feedforward artificial neural network (ANN) class that has multiple layers of perceptrons. The minimum number of layers is three: input, hidden, and output. The hidden layers numbers of units are defined according to the experiment dataset, in our implementation, we have four hidden layers and an output layer with four nodes each represent a workload performance level.

MLP has an activation function, and cost function where the activation function converts the entering signal (neuron) into an output signal (response) and the cost function compute the error that quantifies the gap between the prediction values and the expected value. For activation function we use $softmax$ function

Dataset:

The examined dataset is Hadoop execution trace-data provided by the ALOJA open-access dataset [3] within over 33,147 Hadoop executions and 87 MB. The data has 196 features such as workload type, benchmark type, Hadoop versions, cloud service types, and cloud providers, and it also contains information about the job execution time and resource utilization such as CPU, memory, and disk. An example of the dataset represents in Figure 3.2. The data set is prepared in R, where preparation includes feature selection, normalization, and one-hot encoding for the output variable (label). Each case study represents a feature in the dataset, and the source and target scenarios values are the feature values. The dataset will be filtered to create a case study sub-dataset that contains only samples with the desired source and target scenarios values than the sub-dataset will split into a source and target datasets. For example, in software versions cases study when we examined Hadoop major versions, the sub-dataset has only the rows with Hadoop major versions than the sub-dataset split into source scenario data set that has Hadoop version 1 where $Scenario_{src} = Hadoop1$ and a target scenario data set that has Hadoop version 2 where $Scenario_{tgt} = Hadoop2$. The data is a set of other features, and the data out is the execution time. Due to filtration and splitting processes, some cases studied suffer from unbalanced labels or limitations in the number of samples. Therefore we apply a label balancing method (R library(ROSE))and a stratified sampling with an 80% training size.

Table 3.1: Experimental Results: Hadoop Versions Hypothesis

Hypothesis	(Hadoop-1.0.3→	Hadoop-1.2.1)	(Hadoop 1→	Hadoop 2)	(Hadoop-1.2.1→	Hadoop-2.7.1)
Sample size	Stand-alone	TL	Stand-alone	TL	Stand-alone	TL
50	0.236± 0.043	0.371± 0.100	0.270± 0.040	0.391± 0.017	0.243± 0.070	0.278±0.063
150	0.310± 0.035	0.482± 0.122	0.310± 0.029	0.393± 0.017	0.344± 0.025	0.506±0.122
250	0.413± 0.057	0.485± 0.067	0.397± 0.041	0.509± 0.069	0.412± 0.022	0.573±0.097
350	0.232± 0.111	0.486± 0.113	0.445± 0.034	0.567± 0.151	0.451± 0.020	0.569±0.077
450	0.175± 0.041	0.388± 0.128	0.468± 0.017	0.572± 0.053	0.507± 0.018	0.636±0.076
500	0.120± 0.090	0.312± 0.227	0.500± 0.075	0.676± 0.055	0.526± 0.032	0.667±0.025

3.5 Case Studies and Experimental Result

In order to evaluate the proposed approach, four different case studies are defined as Hadoop software versions, benchmark types, cloud provides and cloud service types. Each case study contains real-life scenarios that are used to determine the examined cross-scenario transfer.

3.5.1 Software Versions

Commercial and open-source software companies produce new software versions either to add new features or fix the software bugs. This can happen at any stage of the software life cycle. The frequency of producing new versions is in accordance with the software design model. In general, open-source software, such as big data ecosystems, release new minor and major versions more frequently than commercial software.

Versions have different configurations and therefore, the trace data that is produced is different in products. The trace-based method is the most used workload modeling method. Following how versions change is not a straightforward task since several components are involved in the process. Although some work try to simplify how versions are changed, for example model SW versions [80], it is still hard to predict how versions change and impact trace data behaviour [81].

Traditional workloads modeling assumes that trace data comes from a single Version

Under Test (VUT) since trace data behaviours are relative to the software version. Such an assumption requires the model to rebuild as often as the software versions release.

In this section, we investigate how the new VUT performance model's accuracy improves when transferring knowledge from another VUT model. We evaluated three cross-Hadoop.Version transfer hypotheses: Cross-Major.Version transfer, Cross-Minor.Version transfer and Cross-Version transfer.

Cross-Major.Version Transfer:

We examine the model accuracy of Hadoop 2 when it uses knowledge from the Hadoop 1 designed model where $Scenario_{tgt} = Hadoop2$ and $Scenario_{src} = Hadoop1$. There is a significant component change between the two major versions. For example, while the MapReduce framework takes the case of resource management on Hadoop1, Hadoop2 has a new resource management stage called YARN. Although changing the ecosystem architecture impacts trace data distribution and therefore performance mode, the version feature remains quite the same. Our results demonstrated that the Target scenario has by up to $\sim 45\%$ more improvement than the stand-alone model.

Cross-Minor.Version Transfer:

Each software major version has at least two minor versions. While the new major version contains the significant change, minor ones are usually developed to fix bugs or make a slight modification. This hypothesis tests two Hadoop1 minor versions 1.2.1 and 1.0.3 where $Scenario_{tgt} = Hadoop{-}1.2.1$ and $Scenario_{src} = Hadoop{-}1.0.3$. Cross-Minor.Version hypothesis designed to evaluate the accuracy improvement when transferring knowledge between a pre-built model and a new one from the same major version but a different minor version. Minor version can learn from each other better than the major versions since there are no huge changes between minor versions; most changes done to fix bugs. The experiment showed that this hypothesis had an improvement between $17\% - 160\%$, where the accuracy improves as the sample size grows.

Cross-Versions Transfer: The Cross-Minor.Version hypothesis tests model accuracy when both major and minor versions change. In Cross-Versions hypothesis, $Scenario_{tgt} = Hadoop - 1.2.1$ and $Scenario_{src} = Hadoop - Hadoop - 2.7.1$. This hypothesis had the same accuracy improvement as the major one reflected by up to $\sim 47\%$.

Figure 3.3 shows the stand-alone and the transfer model accuracy of all cross-Hadoop. Versions transfer hypothesis. Our TL approach represents by the dot lines are always outperform the stand-alone (dashed line). They are outperforming means better accuracy where both stand-alone and TL model accuracy calculated by using the 3.4.1 accuracy formula. For example, the top left Figure is the results of cross-minor versions transfer where the y-axis represents the accuracy of the stand-alone, and the TL mode and x-axis represent the sample sizes. In the cross-minor versions transfer, we notice a rose steadily on both models in the small size. The TL is stable between 150 and 350 samples, while the stand-alone model is reaching a peak when the sample size is 250. From then onwards, the accuracy of the stand-alone model declined steadily. In another direction, both cross-major and cross-version transfers have increased in general from the smaller to bigger size.

Table 3.1 contains the execution result of our TL approach and the Ml stand-alone methods with the three hypothesis minor, major, and versions transfer. The source and target data are split based on the value in the Hadoop version feature. The cross-miner transfer source data has only observations with Hadoop 1.0.3 , while target data has Hadoop 1.2.1. Similarly, the source and target data in the cross-major and cross-versions. The input data on both source and target data are the workloads execution details, and the output is the workload execution time. The data in the table represent the accuracy of the TL and the ML model with six sample sizes.

3.5.2 Benchmark

Computer architecture contains many hardware and software components with different performances. Testing and comparing the performance of the components is an essential step for many further management actions. However, it is not easy to compare the

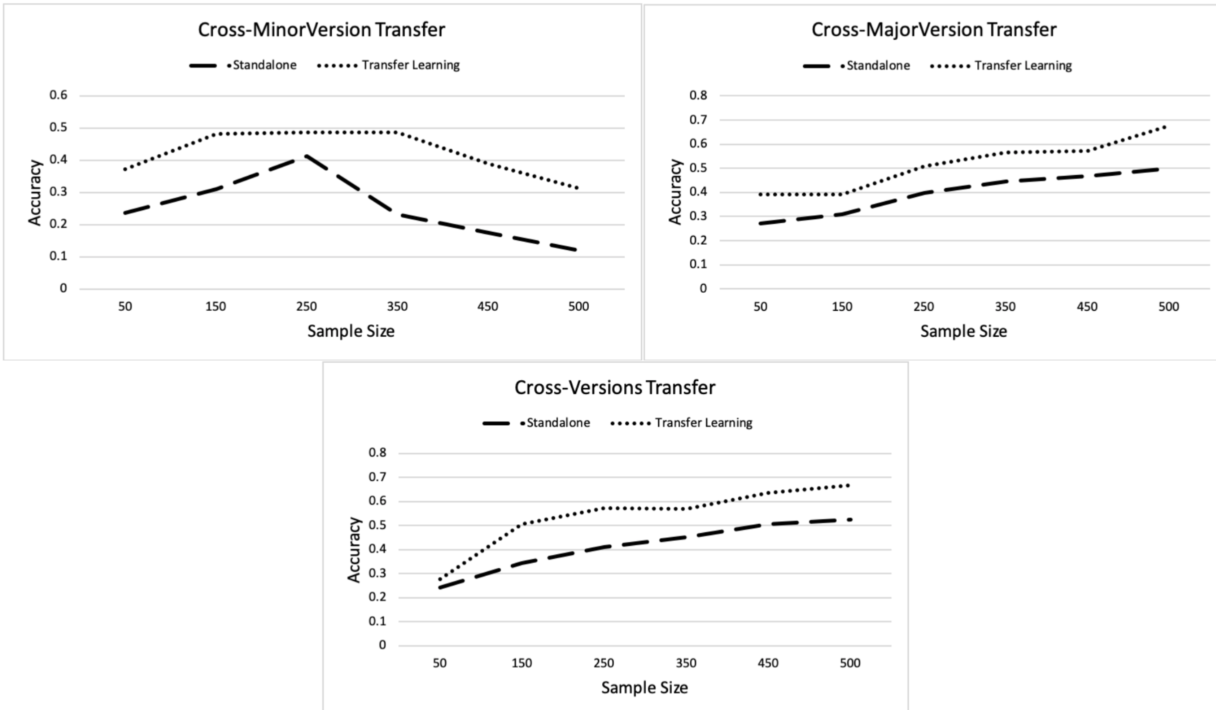

Figure 3.3: Cross-Hadoop.Versions Transfer Hypothesis

performance of various components.

A workload benchmark is a measurement tool that provides a standard set of workloads with the corresponding dataset to test a particular goal like performance under a particular object like software. So, for example, vendors use a benchmark to compare their new product with the previous one. Besides, to provide testing standard suite, benchmarks can be used for simulation [82] and validating particular assumptions.

The traditional benchmarking methods which to test a particular object like SW or HW under a single benchmark [83]. Despite the single benchmark usefulness, it has certain limitations like lack of supporting diversity objects, a small number of benchmarks, and diversity accuracy.

Benchmarking Big Data ecosystems:

In addition to the benchmarking challenges seen in Section 3.3.1, big data ecosystem benchmarking has other issues like scalability, service types variety, use cases diversity, distributed environments, ecosystem complexity and the ecosystem repeatedly changing

Table 3.2: Experimental Results: Benchmarks Hypothesis

Hypothesis	(TPC-H→	TPCH-hive)	(HiBench→	Hadoop-examples)	(HiBench→	HiBench2)
Sample size	Stand-alone	TL	Stand-alone	TL	Stand-alone	TL
50	0.344± 0.027	0.587± 0.310	0.213± 0.040	0.280± 0.143	0.262± 0.007	0.467±0.093
150	0.425± 0.060	0.462± 0.032	0.332± 0.015	0.379± 0.075	0.293± 0.043	0.380±0.149
250	0.500± 0.023	0.595± 0.100	0.429± 0.029	0.566± 0.081	0.335± 0.015	0.416±0.075
350	0.463± 0.060	0.535± 0.085	0.461± 0.011	0.577± 0.117	0.415± 0.050	0.476±0.096
450	0.564± 0.039	0.615± 0.011	0.451± 0.014	0.539± 0.030	0.393± 0.022	0.437±0.088
500	0.533± 0.027	0.585± 0.040	0.482± 0.054	0.603± 0.060	0.418± 0.028	0.507±0.056

[75]. Benchmarking design has many aspects that can be modified to meet one or more of the ecosystem characteristics such as the number of supported applications, workload types, a number of workloads, data set size, a number of workloads implementations and simulation support [84].

Defining the big data workloads model based on a single BUT cannot guarantee accuracy when the big data ecosystem case study changes. In this chapter, we use Hadoop as a case study for big data ecosystems. Hadoop has a complex software stack and different configuration setups that affect workload behavior.

Several benchmarks are designed to meet the needs of the Hadoop ecosystem like workload evaluation [85] and workload simulation [84]. Furthermore, some benchmarks are designed for Hadoop-related Application Programming Interface (API)s such as HBase and Hive. These benchmarks can evaluate performance by measuring job running time or throughput that is calculated at the computing stage MapReduce. These benchmark can also evaluate resource utilization through measuring CPU, memory and I/O that is calculated at the HDFS.

It should be clear that the benchmarking of the big data ecosystem is a challenging task, and select the best benchmark suite is also difficult. For these reasons, we utilize previously built models with a particular benchmark to design a new model with another benchmark, through the cross-benchmark modeling approach.

In this section, we investigate how the new BUT performance model's accuracy improves when transferring knowledge from another BUT. We evaluated three cross-Benchmark transfer hypotheses: Cross-Benchmark.Type transfer, Cross-Benchmark-Version transfer and Cross-Benchmark.Execution.API transfer.

Cross-Benchmark.Type Transfer: Commonly, multi benchmarks are used to measure the same object. Trace data generated from different benchmarks may have a different distribution even when running on the same object. In Cross-Benchmark.Type hypothesis, we examine HiBench [85] and Hadoop-Examples benchmarks. So, $Scenario_{tgt} =$ HiBench and $Scenario_{src} =$Hadoop-Examples. The HiBench benchmark contains the same fundamental read and write workloads on Hadoop-Examples and it also contains some ML workloads. This hypothesis focuses on evaluating model accuracy improvement when it reuses the built model from another benchmark type. Our results of the correlational analysis show 32% improvement in the transfer model.

Cross-Benchmark.Version Transfer: Usually, by the time, the same benchmark changes the tested workloads or the corresponding dataset to fix detected bugs, improve the benchmark or meet new requirements. In Cross-Benchmark.Version hypothesis, we examine two versions of the HiBench benchmark, HiBench and HiBench2 where $Scenario_{tgt} = HiBench$ and $Scenario_{src} = HiBench2$. Our results demonstrated that the Target scenario has by up to $\sim 78\%$ more improvement than the stand-alone model .

Cross-Execution.API Transfer One of the most popular cases in big data ecosystems is executing the same workload from a different API. For example, a fundamental read workload can be executed directly in Hadoop or from the associated API such as Hive. In cross-Execution.API hypothesis, we examine model accuracy when it builds based on data from the TPC-H benchmark [86] only and when it uses knowledge from the TPCH-hive benchmark. $Scenario_{tgt} = TPC-H$ and $Scenario_{src} = TPCH-hive$. The result shows that the transfer model has a 70% higher accuracy than the stand-alone model.

Figure 3.4 shows the stand-alone and the transfer model accuracy of all cross-Benchmark type transfer hypothesis. Our approach provides better accuracy than the stand-alone ML

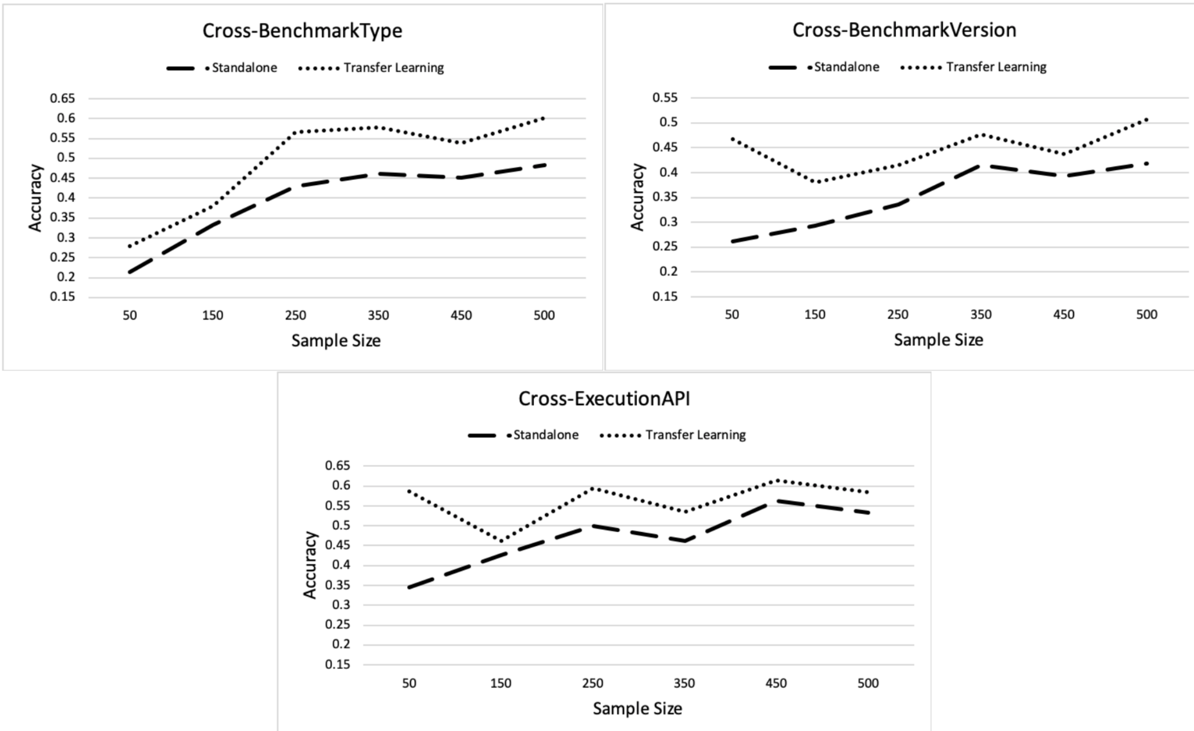

Figure 3.4: Cross-Benchmark Transfer Hypothesis

method across different sample sizes. Both methods in the cross-type transfer rose steadily during the first two sample sizes than the TL accuracy rose sharply from 150 until 250. However, after 350, the TL accuracy declined but recovered right away. On the other hand, both methods fluctuated in cross-execution API.

Table 3.2 contains the execution result of our TL approach and the Ml stand-alone methods with the three hypotheses cross-type, cross-version, and cross-execution API. The original data split into a source and target data depending on the benchmark feature. In the cross-execution API transfer, the source and target have different benchmark types of values where the source data has information only about TPC-H, and the target data has only information about TPCH-hive. Similarly, the source and target data in the cross-types and cross-benchmark version. The input data on both source and target data are the workloads execution details, and the output is the workload execution time. The data in the table represent the accuracy of the TL and the ML model with six sample sizes.

3.5.3 Cloud Services

Cloud computing becomes an essential environment for individuals and big enterprises. At the same time, cloud is a competitive platform of where new service types are produced every day. It is very common that the user moves from one to another to satisfy their needs. Migration in the cloud can pose its own challenges. For example, a user collects data and builds models based on their environment to achieve a particular objective. In most cases, when moving to a new environment, the objectives remain the same. However, the collected data and the built model cannot be reused in the new environment. It is time-consuming and causes resource exhaustion. The proposed approach saves time, saves resources and improves accuracy.

The user moves to cloud or uses it as a solution for many things like getting higher performance or having more capacity. Our work tries to take advantage of the previously built model to design a new one. In reality, users may move from cloud to on-premise or the opposite to meet their desired needs. The performance model is needed in both environments. The question is, does the user need to start a fresh performance model whenever moving from one solution to another? If yes, how long should the user wait to start collecting new data for building a new model?

Next, we will investigate how different cloud service types and cloud provider performance model accuracy improves when transferring knowledge from another service type or provider model. We also will evaluate two Cross-Cloud.Service transfer scenarios: Cross-Service.Type transfer and Cross-Providers transfer.

Cross-Service.Type Transfer:

The big data ecosystem can deploy one of the cloud computing service type models such as IaaS or SaaS, or it can deploy on-premise infrastructure. Each of the models has a unique environment and setup, and therefore, each produces different workload distributions.

