Analysis and comparison of interfacing, data generation and workload implementation in BigDataBench 4.0 and Intel HiBench 7.0

ALEXANDER BAROSEN
SADOK DALIN
Analysis and comparison of interfacing, data generation and workload implementation in BigDataBench 4.0 and Intel Hibench 7.0

SADOK DALIN & ALEXANDER BAROSEN
Abstract

One of the major challenges in Big Data is the accurate and meaningful assessment of system performance. Unlike other systems, minor differences in efficiency can escalate to large differences in costs and power consumption. While there are several tools on the marketplace for measuring the performance of Big Data systems, few of them have been explored in-depth.

This report investigated the interfacing, data generation and workload implementations of two Big Data benchmarking suites, BigDataBench and Hibench. The purpose of the study was to establish the capabilities of each tool with regards to interfacing, data generation and workload implementation.

An exploratory and qualitative approach was used to gather information and analyze each benchmarking tool. Source code, documentation, and reports published by the developers were used as information sources.

The results showed that BigDataBench and HiBench were designed similarly with regards to interfacing and data flow during the execution of a workload with the exception of streaming workloads. BigDataBench provided for more realistic data generation while the data generation for HiBench was easier to control. With regards to workload design, the workloads in BigDataBench were designed to be applicable to multiple frameworks while the workloads in HiBench were focused on the Hadoop family. In conclusion, neither of benchmarking suites was superior to the other. They were both designed for different purposes and should be applied on a case-by-case basis.

Keywords: Big Data, Benchmarking, BigDataBench, HiBench, Analysis, Comparison, Interfacing, Data Generation
Sammanfattning

En av de stora utmaningarna i Big Data är den exakta och meningsfulla bedömningen av systemprestanda. Till skillnad från andra system kan mindre skillnader i effektivitet eskalera till stora skillnader i kostnader och strömförbrukning. Medan det finns flera verktyg på marknaden för att mäta prestanda för Big Data-system, har få av dem undersökt djupgående.

I denna rapport undersöktes gränssnittet, datagenereringen och arbetsbelastningen av två Big Data benchmarking-sviter, BigDataBench och HiBench. Syftet med studien var att fastställa varje verktygs kapacitet med hänsyn till de givna kriterierna.

Ett utforskande och kvalitativt tillvägagångssätt användes för att samlar information och analysera varje benchmarking verktyg. Källkod, dokumentation och rapporter som hade skrivits och publicerats av utvecklarna användes som informationskällor.

Resultaten visade att BigDataBench och HiBench utformades på samma sätt med avseende på gränssnitt och dataflöde under utförandet av en arbetsbelastning med undantag för strömmade arbetsbelastningar. BigDataBench tillhandahöll mer realistisk datagenerering medan datagenerering för HiBench var lättare att styra. När det gäller arbetsbelastningsdesign var arbetsbelastningen i BigDataBench utformad för att kunna tillämpas på flera ramar, medan arbetsbelastningen i HiBench var inriktad på Hadoop-familjen. Sammanfattningsvis var ingen av benchmarkingssuperna överlägsen den andra. De var båda utformade för olika ändamål och bör tillämpas från fall till fall.

Nyckelord: Big Data, Benchmarking, BigDataBench, HiBench, Analys, Jämförelse, Gränssnitt, Datagenerering
# Contents

1 Introduction  
  1.1 Problem ................................. 2  
  1.2 Purpose ................................. 3  
  1.3 Goal .................................. 4  
  1.4 Benefits, Ethics and Sustainability .......... 4  
  1.5 Methodology ............................. 4  
  1.6 Delimitations ........................... 5  
  1.7 Outline ................................ 6  

2 Background  
  2.1 Concepts and Terminology .................. 7  
  2.2 Big Data Software Stack .................... 8  
  2.2.1 Data Processing ........................ 9  
  2.2.2 Data Storage .......................... 10  
  2.2.3 Resource Management .......................... 11  
  2.3 The Benchmarking Process .................... 11  
  2.3.1 Data Generation ....................... 11  
  2.3.2 Workload Execution .................... 12  
  2.3.3 Report Generation .................... 13  
  2.4 Previous Work ............................ 14  

3 Method  
  3.1 General Methodology ..................... 17  
  3.2 Benchmarking Tool Comparison Criteria ...... 18  
  3.2.1 Interfacing ............................ 19  
  3.2.2 Data Generation ....................... 19  
  3.2.3 Workload Implementation ............... 21
4 Analysis Method
  4.1 Interfacing ........................................... 24
  4.2 Data Generation .................................... 25
  4.3 Workload Implementation ........................... 26

5 Results
  5.1 BigDataBench ........................................ 29
    5.1.1 Interfacing ...................................... 29
    5.1.2 Data Generation ................................. 30
    5.1.3 Workload Implementation ...................... 33
  5.2 HiBench .............................................. 36
    5.2.1 Interfacing ...................................... 36
    5.2.2 Data Generation ................................. 37
    5.2.3 Workload Implementation ...................... 42

6 Discussion
  6.1 Interfacing ........................................... 48
  6.2 Data Generation .................................... 48
  6.3 Workload Implementation ........................... 50

7 Conclusion ............................................... 53

References .................................................. 55
## Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>4V</td>
<td>Acronym for Volume, Variety, Velocity and Veracity.</td>
</tr>
<tr>
<td>Activation Function</td>
<td>Determines the output of a neuron in a neural network.</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence.</td>
</tr>
<tr>
<td>Analytics</td>
<td>Discovery of meaningful patterns in data.</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface. Set of methods for communicating with software.</td>
</tr>
<tr>
<td>Benchmarking</td>
<td>Measurement of performance resulting in a metric which can be compared.</td>
</tr>
<tr>
<td>Bioinformatics</td>
<td>Umbrella term for software tools that process biological data.</td>
</tr>
<tr>
<td>Collaborative Filtering</td>
<td>Recommender system technique.</td>
</tr>
<tr>
<td>DAG</td>
<td>Directed Acyclic Graph. Finite and directed graph with no cycles.</td>
</tr>
<tr>
<td>Data Center</td>
<td>Industry scale facility for housing computers and data storage systems.</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database Management System. For creating and managing databases.</td>
</tr>
<tr>
<td>DFSIO</td>
<td>Distributed File-System IO. Hadoop benchmark for measuring the HDFS read and write capacity.</td>
</tr>
<tr>
<td><strong>Term</strong></td>
<td><strong>Definition</strong></td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DMS</td>
<td>Distributed Messaging System. Sends and receives messages between systems.</td>
</tr>
<tr>
<td>Dwarf</td>
<td>Abstraction of frequently occurring computations.</td>
</tr>
<tr>
<td>EIBD</td>
<td>External Interface Block Diagram. Illustrates the system of interest and other elements on the same architectural level.</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier Transform. Algorithm that samples and divides a signal.</td>
</tr>
<tr>
<td>Framework</td>
<td>Software providing generic and modifiable functionality.</td>
</tr>
<tr>
<td>GDPR</td>
<td>General Data Protection Regulation. EU law regulation regarding data privacy.</td>
</tr>
<tr>
<td>Gearpump</td>
<td>Event-based real-time streaming engine.</td>
</tr>
<tr>
<td>Hadoop</td>
<td>A collection of software-based libraries for processing data over multiple computers.</td>
</tr>
<tr>
<td>HDFS</td>
<td>Hadoop Distributed File System.</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output.</td>
</tr>
<tr>
<td>K-means</td>
<td>Clustering method. Assigns observations to k different clusters.</td>
</tr>
<tr>
<td>Kronecker product</td>
<td>Operation of two matrices resulting in a block matrix.</td>
</tr>
<tr>
<td>Label</td>
<td>Machine Learning concept. Output which classifies a phenomenon.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>Method used for modelling the relationship between variables.</td>
</tr>
<tr>
<td>Machine Learning</td>
<td>Subfield of AI which focuses on using previously known data to recognize patterns.</td>
</tr>
<tr>
<td>MapReduce</td>
<td>Programming model and implementation based on parallel programming.</td>
</tr>
<tr>
<td>MD5 Hash</td>
<td>128-bit hash function.</td>
</tr>
<tr>
<td>MLlib</td>
<td>Machine Learning Library. Included in Apache Spark.</td>
</tr>
<tr>
<td>MPI</td>
<td>Message Passing Interface.</td>
</tr>
<tr>
<td>Neural Network</td>
<td>A collection of neurons.</td>
</tr>
<tr>
<td>NoSQL</td>
<td>Non Structured Query Language. For management of data in other forms than tabular relational databases.</td>
</tr>
<tr>
<td>Nutch</td>
<td>A extensible and open source web crawler.</td>
</tr>
<tr>
<td>NWeight</td>
<td>Calculates the association(weight) between two vertices that are n-hop away.</td>
</tr>
<tr>
<td>Pagerank</td>
<td>Ranking algorithm used to rank websites in search engine results.</td>
</tr>
<tr>
<td>Principal Component Analysis</td>
<td>Statistical procedure.</td>
</tr>
<tr>
<td>Random Forest</td>
<td>Machine Learning method used for classification.</td>
</tr>
<tr>
<td>RandomTextWriter</td>
<td>Hadoop function. Generates random text.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
</tr>
<tr>
<td>RDD</td>
<td>Resilient Distributed Dataset. Immutable storage abstraction found in Spark.</td>
</tr>
<tr>
<td>SIGMOD</td>
<td>Special Interest Group on Management of Data.</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language. Programming language for managing relational database data.</td>
</tr>
<tr>
<td>SVD</td>
<td>Singular Value Decomposition. Factorization of a matrix.</td>
</tr>
<tr>
<td>Teragen</td>
<td>A Hadoop data generator. Generates data for the Terasort benchmark.</td>
</tr>
<tr>
<td>TeraSort</td>
<td>Benchmark. Measures the time it takes to sort a terabyte of random data.</td>
</tr>
<tr>
<td>Topic</td>
<td>Organization of streaming messages. All streaming messages are organized into topics.</td>
</tr>
<tr>
<td>Veracity</td>
<td>Accuracy of data from a data generation.</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine. Emulation of a computer.</td>
</tr>
<tr>
<td>VoIP</td>
<td>Voice over IP.</td>
</tr>
<tr>
<td>Wordcount</td>
<td>A word count algorithm which uses the MapReduce features.</td>
</tr>
<tr>
<td>Workload</td>
<td>The processing done by the framework as part of a benchmark.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Big Data is quickly growing to become one of the largest industries in the IT sector[50]. To support the increase in popularity of services such as Google, Youtube and Amazon new data centers are constantly being built in various parts of the world. A special characteristic of Big Data system is that minor differences in efficiency can escalate to large differences in costs and power consumption. One important stepping stone in improving the efficiency and design of Big Data systems is accurate benchmarking [27, 28].

One of the major challenges in Big Data is the accurate and meaningful assessment of system performance [28]. While there are several benchmarking tools available, there is no established standard that they can be designed against. Stacked software and complex application scenarios further increase the difficulty of measuring performance metrics such as throughput, latency and execution time. Well developed benchmarking suites are also becoming increasingly important in order to find flaws in Big Data systems such as high operational system costs or lacking large state operation performance. With accurate information about low and high-level performance metrics, the design of large systems could be improved [26, 27].

The benchmarking process of Big Data frameworks can be divided roughly into three sequential steps [27]. In the first step, input data is generated or fetched for a benchmark. The second step consists of executing a workload i.e. executing the benchmark code on a framework. After the workload has been executed a report is generated with
measurements of the framework’s performance. There are multiple ways to design each step in the benchmarking process with different strengths and weaknesses.

Currently, benchmarking tools are designed after two major philosophies. The first and most practical philosophy is to create a benchmark for each possible application scenario [21]. A problem with this approach is scaling the number of benchmarks to the ever-increasing number of application scenarios. The second and more recent approach is to abstract out common units of computation (Dwarfs) from a large number of application scenarios and then use these basic components to simulate arbitrary workloads [21]. The lack of standards in benchmarking is making it difficult to compare results from different benchmarking tools and evaluate performance in a reliable manner. While there are organizations that are working with standardizing benchmarking of Big Data systems [19, 55], there is no consensus on which methodology is the most appropriate.

There exists a couple of studies where Big Data benchmarking tools have been analyzed and compared. Han et al. [28] provided an overarching comparison between 16 different benchmarking tools with a focus on data generation in a comparison study published in 2015. A study by Han, Lu, and Xu [27] published in 2014 also provides an in-depth comparison of the data generation methodology used by ten different benchmarking tools and an overview of the difference between the tools workloads and software stacks. Since 2015, however, both Hi-Bench and BigDataBench have received major updates adding several new benchmarks and enabling compatibility with more frameworks rendering both of these reports outdated.

1.1 Problem

There are several gaps in current research. One major problem is establishing a standard or a set of metrics which benchmarking tools can be compared against. Nested within this problem is establishing which metrics are beneficial to measure, and the methodology for measuring them properly. Another problem is estimating the current development of benchmarking tools and in what scenarios each tool is favor-
able to use. This problem is related to the lackluster documentation provided with most benchmarking tools. In this report, we focused on the second problem by comparing two popular benchmarking suites: BigDataBench 4.0 and Intel Hibench 7.0.

Both BigDataBench and HiBench have lacking documentation regarding details about their respective software. The documentation does not cover the specifications of each test, about their limitations and boundaries. This could make it harder to make an informed decision on which to use.

To compare BigDataBench and HiBench several comparison criteria were considered. The most important considerations were choosing criteria that correlated with each part of the benchmarking process while providing more in-depth information than the original documentation. Data generation was an obvious first choice since it covers a large part of the benchmarking process. The interfacing between each tool and the framework was also a suitable criterion since it provided a deeper understanding of the data flow between components beyond the specification. An investigation of the workload implementation bridged the gap between the specification and the source code while also illustrating the breadth of BigDataBench and HiBench.

The research questions are stated as: How does Intel HiBench 7.0 and BigDataBench 4.0 differ with regards to interfacing, data generation, and workload implementation?

1.2 Purpose

The purpose of this report was two-fold. One purpose was to analyze and compare the interfacing, data generation methodology, and workload implementation of Intel HiBench and BigDataBench, establishing the capabilities of each tool. The second purpose was to find an appropriate methodology for analyzing the benchmarking tools.
1.3 Goal

The main goal of this thesis was to establish the capabilities of Big-DataBench and HiBench with regards to interfacing, data generation, and workload implementation. Achieving this goal would contribute to the performance tuning of Big Data systems.

1.4 Benefits, Ethics and Sustainability

Data centers operating around the clock present a large and increasing power drain. In 2017 data centers generated 2% of the global CO2 emissions[43]. With the increased amount of data centers being built, the power and cooling costs have increased in proportion[50].

The costs associated with the operational costs of data centers have shifted the focus from the performance of hardware to energy efficiency. To ascertain the energy efficiency of different hardware and software designs extensive and accurate benchmarks are necessary. This work benefits this cause by analyzing and contrasting two major Big Data benchmarking suites.

An ethical concern with Big Data systems is the storage and usage of private information. Before GDPR private information could be used freely by companies such as Facebook and Twitter to track users and give personalized suggestions. Even though the legislation gives more control to users the data is still collected.