Table 3.3: Experimental Results: Cloud Service Type Hypothesis

Hypothesis	(IaaS→	SaaS)	(IaaS→	On-Premise)
Sample size	Stand-alone	TL	Stand-alone	TL
50	0.190± 0.01	0.269± 0.017	0.227± 0.010	0.266± 0.100
150	0.231± 0.012	0.285± 0.084	0.316± 0.011	0.335± 0.076
250	0.288± 0.02	0.358± 0.112	0.316± 0.009	0.409± 0.154
350	0.272± 0.022	0.299± 0.1056	0.376± 0.039	0.413± -0.069
450	0.294± 0.013	0.382± 0.023	0.448± 0.004	0.507± 0.150
500	0.278± 0.020	0.319± 0.030	0.392± 0.026	0.450± 0.034

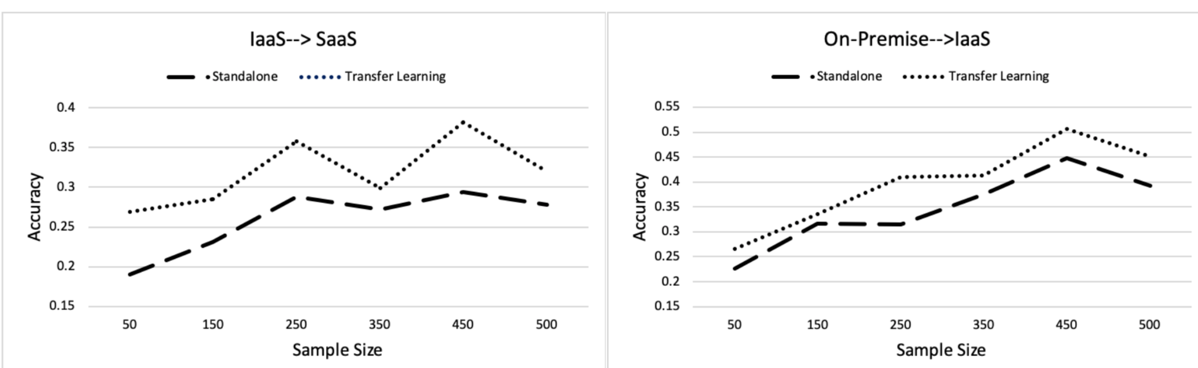

Figure 3.5: Cross-Cloud Services Type Transfer Hypothesis

Table 3.4: Experimental Results: Cloud provider Hypothesis

Hypothesis	(Minerva→	Azure)	(Minerva→	Rackspace)	(Rackspace→	Azure)
Sample ratio	Stand-alone	TL	Stand-alone	TL	Stand-alone	TL
50	0.269± 0.023	0.399± 0.017	0.294± 0.030	0.396± 0.013	0.261± 0.037	0.329±0.043
150	0.361± 0.027	0.433± 0.067	0.406± 0.051	0.580± 0.140	0.365± 0.029	0.518±0.173
250	0.407± 0.016	0.473± 0.050	0.456± 0.040	0.552± 0.097	0.392± 0.030	0.597±0.126
350	0.477± 0.029	0.600± 0.154	0.488± 0.013	0.584± 0.109	0.484± 0.039	0.690±0.042
450	0.532± 0.023	0.671± 0.079	0.512± 0.014	0.588± 0.064	0.564± 0.013	0.704±0.063
500	0.591± 0.028	0.658± 0.040	0.569± 0.035	0.664± 0.042	0.588± 0.022	0.716±0.069

It is common for the user to move from one solution to another. The question is, can the user use the built model on one solution to help improve the accuracy of another one $?$ The result shows by up to $\sim 30\%$ improvement in the target model when the $Scenario_{tgt} = On - Premise$ and the $Scenario_{src} = SaaS$ and by up to $\sim 42\%$ when the $Scenario_{tgt} = IaaS$ and the $Scenario_{src} = SaaS$. It is clear that there is a benefit of using the previously built model in the cloud.

Figure 3.5 and Table3.3 shows the stand-alone and the transfer model accuracy of the two cases in the cross-service.type transfer. In the IaaS–SaaS cases the source data has only information about IaaS, and the target data has SaaS information. The stand-alone accuracy, in that case, rose steadily until 250 when the accuracy began to fall. At the same time, the TL accuracy fluctuated but stayed more than the stand-alone accuracy. In the On-Premise– IaaS case, the accuracy of the TL and the stand-alone model rose, but both have slightly fluctuated.

Cross-Providers Transfer:

There are hundreds of cloud providers in the market and all have different environments, loads and policies. The provider environment setup such as resource capability and amount of accept load impacts the behaviour and distribution of the workload. In Cross-Providers hypothesis, we evaluated the model with data generated from three providers: Azure, Rackspace and Minerva. The result shows by up to $\sim 48\%$, $\sim 43\%$

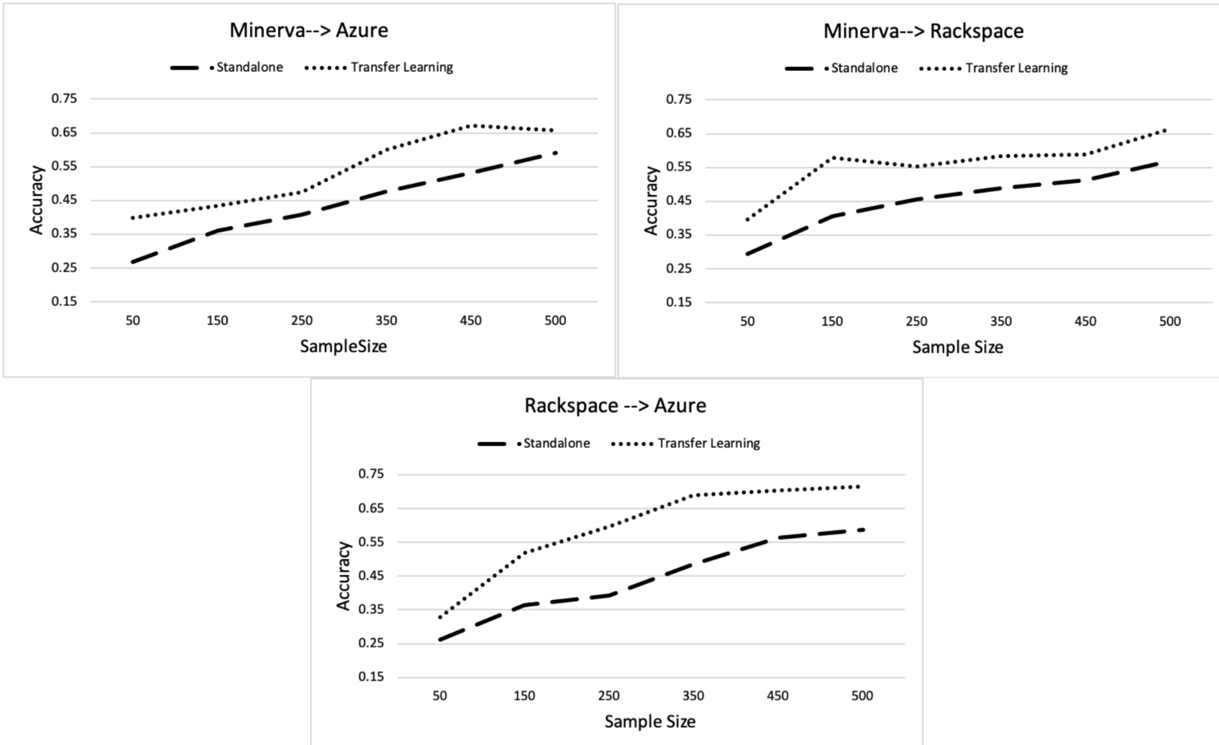

Figure 3.6: Cross-Cloud Provider Transfer Hypothesis

and $\sim 53\%$ where $Scenario_{tgt}$ =Minerva and $Scenario_{src}$ =Azure , $Scenario_{tgt}$ =Minerva and $Scenario_{src}$ =Rackspace, and $Scenario_{tgt}$ =Rackspace and $Scenario_{src}$ =Azure respectively.

Figure 3.6 shows the stand-alone and the transfer model accuracy of cross-providers transfer between three loud provides Azure, Rackspace, and Minerva. Our TL approach represents by the dot lines are always outperform the stand-alone (dashed line) on all provide type. In the top left figure, the dashed line means executing the Azure model without benefiting from the previously built Minerva model, while the dotted line means creating the Azure model with help from the Minerva model. In all provide models, we notice a rose steadily on both TL and stand-alone models.

Table 3.4 contains the execution result of our TL approach and the Ml stand-alone methods for three Cloud providers. The source and the target data in the TL method are defined based on the provider feature. For example, in Minerva –Azure cases, the source data has only observations with Minerva, while target data has Azure; similarly, the two other cases. The input data on both source and target data are the workloads execution details, and the output is the workload execution time. The data in the table represent the accuracy of the TL and the ML model with six sample sizes.

3.6 Chapter Summary

Improving performance model accuracy is an essential step in big data ecosystems because it plays a critical role in many important management decisions and actions. Performance modeling's original assumption relies on a signal SUT that does not fit the big data ecosystem characteristic and causes low model accuracy. It is time-consuming and resource intensive. The proposed approach in this chapter provides the cross-scenarios transfer approach that allows knowledge sharing between performance models with different but related scenarios. Transferring knowledge between models helps in improving target model accuracy. Big data ecosystems have many possible scenarios related to SW or HW changes. In this chapter, we explored the approach with four case studies: benchmarks, cloud service types, and Hadoop versions. Each of the examined case studies have several scenarios. We examined the cross-scenarios transfer approach between the same case study scenarios to evaluate the target model's accuracy. Our result shows that the target performance model in the proposed approach is more accurate than SUT approach. Such an improvement would make a significant change in management design decisions and actions and therefore, on the overall performance.

Chapter 4

Transfer Learning for Adaptive Policy-based Storage Management

Storage systems are of particular interest to researchers from both industry and academia. They have to be able to support different data types, serving heterogeneous workloads and containing complex components. These systems can generate valuable knowledge and control a lot of critical decisions, making their management a key concern for enterprises [87].

Storage systems include heterogeneous components and they provide several services to meet various workloads' needs. Their components include HW elements, such as drives and SW elements, such as file system. Their services include providing availability, space reduction, data splitting and workload placement. Their components and services can be managed from the perspective of HW or SW. For example, to provide availability service, HWs can be used to control RAIDs and SWs can be used to control replication factor. The SW storage management is more fixable than its HW counterpart and in most of the cases, the storage elements have to be entirely redesigned to provide new services. Today's applications change their requirements repeatedly, and to guarantee meeting those requirements, applications adopt SW-based storage management.

Policy-based management can represent one of the most effective controlling mechanisms used to manage storage at both HW and SW levels [63]. In theses types of management schemes, policies are adopted using software-defined technologies like SDS [88] to provide a dynamic management environment where the controller sets storage service policies based on the workloads behavior and requirements. An example of policy used may look as: IF $<$*Condition: Workload I/O pattern* $=$ *High read* $>$ AND $<$ *Goal*$_{desired}$ $=$ performance $> THEN <$ *PolicyAction*: Compressions $=$ False $>$.

Policy action can be formulated to be proactive or reactive. The above example is a proactive policy where the action is defined based on predicting the workload's characteristics. On the other hand, the action on reactive policies is activated when a predefined threshold is reached. For example, IF $<$*Condition: Storage available capacity* $=$ *X* $>$ AND $<$ *Goal*$_{desired}$ $=$ capacity $> THEN <$ *PolicyAction*: Compressions $=$ TRUE $>$ is a reactive policy. Typically, SLA cannot be guaranteed on reactive policies, since the actions are only activated after recording an observation, while an SLA is more likely to be achieved in proactive policies that predict workload needs an advance [65]. It is impotent to be able to design robust proactive policies that can achieve desired goals. However, most of the proactive design-based policy approaches rely on historical data collected from the same scenario, where scenario includes the HW and SW testing setups, such as storage drive type and software version.

Components in a big data ecosystem may have many setup values, where each particular value can represent a different scenario. An example of a scenario is testing software version X with cloud service type Y, where X and Y are the setup values. Both of these setup values may be frequently changing to satisfy the user's and application's needs. However, most of the storage management proactive policies are designed based on only one scenario. A single SUT approach is not the optimal option to design new storage policies in changing environment.

Therefore, in this thesis, we introduce a new policy management framework that address this issue by leveraging previously designed policies. We will show that our framework can improve the quality of policies with different storage policies: workloads placement, compression, replication and block size.

Guaranteeing workload performance is associated with the quality of policies where policy actions such as workload placement location, disabling/enabling compression, replication factor and block size are key aspects of meeting a desired performance. The ecosystems interact with distributed storage either, locally or remotely. Storage and workload characteristics in each of these locations are different impacting the performance. In this thesis, we will design separated placement policies for each location. Compression is a powerful data reduction technology, but it may result in degradation of performance. So, part of our proposed framework focuses on developing policies on compression that define if it has to be enabled or disabled to meet the needed performance. We also will design policies to achieve the optimal replication factor for the desired performance since the replication factor guarantee availability, but it may increase the workload's execution time. Finally, we will define a block size policy, as this size affects the workload's performance [31].

The main contribution in this chapter is to provide an optimal storage policy management framework in big data ecosystems that can guarantee the required performance even when the scenarios change. The framework is a policy-based controller that transfers knowledge between test cases to improve the policy's quality by adopting the TL method. We study four policies on two levels: storage and workload. At the storage level, we will define local and remote workload placement. At the workload level, we define compression, replication and block size policies. All policies will be examined with four case studies: benchmark types, software versions cloud service types and cloud provides. We will show that the framework provides higher quality policies than the single SUT policies.

4.1 Policy-based Storage Management

Policy-based management is an effective solution for controlling the heterogeneous environment with various performance requirements. It has been successfully applied in many critical storage decisions such as employing deduplication on data [89], allocating objects in the storage system [67], configuring storage components [66], and scheduling workload [51].

While the structure of the policy-based concept looks simple, applying the concept in a complex environment such as storage is a complicated task. Policy-based follow IFTTT structure [90], and it can contain many in-depth details such as where, when and how the policies should be applied [64] in order to satisfy a diversity of storage and workload requirements.

Storage policies can be defined in any storage I/O stack stage such as workload, ecosystem, hypervisor or back-end storage. The policies in each stage are very dependent on the I/O stage characteristic. For example, a big data ecosystem like Hadoop processes the workloads on Map and Reduce phases, where each phase interacts with the storage differently. The Map phase reads data from a remote storage and stores the intermediate data on local storage, while the Reduce phase reads data from local storage and stores data on the remote storage. Workload performance varies in each phase based on the workload I/O access pattern [8] and the back-end storage capability [62]. High-performance storage drives such as an SSD can improve workload performance. Therefore, we consider both the processing phase and back-end storage when designing workload placement policies.

At the workload level, there are several storage related policies that affect the workload's performance such as replication, block size and compression. The number of replications is essential for providing availability services, but when the number of replications increase, the storage capacity decreases and the workload performance is affected [54]. There is a need to define the right number of replications in the workload level to meet a specific performance. In big data ecosystems, the data is divided into splits with a particular block size. The splits are distributed into the cluster nodes for processing and storing operations where various block sizes cause different performance [54]. Since there is a relationship between block size and performance, defining the right block size is needed to optimize workload performance. Another critical storage policy is compression, which usually defines to reduce data size and save storage capacity. However, compression has an impact on workload performance [91]. Typically, compression policies in big data ecosystems are predefined and is applied for all workloads similarly.

Workload-aware policies designs are often on analyzing historical data to predict the best policy that can be used to meet the required objectives. However, in most work in the literature, the policies are designed based on collecting data from a single SUT. Big data ecosystem users usually change the setup to meet specific objectives such as updating software versions or move deployment from one cloud service type to another. The policies have to be redefined after each change, resulting in addition time, efforts and resources.

4.2 Adaptive Storage Policy Management Framework

In this chapter, we will present an innovative adaptive storage policy management framework that can enhance the quality of the storage policies and guarantee workload performance. The innovation of the framework is on using knowledge from previously designed policy model when updating policy model due to scenario changes. The proposed framework contains two main components: transfer learning engine and policy-based storage controller. The framework is illustrated in Figure 4.1.

4.2.1 Transfer Learning Engine:

TL is a powerful learning concept used with ML tasks such as classification to improve the *target* model accuracy through reusing a *source* model knowledge [38]. The fundamental

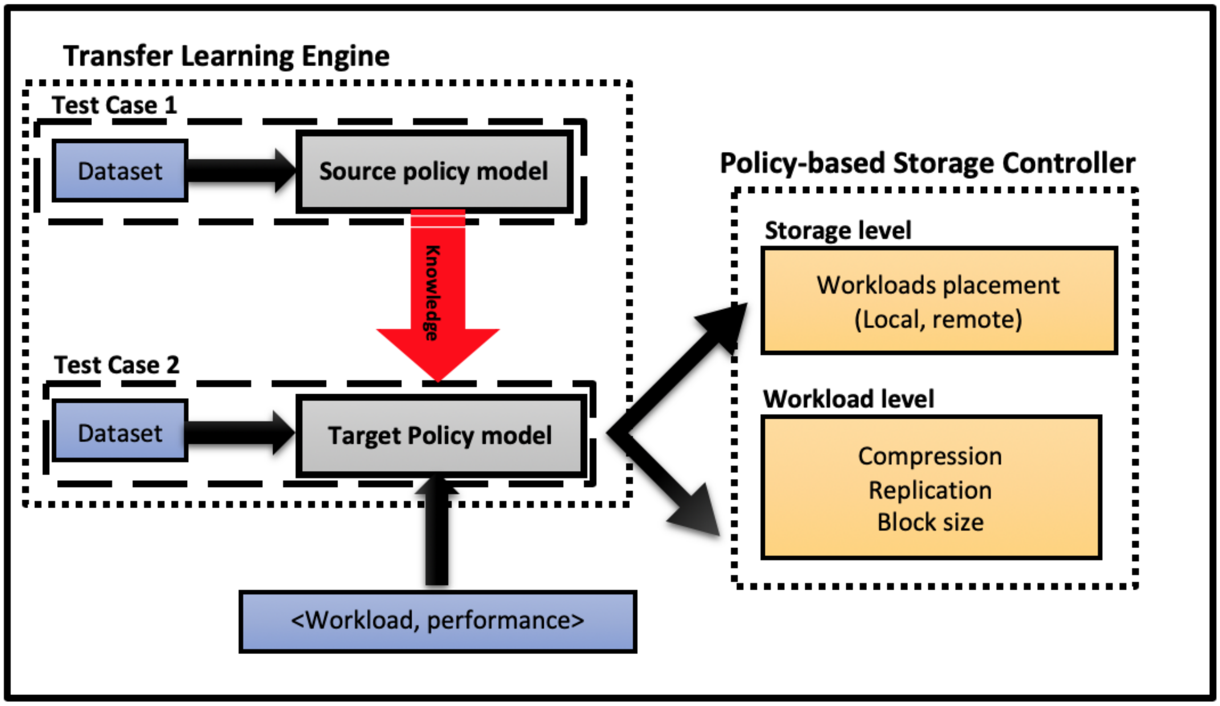

Figure 4.1: Adaptive Storage Policy Management Framework

TL procedures are :(A) having two datasets with different distributions,(B) construct the source model from one of the datasets, (C) apply the TL method on the source model to extract knowledge that is useful for the target model, and finally (D) construct the target model from the other dataset and with the extracted knowledge. More information about TL can be found in Subsection 2.2.2.1.

Data collected under different scenarios may have different distributions. A scenario is a set of setup values, and a scenario consider different than another when it has one or more different setup values. In this thesis, we consider scenarios different when they have one different setup value while other setup values remain the same where that different value comes from the same case study. For example, in software version cases study, the examined scenarios have the same setups values but different software versions. The data that is collected under different software versions have different data distributions [45]. In this chapter, we consider collecting data under different scenarios among four case studies: benchmark type, software version, cloud service type, and cloud providers, where each case study has two or more scenarios. The details about how we used the case studies' scenarios with TL were discussed in Subsection 3.5.

The TL method notations adopted from [38] and is explain below:

We define a classification task $\mathcal{T}$ for all examined policy $\mathcal{P}$, where each has X learning sample represent as $< Workload\ ,\ Desired\, Objective >$, and $\mathcal{Y} = 0, 1, ..., L-1$ set of L possible actions. The training data is a set of learning samples x and actions y, (x_i, y_i), where $x_i \in X$ and $y_i \in \mathcal{Y}$.