1.5 Methodology

An exploratory approach was used during the project. Initially, an overview of benchmarking and data generation approaches was done by investigating relevant literature. The literature included both overviews of benchmarking methodology and specific benchmarking tool documentation. The analysis of Intel HiBench and BigDataBench was done by reading relevant documentation, source code and then evaluating the capacity and scope of the tool with regards to interfacing, data generation, and workload implementation. The analysis criteria
were chosen to have a clear relation to the data generation and workload implementation phases of the benchmarking process.

1.6 Delimitations

There were several delimiting decisions made during the project. One major choice was avoiding an experimental approach, which the initial work was based on. The primary problem with an experimental approach would have been to establish an accurate baseline for relevant metrics such as latency and throughput. Without knowing the correct value of the metrics beforehand measuring the accuracy of the HiBench and BigDataBench would have been impossible. The analysis of the frameworks which the benchmarking tools work against was also minimized since it is out of the scope of a thesis project.

The choice of tools was also a delimiting factor. The most common approach in similar works was to give an overview of every benchmarking tool that existed. A problem with this approach is that it is hard to make an in-depth analysis of each tool and subsequently a meaningful comparison. In this project, the purpose was to make an in-depth comparison which is why the number of tools was limited to two. HiBench and BigDataBench were chosen because they were recently updated\cite{30, 36}, extensive\cite{52} and popular.

Another limiting factor was the lack of standards in Big Data benchmarking. The lack of standards meant that analyzing and comparing a single benchmarking tool to an established practice or requirements specification was not possible. Ergo, a comparison of two different benchmarking tools was chosen as the aim of the project. Comparing a single tool against an established practice is also not so interesting since different benchmarking tools are created for different application scenarios.

To be able to investigate the data generation and workload implementation in depth in the time allotted the report generation of the benchmarking process was not analyzed.
1.7 Outline

In Chapter 1 the thesis is introduced. The first section gives a brief background on Big Data benchmarking and current gaps in research followed by a presentation of the problem and research statement. Ethics and sustainability issues related to Big Data and this project are also explained. The chapter ends with a presentation of the methodology and delimitations of the study.

In Chapter 2 the background of the project is presented. The chapter begins with a definition of Big Data and other important concepts. In the following sections the software stack, benchmarking process, and previous work are explained.

In Chapter 3 the methodology used during the thesis is explained and motivated. The first section explains the general methodology. The latter sections explain and motivate the comparison criteria in detail.

Chapter 4 covers the analysis methodology used during the thesis work. The first section explains how information was collected from different sources. Each following section explains how each of the comparison criteria was applied to the gathered information.

In Chapter 5 the results of the study are given. The results were divided into 2 major sections, one for each benchmarking tool. For each benchmarking tool results related to each evaluation criteria were listed in the same order as Chapter 3.

In Chapter 6 the results are discussed and providing a foundation to our conclusions about the thesis. Initially, the problem is reiterated and the methodology used during the project is discussed. In the following sections, discussions regarding interfacing, data generation, and workload implementation are presented.

In Chapter 7, our conclusions are presented. The report is concluded with comments on future work.
Chapter 2

Background

In this chapter, the background for the study is presented. In section 2.1 the important concepts and major components and benchmarking of Big Data are explained. In section 2.2 the benchmarking process i.e. data generation, workload execution and report generation is explained. The common components of a Big Data software stack are explained in section 2.3. Related studies on benchmarking tools are covered in section 2.4.

2.1 Concepts and Terminology

Big Data is defined in relation to the complexity of the data. While the word "big" is usually interpreted to indicate the size of the data, large data sets that are well structured are not considered as Big Data. The word big should rather be interpreted to indicate the number of permutations of the data that can be used to extract useful information i.e. the complexity of the data[15]. A part of the definition of Big Data is that it cannot be handled using traditional data-processing methods[15].
One of the primary strengths of Big Data frameworks is that they are, unlike traditional data processing systems, designed to handle semi-structured and unstructured data in addition to structured data [56]. Traditional or structured data is clearly organized with the help of records containing fields and is usually stored in blocks of data. Semi-structured data is similar to structured data in that it contains some identification or tags for managing the data. Unstructured data does not have a predefined model. Examples of unstructured data are the contents of a VOIP call, posts on Facebook and sensor data.

Big Data software and frameworks can be grouped and divided into three major layers and are illustrated in Figure 2.1 [37]. The top layer consists of data processing frameworks that are responsible for transforming input data. Storage systems in the middle layer are responsible for storing data in a distributed fashion. The lowest layer consists of resource management systems which are responsible for scheduling jobs and allocating resources. The components in each layer can generally be exchanged for another on the same layer, which affects the stacks functionality and performance.

A benchmark is an act of running a program in order to assess the performance of an object. The result of running a benchmark is a measurement of a set of evaluation criteria such as execution time, throughput and power draw.

### 2.2 Big Data Software Stack

A Big Data system can be divided into three layers [37]. The top layer is responsible for processing and transforming input data, and is explained in chapter 2.2.1. Data is stored and distributed in the middle layer which is presented in chapter 2.2.2. The frameworks in the lowest layer are responsible for managing resources when multiple frameworks run on the same cluster and are briefly explained in chapter 2.2.3. Hadoop is a collection of open source distributed processing and storage tools that span all three layers and include MapReduce [24], HDFS [23], and YARN [4].
2.2.1 Data Processing

Data processing frameworks and engines compute the data in a Big Data System. The goal of the processing systems is to find patterns and understand interactions in the data. A simple way to categorize processing frameworks and engines is by the type of data they are designed to handle.

Batch processing systems take abounded, often persistent dataset as input and perform calculations on every element in the set. MapReduce [24] is a core programming model and implementation designed to process large datasets. The MapReduce model is composed of a number of mappers and reducers. Mappers filter and sort while re-
Producers summarize the data. Apache Spark [11] trades the capacity of MapReduce to process large datasets for processing speed. Apache Mahout [9] is a linear algebra framework which can be run on Spark.

Stream Processing systems compute continuously as data enters the system. Unlike batch processing, the dataset is defined as the total amount of data that has entered the system. Apache Storm [14] is dedicated stream processor. JStorm [38] is an alternative implementation of Storm. Apache Flink [2] is a unified stream and batch processing framework.


Graph processing frameworks are tailored mine data from graphs with vertices and edges as input. Graphx [12] is a graph processing framework which expands upon Spark to store and analyze graphs.

Machine Learning frameworks are used to identify patterns in recorded data. These patterns are then used to predict future outcomes. Caffe [18] is an open source machine learning framework specialized in processing images. TensorFlow [53] is an open source machine learning software library.

2.2.2 Data Storage

Data storage frameworks and systems are responsible for handling and distributing data in a server cluster. The commonality between distributed file storage systems is that they are designed to run on low-cost hardware with a high amount of fault-tolerance. HDFS [23] is the archetypal example of such a system, which replicates and manages data in a server cluster.

NoSQL databases support data modelled in other means than SQL data. Common types of storage include column, key-value, document, multi-model and graph stores. Cassandra [1] and HBase [5]
are column-store database management systems. MongoDB [45] is a
document-store database management system.

Distributed Messaging Systems (DMS) are used to stream data to pro-
cessing frameworks. DMS are built upon the publish/subscribe archi-
tecture where an application can publish a stream of records to a topic
or subscribe to a topic to receive a stream of records. Examples of DMS
include Apache Kafka [8], Gearpump [3] and Mpich [46].

2.2.3 Resource Management

Resource management systems were designed to handle several Big
Data frameworks running on the same platform. Unlike traditional
solutions such as using VMs or separate clusters, running multiple
frameworks on a single cluster enables elasticity and maximizes uti-
lization of available resources. YARN (Yet Another Resource Negotia-
tor) [4] is an example of a resource manager.

2.3 The Benchmarking Process

The benchmarking process of Big Data systems can be divided into
three major parts or aspects, usually performed in sequence. The ini-
tial step is workload input data generation which is explained in sec-
tion 2.3.1. After the data has been generated a workload is executed
with the data as input. This step is presented in section 2.3.2. In the
last step, a report is generated with a measure of different metrics such
as latency, execution time and availability. Report generation and the
related performance metrics are explained in section 2.3.3.

2.3.1 Data Generation

The data in Big Data workloads is commonly characterized by four
different properties. These four V are known as Volume, Variety, Ve-
locity, and Veracity. Volume represents the size of the data, often mea-
sured in bytes or in the number of data entries. Velocity represents ei-
ther the data generation rate of the system, the data update frequency
or the data processing speed depending on the system type. Variety represents the different types of data in the system. Veracity is a measure of how much the input data conforms to the characteristics of raw data[28].

Input data can be generated using different methods, each with its own strengths and weaknesses. Real life datasets are used to preserve the veracity of the data and can be obtained from various sources such as Amazon[47] and Wikipedia[58]. The main limitation of real-life datasets is that they are hard to scale.

Using data generators is the alternative to using ready-made datasets. Data generators can be based on synthetic distributions, real data or a hybrid. Synthetic distributions include uniform and power-law and are used differently depending on the test. Generators based on real data learn properties from real data and then generate data with a similar distribution, maintaining the veracity of the data with the option to control the volume and velocity. Hybrid data generators use samples from real datasets in combination with generated data.

### 2.3.2 Workload Execution

The second part of the benchmarking process is the workload submission and execution. The workload is the work done by the framework as part of the benchmark. Submission and execution are done by injecting an executable file into the framework. Workloads can be divided into different categories depending on how extensive the test is or what its purpose is.

On a high abstraction level, workloads can be categorized according to their target application domain[36]. Common application domains include search engines, social networks, e-commerce, multimedia processing and bioinformatics.

Another common categorization is dividing Big Data workloads in relation to their scope. The most prevalent categories are micro, functional, genre-specific and application-level workloads. Micro workloads evaluate lower-level functions and include sort, write and read
workloads. Functional workloads test a specific high-level function. Genre-specific workloads are executed using a specific type of data such as graphs or SQL records. Application-level workloads are used to provide a realistic measurement of the performance of the system by using real-world input data and workloads[20].

Composing benchmarks that can be used to evaluate systems with different data processing methods is difficult. The simplest and most common method is to create an application-level workload for each use case. While the accuracy using method is relatively high, it is not scalable. The alternative suggested by Gao et al. [21] is to identify frequent units of computation, "dwarfs", that are common to each workload through workload analysis and decomposition. By profiling larger workloads, these dwarfs can be combined in a weighted DAG-like structure to simulate the workload.

2.3.3 Report Generation

In the last part of the benchmarking process, a report is generated containing the statistics about a specific test. Different evaluation metrics are analyzed and compiled in the report depending on the benchmarking tool and type of test. Evaluation metrics can be categorized into performance, price and energy metrics [26].

Performance metrics can be divided into user-perceivable metrics and architecture metrics. User-perceivable metrics are used to test the performance of the system of interest while architecture metrics measure the performance of the underlying hardware.

Response time is a performance metric used to measure the latency between the submission and response of a workload execution. Throughput measures the number of operations per time unit. Operations can be defined as queries, processed data or completed jobs depending on the nature of the benchmark. Reliability measures the relative amount of successful operations. Availability measures the ratio of available system time.
2.4 Previous Work

There are several reports which analyze and compare benchmarking tools and suites. In the BigDataBench 4.0 report by Huang et al. [36], an overview of 11 different benchmarking suites was provided. The suites were categorized according to overall methodology, application domains, amount of workload types, amount of workloads, amount of datasets and amount of software stacks. A weakness with this categorization is that it does not estimate the scope of the suites in each workload type.

Han, Lu, and Xu [27] published a report in 2013 containing an overview of the data generation and workload implementation of ten Big Data benchmarking suites. In addition to the overview, the benchmarking process, 4V, and benchmarking techniques were discussed. In the report it is not mentioned which version of BigDataBench and HiBench is analyzed which makes it hard to verify the results. The report is also outdated. Both Bigdatabench and HiBench have been modified since this report was published. BigDataBench 4.0 [36], released 1st April 2018, added seven real-world datasets and 15 workloads over BigDataBench 3.0 [16]. A new category of AI workloads was also implemented. HiBench 6.0 [33], released in November 2016, added four streaming and one graph workloads. The configuration file architecture was also revised. HiBench 7.0 [33], released in November 2017, added 8 Machine Learning workloads.

Han et al. [28] published a report in 2015 detailing comparing 16 different Big Data benchmarking suites. In addition to explaining the benchmarking and data generation process Han et al. [28] also categorized the 4V in relation to each suite. The workloads and software stacks for each suite was also presented in an overview. The major weakness of this report is that it only gives an overview of suites without detailing any particular workloads or categories. Another problem is that it is outdated. Both BigDataBench and HiBench have been modified since this report was published.

Han, John, and Zhan [26] published an extensive report in 2017 which explained and categorized a large number of frameworks and systems related to Big Data. 32 Big Data benchmarking suites were analyzed in
the report, listing among others the workloads, operation types, submission policies and workload mixes for the different suites. Similar to previous reports, Han, John, and Zhan [26] only provided an overview of each suite without going into detail about the workloads or categories of workloads. The data generation technique of different suites was also categorized depending on how the data was generated or obtained. Architecture and performance were explained in detail. The major weakness of this report is that is too general. Statements are only given about benchmarking suites as a whole rather than components in the suites.

While the previous works give an overview of many different benchmarking suites, one of the problems is that the majority are outdated. Our report aims to give an update on the topic by comparing the recently updated BigDataBench and HiBench. In addition to giving an update, this report is also more in-depth than previous studies, focusing more on individual components in the tools rather than the tools themselves. The similarity between this work and the previously published reports are the criteria used for comparison i.e. data generation and workload implementation.
Chapter 3

Method

In this chapter, we describe the method used in order to extract and process the information that was needed for the result section. In section 3.1 the general methodology is presented. In section 3.2 the comparison criteria used for comparing Intel HiBench and BigDataBench are explained and motivated.

3.1 General Methodology

A qualitative research methodology was used for establishing analyzing and comparing HiBench and BigDataBench. A quantitative approach was not feasible since a fully controlled environment was impossible to set up. Even if a controllable environment could have been set up, having a 100% accurate baseline by which to compare benchmark results to would have been too time-consuming to set up. Another complication was that each benchmarking tool generated different data sets for the same experiment, which made the results hard to compare. The non-experimental approach allowed us to examine the data generation and workload implementation code without having to take into account surrounding software.

While an exploratory approach was suitable for getting an overview of benchmarking approaches, it was suboptimal for comparing the benchmarking tools. A major limitation of a non-experimental approach was that the results were hard to verify.

A literature study was done to get an overview of Big Data frame-
works, benchmarking methodologies and benchmarking tools. The primary data sources used for analyzing and comparing each benchmarking tool was the documentation and source code for each suite. Benchmarking methodologies and the design choices for each of the tools were gathered from reports. Comparison criteria for the tools were drawn from other areas in distributed benchmarking and the design methodology for Big Data benchmarking tools.

3.2 Benchmarking Tool Comparison Criteria

In this section, the comparison criteria used to compare Intel HiBench and BigDataBench are explained and motivated. The criteria were refined throughout the thesis. At first, a list of possible criteria was adapted from two similar reports [28, 49]: Usability, software stacks, performance metrics, modeling scope, modularity, scalability, data generation and correctness. The criteria were narrowed down to minimize the consideration of external factors such as human interaction, system architecture and external software. Interfacing, data generation, and correctness were the initially chosen criteria because they could be investigated using the individual tools source code, documentation and published reports. Correctness as a criterion was later changed to workload implementation because the verification of correctness required a formal method. Workload implementation also fulfilled the initial goal of gaining a deeper understanding of the BigDataBench and HiBench. The specific method used for applying the criteria to BigDataBench and HiBench is explained in chapter 4.