The classification task $\mathcal{T}$ has actions $\mathcal{Y}$ and an objective predictive function $f(.)$ that is defined based on the training data. The TL methods can either have a different task or different domains. Here we are focusing on having different domains and the same task.

Each policy examined with two different distributions domains: source and target; we refer to these as source policy $\mathcal{P}_S$ and target policy $\mathcal{P}_T$. These policies are used to construct source policy model $\mathcal{M}_S$ and target policy model $\mathcal{M}_T$ respectively. Each policy domain has two attributes: feature space $\mathcal{X}$ and marginal probability distribution $P(X)$. For a particular learning sample X, the feature space is represented as $\mathcal{X} \subseteq \mathbb{R}^n$ and the corresponding probability distribution is represented as $P(X)$. Where $X = \{x^{(i)}\}_{i=1}^{N}, x^{(i)} \in \mathcal{X}$. TL methods assume source and target domains to have either different feature spaces in $\mathcal{X}$ or different marginal probability distributions $P(X)$. In this work, we assume different data distributions between $\mathcal{P}_S$ and $\mathcal{P}_T$.

4.2.2 Policy-based Storage Controller

The designed controller receives and processes the TL engine result to define a high-quality policy that guarantees the required performance. In this work, we examine four storage policies: workload placement, compression, replication, and block size. The controller notations and operations are following policy-based management terminologies [64].

- $***Policy***: is a statement that define an $*Action* based on a given $*Condition* to achieve $*Goal*$_{desired}$. An example of a policy is:

$IF < \$\mathit{Condition} > AND < \$\mathit{Goal}_{desired} > THEN < \$\mathit{Action} >$

- $\$$***Condition***: The circumstances that affects the $\$$*Action* and it is defined based on the $\$$*Policy* purpose. For example, it could be the limited capacity, type of service, or job priority. In this work, the purpose of defining the policies is to guarantee workload performance, so the condition is the workload types.

- $\$$***Goal***: The objective of defining or modifying the storage policies. It is a predefined general parameter like capacity, reliability, performance, and cost.

- $\$$***Goal***$_{desired}$: A specific $\$$*Goal* threshold that is needed to be achieved such as high-performance. It is used to measure the quality of the policies through comparing it with the $\$\mathit{Goal}_{predict}$ that gain over applying the policy.

- $\$$***Action***: The decision that it makes to achieve $\$\mathit{Goal}_{desired}$. In this work, actions are placement, compression, replication, and block size. Specifically, for example, in the $placement$ case, the actions are a set of locations, and in $replication$ case, the actions are a set of factors.

- $\$$***PolicyGroup***: a set of related policies. We have two groups of storage policies, storage level, which includes local and remote placement and workload level that include compression, replication, and block size.

The policies are defined automatically by the controller based on the $\$\mathit{Goal}_{desired}$. An example of the designed $\$$*Policy* is:

IF<Workload types= X>AND< $\$\mathit{Goal}_{desired}$ =Performance> THEN <Compression=TRUE >

4.3 Experimental Evaluation and Results

In this work, we use Hadoop as a big data ecosystem [23]. Hadoop supports many types of MapReduce workloads like batch and stream, where the workloads have different I/O patterns. We examine the proposed framework in two policies level storage and workload. Storage level policies include : workload local placement and workload remote placement, and workload level policies include: compression, replication, and block size. We compare the quality of these policies with a single scenario approach (stand-alone) and with our proposed framework. The experiment includes testing four case studies: benchmark type, Hadoop version, cloud service type, and cloud provider. The source policy $\mathcal{P}_S$ by itself represents a single scenario approach, and source policy $\mathcal{P}_S$ and target policy $\mathcal{P}_T$ use to represent our approach. For example, in a benchmark type case study, source policy can be benchmark tool X, and target policy can be benchmark tool Y; details about the tested case studies explored in Subsection 3.5.

To examine Hadoop workloads, we use Hadoop execution traced data from an open-access collection by ALOJA framework [3]. More details about the data can be found in Subsection 3.4.1. The data set have information about workloads type, execution time, local and remote beck-end storage type, compression algorithm, replication factor, and block size. The dataset is collected from executing different types of workloads, such as I/O-bound and CPU-bound workloads.

4.3.1 Storage Level Policies

In short, our Hadoop MapReduce workloads interact with storage on two-levels: locally (node) and remotely (HDFS). Both local and remote storage could have various storage drives with various capabilities. In our experiments, both local and remote storage have HDDs and SSDs drives. The design placement policy defines if the workload local and remote data have to be placed in the HDDs or SSDs to achieve the required performance. Typically, the placement policies in Hadoop are defined by the Hadoop resource manager, YARN [20], which is an efficient resource management infrastructure. However, YARN does not consider the workload type or required performance.

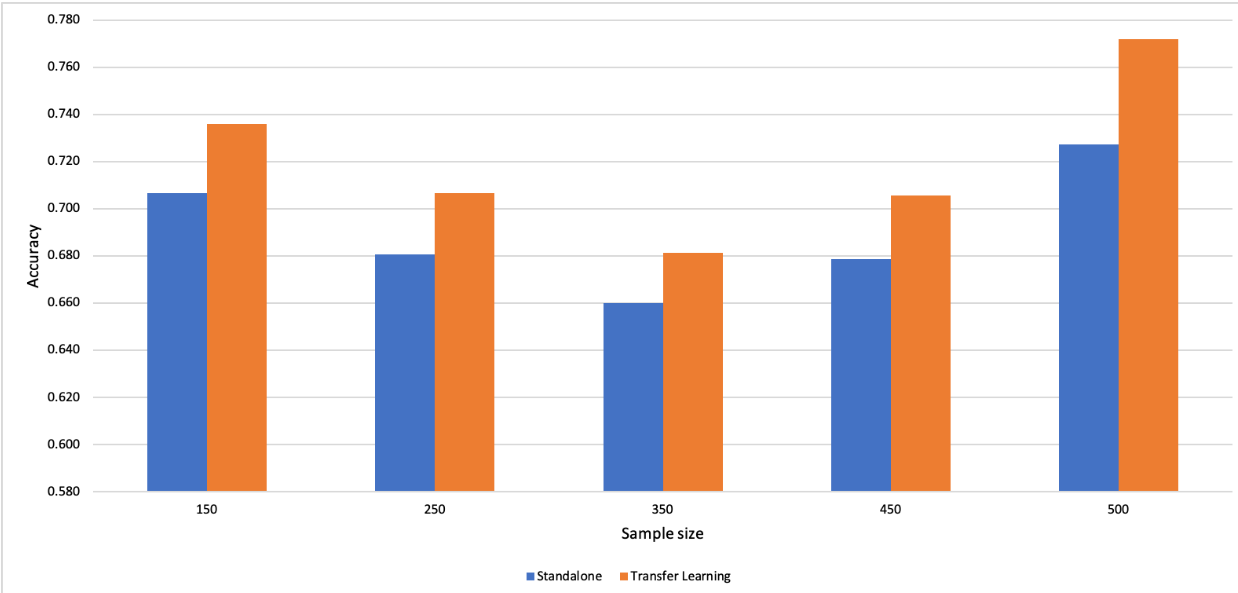

Figure 4.2: Workload Remote Placement Policy

Workload remote placement:

The remote placement policy defines if the workload distributed data has to store in HDDs or SSDs to achieve a particular performance. We examine the framework with the benchmark case study, where the source and target policies are coming from the same benchmark, but different versions represent $\mathcal{P}_S$ = HiBenchV1 and $\mathcal{P}_T$ =HiBenchV2. Our result shows that the designed remote placement policy has improved in the quality than the stand-alone approach by up to 6%. The quality improvement in remote policies with different sample size represents in Figure 4.2.

Workload local placement:

The local placement policy help decide which storage type is suitable for the temporary data to get the needed performance for a workload. We examine the policy with all case studies with five simple sizes. From Figure 4.3, we can see that the proposed framework provides an improvement on all case studies, but the benchmark case has the most considerable improvement by up to 21% comparing with the stand-alone approach.

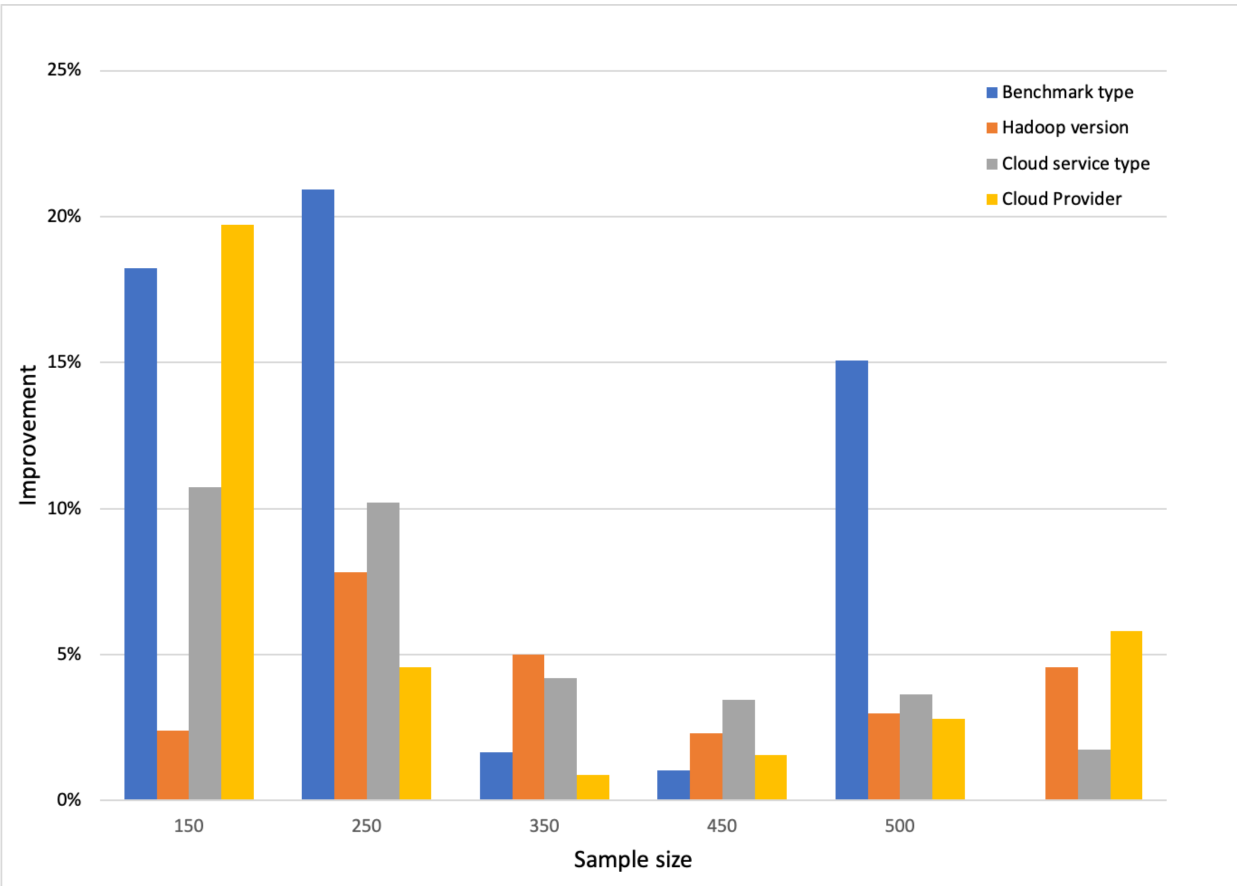

Figure 4.3: Workload Local Placement Policy

4.3.2 Workload Level Policies:

Policies at the workload level are compression, replication, and block size. In Hadoop ecosystems, these policies can define at the ecosystem level or workload level. In the case of workload level, mostly these define manually for each workload separately, which takes time and effort. As previous policies, we measure and compare the quality of the standalone policies and the proposed policies to find out the amount of improvement. The quality of the policies is calculating based on comparing the $\$Goal_{desired}$ with the $\$Goal_{predict}$. We examine each policy with our four case studies: benchmark types, Hadoop version, cloud service type, and cloud provides.

Compression:

Compression is an effective data reduction technique; however, it is not the best option for all workload as it is required some performance overhead. The designed policy in this section defines if the compression factor has to be enabled or disabled in the workload to obtain the required performance. We examine the compression policy with two case studies: cloud service type and cloud providers. In the cloud service type, we study how utilizing the policy designed for the on-premise environment helps improve the quality of the IaaS compression policy, where the result shows improvement by up to 32%. In the cloud provider cases study, we have Cloud Minerva as the source policy and Microsoft Azure as the target policy. Our result shows an improvement in the quality of the Microsoft Azure policy by up to 27%. The result is seen in Figure 4.4.

Replication: The MapReduce workload process over multi phases Map and Reduce where the data split on each phase into chunks and distribute in the HDFS. Each chuck can have one copy or more depending on the workload replication factor, which is usually defined by the user manually. The designed replication policy attempts to define the best number of replication that guarantees the needed performance. We examined all four case studies in the replication policy, as seen in Figure 4.5. However, Hadoop version and cloud

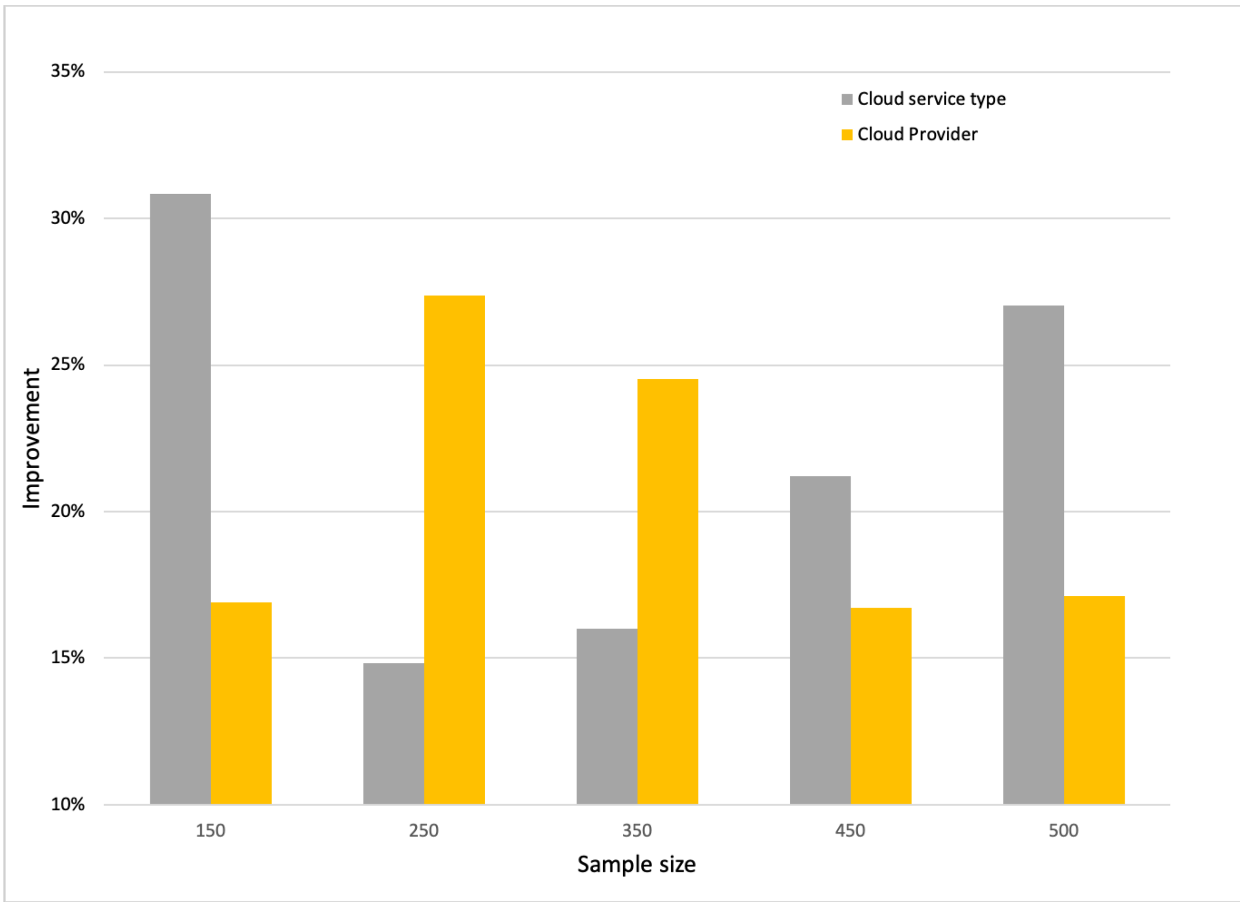

Figure 4.4: Compression Policy

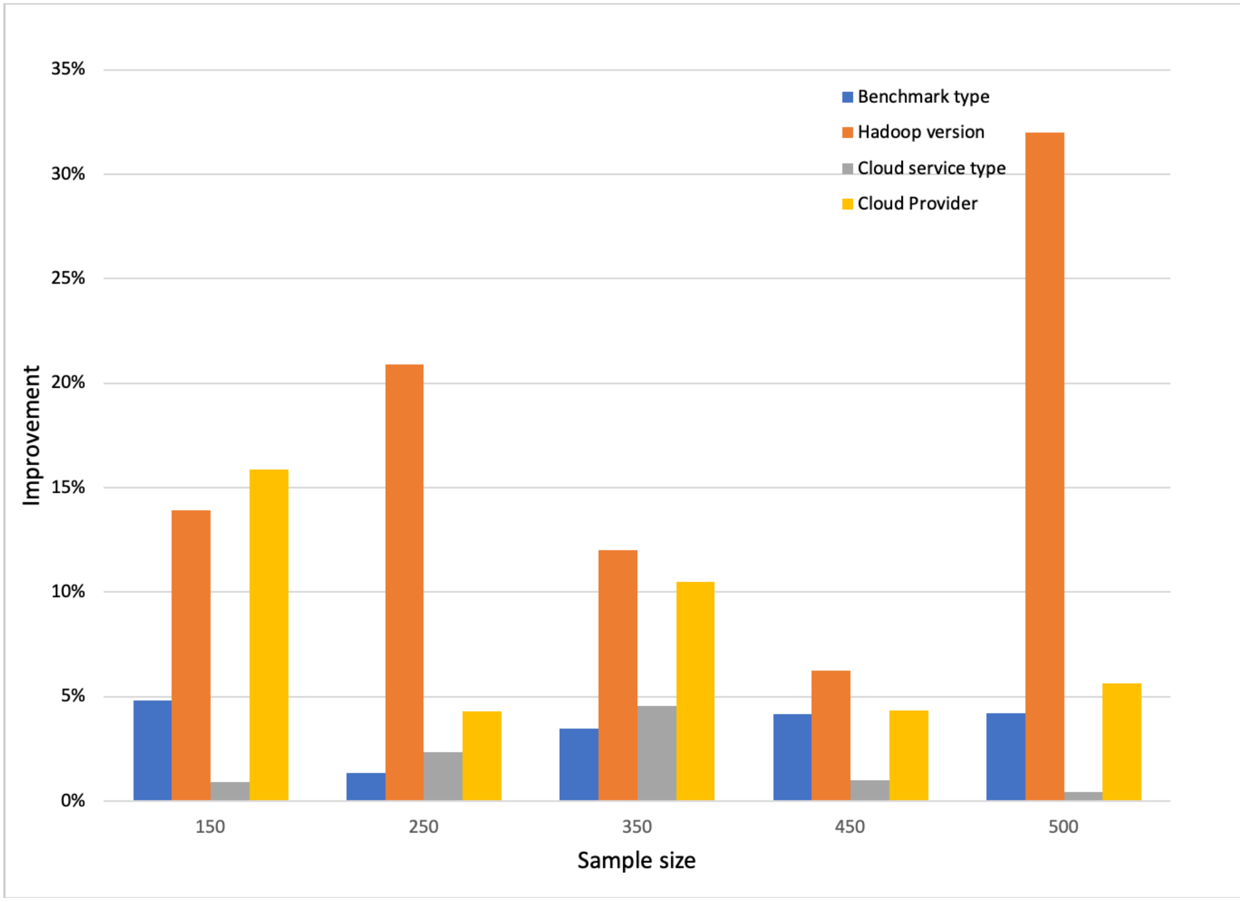

Figure 4.5: Replication Policy

providers cases studies show the most significant improvement compared with the other by up to 33% and 16%, respectively.

Block size Block size is defined as a workload configuration or as a general factor for all workloads in the Hadoop ecosystem, where it represents the size of distributing block in HDFS. The block default size is 128 MB, but it can be smaller or bigger, depending on the needs. There is a strong relationship between the block size and the workload performance [92]; therefore, we consider designing block size policy to guaranteed performance. We examine the policy with all case studies, where results show up and down improvement in all cases, as seen in Figure 4.5.

From the results, we can see that the improvement percentage varies between case studies and between sample size. In general, most of the results show butter performance

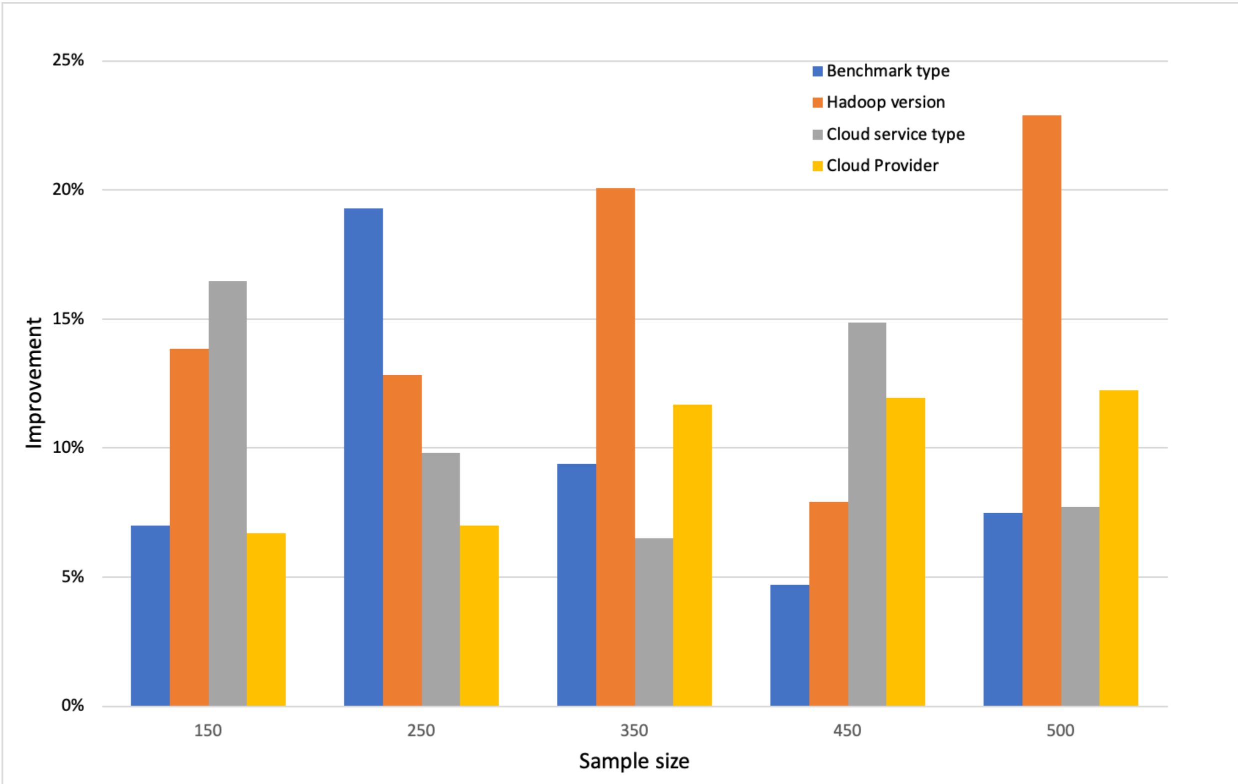

Figure 4.6: Block Size Policy

with a low sample size because TL methods are sensitive to data noise and the quality of the samples. Unfortunately, those are common issues in the big dataset. Moreover, we have noticed that some case studies benefit from the TL more than the other; this is because the Source and Target have a strong relationship. The relatedness between domains helps TL transfer more useful knowledge [93].

4.4 Chapter Summary

In this work, we designed an adaptive policy-based storage management framework for big data ecosystems. The framework policy-based controller gets information from more than one policy model by adopting the TL method. The framework defines high-quality storage policies by transferring knowledge between two policy models where each model is constructing from different scenarios. We examined various case studies include :

benchmark types, software versions, and cloud service types with four storage policies: workload placement, compression, replication, and block size. Then we measured and compared the qualities of the policies across the test cases scenarios when the policies are defined based on: the proposed framework and the traditional single test case approach. Our results show significant improvement in achieving storage policies that can meet performance requirements.

Chapter 5

Cache or Tier : An Evaluation of SSD Cost With MapReduce Workloads

5.1 Introduction

In recent years, there has been a growing interest in the use of big data ecosystem with large sets of data generated by sensors, mobile platforms etc. When used with heterogeneous storage systems supporting analytic dynamic workload requirements, a significant challenge facing these applications is the storage capacity and performance. A control storage utilization strategy has to balance the increased cost per MB with the workload performance [87].

SSDs have been drawing considerable interest in heterogeneous system architecture design, given that they outperform their HDD counterparts. Compared to HDDs, SSDs provide high-density storage with low latency, but with a higher cost and limited capacity. The improvement in performance and impact on the cost efficiency level depends on the way that the workload reads and writes data into SSD, data size and the R/W ratio [94].

SSDs are utilized in a wide range of storage setups, such as a primary storage, as an additional *tier* or as a *cache*. The storage efficiency and the I/O performance varies in each usage approach. For example, in the tier approach, the actual data resides on the SSD. However, in the cache approach, only a copy of the data is stored on the SSD. The cost of moving the actual data is different than the cost of copying the data. In general, moving data from the tier is more expensive than moving data in the cache. Earlier work has examined performance effects when introducing SSDs as a tier [12] [13] or when introducing SSDs as a cache [14] [15], but no work compares the performance and the cost of each usage approach with the same workload.

Due to the capacity limitation on SSDs, space utilization reduction techniques such as compression play an important role in managing the media efficiency, reducing the footprint and increasing the lifespan. A number of studies have investigated the association between compressing SSD and the SSD lifespan [73], efficiency improvement [74] and workload latency [95].

SSD performance and efficiency are affected directly by the data properties, system workload characteristics and behaviour. Evaluating the SSD usage approaches with simple read and write operations is different than evaluating complex analytic workloads. Analytic workloads require a multi interacting read and write, where the read and write could have a different level of skewness and therefore require different workloads placement policies.

In this chapter, we will investigate and analyze the performance and the cost difference between using SSD as a tier and a cache. Our work will look at a number of questions including: How does a given SSD usage approach improve MapReduce workload performance ? How does a MapReduce workload affect the SSD lifespan? Also, which of the SSD usage approaches benefit from space reduction techniques, such as compression to increase the SSD lifespan without decreasing the MapReduce workload performance?

We employ several tools in assessing our MapReduce workload's performance with different SSD usage approaches, and look at the impact of these workloads on the SSD lifespan. We will adopt Hadoop as a ecosystem for MapReduce workloads. Nine MapReduce workloads containing three different intensities, I/O, CPU and Hybrid, have been evaluated with two SSD usage approaches (tier and cache). Each approach was also evaluated with two setups (compress and uncompress) to observe the influence of the storage reduction technique on the SSD lifespan and workload performance.

The result of our work can provide enterprises with an important tool to understand how to maximize the dollar per MB on a storage system without declining the workload performance by optimizing the workloads placement.

The rest of the chapter is organized as follows. In Section 5.2, tier and cache SSD usage approaches are described. Hadoop as the MapReduce-based ecosystem is briefly discussed in Section 2.1.1. Section 5.4 describes our experimental setup, followed by the evaluation of the result in Section 5.5.

5.2 SSDs Usage Approaches

SSDs use solid-state memory with a NAND chips design that provides a high IOPS compared to the spinning platter design on HDDs. This is because NAND chips avoid spinning platter rotation and seeking which causes extra time on reading and writing. Furthermore, the NAND chip's durable design does not require moving physical parts which could get damaged over time. This lowers disk failure rates. The SSD design and performance makes it one of the key components of today's storage system. However, it should be added to a storage system carefully due to its high price and capacity limit.

Another issue with SSDs is their short lifespan. The lifespan and price vary based on the SSD's capabilities such as the amount of bit that could be stored on the NAND memory cell.

Two typical usage of SSDs into a system are tier and cache. SSDs can be used as a primary or an additional tier storage to store data permanently. They can also be used as cache storage to store data temporarily. All logical usage approaches perform and cost different depending on how they are defined and what they serve.

The logical design plays a critical role in meeting the system's goals and providing the requirements for the workloads. The challenges of today's systems are the workload characteristics and heterogeneity requirements. Each workload behaves differently and varies in the performance, cost, power, reliability etc. Therefore, understanding how each SSD logical approach works and affects the performance of each type of workload is a necessary step to optimize the storage model.

5.2.1 Tiering

In recent years, utilizing SSDs in high-performance storage media as a tier in a multi-tier storage environment has become more common. Attaching SSD as another tier improves the overall performance and reduces SAN traffic in the storage system. The tiering approach gives the system different storage capabilities, which provides more options for workload allocation and data placement.

Usually, SSDs are used as a tier when there is a need to store data permanently with high throughput and fast I/O. It is the optimal usage approach for sensitive data, high priority data and high read ratio data. For example, the hot/cold data classification-based tiering model places the active data into expensive and high-performance approaches and the inactive data into the cheapest and low efficient approaches [96]. The issue of using SSD as a tier is its capacity limit. Storing data permanently on SSD requires some level of redundancy to provide fault tolerance into the system, but replica comes with a drive utilization cost. The replica double the utilization of the drive which increases the drive footprint and reduces the drive lifespan. Data drive utilization $=$ actual data size $*$ number of replicas.

For workload placement, the SSD tier is the optimal option when a high I/O is required, and when the workload has a low writing ratio and low locality. There is no need to move data between drives. Understanding how the workloads utilize SSDs and optimize the placement decision can maximize the SSD lifespan.

This chapter explore the relationship between MapReduce workload's characteristics and the SSD lifespan. The impact of the SSD usage approach and workload performance. The result can be utilized to define the data placement and workload placement policies.

Today's storage systems are designed to handle dynamic requirements with an awareness of the workload characteristics and the storage capability as part of what is referred to as SDS. The SDS abstract the storage components capabilities, such as throughput and abstract the requirements of the workload, such as latency which can be defined to optimize storage management policies that help meet the workload requirements.

Some of the work in SDS introduce SSD as a tier [97] [98] to have storage capability diversity and be able to serve the workload with the needed requirements. SDS adapts policy-control methods, where policies define with an awareness of the front-end requirements and back-end capabilities. The controller automates storage workload and defines data placement, compression, deduplication and even encryption policies to customize the system for workload needs [99], [100] [98]. Although some SDS work considers SSD capability when defining the policies, no work considers SSD deployment approaches for defining the policies.

5.2.2 Caching

Caching is one of the most efficient methods to improve I/O performance. Instead of searching for the data on the entire tier, the data is fetched from the cache. The cache table is checked first to see if it has the data or not. If the data does not exist on the cache (miss), the request is directed to the tier. SSD is used for caching when the data has a high read access ratio. Data is stored temporarily on the SSD to gain a higher I/O for sufficient space compared to other caching methods. The cost of moving data from and into the cache is cheaper than moving data from and into a tier. The data in the cache is just a copy, but the one in the tier is primary, which is subject to the different storage restrictions such as data integrity. Besides, cache requires less space than a tier because it does not require the replication technique to support fault-tolerant. If for any reason, the data goes missing in the cache, it does not mean that it is lost unlike missing data on the tier. Number of work have investigated utilization of SSD as a cache [14], [101] and [102].

Today's big data ecosystems requires a massive amount of capacity for caching and SSDs play a key role in these ecosystem [8]. However, even if SSD provides more cache space in a big data ecosystem, the resulting available space is limited even when used with efficient space allocation/utilization techniques, such as caching page replacement algorithms and with ejecting data methods to keep the most needed data such as *Least Recently Used (LRU)*. The limited lifespan of SSDs is another significant challenge in highly interactive big data ecosystems. As discussed in the next subsection, applying additional data utilization techniques such as compression could potentially increase cache capacity and durability.

5.2.3 Compression and SSD

Compression can potentially improve SSD capacity-efficiency for both SSDs usage approaches: tier and cache. Studies have shown that applying compression in a SSD tier approach could reduce SSD drive utilization between $30\%$to 50% [53] [12]. It can also extend the SSD lifespan and reduce writes by up-to 22% when applied in the SSD cache approach [14].

In addition to the positive effect that compression brings in the SSD logical usage approaches, compression also contributes to enhancing the storage system when it applied to the hardware level [73]. However, the trend of to these benefits are added in the system with an additional computation and energy overhead [95]. Although compression improves the SSD lifespan, reduces the footprint, and achieves low-cost drive utilization, the overhead becomes a high I/O-intensive process in some cases and can challenge out the benefit gain [73].

The overhead varies based on the workload characteristics and behaviour, the compression algorithm and the system configuration. In this work, we investigate the impact of compression on both SSD usage approaches and with various MapReduce workloads types by measuring performance gain and SSD lifespan.

5.3 Hadoop performance and SSD efficiency

Hadoop execute a huge amount of data and therefore, the storage system capacity and capability play a critical role in the workload's overall performance. The heterogeneous storage system has been widely used in Hadoop to improve the performance. The storage media type in the heterogeneous storage system impacts the performance improvement. Research shows that Hadoop could gain a higher performance from SSD storage than HDD storage [59], [61]. SSD has been embedded into the HDFS by many to improve I/O [103], reduce cost [60] [62] and decrease energy consumption [56]. However, SSD is not always the optimal storage for MapReduce workloads. SSD performs well with a high read ratio, and MapReduce workloads have various R/W ratios that are affected by the map and the reduction process.

SSD could be used on Hadoop with many tiers. For example, it could be used to cache the intermediate data (cache approach) or to store the input and output data (tier approach). The performance and the cost in each usage approach depends on the data size, read/write ratio and other MapReduce-based factors. To the best of our knowledge, there is no work comparing and evaluating both SSDs usage approaches for MapReduce workloads.

5.4 Experiments

5.4.1 Setups

The experiment's environment contains five nodes: one master node and four slave nodes. The master node has 12 CPU, 64 GB memory, 120 GB SSD and 5 TB SATA HDD. Each slave node has 2 CPU, 24 GB memory, 60 GB SSD and 2 TB SATA HDD. The hardware configuration is listed in table 5.1. The software and compiler configurations are: Ubuntu 18.04.1 with Linux kernel 4.1.13, JDK 1.8.0_191, Hadoop2.9, Apache Mahout 0.13.0 [104] and Vsphere 6.7. To provide the same I/O load for all nodes, the workloads are distributed uniformly. Also, the same network setup is used for all setups so network factors are not considered in the evaluation. The recorded result is the average of executing each workload three times.

Component	Description
HDD	HGST HUH721212AL SATA
SSD	MZ7KM480HMHQ0D3 SATA
Network	Dual 10 Gigabit Ethernet Brocade
CPU	Intel(R) Xeon(R) Gold 5118 CPU @2.30 GHz
Memory	DDR-4 2400 MHz

Table 5.1: Hardware Components Description

5.4.2 Benchmark

To evaluate our work, we used BigDataBench 4.0, which could be accessed from `http://prof.ict.ac.cn/BigDataBench`. The benchmark is designed to be executed on 17 software stacks like Hadoop, Spark and TensorFlow. It provides 47 workloads with 7 types (online service, offline analytics, graph analytics, AI, data warehouse, NoSQL and streaming) and for 5 application domains (Search engine SE, social network SN, e-commerce EC, Multimedia processing MP and bioinformatics BI) [1]. It also provides a data generator workload to serve the workloads with the required data. The benchmark provides each software stack with different workload types based on the software properties.

From BigDataBench, we used 5 offline analytics workloads (Sort, Grep, WordCount, Kmeans and Naive Bayes) and 2 graph analytics (Connected Component and PageRank).

Some of the workloads are from the micro-benchmark and the others are from the component-benchmark. While micro workloads contain a single operation, component workloads contain multi operations [105]. The benchmark provides the workloads with an associated data generator that allows us to examine the workload in our experiment directly. The Connected Component workload defines the friendship connection on Facebook social network data and PageRank workload rank web pages on Google web graph data. Table 5.3 lists the utilized workloads. All of the workloads are executed directly on the MapReduce framework, except K-means clustering and Naive Bayes which are executed on Hadoop through Mahout. To measure the Hadoop I/O, we also utilized the write and the read workloads from the TestDFSIObenchmark, which are provided by the Hadoop environment. To generate the test data set, we utilized *Big Data Generator Suite(BDGS)* from BigDatabenchmark. The used data set includes: Wikipedia entries, Amazon movie reviews, Google web graph and Facebook social network. The summary of data sets based on the BigDataBench website represents in Table 5.2

Data sets	Data size	Data type
Wikipedia entries	4,300,000 English articles	unstructured text
Amazon Movie Reviews	7,911,684 reviews	semi-structured text
Google Web Graph	875713 nodes, 5105039 edges	unstructured graph
Facebook Social Network	4039 nodes, 88234 edges	unstructured graph

Table 5.2: Summary of the data sets

5.5 Evaluation

In this section, we are trying to answer the following questions: (1) How does the throughput of various MapReduce workloads change with SSDs usage approaches? (2) How do MapReduce workloads' characteristics impact the SSDs lifespan for both SSDs usage approaches? (3) When applying the compression method, what is the impact on the SSDs lifespan and workload throughput for both SSDs usage approaches?

Benchmark	Workload	Workload type	Data Set
Micro	Sort	Offline analytics	Wikipedia entries
	Grep	Offline analytics	Wikipedia entries
	WordCount	Offline analytics	Wikipedia entries
	Connected component	Graph analytics	Facebook social network
Component	Kmeans	Offline analytics	Facebook social network
	Naive Bayes	Offline analytics	Amazon movie review
	PageRank	Graph analytics	Google web graph

Table 5.3: Workloads Description [1]

We evaluated and analyzed performance-efficiency, defined as the MapReduce workload throughput, and cost-efficiency, defined as the SSD lifespan in SSD usage approaches for MapReduce workloads. The SSD usage approaches are: SSD tier usage approach (T), SSD tier with compression approach (TC), SSD cache approach (C), and SSD cache with compression approach (CC). MapReduce workloads are shown either with offline analytics or graph analytic workloads which are explained in the benchmark Section 5.4.2.

In the tier approach (T), the SSD serves as the storage media for HDFS. So in this setup, all of the distributed data is only stored on SSD, while HDD is used for other data in Hadoop such as the temporary files. The second setup is a SSD tier with a compression approach (TC). It is the same as the previous setup except the data processed by the MapReduce framework is compressed before it is stored on SSD. The third setup has the SSD cache approach (C). It handles intermediate data processed by the mapper in cases where it does not fit in the memory while HDD provides the media for HDFS. The fourth setup is a SSD cache with compression approach (CC). This is similar to the third setup except the intermediate data is compressed ahead of storage on SSD. All SSD setups are defined at the software stack level (Hadoop ecosystem).

5.5.1 Performance-efficiency

Firstly, we have shown how MapReduce workloads' performances are changing with various SSD usage approaches. The goal is to explore the relationship between workload throughput and the SSD approach to define the optimal performance-efficiency approach for the MapReduce framework in general and each workload individually.

As seen in Figure 5.1, in general, we get the highest overall workload throughput by using SSD as a tier on HDFS. The MapReduce workload performs better by up to 66% in a tier approach compared with the cache approach. This is because Hadoop workloads load and store a massive amount of data from and into HDFS. A SSD high I/O bandwidth helps improve HDFS I/O clearly and therefore, the workload throughput improves.

In the cache approach, the amount of intermediate data is low compared with the amount of data stored on HDFS. Therefore, the cache approach does not benefit from SSD as much as the tier approach. Additionally, Hadoop provides a high data locality approach on HDFS, so tasks run close to the process data. High data locality helps SSD perform well as a tier on HDFS. Most of the workloads perform better with the tier approach than the cache approach. The difference is evident in the workload with a mostly high number of maps such as Sort, Grep, and Page Rank workloads.