Interfacing, which was used to get an overview of the data flow between components in the system, is covered in section 3.2.1. Data generation was adapted as evaluation criteria from a similar study by Han et al. [28] where Big Data benchmarking suites were compared and is presented in section 3.2.2. Workload implementation as a criterion was adapted from a study by Han, John, and Zhan [26] where the data generation, workload implementation and performance metrics of 32 Big Data benchmarking suites were compared. The method used for analyzing the workload implementation of BigDataBench and HiBench can be found in section 3.2.3.
3.2.1 Interfacing

The internal and external interfacing of BigDataBench and HiBench was described in order to gain an increased understanding of each how the benchmarking tools interact with the target frameworks throughout the benchmarking process.

External Interface Block Diagrams (EIBD) were drawn to illustrate how each component in the software stack interacts with the benchmarking tool. Four diagrams were drawn in total. Two diagrams were drawn to illustrate how each of the benchmarking tools interacts with the frameworks during the execution of batch, graph, SQL and machine learning workloads. Another two diagrams were drawn, one for each of the benchmarking tools, which illustrate the interaction between the tools and the frameworks during streaming workloads.

In the last step of the interfacing analysis, the data flow between each of the subsystem components in each of the benchmarking tools was investigated. The data flow between each tool’s subsystem and the subsystems of the software stack was also investigated.

3.2.2 Data Generation

The data generation technique used by Intel HiBench and BigDataBench was a relevant point of comparison in this study. Data center input data is usually complex and possesses one or more of the 4V properties. In particular, the veracity of the data makes realistic data hard to generate. Several approaches exist for generating input data for workloads. One approach is to use samples from real-world data. Another approach is to use statistical models to generate data that closely resembles real-world data. The problem with generating synthetic data independently of real-world data is that the veracity of the data is not taken into account[28]. Another difficult aspect of data generation is to simulate realistic data arrival patterns e.g. the velocity of the data.

To ascertain the method used for data generated by each tool the individual tool documentation and source code was read. At first, the data generators and data sets were organized into categories since differ-
ent workload categories require different data. For BigDataBench the
categories included Offline Analytics, Online Service, AI and Stream-
ing. The data generators in HiBench were sorted into Micro, Machine
Learning, SQL, Graph, and Streaming. The data generation input, data
generator with associated parameters and output dataset was then de-
scribed for each data generation component.

The data generation input denotes the data that is needed to create the
output dataset. This could, for example, be a dictionary in the case of
generating a list of text reviews. The data generator and the associated
parameters are used to actually generate the data. Optional param-
eters can be used to vary the size, rate and other properties of the data
depending on the generator.

The data generator input and parameters are there to control the verac-
ity, velocity, volume, and variety of the output dataset. The degree to
which the 4V could be controlled was assessed by relating parameters
and input data to each of the 4V:

- **Volume.** The volume indicates the amount of data generated
  for a workload. One important consideration is that the term
  volume implies different types of data for different workloads.
  In a sort workload, for example, the volume means the number
  of words. In a graph analysis workload, the volume indicates the
  number of vertices [27].

- **Velocity.** The velocity of data can be broken down into data gen-
eration rate and data updating frequency. A good benchmarking
  tool can vary each of these parameters to simulate different Big
  Data scenarios. In a social graph workload, for example, data is
  continuously updated[27]. A good Big Data benchmarking sys-
  tem can also generate batch, real-time and streaming workload
  input data [26].

- **Variety.** The variety of data indicates how different it can be. A
  good benchmarking tool can support a variety of data types by
  being able to generate unstructured, semi-structured and struc-
tured data. A good benchmarking tool should also support vary-
ing data sources such as tables, texts, images, streams, and graphs
  [27].
• **Veracity.** Veracity represents the biases in the input data. A good benchmarking tool can generate and scale data while preserving the veracity of raw data. Veracity is arguably the most important property for generating realistic data.

The output data set is not necessarily dependent on a data generator. Ready-made datasets from sources such as Wikipedia\[58\] do not need to be processed. To differentiate between the general method used by BigDataBench and HiBench to generate datasets they were categorized into ready-made datasets, data generators based on synthetic distributions, data generators based on real-world data and hybrid generators[26].

### 3.2.3 Workload Implementation

A major part of the benchmarking process is the execution of a workload on some input data. To gain a deeper understanding of the workload design in BigDataBench and HiBench the workloads in each suite were analyzed and categorized. Because of the different design philosophy behind each suite, the categorization method was different for BigDataBench and HiBench.

For BigDataBench the workloads were first divided into two major groups, micro and component benchmarks. The reason for this division was that micro and component benchmarks were consistently different in scale. Each of the micro-benchmarks corresponds to a single Dwarf while component benchmarks correspond to weighted combinations of Dwarfs. Because component benchmarks are weighted combinations of micro-benchmarks, only the microbenchmarks were taken into consideration in the analysis.

For each workload in BigDataBench the workload type, operation type, and software stack was identified. The workload type identified the categories each workload fits in, and the basic computations used in each workload. These categories were used to estimate the diversity of the workloads in BigDataBench and HiBench, showing the difference between the implementation of the tools. The software stack was taken into consideration in order to understand the diversity of BigDataBench and HiBench.
The operation type, which was used to classify benchmarking tools in their entirety in a similar report by Han, John, and Zhan [26], was used in this case to describe each workload. The operation type was chosen as a category because it showed the difference in implementation between the workload implementations of each tool. Operation types consist of I/O operations, Algorithms and Elementary Operations. I/O operations include operations on data input or output data. Reading, writing, shuffling, creating and deleting data are all I/O operations. An algorithm, in this case, is a type of operation that performs CPU intensive step-by-step operations on data. Wordcount and Map among others are CPU intensive operations and fall within the algorithm operation type. Elementary Operations support a dynamic combination of operations e.g. SQL commands or similar.

The workload submission policy, which was used to classify benchmarking tools in their entirety in a similar report by [26], was used in this case to classify each workload. The workload submission policy was chosen as a category because it showed the difference in implementation between the workload implementations of each tool. A workload can have a pre-specified process, parameter control, trace-driven submission or a hybrid submission policy.

HiBench was categorized similarly to BigDataBench. The primary categorization of the workloads in HiBench was a division into application domains. These included Micro, Machine Learning, SQL, Websearch, Graph and Streaming categories.
Chapter 4

Analysis Method

In this chapter, we describe how our exploratory approach was made which includes how information was obtained and gathered in order to produce the results. Sections are divided into the respective tool to be analyzed. The sections describe the type of information found and how we chose to further investigate each tool in order to cover each point of interest. These point of interests are based on the criteria described in the method section.

By starting at homepages of the tools we were able to get an overview of the benchmarking tools. The initial aim was to download documentation of the tools such as manuals, source code and any other figures which can provide helpful information in order to establish an overview of the products. This was done to figure out limitations and necessary prerequisites. By gathering and compiling this initial overview, we were able to derive the possible environments each tool able to run in.

Once the initial information was gathered, we proceeded to examine each tool more in-depth. This was done initially by reading the reports published by the developers of BigDataBench and HiBench[22, 35, 52]. To gain a further understanding of the implementation and output of each tool the source code was also examined.

An overview was created given the comparison criteria. The source code was checked first because of the clear labeling of input, parameters, and input. For BigDataBench the handbook for version 3.1[41]
served as an information source. For HiBench there were several on-
line pages of instructions on their Github page[30] explaining the pa-
rameters of the various data generators.