In addition, Figure 5.1 depicts the SSD usage approach throughput of each workload. It is notable that the tier with the compression approach has poor performance on all workloads with a percentage drop between 5% and 65%. The only workloads that have a better performance with a tier compression approach are read and component connected workloads because they are read-intensive. The rest of the workloads have a high write ratio on HDFS with a huge amount of data which is affected directly by the compress and decompress overhead. In addition, our investigation shows that the disparity of the average throughput between different usage approaches in I/O intensive workloads is small compared with the disparity in CPU intensive workloads and hybrid workloads as

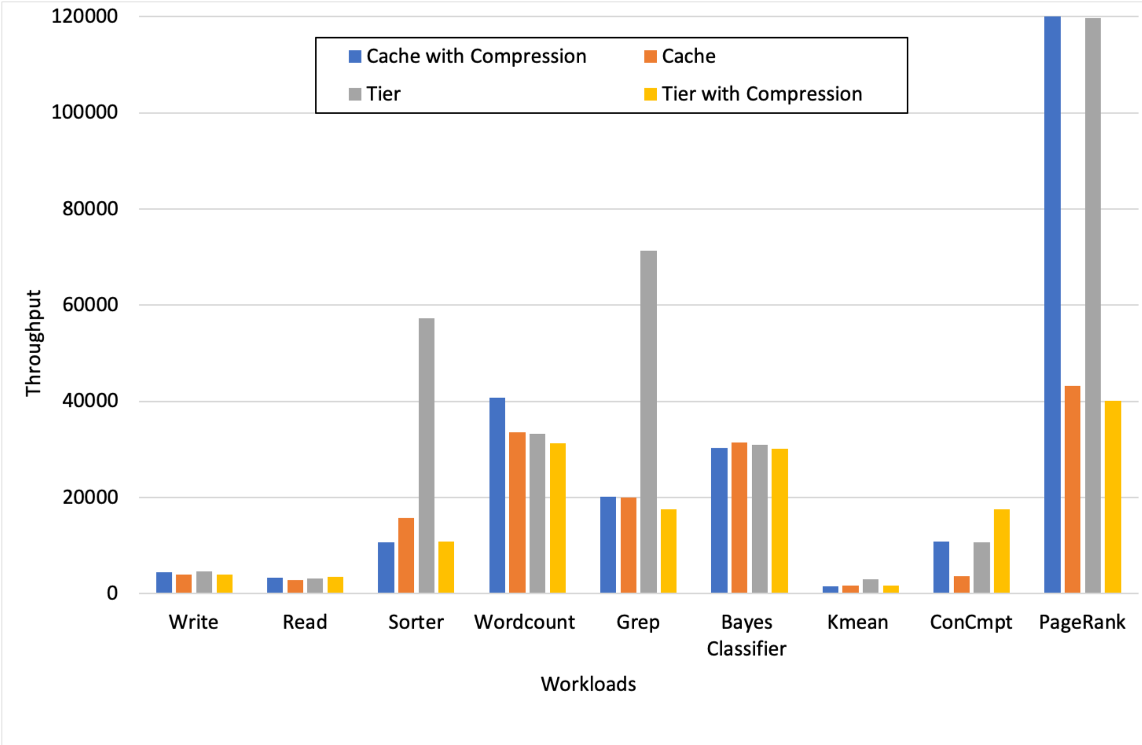

Figure 5.1: Workloads' Throughput

presented in Figure 5.2. The I/O intensive workloads include Write, Read and Sorter, the CPU intensive workloads include Wordcount, Grep, Bayes classifier and Kmean, and the hybrid workloads include ConCmpt and PageRank.

5.5.2 Cost-effective

The SSD lifespan is the cost model used to investigate SSD cost-effective while MapReduce workloads use it. NAND Flash has a fixed number of pages on each data block which is typically 64 or 128 pages. The flash page read and write operations help SSD achieve up to 100× faster random access than HDD. However, the data in SSD only writes to clean pages and therefore, it is slow by around 2 ms. Erasure operations have to be executed first. For example, Multi-Level Cell (MLC)s has a limited block cell erasure to 10K, while Triple-Level Cell (TLC) only allows around 1K block cell erasures. Such limitations make the SSD lifespan a critical aspect for controlling workload placement and data placement.

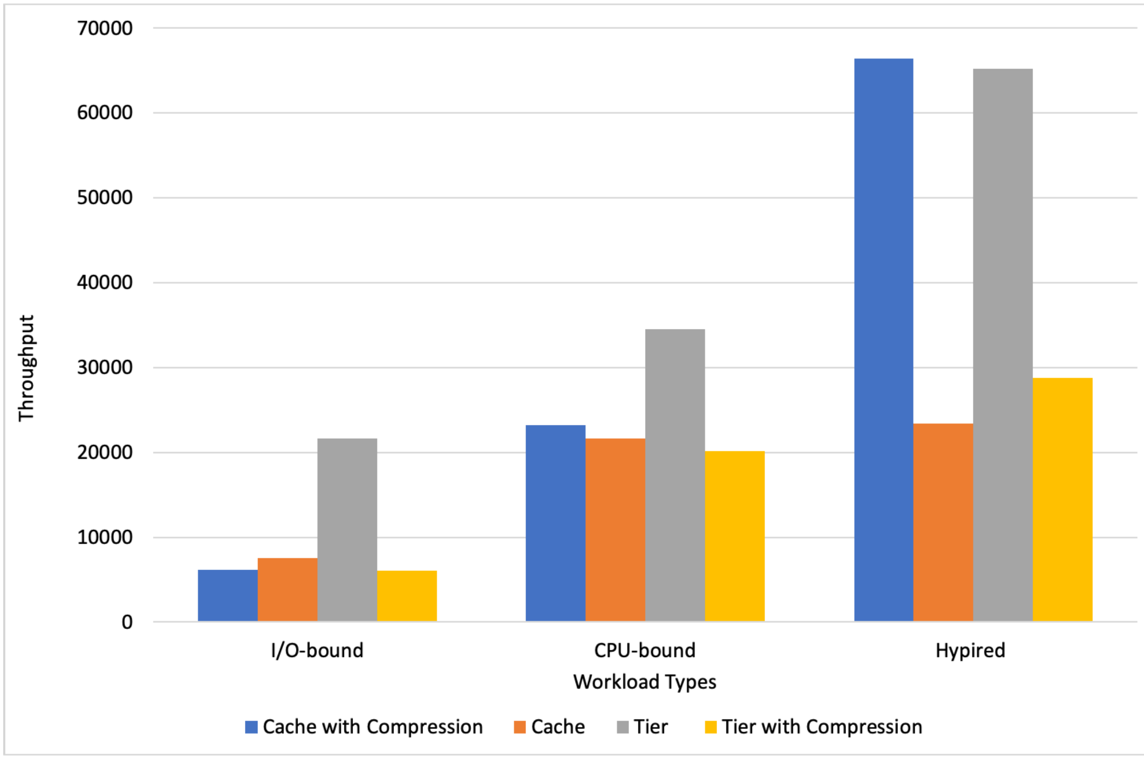

Figure 5.2: Workloads' average aggregate throughput per workload type

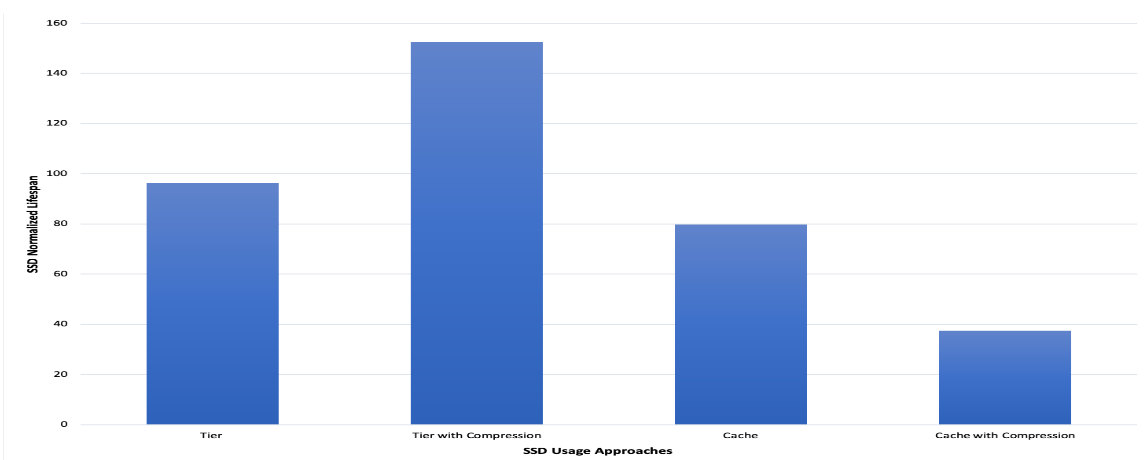

Figure 5.3: SSD normalized lifespan with SSD approceches. Taller means longer lifespan.

Understanding the lifespan could help us take full advantage of SSD capability. When the number of write operations is over the suggested limit by the vendor, SSD reliability drops and the error rate increases, which is not acceptable in most systems.

The SSD drive comes with warranty years and with a certain number of writes per day during the years, the reliability starts to drop. For example, if the drive comes with three years of warranty and 20 GB suggested writes per day, reliability declines after around $20 * (3 * 365) = 21.36TB$ writes. There are many methods to estimate the SSD lifespan. In our work, we adopted the SSD estimate lifespan formalized by the VMware documentation center [1].

`SSD estimation lifespan = (number of writes per day * warranted lifespan by year) / actual average number of write per day`

where the number of writes per day and lifespan are provided by the vendor and the actual average number of writes are collected from the VMware ESXi. For example, the SSD estimated lifespan for the above example when the aggregated workloads write is 30 GB per day is $(20 * 3)/30 = 2$ years.

We examined the SSD lifespan in MapReduce workloads with SSD tier and cache usage approaches, and each approach with compress and uncompress setups. The recorded write operations in each step are the aggregation of write operations generated by executing the explained benchmark in Section 5.4.2 three times. As evident in Figure 5.3, the higher SSD lifespan is achieved by utilizing SSD as a tier with compression setup and the lowest lifespan is found by utilizing the cache approach with a compression setup. The results are driven by the MapReduce workload behaviour and R/W pattern, cache location and the number of replications in the tier approach.

In general, while MapReduce workloads have a high read ratio and low write ratio for the data stores on the cluster where we applied the tier approach, there is a high write ratio and low read ratio for the intermediate data that we applied in the cache approach.

[1] `https://pubs.vmware.com/vsphere-50/index.jsp?topic=%2Fcom.vmware.vsphere.storage.doc_50%2FGUID-3CFD7350-E81E-446A-A0C8-2239A16A77E0.html`

SSD performs well with a high read ratio and low write ratio because it provides 8$\times$ read speed than write. The caching approach shows low SSD lifespan utilization by the huge number of blocks writes sends to SSD from MapReduce workloads intermediate process. MapReduce intermediate write exhausts SSD erase operations and expanding write traffic and therefore decreases SSD reliability.

Hadoop has multi-stage architecture, where each stage has its own cache allocation policies that are used by individual services and workload components. Defining cache in the YARN stage could improve service performance more than workload performance due to the lack of workload R/W pattern information in this stage. Yarn does not provide intermediate data placement policies to control placing hot intermediate data only into the cache.

The number of replicas in the tier approach affect drive utilization and therefore drive lifespan. In our experiments, we set replica to 1 for a fair comparison between the tier approach and cache approach when they are used by MapReduce workloads. The replication number policies could be defined based on the workload's priority level and the data sensitivity, which are not covered in this work.

Figure 5.3 shows that the tier approach provides around a 20% longer lifespan than the cache approach. Also, applying compression in the tier approach enhanced the lifespan by around 60%. On the other hand, applying compression in the cache approach reduced the lifespan by around 50%. The cache approach with a compression setup provides the shortest SSD lifespan. Compression does not reduce the intermediate data size efficiently due to the high skewness data output produced by the map phase that causes a compression ratio reduction. Besides, the compression service in Hadoop converts the fixed- sized blocks into variable-sized blocks and SSD performs better with a fixed-sized block than variable-sized block [14].

As shown in Figure 5.4, the offline analytic workloads like Sort, Grep, WordCount, Kmeans and Naive Bayes have a shorter SSD lifespan compared to the graph analytics

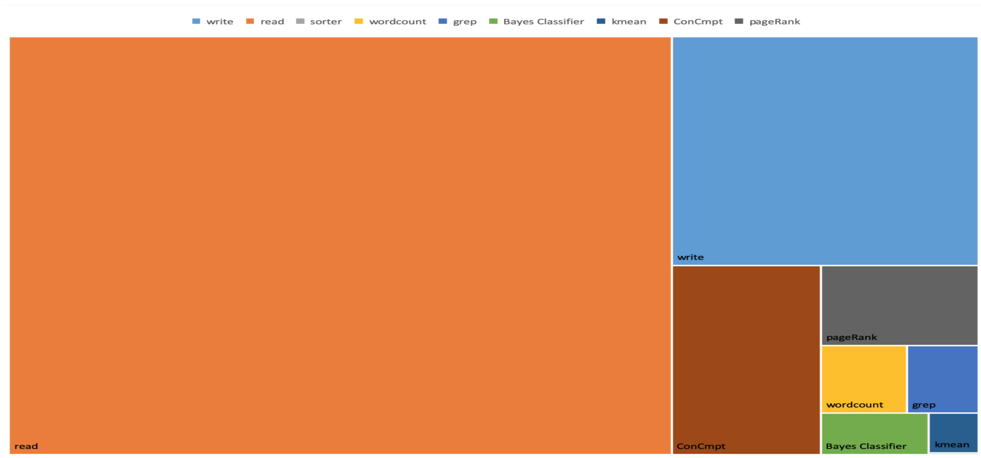

Figure 5.4: SSD normalized lifespan per workloads. Bigger means longer lifespan.

workloads such as Connected Component and PageRank. This is due to the high iteration ratio in the graph analytic workloads where the same data is used several times. Offline analytic workloads are I/O-bound so it exhausts the SSD erase operations and increases the write traffic.

5.6 Chapter Summary

SSD has great potential for improving the MapReduce workload performance when it is either used as a tier or cache. However, SSD is expensive and it has a limited capacity compared with drive storage. In this work, we demonstrated the relationship between MapReduce workloads' performance gain when introducing SSD into a MapReduce-based ecosystem and SSD lifespan. We showed that utilizing SSD as a tier on the MapReduce-based ecosystem would improve performance and increase the SSD lifespan more than utilizing SSD as a cache for the intermediate data. We also analyzed the impact of the data reduction method, such as compression on the performance and the cost for both usage approaches. We observed that SSD as a tier could benefit from compression to improve the SSD lifespan more than the cache approach. The investigation helps obtain intelligence workload placement policies for high workload performance and low storage cost. For a fair comparison between cache and tier, we set the number of replications to one to avoid the replication effect on the tier approach. However, the work could be extended to examine the tier approach with other ecosystems and job configurations. Hadoop is a multi-stage ecosystem. In our work, we examined cache in the YARN stage. The SSD cache approach could be investigated in other Hadoop stages like MapReduce and HDFS.

Chapter 6

HMM-Based I/O Modeling to Extend The Lifespans of SSDs

6.1 Introduction

Flash-based SSDs have become one of the main components of most of today's storage system. They provide highly attractive characteristics like low latency and low power consumption. They can also play various roles in systems and be deployed as a cache or as an additional storage device to support heterogeneous storage architecture.

Nevertheless, SSDs' valuable features come with a price. They are expensive and they have limited capacity compared to their HDDs counterparts. Organizations have to add them into the system carefully and utilize them intelligently to minimize the total cost of ownership. There is a strong correlation between the system's performance and managing storage system, especially with today's data growth. However, balancing the big data-based system capacity requirements and the SSD capacity limitations is not a straightforward task. For this reason, there is a great deal of desire both in industry and academia to use solutions that can manage and control SSDs in order to reduce storage costs and to improve the overall system's performance.

SSD's high capabilities drop significantly with use. In particular, SSD performance decreases as the number of drive writes increases.

Workload I/O access patterns have a strong correlation with the storage cost and performance [9]. Workload I/O patterns could be different in terms of intensity (read or write), sequentiality (sequential or random) or request size. For example, web server workloads like search engines and web services typically have random large I/O patterns while workstation workloads generally have sequential small I/O patterns. At the same time, storage devices have a dissimilar throughput with various access patterns. In general, HDDs provide a very high throughput for sequential reads compared with the throughput for random reads and a very low throughput for both random and sequence writes. On the other hand, SSDs provide a fairly similar throughput for both random and sequence reads and lower throughput for writes. Despite this, the SSD write operation throughput still outperforms its HDD counterpart.

Investigating the workload's I/O patterns brings many cost and performance benefits into the storage system through optimizing the system's design like optimizing the scheduler and placement policies. Consequently, analyzing and characterizing I/O patterns has drawn a grate deal of attention [106] [107].

In today's system, I/O flows from the front-end user until the back-end storage by visiting many system stages such as Virtual Machine (VM), application, hypervisor etc. Each stage has a unique storage functionality like compression and replication as well as a unique I/O processing method like splitting, scheduling and allocation. The stage's functionality and processing methods affect the I/O patterns. Recently, SDS related research such as [68] have focuses on understanding I/O end-to-end patterns in complex environments and establishing stage-based storage policies to guarantee SLA.

In addition to I/O complexity from the system stages, workloads executed on big data ecosystems like Hadoop [23] or big data engines like Spark [21] are suffering from interactions with the ecosystem component's functionality and methods. Besides, the same workload is usually divided into sub tasks where each of them is executed in one or more stages. Accordingly, big data workloads often have mixed I/O patterns during the execution time. For instance, an individual workload may have a high random read on time t_x and a low sequence read on time t_y. In this chapter, we use MapReduce workloads as an example of big data workloads. The MapReduce workloads are splitting into multiple tasks executed on the map and reduce phases consecutively.

The HMMs [108] is a well-known and successful model for many applications such as speech and handwriting recognition. It has also been adopted in I/O research to categorize and predict I/O patterns.

In this chapter, we will apply a mixed categorical HMM on the Hadoop workload historical data to discover the R/W ratio that is changing during the map and reduce process. The model result is used to define optimal workload placement policies for SSD cost efficiency.

We focus on modeling MapReduce workload's I/O patterns in both local and remote storage to guarantee SSD's long lifespan on the Hadoop ecosystem. The designed workload placement policies reduce SSD utilization and improve the SSD lifespan by up to 40%.

We will design a cost-efficiency SSD lifespan controller with two phases: constructing model and defining policies. First, we will construct an HMM-based model for prediction of SSDs lifespan usage by different MapReduce I/O patterns. Then, we will use the result to define workload placement policies.

The HMM-based model is constructed based on a trace approach where the trace data is MapReduce workload's execution history. The model analyzes the MapReduce workloads R/W ratios on the map and reduce phases when the workloads interact with the local and remote storage. The model is then used to define workload placement policies to maximize the SSD lifespan.

The rest of the chapter is structured as follows. In Section 6.2, we outline flash-based SSDs and the workloads. Section 6.3 introduces the MapReduce framework and the interaction between big data ecosystems and SSD. In Section 6.4, we describe the proposed SSD cost-effective controller framework. In Section 6.5, we describe the SSD controller and the corresponding results in detail. Finally, Section 6.6 has a summary of the chapter.

6.2 Flash-based SSDs and Workloads

SSDs have been drawing considerable interest in big data ecosystems, given that they outperform these HDD counterparts. Compared to HDDs, SSDs provide high-density storage with low latency, but with higher costs and limited capacity. Nevertheless, the SSD cost and performance is not stable during the time and use. They are affected by many external factors, including the workload's I/O patterns like R/W ratio and request sequentiality [94].

An SSD gains its capabilities from its unique design and from the way that it reads and writes data. While an HDD contains moving parts to read data, an SSD reads data through a lookup table. Therefore, in contrast to an HDD,the SSD performance is not profoundly affected by the read sequentiality. It performs well for both random and sequence reads.

However, today's SSD design still suffers from some limitations despite significant improvement in design since it was invented in the 1950s. The two main limitations are related to the writing operation. First, the NAND cells on flash-based SSDs can only tolerate a limited number of write operations. Second, extra procedures (erase and program) are required for every writing operation. Those limitations cause a sharp decrease in SSD's overall performance within and after each writing operation. Therefore, an SSD performs better with the read operation than the write operation.

An SSD includes a set of controllers to minimize the effect of those limitations and to manage its other properties. Various models have been developed to provide specific capabilities as needed. For example, a NAND cell could be designed to store one or more bits and thus offer a different performance and cost. The performance and the cost decrease as the cell stores more bits. Single-Level Cell (SLC) provides the highest performance and has the highest cost, and Quad-Level Cell (QLC)provides the lowest performance and has the lowest cost. Performance and cost fall between those of SLC and TLC.

6.2.1 I/O Patterns

There is a significant correlation between the workload's I/O patterns and SSD. The I/O patterns cause short-term and long-term effects on SSD performance and cost. Therefore, modeling I/O patterns is a significant procedure for SSD management decisions. Enhancing SSD usage can help obtaining the manufacturer's promise of performance for a longer time and also reduce the storage cost. Modeling is an effective method to discover the I/O patterns.

SSD is very responsive to the I/O patterns such as I/O size, R/W ratio, locality, and sequentiality. An SSD provides a different performance and cost for various I/O patterns [109]. For example, as previously discussed, write overstrains SSDs. Therefore, the workload with a low write ratio is more suitable for an SSD than the ones with high write ratio. Every write bit reduces the NAND cell's tolerance and has a permanent impact on SSD even after the operation is completed [110]. Subsequently, both current and previous I/O patterns should be considered in storage-based decisions.

Modeling is an effective method for simplifying workload I/O characteristics and describing the relationship between the I/O and the SSD lifespan usage. Models can be used for evaluating [25], simulating, predicting or defining I/O patterns. The discovered patterns are then used to achieve the system requirements [2]. There are many methods to build the model, but the most suitable method for workload I/O is the trace-driven approach that is built based on a collected history.

Discovering the workload's I/O patterns can help optimize storage-based decisions and achieve storage and workload desired requirements. Storage-based decisions can include storage configuration, deployment plan, scheduling and placement. The patterns can also be utilized to optimize the workload execution plan which is out of this work's scope.

In this chapter, we will design an I/O MapReduce workload model with the trace-driven method to optimize workload placement policies and the SSD deployment plan on the ecosystem.

6.2.2 Workload Placement Policies based on I/O Patterns

Storage systems that interact with big data applications control the heterogeneousness I/O workloads and storage. The systems are expected to execute a large number of workloads which are typically processing and producing a massive amount of data. Failure to design the storage could drop the overall performance and can cause I/O bottleneck even if other resources such as network and CPU are optimized. Performance is not the only concern in the big data ecosystem. Enterprises also plan to minimize the storage cost. They try to balance between meeting the application requirements like performance and meeting the system's needs to be cost-effective.

The workload's I/O patterns have a direct influence on the overall performance and SSD cost [9]. Workloads with different patterns demand a different amount and type of resources. For example, I/O-bound workloads have higher storage utilization compared to CPU-bound workloads. Besides, even I/O-bound workloads utilize storage differently depending on the I/O characteristics. Defining the placement policies based on the workload access patterns and the storage capability can provide a powerful tool in storage management.

The model workload's I/O patterns provide useful information for constructing effective placement policies. Today, the I/O stack could include multi applications, ecosystems, storage systems etc. The model can be designed to study the I/O flow in one or more components of the system where each stage has its unique features. Since system stages are not the same, the information provided by the I/O pattern model is mostly useful for placement policies in the same modeling stage. In this chapter, we will focus on model I/O patterns on the MapReduce framework and design workload placement policies on the Hadoop ecosystem.

6.3 HMM-based MapReduce Workload's I/O Patterns Model

In this chapter, we will employ the HMM to design the I/O patterns model for MapReduce workloads. The model is derived from a trace-based analysis approach that utilizes MapReduce trace data collected during the workload execution and from the log file.

The I/O pattern model can be designed for a particular element such as data, file, block, workload or application [107]. The model target element is selected according to the system's desired goal. The I/O activities of the element are collected either from one source such as one block or from multiple sources such as workloads. The main goal of our work is to investigate I/O patterns of MapReduce workloads to optimize SSD utilization through the placement policies. Consequently, our proposed model focuses on workloads as the target element. The I/O-based model will take into account the element and I/O characteristics. Element characteristics are like skew, size and how it changes over time as well as I/O pattern characteristics like I/O size, number of reads, number of writes, R/W ratio and locality. Another factor that are considered in designing an I/O patterns model is the system setup. I/O activities are not isolated from the system. Each configuration factor impacts the I/O behaviour. I/O behaves differently in the unlike system setups. In this chapter, we trace workloads with various characteristics and focus on R/W ratio. To avoid complexity, we collected the trace data from executing workloads on the same system setups.

Designing models for workloads executed on a simple ecosystem with one stage is different than designing it for workloads executed on the ecosystem with several processing stages like Hadoop. There are many challenges for designing I/O patterns in a multi-stage level environment. Considering that the stages interact, the same workload is executed differently with a slight change in the setup, and the workloads are affected by the placement policies in each stage.

I/O and MapReduce Framework

The MR workload's I/O activities need to be measured in two locations: *DataNodes (local)* and *HDFS (remote)*. Local locations serve the intermediate data and the remote location stores the input and output data. More details about the MapReduce framework is explained in Subsection 2.1.1.

6.3.1 The Interaction between Big Data ecosystems and SSD

The interaction between hardware and software components makes managing storage systems a complicated task. In fact, the storage system is accessed by many software components such as big data ecosystems. Each of the ecosystem has its own method for dealing with the back-end storage. Although storage media comes with its particular controller systems, the storage media could be managed by an external software controller from other system components through a software-defined architecture.

The SSD management design has an impact on workload performance and costs. The characteristics of the workloads also have an impact on SSD performance and costs. The performance of the same workload could be changed dramatically when the SSD design is changed and when the SSD utilization is changed. big data ecosystems have different workload characteristics. Therefore, having a reliable correlation between the workload's I/O pattern is a critical factor in optimizing SSD management policies such as scheduling and allocation [103]. Correlation can be used to modify the ecosystem components to optimize SSD utilization. For example, the workload execution plan can be adjusted to read data on a sequence order instead of random order. For instance, [9] we can change the data layout by transforming the random write access into sequential access to meet the SSD capabilities.

It is evident that controlling SSD from one level cannot guarantee SSD efficiency or workload requirements. In the SDS parading the control plane and the data plane are separated. SDS objects to guarantee the overall requirements by enforcing policies in the system stages. The software controller has comprehensive information about the I/O flow from the front-end user to the back-end storage. The information is used to define automatic policies for specific applications, workloads or ecosystems based on their capabilities, instead of static general policies.

We adopt the SDS approach by designing placement policies to guarantee SSD efficiency in the MapReduce framework.

Controlling SSD from a Hadoop level provides advantages for both the workload and SSD. It is widely recognize that SSD brings huge benefits into the Hadoop ecosystem [59], [61].The benefits include improved I/O [103], decreased energy consumption [56] and reduced overall cost [60] [62]. It is also evident that SSD is expensive and has limited capacity. Hadoop processes huge amounts of data and stores it in the SSD storage, which does not fit on SSDs, at least with today's SSD capabilities. In this chapter, we have designed a policy-based controller that balances allocating workloads in SSD and maximizing the SSD lifespan.

6.4 The Proposed Ecosystem-level SSD Cost-effective Controller

The main goal of this work is to manage SSD utilization from the ecosystem level and therefore guarantee SSD's long lifespan. The designed placement policies reduce SSD utilization and improve the SSD lifespan by up to $\%40$. The designed, cost-effective SSD utilization controller has two phases: construct HMM for modeling MapReduce workload's I/O patterns and define workload placement policies. Both phases are done on the Hadoop ecosystem.

The proposed controller has a number of procedures which are illustrated in Figure

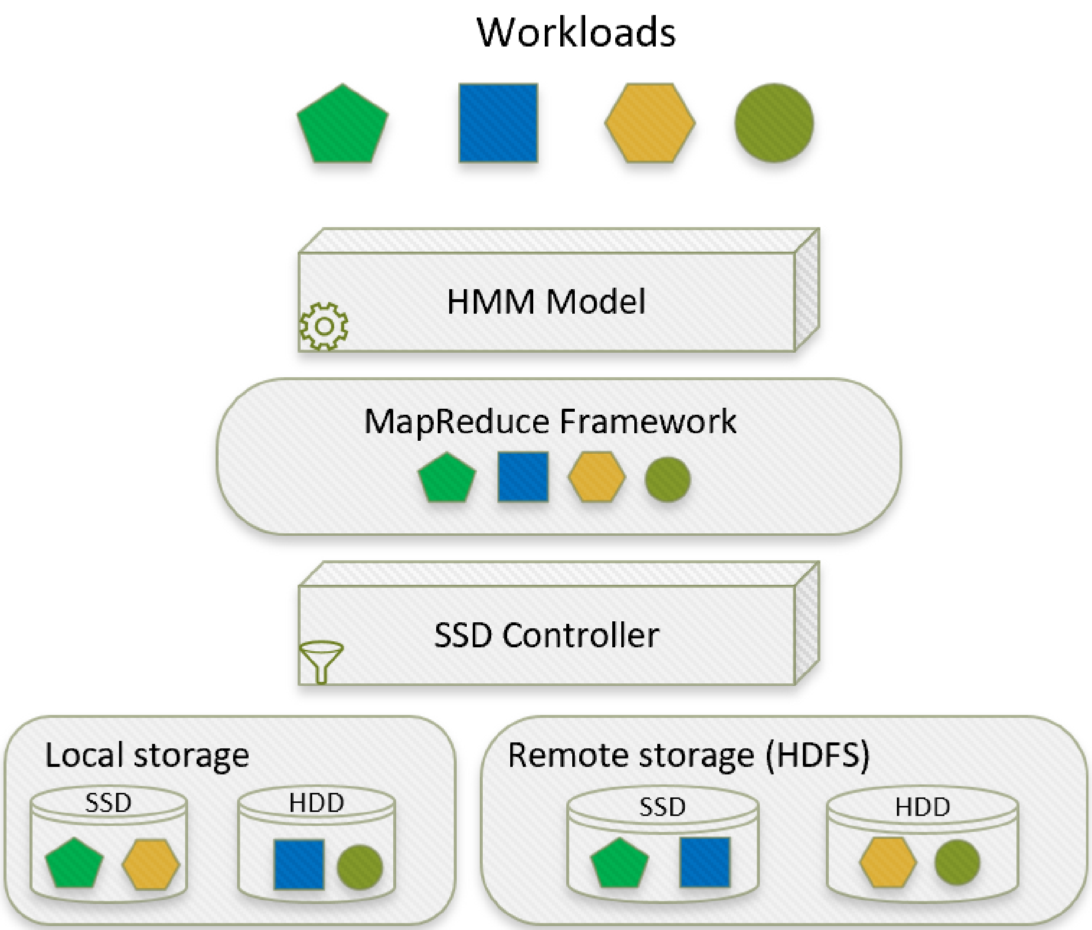

Figure 6.1: SSD Cost-effective Controller

6.1. First, workloads with diverse I/O patterns are delivered to Hadoop. The workload's execution data is collected to be used in the modeling phase. The HMM-based I/O pattern's model is constructed for MapReduce workloads. The model output is used as a seed for the placement phase. The placement policies are defined to maximize the SSD lifespan. Further details are provided in the next section.

6.4.1 Construction HMM for MapReduce Workloads

The HMM is a well-known statistical model for many applications like speech, and handwriting for recognition and prediction. It is also successfully used in I/O patterns research [111]. HMM can be used as supervised and unsupervised learning algorithms. It is designed based on Markov's chain stochastic model which produces information about the probabilities of sequences of random variables with an assumption that (*states*) are unobservable. *States* could represent any information depending on the application design.

HMM could solve three fundamental issues: learning, decoding and likelihood [112]. Various learning algorithms such as Forward-Backward, Baum Welch and Viterbi training algorithms are used for the leaning problem. The learning algorithm is used either for initially estimating the HMM parameters such as transition probabilities matrix and emission probabilities matrix or for updating the parameters. The observation likelihood $p(O|\lambda)$ and the state sequence (S) decoding could be estimated for an HMM model (λ) when the transition probabilities matrix (A), and the emission probabilities matrix (B) are known. The HMM notations are listed below:

- λ is the HMM model.
- S is a set of hidden states
- N is the number of hidden states, $S = s_1 ... s_n$.
- O is the set of observed states.
- M is the number of observed states, $O = o_1 ... o_m$.
- Y is the sequence of observations with in time T , $Y = y_1, ..., y_t$
- A is the transition probability matrix with size $N \times N$ that represents the moving probability between one state to another. For example, element a_{ij} on the matrix holds the moving probability from state i to state j. It is represented as an arrow between the states.
- B is the emission probabilities matrix represents the likelihood of states producing the observations. For example, element $b_i(o_t)$ on the matrix holds the probability of

state i to produce the observation o_t. It is symbolized as an arrow between the state and the observations.

- π is the states initial probability distribution vector with length N. Each state s_n has a starting probability π_n where $\pi_n = P(x_i = s_n)$, $0 \leq \pi_s \leq 1$.

The HMM makes a number of assumptions and has the following properties:

1. Markov's chain method assumes that the probability prediction of a sequence of variables considering only a current state while all previous states make no impact on the probability prediction. So, for a sequence of state variables $s_1, s_2, ..., s_i$,the assumption is: $p(s_i = a|s_1...s_{i-1}) = p(s_i = a|s_{i-1})$.

2. The output observation is only dependent on its produced state regardless of what the other observations are or other states are $p(o_i|s_i)$.

3. The sum of moving probabilities into a state i should be 1 that us $\sum_{j=1}^{n} a_{ij} = 1$

4. The total of the initial probability distribution on all states has to be 1, when $\pi = \pi_1, ..., \pi_N$ than $\sum_{i=1}^{n} \pi_i = 1$.

5. Ergodic Markov Model is defined when the initial probability on any states is non-zero.

We have applied seqHMM package in R [113] to our HMM model. SeqHMM is a sequence analysis method that analyzes sequences of multivariate categorical data to define a set of correlations [114]. This package is suitable for our work because it supports multiple subjects of the data and also supports multiple interdependent sequences. In MapReduce workloads, there are multi-sequences of reading and writing in multiple locations (local and remote).

In our work we have employed the sequence-HMM on the MapReduce workload traces data to classify the workloads based on the I/O intensity. HMM provides a probability distribution of the upcoming workload when observing the access sequence where the model's focus is the workload R/W ratio level. Later, we use the I/O intensity to define the optimal workload placement policies that guarantee the SSD lifespan.

I/O patterns vary during stages in the MapReduce framework. The stages represent the time sequences in our model. Furthermore, there are two locations that the data reads and writes from: local and remote. Data access location and operation types are represented as a channel. Each of the four channels have two states low and high. The model is designed with four hidden states as illustrated in Figure 6.2. Channel 1: remote read has R-high-read and R-low-read , channel 2: remote write has R-high-write and R-low-write, channel 3: local read has L-high-read and L-low-read, and channel 4: local write has L-high-write and L-low-write. The lows and highs are marked based on the channel operation mean.

The MapReduce workload traced data is collected from executing nine workloads that are each repeated three times. Seven of the workloads were part of BigDataBench 4.0 which can be accessed from `http://prof.ict.ac.cn/BigDataBench` and the resets of the workloads are from the TestDFSIO benchmark. The seven workloads from BigDataBench are portrayed as five offline analytics workloads (*Sort, Grep*, *WordCount*, *K-means* and *Naive Bayes*) and two graph analytics workloads *Connected Component* and *PageRank*. The workloads are a mix of micro-benchmarks with a single operation and component-benchmarks with multi operations [105]. The Hadoop TestDFSIO benchmark contains read and write workloads. The workloads are executed on the MapReduce framework with an additional tool excluding K-means clustering and Naive Bayes which are executed over Mahout [104].

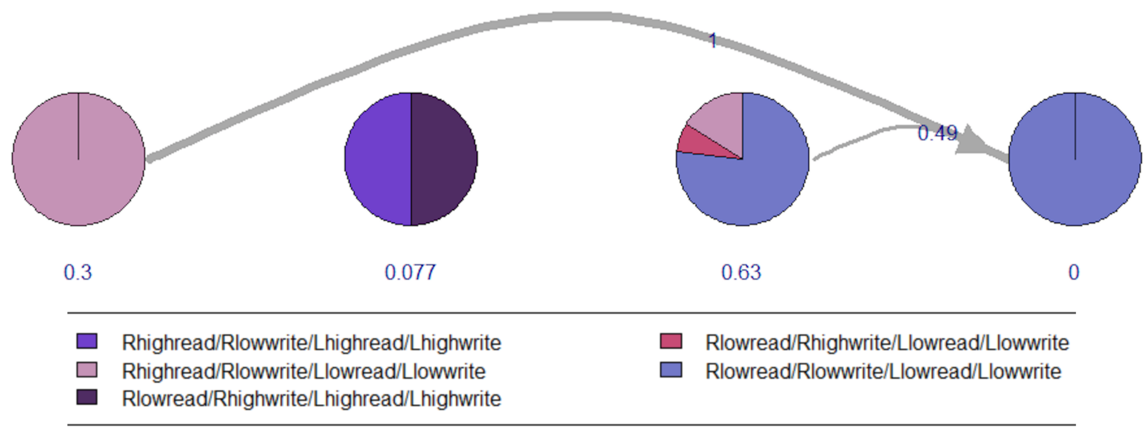

Figure 6.2: HMM-based MapReduce Workload's I/O patterns

6.5 SSD Controller

The big data's workloads goes through many I/O stages such as applications, ecosystems and storage systems before arriving at the back-end storage. Each I/O stage has it is own storage-based policies that define how it interacts with each back-end storage type and with the whole storage media. Defining general static workload policies regardless of the behavior of the workload cannot guarantee SLA or achieve system storage-based requirements. These policies are supposed to be designed at each stage based on the system's need, the workload'sI/O patterns and the stage properties. SDS architectures [68] focus on controlling storage-based policies at the stage-level. Policies are defined at each stage to enforce the SLA and guarantee storage efficiency at each stage.

Our proposed SSD controller utilize the SDS controller concept for SSD provisioning on the Hadoop ecosystem. First, the proposed HMM model provides the SSD controller with stage-based I/O pattern information. Then, the information is used by the controller to improve the SSD lifespan under storage-based policies.

The SSD provisioning metric used on the SSD controller is the SSD lifespan. The controller defines the policies based on how MapReduce workloads alter the SSD lifespan. The NAND cells on SSD can have a limited number of writes and as the number of writes increases, lifespan decreases. For instance, if the SSD comes with a limited number of

10K for cell erasure, the SSD performance and reliability drops when the limited number is reached. Therefore, the manufacturer guarantees the SSD performance for a particular number of years and a particular number of writes per day.

There are various methods for calculating the SSD lifespan. This work employs the SSD estimated lifespan formalized by the VMware documentation centre[1].

`SSD estimation lifespan = (number of writes per day * warranted lifespan by year) / actual average number of write per day`

The manufacturer specifies both the number of writes per day and the warranted lifespan, while the actual average number of writes per day is collected by the user from the VMware ESXi. For example, if the manufacturer specified a two-year warranty for 30 GB suggested writes per day and the user generates around 40 GB writes per day, the estimated lifespan is one and a half years $(30 * 2)/40$.

6.5.1 Workload Placement Policies

The MapReduce framework has two storage locations: local and remote. SSD could be deployed on the Hadoop ecosystem to serve local or remote storage. MapReduce workloads have different I/O patterns at each storage location as investigated in the HMM model Section 6.4.1. Consequently, the SSD lifespan varies at each location as evident in the Figure 6.3.