The initial focus on BigDataBench was the wide variety of availability.
BigDataBench offered tests on many types of software stacks therefor
working on a wider set of Big Data processing frameworks. This re-
quired us to organize and later analyze each category separately, based
on the type of test to be performed. With the help of the dwarf-based
system which the benchmarking tool was based on [52], the examin-
ing of source code was made easier and grouped into the data dwarfs
which were used by the developers.

When reading the source code several external libraries and algorithms
were found. The algorithms were looked up and summarized briefly.
A more in-depth study was out of the scope of this thesis project.

4.1 Interfacing

The data flow between different components was determined by ana-
lyzing the source code and folder structure created by each BigData-
Bench and HiBench. The components were derived by examining the
setup instructions, combined with the expected components which
were obtained by different phases described by Han, Lu, and Xu [27].

For BigDataBench the data flow could be derived from the launch
scripts for each workload. Datasets were specified in parameters in
the launch script for each workload. By cross-referencing the parame-
ters and folder structure, the data flow from the data generator and the
workload could be found. For the data generators the resource folder,
a subfolder for each data generator, contained all relevant input for the
data generators.
HiBench has configuration files located in the folder labeled, which provided an easy method to organize which configurations affected which components. The configuration files contained variables denoting paths to input and output of the workloads. From these inputs and outputs, the data flow between components was derived.

4.2 Data Generation

The source code of the data generators in BigDataBench was found in the BigDataGeneratorSuite folder. This folder contained a number of subfolders, each representing a generator for a type of data. The workload table on the BigDataBench website listed the relation between the data generators, each workload type, and specific workload. The generators were categorized into workload types by referencing the table. A description of the generators and the real world datasets used in BigDataBench was found in [44]. A similar approach was used for investigating HiBench but with the use of HiBench online documentation[30, 34] instead of a published paper.

The AI data generators were investigated by first checking the source code for any dedicated data generators. Ready-made datasets were investigated by checking the website of BigDataBench [17] for any datasets associated with workloads in the AI category. Information on each ready-made dataset was gathered by searching for the names of the datasets on Google. The streaming data generators were investigated using only the source code since the BigDataBench 3.1 handbook[41] contained no information about data generation related to streaming. In each streaming generator execution script, the parameters were written as comments in the code.

The parameters for controlling the data generation in HiBench were searched for in either the configuration files for HiBench or the individual launch scripts for each generator. To find out if the data generators in HiBench were using a distribution to generate the output dataset, the data generator code files were checked after variables setting a distribution model.
The streaming data generation was investigated by reading the source code and the configuration help on the Github of HiBench [34]. Machine learning data generation was investigated primarily by reading the source code [31]. Each generator was labeled and contained in their own subfolder by the developers. For each file corresponding to a data generator, all parameters were labeled with "@param" at the top of each file by the developers. The parameters were later associated with different properties of the data generators output datasets.

4.3 Workload Implementation

By looking on which method calls that were used, we could determine what the workloads were supposed to accomplish. Most of the basic micro-benchmarks of BigDataBench made use of, and also tested the MapReduce paradigm of the frameworks. More complex workloads relied on the framework’s own library of functionality in order to carry out the task. By looking at the method calls used, we could, therefore, confirm the design and intent of the workload. This would later be used to understand and evaluate the workload implementation with the help of the reports [52, 57] from the developers of BigDataBench, which underlines the categories of the different workloads.

HiBench offered most of the documentation through their Github page [30]. The initial approach was to find suitable categories to divide and describe the individual workloads. By examining the user-manuals it was found that HiBench focused their tests on a small set of frameworks. While the tests were plenty, the available software stacks these were available on were limited to Hadoop, Spark [11], Gearpump [3], Flink [2], Kafka [8], and Storm [14]. Since these frameworks were similar to each other in terms of functionality, we kept using the categories which the developers of HiBench used. HiBench also offered to benchmark stream-based services. Because of this, we separated the batch-based workloads from the streaming workloads.
The operation type of each workload type was determined by reading the source code. Excluding the input and output of the workload, the operations in between were classified. If the operation included a modification of data on disk, it was classified as an I/O operation. If the operation performed computations without modifying data on disk, the operation was classified as an algorithm. Workloads that were used to test SQL and NoSQL data processing were classified as Elementary Operations.

The workload submission policy was determined by analyzing the source code. If there were no parameters for a workload, the workload was classified as a pre-specified process. If there were any parameters that could control the execution of the workload, it was classified as parameter controlled. If the workload took a real-life based data (trace) as input, its submission policy was classified as trace-driven.

The workload submission policy for individual workloads in BigData-Bench was found by reading the launch script, for each workload. In the launch script, the parameters could be found in the run command. If there were no optional arguments for running the benchmark the workload was classified as a pre-specified process. If there were any parameters the workload was classified as parameter controlled.

An overview of the workloads in HiBench was made by reading the HiBench Github[30]. The website was cross-referenced with the folder structure of the HiBench source code to find the code for the individual workloads. The workload submission policy for individual workloads in HiBench was found by reading the launch script for each workload. In the launch scripts, the OPTION variable was used to determine the type of submission policy. If the OPTION variable was absent, the workload was classified as a pre-specified process. If the OPTION variable was present, it was specified as parameter controlled.
Chapter 5

Results

5.1 BigDataBench

In this section our findings on data-generation, interfacing and workload implementation related to BigDataBench is presented. In section 5.1.1 findings related to data-generation is presented. In section 5.1.2 interfacing is described. In section 5.1.3 workload implementation is covered.

5.1.1 Interfacing

In this section, the interfacing for BigDataBench is presented. The Data Generation component took input data in the form of parameters from the user and input data from either a file or online source. The workload execution component uploaded data it had received from the data generator to the distributed file system. After the data had been uploaded to the file system the workload execution component uploaded the specific benchmark instructions to the framework which were then executed. After the execution of the workload, processed data was sent to the distributed file system. The report generation component then took the processed data and analyzed it to find out data about the performance metrics. The external interface block diagram in 5.1.1 illustrates this process.

The Data Generation component took input data in the form of parameters from the user and input data from either a file or online
Chapter 5. Results

Figure 5.1: External interface block diagram for BigDataBench benchmarks

source. The workload execution component uploaded data it had received from the data generator to the distributed file system. After the data had been uploaded to the file system the workload execution component uploaded the streaming and workload instructions to the distributed file system. The distributed file system uploaded both the generated data and the workload instructions to the framework. One workload streamed the data from the framework to itself while the streaming workload was executed concurrently. Processed data was sent to the distributed file system. The report generation component took the processed data and analyzed it to measure the performance metrics. The data flow is illustrated in 5.1.1.

5.1.2 Data Generation

In this section, the data generation phase of the benchmarking process is explained. The different implementations are divided into groups depending on workload type. The groups were based upon the workload types used by BigDataBench. Individual generators are presented
using the source code labels. The section also provides a description of the data generator implementations, inputs, and parameters. Information used in this section was compiled and extracted from the available packages which contain the actual workloads. These packages often included source-code which made it possible to derive how a certain component was implemented. Most of the data generators were implemented in C or C++. Some of the implementations were designed to not rely on a function which outputs a uniform-random sample. These functions were further explained in the handbook of the version 3.1 of BigDataBench [41].

**Offline Analytics**

The data generators available for Offline Analytics generated data containing text and matrices. In total there were two generators for text and three generators for matrices. The text generators were implemented as C++ functions. The text generators took a word list as input and then used Latent Dirichlet Allocation to assign words to topics. Words were then randomly selected from a topic to form a text which became the output of the generator. The sentences in the text had a

---

**Figure 5.2:** External interface block diagram for BigDataBench benchmarks when streaming
multinomial distribution of words in the selected topic. The parameters for the text generators were the following: Dictionary path, number of files, lines per file, words per line, and a path to the output folder. The output data was one or multiple files containing random text where each line represented a sentence. The text had no grammatical structure.