Workloads with a high write ratio on local storage, such as Sorter, Connected Component and PageRank, provide a shorter SSD lifespan on local storage than remote storage. Connected Component and PageRank are graph analytic workloads with high intermediate computing and they generate big data. On the other hand, the write workload, for example, provides a longer lifespan on local storage than remote storage due to its high write ratio on the remote storage. Overall, remote storage provides a slightly longer SSD lifespan compared to local storage. This is caused by the massive number of blocks

[1] `https://pubs.vmware.com/vsphere-50/index.jsp?topic=%2Fcom.vmware.vsphere.storage.doc_50%2FGUID-3CFD7350-E81E-446A-A0C8-2239A16A77E0.html`

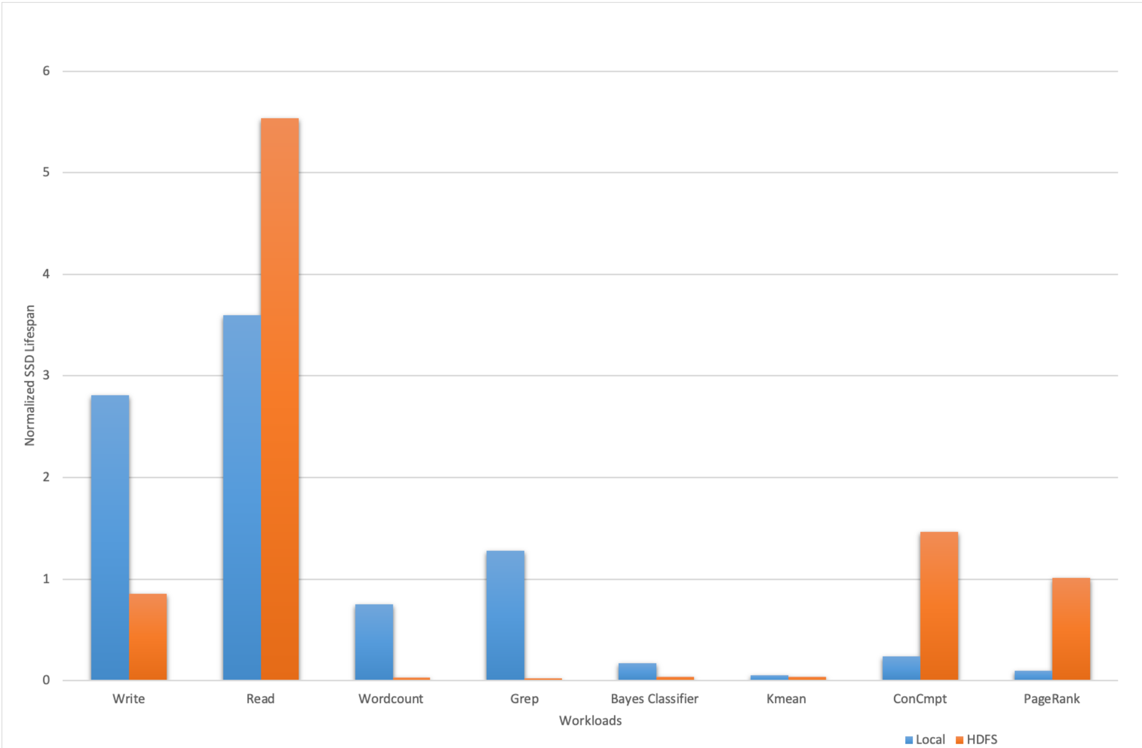

Figure 6.3: Local and remote storage's SSD lifespan of MapReduce workload's where higher represents longer lifespan.

generated during the MapReduce workload's intermediate process. Besides, for equitable compression, the replication factor on remote storage is set at one. An increase in the number of replications on the remote storage can change the SSD lifespan.

The SSD controller defines placement policies for both local and remote storage based on the workload's I/O patterns with the intention of maximizing the SSD lifespan. On the lifespan function l, the objective of the selection model is to select optimal policy P for a workload wa where $l(w,p)$ is maximized. It is defined as $argmax_{p_i \in P} l(w,p_i)$. The defined policies are listed in Table 6.1.

Policy	Condition	placement policy
policy 1	if Write $=$ High $\&$ Read$=$ Low	All blocks _HDD
policy 2	if Write $=$ High $\&$ Read$=$ High	All blocks _HDD
policy 3	if Write $=$ Low $\&$Read$=$ High	All blocks _SSD

Table 6.1: I/O patterns and storage policies

Figure 6.4 shows a normalized SSD lifespan of the defined placement policies, allocated

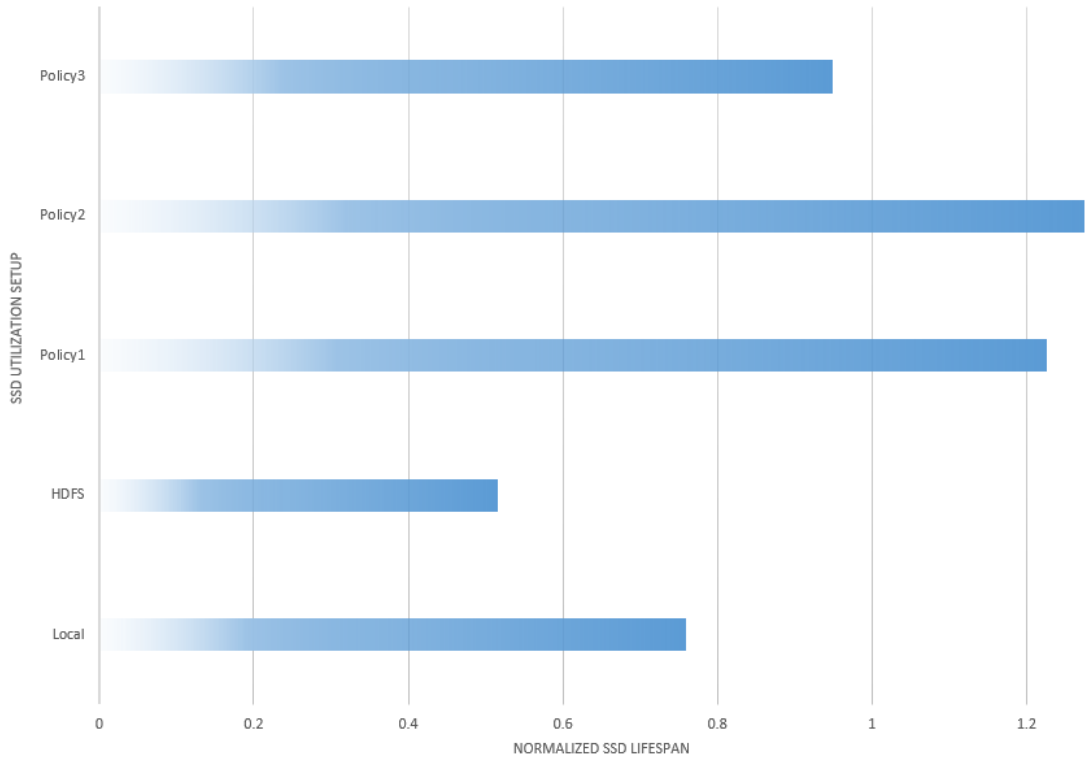

Figure 6.4: Placement policies normalized SSD lifespan where a long bar represents a longer lifespan.

workloads on SSD local storage and allocated workloads on SSD remote storage. It clearly shows that the defined policies indicate a longer lifespan than both local and remote SSD storage. Avoiding allocating the workload into SSD when it has a high write ratio could extend the SSD lifespan when the read ratio is low or high (policies 1 and 2). Allocating the workload into SSD if it has a high read and low write ratio could improve the SSD lifespan (policy 3) but not as (policies 1 and 2). The achieved results reflect the fact that SSDs are sensitive to write operations. However, placement policies for MapReduce workloads on both local and remote storage cannot be done manually. The designed HMM model and the SSD controller provide the automatic placement mechanism to maximize the SSD lifespan. The policies are defined based on the map and they reduce write operations in both local and remote storage during the workload's execution.

6.6 Chapter Summary

The goal of the SSD controller proposed in this chapter is to optimize placement policies on the Hadoop ecosystem for maximizing the SSD lifespan. HMM-based model was designed to investigate the relationship between MapReduce workload I/O patterns and SSDs lifespan consumption when the SSDs use on the local and remote storage. The result of the model use to define cost-efficiency workload placement policies.

The HMM model abstracts the MapReduce workloads I/O pattern, where the workloads are executing on several processing stages and interacting with multiple storage locations. The same workload has different I/O patterns in various processing stages and storage locations. Defining placement policies in local and remote storage based on the workloads I/O pattern can result a significant improvement in the SSD lifespan.

The evaluation lifespan metric showed that the designed placement policies could achieve a 40% longer lifespan compared to static workload placement policies in the local and remote storage. The evolution estimate based on executing nine MapReduce workloads includes offline analytics, graph analytics and I/O workloads. The designed SSD controller can be employed to automate placement policies to extend SSDs lifespan.

Chapter 7

Conclusions and Future Work

Performance prediction is an essential aspect of several critical system design decisions, such as workload scheduling and resource planning. However, developing a model with higher prediction accuracy is a challenging task in big data systems due to the stack complexity and environmental heterogeneity. We presented a new transfer learning modeling approach and policy-based storage management solution designed for effective big data ecosystems. The proposed modeling approach is driven by the complexities of the ecosystem's setups and the need for a dynamic modeling approach. In this thesis, we showed that a single testing setup approach, which we call a single SUT, is not the optimal workload modeling approach in a complex environment. Thus, we developed a transfer learning modeling approach, which we call cross-SUTs, that considers setups changing and improves the model's accuracy. The approach benefits from the previously built model (source) to improve the new model's (target) accuracy, where the source model is constructed using a dataset collected from testing setups that are different than the ones in the target model. The cross-SUTs approach was evaluated with four testing setups: benchmark types, software versions, cloud providers and cloud service types. Our approach showed that the target model's accuracy can be enhanced with the proposed approach compared to the standalone model by up to 78%.

The next objective of this thesis was to investigate storage management solutions in the ecosystems to improvement in the quality of storage policy, enhance workload performance, and reduce SSDs costs.

First, we designed an adaptive policy-based storage management framework based on the TL method that helps improve the quality of storage policies to guarantee workload performance even when the ecosystem setup changes. The proposed storage framework shows quality improvement in four storage policies: workloads placement, compression, replication, and block size by up to 33%.

Second, we focused on exploring SSDs usage approach to enhance workload performance. The storage system usually contains drives with various capabilities to serve mixed workload I/O patterns and meet their needs. Highly capable storages are typically expensive ones. For example, SSDs provide high performance, but have a short lifespan. Balancing between required performance and storage cost constraint is an urgent and complicated task. To control the SSD's lifespan and therefore reduce overall storage cost, we investigated two storage management solutions: workload placement policies and storage usage approach selection. We investigated the cost and performance of SSD usage approaches in distributed storage systems. High-performance drives like SSDs can be used as cache or tier in the local and remote storage sequentially. In the thesis, we studied how the SSD's lifespan is affected by the usage approach and how various MapReduce workloads I/O pattern impact the lifespan. MapReduce workloads are interacting with the local storage for intermediate results data and with remote storage for final result data where the I/O patterns are dissimilar in each storage location. We also studied the compression technique's impact on SSDs' lifespan on both usage approaches where the study is done on mixed I/O patterns. Understanding how SSD's lifespan is affected by the usage approach and I/O patterns helps us select a cost-efficient usage approach in the big data ecosystems. Our results showed that by using SSD as a tier, MapReduce workload performance can be improved by up to 66% and SSD lifespan can be increased by up to 20% when comparing with cache approach. We also observed that applying compression on the tier approach enhanced the lifespan by up to 60% but reduced lifespan of cache tier by 50%.

Third, we proposed and implemented an HMM-based framework that analyzes the I/O patterns of MapReduce workloads to define storage cost-efficient workload placement policies. We evaluated the framework with nine MapReduce workloads including offline analytics, graph analytics and high I/O workloads. The designed framework controls the placement of the workload on the local and remote storage to extend the SSD's lifespan by up to 40%.

7.1 Future Directions

In this Subsection, we discuss some of the future directions on workload modeling and storage management solutions in the big data ecosystems.

Modeling Efficiency and TL Methods:

Big data ecosystems have various, complex components that are frequently changing. Designing an effective workload modeling approach in a dynamic environment is a challenging task. In this thesis, we focused on designing an efficient modeling approach when the change in the ecosystems happens in four case studies: benchmark types, cloud service types, cloud providers and software versions through adopting a supervised transfer learning method. There are several ways we can apply TL methods to develop the modeling approaches and we discuss some of them below.

- In this thesis, we examined TL when a single setup changes, the TL methods could also be considered when more than one setup changes like software version and cloud service type.
- Collecting data is an expensive activity in big data ecosystems and most of the collected data has missing labels. Therefore, there is a need to study unsupervised and semi-supervised transfer learning methods [115] when modeling the ecosystems.

- TL methods typically solve the issue of having different data distributions from two domains. However, some work presents TL generalization methods such as Domain Generalization (DG) [116] to solve the issue of having multiple data distributions from more than two domains. Generalization TL methods can be used to design an efficient general modeling approach to decrease the modeling management effort instead of constructing and controlling multiple models.

- Users typically deploy more than one big data ecosystem or move from one ecosystem to another to satisfy their needs. New models have to be constructed after deploying a new ecosystem or moving to a new one. Big data ecosystems have a lot of similar features and therefore, applying TL methods between their models can save time and resources.

- TL methods have different time and space complexities and energy consumption [117], which have to be measured when designing modeling approaches to decide if the gain in performance is worth it.

- The knowledge in TL methods can be transferred between models (tasks) or domains (data). In this thesis, we transferred knowledge between models to improve model accuracy. Transferring knowledge between domains can be used as a sampling method to minimize training data size. Investigating TL methods that are focusing on reusing data would benefit modeling massive amounts of data.

Big data Storage Management Solutions :

The storage system is one of the most critical components in big data ecosystems due to the size of producing and serving data. In this thesis, we focused on two storage management solutions: workloads placement and storage usage approaches. There are several directions on storage management solutions; we are going to discuss some of them below.

- Our study results about using SSDs as a cache can be aggregated with some of the software-defined cache (SDC) [118] controller functionality. For example, the SDC controller defines the cache amount, and location policies based on a designated objective, if the objective is cost, then SSDs lifespan can be considered in the metric.

- Compression techniques can be applied at the software level, like what we studied in this work, or applied in hardware level [73]. There is a need to investigate how SSDs' lifespan and workloads performance are affected by different applying locations.

- Applying *Multi-Agent System (MAS)* to control storage policies are among the complex big data ecosystem components. A MAS is a system with multiple intelligent agents who interact with each other to resolve a complex problem that is not easy to solve with only one agent. The MAS agents can design to control big data ecosystem components.

Bibliography

[1] W. Gao, J. Zhan, L. Wang, C. Luo, D. Zheng, X. Wen, R. Ren, C. Zheng, X. He, and H. Ye, “BigDataBench: A scalable and unified big data and unifiede benchmark suite,” *arXiv preprint arXiv:1802.08254*, 2018.

[2] D. G. Feitelson, *Workload modeling for computer systems performance evaluation*. Cambridge University Press, 2015.

[3] J. L. Berral, N. Poggi, D. Carrera, A. Call, R. Reinauer, and D. Green, “Aloja: a framework for benchmarking and predictive analytics in hadoop deployments,” *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 4, pp. 480–493, 2015.

[4] Y. Demchenko, C. De Laat, and P. Membrey, “Defining architecture components of the big data ecosystem,” in *Proceedings of the International Conference on Collaboration Technologies and Systems (CTS)*. IEEE, 2014, pp. 104–112.

[5] Y. Chen, S. Alspaugh, and R. Katz, “Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads,” *In proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1802–1813, 2012.

[6] Z. Jia, W. Gao, Y. Shi, S. A. McKee, Z. Ji, J. Zhan, L. Wang, and L. Zhang, “Understanding processors design decisions for data analytics in homogeneous data centers,” *IEEE Transactions on Big Data*, vol. 5, no. 1, pp. 81–94, 2017.

[7] T. M. Madhyastha and D. A. Reed, "Learning to classify parallel input/output access patterns," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 8, pp. 802–813, 2002.

[8] Z. Tang, W. Wang, L. Sun, Y. Huang, H. Wu, J. Wei, and T. Huang, "IO dependent SSD cache allocation for elastic hadoop applications," *Science China Information Sciences*, vol. 61, pp. 1–17, 2018.

[9] R. Stoica, M. Athanassoulis, R. Johnson, and A. Ailamaki, "Evaluating and repairing write performance on flash devices," in *proceedings of the 5th International Workshop on Data Management on New Hardware*. ACM, 2009, pp. 9–14.

[10] Z. Yang, M. Awasthi, M. Ghosh, J. Bhimani, and N. Mi, "I/O workload management for All-Flash datacenter storage systems based on Total Cost of Ownership," *IEEE Transactions on Big Data*, 2018.

[11] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, "Extending SSD lifetimes with disk-based write caches," in *proceedings of the 8th Conference on File and Storage Technologies (FAST)*, vol. 10. USENIX, 2010, pp. 101–114.

[12] Y. Cheng, M. S. Iqbal, A. Gupta, and A. R. Butt, "Cast: Tiering storage for data analytics in the cloud," in *proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2015, pp. 45–56.

[13] J. D. Strunk, "Hybrid aggregates: Combining SSDs and HDDs in a single storage pool," *ACM SIGOPS Operating Systems Review*, vol. 46, no. 3, pp. 50–56, 2012.

[14] C. Li, P. Shilane, F. Douglis, H. Shim, S. Smaldone, and G. Wallace, "Nitro: A capacity-optimized SSD cache for primary storage," in *proceedings of the Annual Technical Conference (ATC)*. USENIX, 2014, pp. 501–512.

[15] J. Kim, H. Roh, and S. Park, “Selective I/O bypass and load balancing method for write-through SSD caching in big data analytics,” *IEEE Transactions on Computers*, vol. 67, no. 4, pp. 589–595, 2018.

[16] P. Valov, J. Guo, and K. Czarnecki, “Empirical comparison of regression methods for variability-aware performance prediction,” in *Proceedings of the 19th International Conference on Software Product Line*, 2015, pp. 186–190.

[17] F. Alsayoud and A. Miri, “Cross-scenario performance modeling for big data ecosystems,” *to appear in 22nd International Conference on Human-Computer Interaction (HCI)*, 19-24 July 2020.

[18] ——, “SSD: Cache or tier -an evaluation of SSD cost and efficiency using mapreduce,” in *Proceedings of the 16th International Conference on Computer Systems and Applications (AICCSA)*. ACS/IEEE, 2019.

[19] ——, “HMM optimized modeling of SSD storage for I/O mapreduce workloads,” in *Proceedings of the 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2019, pp. 0177–0183.

[20] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, and S. Seth, “Apache hadoop yarn: Yet another resource negotiator,” in *proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.

[21] “Apache Spark™ - Unified analytics engine for big data,” [Online] Available : https://spark.apache.org/, last accessed on 29 January 2020.

[22] “Apache Storm™ - distributed dealtime computation system,” [Online] Available : https://storm.apache.org/, last accessed on 15 December 2019.

[23] "Apache Hadoop," [Online] Available : https://hadoop.apache.org/docs/r2.9.0/, last accessed on 1 April 2020.

[24] D. C. Schmidt, "Model-driven engineering," *Computer-IEEE Computer Society*, vol. 39, no. 2, p. 25, 2006.

[25] D. G. Feitelson, "Workload modeling for performance evaluation," in *proceedings of the IFIP International Symposium on Computer Performance Modeling, Measurement and Evaluation*. Springer, 2002, pp. 114–141.

[26] P. G. Harrison, S. Harrison, N. M. Patel, and S. Zertal, "Storage workload modeling by hidden Markov models: Application to Flash memory," *Performance Evaluation Journal (PEVA)*, vol. 69, no. 1, pp. 17–40, 2012.

[27] S. J. Tarsa, A. P. Kumar, and H. Kung, "Workload prediction for adaptive power scaling using deep learning," in *proceedings of the International Conference on IC Design & Technology*. IEEE, 2014, pp. 1–5.

[28] Y. Zhu, J. Liu, M. Guo, W. Ma, and Y. Bao, "Acts in need: Automatic configuration tuning with scalability guarantees," in *proceedings of the 8th Asia-Pacific Workshop on Systems*. ACM, 2017, pp. 1–8.

[29] K. Wang, X. Lin, and W. Tang, "Predator—An experience guided configuration optimizer for Hadoop MapReduce," in *proceedings of the 4th International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2012, pp. 419–426.

[30] H. Yang, Z. Luan, W. Li, and D. Qian, "MapReduce workload modeling with statistical approach," *Journal of grid computing*, vol. 10, no. 2, pp. 279–310, 2012.

[31] H. Herodotou and S. Babu, "Profiling, what-if analysis, and cost-based optimization of mapreduce programs," *In proceedings of the VLDB Endowment*, vol. 4, no. 11, pp. 1111–1122, 2011.