The data generators for matrices generated a large matrix by calculating the tensor product of two smaller input matrices multiple times. The matrix generators took a matrix equivalent to a weighted adjacency graph as input. The parameters for the matrix generators were the following: number of vertices and the Kronecker model parameters. The output from the matrix generator was a large matrix.

**Graph Analytics**

There was only one data generator for generating graphs in BigDataBench. The generator used the Kronecker graph model [40] to create large graphs by doing a matrix multiplication of two weighted directed graphs adjacency matrix. The provided datasets which served as an input for the generator were "Google Web Search" and "Facebook Social Network" [17]. The parameters for the graph generator controlled the number of iterations and which graphs that were used in the matrix multiplication. According to the handbook [41], the generator served as an extension, implying that the datasets provided could be seen as ready-made datasets of their own. The benefit of using the generator was that the user could scale the volume of the graph while still maintaining a self-similar graph [40] i.e. maintaining the model of the graph while scaling the size.

**Data Warehouse**

For the Data Warehouse workloads, BigDataBench offers two implementations labeled: e-com and personal_generator. These generators generated all the tables needed in order to run table based workloads. The personal data generator took 11 dictionaries in English and Chinese. The generators were extended from an existing solution called PDGF [51]. To construct the table the generator randomly selected a row from each dictionary and inserted it into a column in the table. By
using this generator, the data set could be scaled up while still main-
taining a given distribution by using distributed parallel random gen-
ersators. The generators needed a configuration file in XML-format in
order to run. The parameters and inputs needed to run the generator
was the number of rows(size of the output) and a seed for the random
generator.

The E-com generator generated a table containing customer- and ledger
information. Similar to the personal data generator the e-com genera-
tor took eleven smaller dictionaries as input, where each dictionary is
responsible for randomizing the fields(columns in the table). The gen-
erator used a random generator to select data to insert into the output
table. The e-com generator took the number(size in GB) which would
be generated as a parameter. The output was a single table.

AI

For the AI workloads BigDataBench had no data generators. Ready-
made datasets were used instead from various sources. Seven datasets
were provided by BigDataBench [17]. CFAR-10 was a dataset of im-
ages. ImageNet was a dataset of random images. LSUN was a set of
labeled images. TED Talks was a set of translated text. SoGou Data
was a set of unstructured search query text. MNIST was a set of im-
ages illustrating handwritten digits. MovieLens Dataset was a dataset
of movie user scores.

Streaming

There were no dedicated data generators for the streaming workloads
in BigDataBench. Instead, a number of stream generators were used
to select data, sequentially or randomly, from existing datasets. The
datasets consisted of random text, arrays of numbers and movie re-
view scores. Parameters included data-rate, port and input dataset.

5.1.3 Workload Implementation

Table 5.1 provides an overview and categorization of the individual
workloads in Bigdatabench 4.0. The workload name provides a con-
venient label which indicates the main algorithm or operation used in
the workload. The workload type represents the approximate application domain of the workloads. The operation type is used to classify the majority of the operations in the workload. The workload submission policy indicates the amount of user control a workload provides. The software stack column indicates the compatibility of the workload with different software stacks. It should be noted that the terms AI and Machine Learning were used interchangeably. This also applied to the terms Graph and Graph Analytics. Further details on the individual workloads related to the workload types can be found below the table.

Table 5.1: Overview of the Workload Type, Operation Type, Workload Submission Policy and Software Stacks for each Micro workload in BigDataBench 4.0

<table>
<thead>
<tr>
<th>Workload Name</th>
<th>Workload Type</th>
<th>Operation Type</th>
<th>Workload Submission Policy</th>
<th>Software Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort</td>
<td>Offline analytics</td>
<td>Algorithm</td>
<td>Trace-Driven Submission</td>
<td>Hadoop, Spark, Flink, MPI</td>
</tr>
<tr>
<td>Grep</td>
<td>Offline analytics &amp; Streaming</td>
<td>Algorithm</td>
<td>Trace-Driven Submission</td>
<td>Hadoop, Spark, Flink, MPI</td>
</tr>
<tr>
<td>WordCount</td>
<td>Offline analytics</td>
<td>Algorithm</td>
<td>Trace-Driven Submission</td>
<td>Hadoop, Spark, Flink, MPI</td>
</tr>
<tr>
<td>MD5</td>
<td>Offline analytics</td>
<td>Algorithm</td>
<td>Pre-Specified Process</td>
<td>Hadoop, Spark, Flink, MPI</td>
</tr>
<tr>
<td>FFT</td>
<td>Offline analytics</td>
<td>Algorithm</td>
<td>Pre-Specified Process</td>
<td>Hadoop, Spark, Flink, MPI</td>
</tr>
<tr>
<td>Matrix Multiplication</td>
<td>Offline analytics</td>
<td>Algorithm</td>
<td>Pre-Specified Process</td>
<td>Hadoop, Spark, Flink, MPI</td>
</tr>
<tr>
<td>Connected Component</td>
<td>Graph analytics</td>
<td>Algorithm</td>
<td>Trace-Driven Submission</td>
<td>Hadoop, Spark, Flink, MPI</td>
</tr>
<tr>
<td>Read</td>
<td>NoSQL</td>
<td>EO</td>
<td>Pre-Specified Process</td>
<td>HBase, MongoDB</td>
</tr>
<tr>
<td>Write</td>
<td>NoSQL</td>
<td>EO</td>
<td>Pre-Specified Process</td>
<td>HBase, MongoDB</td>
</tr>
<tr>
<td>Scan</td>
<td>NoSQL</td>
<td>EO</td>
<td>Pre-Specified Process</td>
<td>HBase, MongoDB</td>
</tr>
<tr>
<td>Convolution</td>
<td>AI</td>
<td>Algorithm</td>
<td>Pre-Specified process, Parameter Control</td>
<td>TensorFlow, Caffe</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>AI</td>
<td>Algorithm</td>
<td>Pre-Specified process, Parameter Control</td>
<td>TensorFlow, Caffe</td>
</tr>
<tr>
<td>Relu</td>
<td>AI</td>
<td>Algorithm</td>
<td>Pre-Specified Process, Parameter Control</td>
<td>TensorFlow, Caffe</td>
</tr>
<tr>
<td>Sigmoid</td>
<td>AI</td>
<td>Algorithm</td>
<td>Pre-Specified Process, Parameter Control</td>
<td>TensorFlow, Caffe</td>
</tr>
<tr>
<td>Tanh</td>
<td>AI</td>
<td>Algorithm</td>
<td>Pre-Specified Process, Parameter Control</td>
<td>TensorFlow, Caffe</td>
</tr>
<tr>
<td>MaxPooling</td>
<td>AI</td>
<td>Algorithm</td>
<td>Pre-Specified Process, Parameter Control</td>
<td>TensorFlow, Caffe</td>
</tr>
<tr>
<td>AvgPooling</td>
<td>AI</td>
<td>Algorithm</td>
<td>Pre-Specified Process, Parameter Control</td>
<td>TensorFlow, Caffe</td>
</tr>
<tr>
<td>BatchNorm</td>
<td>AI</td>
<td>Algorithm</td>
<td>Pre-Specified Process, Parameter Control</td>
<td>TensorFlow, Caffe</td>
</tr>
<tr>
<td>Dropout</td>
<td>AI</td>
<td>Algorithm</td>
<td>Pre-Specified Process, Parameter Control</td>
<td>TensorFlow, Caffe</td>
</tr>
</tbody>
</table>

The Offline Analytics microbenchmarks were used to test the basic functionality of MPI (Message Passing Interface), Spark & Flink. For Hadoop, the sort benchmark was implemented using the sort function in Hadoop MapReduce. The sort benchmark sorted a text input file
in alphabetical order. The Grep micro benchmark searched for lines or keyword matching a certain pattern. Wordcount used MapReduce by splitting an input text file into separate words, counting them and sorting them according to the count. MD5 calculated the hash value of every row in an input text file. The FFT and Matrix Multiply workloads tested MapReduce.