[32] C.-C. Chen, Y.-T. Hasio, C.-Y. Lin, S. Lu, H.-T. Lu, and J. Chou, "Using deep learning to predict and optimize hadoop data analytic service in a cloud platform," in *proceedings of the 15th International Conference on Dependable, Autonomic and Secure Computing (DASC), 15th International Conference on Pervasive Intelligence and Computing (PiCom), 3rd International Conference on Big Data Intelligence and Computing (DataCom) and Cyber Science and Technology Congress (CyberSciTech)*. IEEE, 2017, pp. 909–916.

[33] P. C. O'Brien and T. R. Fleming, "A multiple testing procedure for clinical trials," *Biometrics*, pp. 549–556, 1979.

[34] M. Alosh, F. Bretz, and M. Huque, "Advanced multiplicity adjustment methods in clinical trials," *Statistics in medicine*, vol. 33, no. 4, pp. 693–713, 2014.

[35] N. Meinshausen, "Hierarchical testing of variable importance," *Biometrika*, vol. 95, no. 2, pp. 265–278, 2008.

[36] P. Lama and X. Zhou, "Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud," in *proceedings of the 9th international conference on Autonomic computing*, 2012, pp. 63–72.

[37] C.-O. Chen, Y.-Q. Zhuo, C.-C. Yeh, C.-M. Lin, and S.-W. Liao, "Machine learning-based configuration parameter tuning on hadoop system," in *proceedings 2015 IEEE International Congress on Big Data*. IEEE, 2015, pp. 386–392.

[38] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[39] B. S. Kim and T. G. Kim, "Cooperation between data modeling and simulation modeling for performance analysis of hadoop," in *proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*. IEEE, 2017, pp. 1–7.

[40] A. Erradi, W. Iqbal, A. Mahmood, and A. Bouguettaya, "Web application resource requirements estimation based on the workload latent features," *IEEE Transactions on Services Computing*, 2019.

[41] C. Stewart, K. Shen, A. Iyengar, and J. Yin, "Entomomodel: Understanding and avoiding performance anomaly manifestations," in *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 2010, pp. 3–13.

[42] E. Thereska, B. Doebel, A. X. Zheng, and P. Nobel, "Practical performance models for complex, popular applications," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 1, pp. 1–12, 2010.

[43] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 497–511, 2012.

[44] C.-J. Hsu, V. Nair, T. Menzies, and V. W. Freeh, "Scout: An experienced guide to find the best cloud configuration," *arXiv preprint arXiv:1803.01296*, 2018.

[45] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.

[46] E. Kocaguneli, T. Menzies, and E. Mendes, "Transfer learning in effort estimation," *Empirical Software Engineering*, vol. 20, no. 3, pp. 813–843, 2015.

[47] P. Jamshidi, M. Velez, C. Kästner, N. Siegmund, and P. Kawthekar, "Transfer learning for improving model predictions in highly configurable software," in *proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 2017, pp. 31–41.

[48] M. Mahmud and S. Ray, "Transfer learning using kolmogorov complexity: Basic theory and empirical evaluations," in *Advances in neural information processing systems*, 2008, pp. 985–992.

[49] E. Eaton, T. Lane *et al.*, "Modeling transfer relationships between learning tasks for improved inductive transfer," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2008, pp. 317–332.

[50] D. Agrawal, J. Giles, K.-W. Lee, K. Voruganti, and K. Filali-Adib, "Policy-based validation of SAN configuration," in *proceedings of the 5th International Workshop on Policies for Distributed Systems and Network*. IEEE, 2004, pp. 77–86.

[51] J. Yang, Z. Lu, N. Wang, J. Wu, and P. C. Hung, "Multi-policy-aware mapreduce resource allocation and scheduling for smart computing cluster," *Journal of Systems Architecture*, vol. 80, pp. 17–29, 2017.

[52] M. Devarakonda, D. Chess, I. Whalley, A. Segal, P. Goyal, A. Sachedina, K. Romanufa, E. Lassettre, W. Tetzlaff, and B. Arnold, "Policy-based autonomic storage allocation," in *proceedings of the International Workshop on Distributed Systems: Operations and Management*. Springer, 2003, pp. 143–154.

[53] B. Zhang, C. Wang, B. B. Zhou, D. Yuan, and A. Y. Zomaya, "DCDedupe: Selective deduplication and delta compression with effective routing for distributed storage," *Journal of Grid Computing*, vol. 16, no. 2, pp. 195–209, 2018.

[54] M. P. Szymaniak, "Latency-driven replication for globally distributed systems," Ph.D. dissertation, 2007.

[55] M. Al-Ayyoub, Y. Jararweh, E. Benkhelifa, M. Vouk, and A. Rindos, "A novel framework for software defined based secure storage systems," *Simulation Modelling Practice and Theory*, vol. 77, pp. 407–423, 2017.

[56] I. Polato, D. Barbosa, A. Hindle, and F. Kon, "Hybrid hdfs: decreasing energy consumption and speeding up hadoop using SSDs," *PeerJ PrePrints*, vol. 3, p. e1320v1, 2015.

[57] L. C. Miller and S. Fadden, *Software Defined Storage For Dummies*. John Wiley & Sons, Inc, 2014.

[58] S. Zertal, "Exploiting the fine grain SSD internal parallelism for OLTP and scientific workloads," in *proceedings of the Intl Conf on High Performance Computing and Communications (HPCC) , 6th Intl Symp on Cyberspace Safety and Security (CSS) , and 11th Intl Conf on Embedded Software and Syst (ICESS)*. IEEE, 2014, pp. 990–997.

[59] K. Kambatla and Y. Chen, "The truth about mapreduce performance on SSDs," in *proceedings of the 28th Large Installation System Administration Conference (LISA)*, 2014, pp. 118–126.

[60] S. Moon, J. Lee, and Y. S. Kee, "Introducing SSDs to the Hadoop mapreduce framework," in *proceedings of the 7th International Conference onCloud Computing (CLOUD)*. IEEE, 2014, pp. 272–279.

[61] K. Krish, M. S. Iqbal, and A. R. Butt, "Venu: Orchestrating SSDs in Hadoop storage," in *proceedings of the International Conference on Big Data (Big Data)*. IEEE, 2014, pp. 207–212.

[62] N. Poggi and D. Carrera, "Evaluating the impact of SSDsand infiniband in hadoop cluster performance and costs," *technical white paper. Barcelona Supercomputing Center*, 2015.

[63] R. Fuller, L. Avramov, and M. Portolani, *The Policy Driven Data Center with ACI: Architecture, Concepts, and Methodology*. Pearson Education, 2014.

[64] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, and S. Waldbusser, "Terminology for policy-based management," Tech. Rep., 2001.

[65] Y. Liu, N. Rameshan, E. Monte, V. Vlassov, and L. Navarro, "Prorenata: Proactive and reactive tuning to scale a distributed storage system," in *proceedings of the 15th International Symposium on Cluster, Cloud and Grid Computing*. IEEE/ACM, 2015, pp. 453–464.

[66] N. Devireddy and X. Chen, "Policy based storage configuration," 2003, US Patent 6,519,679.

[67] S. L. Howard, "Policy-driven management for distributed object systems," Ph.D. dissertation, 1999.

[68] E. Thereska, H. Ballani, G. O'Shea, T. Karagiannis, A. Rowstron, T. Talpey, R. Black, and T. Zhu, "IOFlow: a software-defined storage architecture," in *proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 182–196.

[69] G. Wu, X. He, and B. Eckart, "An adaptive write buffer management scheme for flash-based SSDs," *ACM Transactions on Storage (TOS)*, vol. 8, no. 1, p. 1, 2012.

[70] Y. Liu, X. Ge, X. Huang, and D. H. Du, "Molar: A cost-efficient, high-performance hybrid storage cache," in *proceedings of the International Conference on Cluster Computing (CLUSTER)*. IEEE, 2013, pp. 1–5.

[71] W. Li, G. Jean-Baptise, J. Riveros, G. Narasimhan, T. Zhang, and M. Zhao, "CacheDedup: In-line deduplication for flash caching," in *proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST)*, 2016, pp. 301–314.

[72] J. Kim, C. Lee, S. Lee, I. Son, J. Choi, S. Yoon, H.-u. Lee, S. Kang, Y. Won, and J. Cha, "Deduplication in SSDs: Model and quantitative analysis," in *proceedings of the 28th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2012, pp. 1–12.

[73] A. Zuck, S. Toledo, D. Sotnikov, and D. Harnik, "Compression and SSDs: Where and how?" in *proceedings of the 2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW)*, 2014.

[74] T. Makatos, Y. Klonatos, M. Marazakis, M. D. Flouris, and A. Bilas, "Using transparent compression to improve SSD-based I/O caches," in *proceedings of the 5th European conference on Computer systems*. ACM, 2010, pp. 1–14.

[75] Y. Chen, "We don't know enough to make a big data benchmark suite-an academia-industry view," *in proceedings of the WBDB*, vol. 74, 2012.

[76] T. Gouasmi, W. Louati, and A. H. Kacem, "Optimal MapReduce job scheduling algorithm across cloud federation," in *proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2017, pp. 88–93.

[77] D. B. Prats, J. L. Berral, and D. Carrera, "Automatic generation of workload profiles using unsupervised learning pipelines," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 142–155, 2017.

[78] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson, "Statistics-driven workload modeling for the cloud," in *proceedings of the 26th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 2010, pp. 87–92.

[79] P. Jamshidi, N. Siegmund, M. Velez, C. Kästner, A. Patel, and Y. Agarwal, "Transfer learning for performance modeling of configurable systems: An exploratory analysis," in *Proceedings of the 32nd International Conference on Automated Software Engineering (ASE)*. IEEE/ACM, 2017, pp. 497–508.

[80] R. Conradi and B. Westfechtel, "Version models for software configuration management," *ACM Computing Surveys (CSUR)*, vol. 30, no. 2, pp. 232–282, 1998.

[81] M. Sharma, "Database environmental change impact prediction for human-driven tuning in real-time (DECIPHER)," Ph.D. dissertation, Dakota State University, 2013.

[82] J. Mieścicki and W. B. Daszczuk, "Proposed benchmarks for PRT networks simulation," *arXiv preprint arXiv:1710.05754*, 2017.

[83] R. Han, L. K. John, and J. Zhan, "Benchmarking big data systems: A review," *IEEE Transactions on Services Computing*, vol. 11, no. 3, pp. 580–597, 2018.

[84] R. Han, S. Zhan, C. Shao, J. Wang, L. K. John, J. Xu, G. Lu, and L. Wang, "BigDataBench-MT: A benchmark tool for generating realistic mixed data center workloads," in *proceedings of the BPOE*. Springer, 2015, pp. 10–21.

[85] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench benchmark suite: Characterization of the mapreduce-based data analysis," in *proceedings of the 26th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 2010, pp. 41–51.

[86] "TPC Benchmarks," [Online] Available : http://www.tpc.org/information/benchmarks.asp, last accessed on 1 January 2020.

[87] Tintri, "State of storage 2016 for virtualized enterprises," [Online] Available : https://www.tintri.com/resources/whitepapers/2016-state-storage-virtualized-enterprises, last accessed on 30 November 2019.

[88] A. Alba, G. Alatorre, C. Bolik, A. Corrao, T. Clark, S. Gopisetty, R. Haas, R. I. Kat, B. Langston, and N. Mandagere, "Efficient and agile storage management in software defined environments," *IBM Journal of Research and Development*, vol. 58, no. 2/3, pp. 5–1, 2014.

[89] R. Kaur, I. Chana, and J. Bhattacharya, "Data deduplication techniques for efficient cloud storage management: a systematic review," *The Journal of Supercomputing*, vol. 74, no. 5, pp. 2035–2085, 2018.

[90] R. Gracia-Tinedo, J. Sampé, E. Zamora, M. Sánchez-Artigas, P. García-López, Y. Moatti, and E. Rom, "Crystal: Software-defined storage for multi-tenant object stores," in *proceedings of the 15th USENIX Conference on File and Storage Technologies(FAST)*. USENIX Association, 2017.

[91] M. Usama and N. Zakaria, "Chaos-based simultaneous compression and encryption for hadoop," *PloS one*, vol. 12, no. 1, 2017.

[92] C. Vorapongkitipun and N. Nupairoj, "Improving performance of small-file accessing in hadoop," in *Proceedings of the 11th international joint conference on computer science and software engineering (JCSSE)*. IEEE, 2014, pp. 200–205.

[93] A. Argyriou, A. Maurer, and M. Pontil, "An algorithm for transfer learning in a heterogeneous environment," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2008, pp. 71–85.

[94] S. Hayashi and N. Komoda, "Evaluation of SSD tier method and SSD cache method in tiered storage system," in *proceedings of the Technical Meeting on Information Systems*, 2013, pp. 51–56.

[95] Y. Chen, A. Ganapathi, and R. H. Katz, "To compress or not to compress-compute vs. io tradeoffs for mapreduce energy efficiency," in *proceedings of the 1st workshop on Green networking(SIGCOMM)*. ACM, 2010, pp. 23–28.

[96] Z. Li, A. Mukker, and E. Zadok, "On the importance of evaluating storage systems costs," in *proceedings of the 6th Workshop on Hot Topics in Storage and File Systems (HotStorage)*. USENIX, 2014.

[97] J. Zhang, D. Feng, J. Gao, W. Tong, J. Liu, Y. Hua, Y. Gao, C. Fang, W. Xia, and F. Fu, "Application-aware and software-defined SSD scheme for tencent large-scale storage system," in *proceedings of the 22nd international conference on parallel and distributed systems (ICPADS)*. IEEE, 2016, pp. 482–490.

[98] W.-S. E. Chen, C.-F. Huang, and M.-J. Huang, "iSDS: a self-configurable software-defined storage system for enterprise," *Enterprise Information Systems*, pp. 1–22, 2016.

[99] S. Al-Kiswany, L. B. Costa, H. Yang, E. Vairavanathan, and M. Ripeanu, "A cross-layer optimized storage system for workflow applications," *Future Generation Computer Systems*, vol. 75, pp. 423–437, 2017.

[100] R. Gracia-Tinedo, P. García-López, M. Sánchez-Artigas, J. Sampé, Y. Moatti, E. Rom, D. Naor, R. Nou, T. Cortés, and W. Oppermann, "IOStack: Software-defined object storage," *IEEE Internet Computing*, vol. 20, no. 3, pp. 10–18, 2016.

[101] W. Li, G. Jean-Baptise, J. Riveros, G. Narasimhan, T. Zhang, and M. Zhao, "CacheDedup: In-line deduplication for flash caching," in *proceedings of the 14th Conference on File and Storage Technologies (FAST)*. USENIX, 2016, pp. 301–314.

[102] R. Koller, A. J. Mashtizadeh, and R. Rangaswami, "Centaur: Host-side SSD caching for storage performance control," in *proceedings of the International Conference on Autonomic Computing*. IEEE, 2015, pp. 51–60.

[103] K. Rishabh, V. Mehta, T. Jha, and V. Varma, "HetStore: A platform for IO workload assignment in a heterogeneous storage environment," in *proceedings of the*

International Conference on Cloud Computing in Emerging Markets (CCEM). IEEE, 2016, pp. 25–31.

[104] Apache mahout. [Online] Available : https://mahout.apache.org/. Last accessed on 1 April 2020.

[105] W. Gao, J. Zhan, L. Wang, C. Luo, D. Zheng, F. Tang, B. Xie, C. Zheng, X. Wen, and X. He, "Data motifs: a lens towards fully understanding big data and ai workloads," *arXiv preprint arXiv:1808.08512*, 2018.

[106] E. L. Miller, "Input/output behavior of supercomputing applications," in *proceedings of the conference on Supercomputing*. ACM/IEEE, 1991, pp. 567–576.

[107] E. Smirni and D. A. Reed, "Lessons from characterizing the input/output behavior of parallel scientific applications," *Performance Evaluation*, vol. 33, no. 1, pp. 27–44, 1998.

[108] Z. Ghahramani, "An introduction to hidden markov models and bayesian networks," in *Hidden markov models: applications in computer vision*. World Scientific, 2001, pp. 9–41.

[109] S. He, Y. Wang, Z. Li, X.-H. Sun, and C. Xu, "Cost-aware region-level data placement in multi-tiered parallel I/O systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 1853–1865, 2017.

[110] L. Bouganim, B. Jónsson, and P. Bonnet, "uFLIP: Understanding flash IO patterns," *arXiv preprint arXiv:0909.1780*, 2009.

[111] J. Oly and D. A. Reed, "Markov model prediction of I/O requests for scientific applications," in *proceedings of the 16th international conference on Supercomputing*. ACM, 2002, pp. 147–155.

[112] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *In proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[113] seqhmm: Mixture hidden markov models for social sequence data and other multivariate, multichannel categorical time series. [Online] Available : https://cran.r-project.org/package=seqHMM. R package version 1.0.13, Last accessed on 1 December 2019.

[114] S. Helske and J. Helske, "Mixture hidden Markov models for sequence data: The seqHMM package in R," *Journal of Statistical Software*, vol. 88, no. 3, pp. 1–32, 2019.

[115] X. Shi, W. Fan, and J. Ren, "Actively transfer domain knowledge," in *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2008, pp. 342–357.

[116] D. Li, J. Zhang, Y. Yang, C. Liu, Y.-Z. Song, and T. M. Hospedales, "Episodic training for domain generalization," in *Proceedings of the International Conference on Computer Vision*. IEEE, 2019, pp. 1446–1455.

[117] J. A. Pereira, H. Martin, M. Acher, J.-M. Jézéquel, G. Botterweck, and A. Ventresque, "Learning software configuration spaces: A systematic literature review," *arXiv preprint arXiv:1906.03018*, 2019.

[118] N. Freris, Y. Jararweh, and M. Al-Ayyoub, "SDCache: Software defined data caching control for cloud services," in *proceedings of the 4th International Conference on future Internet of Things and Cloud (FiCloud)*. IEEE, 2016, pp. 164–169.

Acronyms

API Application Programming Interface. 45, 46

CPU Central processing unit. 7, 27, 40, 45, 61, 70, 76, 79, 80, 91

DSS Decision Support System. 33

HDD Hard Disk Drive. 2, 4, 23, 24, 30, 61, 62, 69, 71, 75, 76, 78, 80, 86, 87, 89

HDFS Hadoop File System. 8, 11, 13, 35, 45, 61, 64, 66, 75, 78, 79, 85

HMM Hidden Markov Model. iv, 4, 8, 9, 88, 92, 94–97, 99, 100, 102, 103, 106

HW Hardware. 2, 11, 24, 34, 44, 52–54

I/O Input/Output. 1–5, 7–9, 11, 22–25, 27–29, 45, 56, 60, 61, 69, 70, 72–77, 79, 80, 84, 87–95, 97–101, 103, 105, 106

IaaS Infrastructure as a Service. 16, 31, 48

IFTTT If-This-Then-That. 26, 56

IOPS Input/Output Operations Per Second. 24, 71

ML Machine Learning. 2, 10, 17–19, 21, 29, 31, 34, 36, 46, 57

MLC Multi-Level Cell. 80

MLP Multi-Layer Perceptron. 38–40

OLTP Online Transaction Processing. 13, 30, 33

P/E Program/Erase. 24, 25

QLC Quad-Level Cell. 89

R/W Read/Write. 3, 69, 75, 82, 83, 88–90, 92

RAID Redundant Array of Independent Disks. 22, 53

SaaS Software as a Service. 31, 48

SAN Storage Area Network. 27, 72

SDS Software-Defined Storage. 27, 54, 73, 87, 94, 99

SLA Service Level Agreement. 27, 54, 87, 99

SLC Single-Level Cell. 89, 90

SSD Solid State Drive. iii, iv, 2, 4, 5, 7–9, 23–26, 28, 29, 56, 61, 62, 69–80, 82–91, 93–95, 98–103, 105, 106, 108

SW Software. 2, 6, 11, 32, 34, 35, 41, 44, 52–54

TL Transfer Learning. iii, 7, 10, 18–21, 31, 38, 39, 43, 47, 50–52, 55, 57–59, 67, 105–107

TLC Triple-Level Cell. 80, 90

VM Virtual Machine. 87

YARN Yet Another Resource Negotiator. 11, 13, 35, 42, 61, 83, 85