The Graph Analytics micro benchmarks consisted of only one benchmark, Connected Components. This workload counts the number of connected components in an input graph.

The NoSQL workloads were used to benchmark the operations of MongoDB & HBase. The Read and Write workloads tested the basic operation of MongoDB [45] and HBase [5] by reading from and setting a key. The Scan workload checked the performance of the ability to search for specific keys set in the database.

The workloads available in the AI microbenchmarks category are designed to benchmark different areas within machine learning. The workloads are labeled according to the different specific areas of machine learning by the developers. There are two workloads designed to test the performance of building layers of neurons, labeled Convolution and Fully Connected. Neurons are nodes which act as building blocks in machine learning. BigDataBench also provides workloads which are designed to tests different setups or techniques within these layers of neurons. There were three workloads, Relu, Sigmoid, and Tanh, which are designed to test the performance of activation functions, i.e. different functions which define the output of neurons. The workloads MaxPooling and AvgPooling benchmarks techniques used to improve accuracy in neural networks(stacked layers of neurons). The workloads CosineNorm and BatchNorm were designed to normalize outputs from the neurons. The Dropout workload is a workload which benchmarks a technique used to probabilistically drop the output for singular neurons.
5.2 HiBench

In this section our findings on data-generation, interfacing and workload implementation related to HiBench is presented. In chapter 5.1.1 findings related to data-generation is presented. In section 5.1.2 interfacing is described. In section 5.1.3 workload implementation is covered.

5.2.1 Interfacing

In this section, the interfacing for HiBench is presented. 5.2.1 illustrates the data flow for HiBench during the execution of every benchmark except for those in the streaming workload type. The Data Generation component took input data in the form of parameters from the user and input data from either a file or online source. The data generator component then uploaded the data it had generated to the distributed file system. After the data had been uploaded to the file system the workload execution component uploaded the specific benchmark instructions to the framework which were then executed. After the execution of the workload, processed data was sent to the distributed file system. The report generation component then took the processed data and calculated the value of the performance metrics. The data flow for most of the common Hibench workloads is illustrated in Figure 5.3.

For streaming workloads the data generation component took input data in the form of parameters from the user and input data from either a file or online source. The data generator component then uploaded the data it had generated to Apache Kafka, a data streaming platform. After the data had been uploaded to the streaming platform the workload execution component uploaded the specific benchmark instructions to the framework which were then executed. The streaming of the data runs in parallel with the execution of the workload. When a part of the streamed data was processed, it was written to the distributed file system. The report generation component then took the processed data from either the distributed file system or the streaming platform and calculated the value of the performance metrics. The data flow is illustrated in Figure 5.4.
5.2.2 Data Generation

In this section, the data generation phase of the benchmarking process is explained. The different implementations are divided into groups depending on workload type. The groups were based upon the workload types used by HiBench. Individual generators are presented using the source code labels. The section also provides a description of the data generator implementations, inputs, and parameters.

Micro benchmarks

There were three data generators for micro-benchmarks in HiBench. For the micro-benchmarks requiring text, the RandomTextWriter in Hadoop was used in order to produce the necessary data [30]. Because of the RandomTextWriter class already residing on the Hadoop platform, data was written directly to the distributed file system. The RandomTextWriter had multiple configurable parameters, listed as follows: Total bytes to be written, bytes per map, number of mappers,
Figure 5.4: Process flow diagram for stream based Benchmarking in HiBench

number of reducers, and the output path. The output was a large collection is lines each containing a random amount word of random length within a given interval. The text then got written to a distributed file system.

In order to prepare data for the benchmark which is based on TeraSort benchmark [54], data was generated using the Hadoop’s TeraGen example program [54]. The TeraGen was also a variant of a MapReduce implementation, requiring similar inputs of RandomTextWriter. The parameters needed in order to run TeraGen is: number of mappers, number of reducers, data size in bytes, and the output folder located on the distributed system. The output file consisted of rows containing a key, rowid, and random byte data.

The workload "Enhanced DFSIO(Distributed File System Input Output)" measured the performance of read and write operations of a Hadoop file-system. In order for this test to run, files may be optionally placed on the file system. The data-generation step was incorporated into the class of the actual workload performing the test, as the
class was meant to be both writing and reading from the file system. Since this workload was running and performing the benchmarking over multiple files, a configuration file was needed in order to specify folder locations for reading files, writing files, temporary files, and results. The parameters needed to run this as a data generation tool apart from the configuration file were: number of files, file size in bytes, and a buffer size in bytes. The argument "-skipAnalyze" was used to specify that no benchmarking would be executed and only the generation of the files.

Machine Learning

In total there were nine different data generators for generating datasets used in machine learning workloads [31]. All data generators for machine learning were based on Spark MLLib (Machine Learning Library) [30]. The GradientBoostingTreeDataGenerator generated random numbers based on a normal distribution. The generator took parameters for controlling the number of examples that would be contained in the RDD, a number of features to be generated for each example, epsilon factor by which the positive examples were scaled, a number of partitions of the generated RDD and the probability that a label was of the numeric value 1.

The LDADataGenerator (Latent Dirichlet Allocation Data Generator) generated a random text based on a random seed. The generator took no input. The LDADataGenerator had six different parameters. The number of documents, vocabulary size, minimum document length, maximum document length, number of partitions of the working memory and the random seed for each partition. The output was a specified number of text files each containing random text according to the vocabulary size.

The Linear Regression data generator [31] generated rows of numbers. The generator took no input. The generator had five parameters. The number of examples(rows), number of features(randomized entries per row), the epsilon factor(distribution), number of partitions and a random seed could be controlled. The output of the generator was one text file containing a specified number of rows. Each row contained
one label and a set of random numbers equivalent to the number of features.

The Logistic Regression data generator [31] was similar to the LDA-DataGenerator but with a given probability might label examples incorrectly for training purposes. The generator took no input. The generator had five parameters. The number of examples (rows), number of features (randomized entries per row), the epsilon factor (distribution), number of partitions and the probability that an example was mislabeled. The output of the generator was one text file containing a specified number of rows. Each row contained one label and a set of random numbers equivalent to the number of features.

The Linear Regression data generator [31] generated a random label paired with an array of random numbers. The generator took no input. The generator had five parameters. The number of examples (rows), number of features (randomized entries per row), the epsilon factor (distribution), number of partitions, the probability that an example is mislabeled and the random seed could be specified. The output of the generator was one text file containing a specified number of rows. Each row contained one label and a set of random numbers equivalent to the number of features.

The Random Forest data generator [31] generated a random label paired with an array of random numbers. The generator had five parameters. The number of examples (rows), number of features (randomized entries per row), the epsilon factor (distribution), number of partitions, the probability that an example is mislabeled and the random seed could be specified. The output of the generator was one text file containing a specified number of rows. Each row contained one label and a set of random numbers equivalent to the number of features specified in the parameters.

The SVD(Singular-Value Decomposition) data generator [31] generated a random label paired with an array of random numbers. The generator categorized examples according to a Gaussian distribution. The generator took no input. The generator had five parameters. The number of examples (rows), number of features (randomized entries per row), the epsilon factor (distribution), number of partitions, the
probability that an example was mislabeled and the random seed could be specified. The output of the generator was one text file containing a specified number of rows. Each row contained one label and a set of random numbers equivalent to the number of features.

The Rating data generator [31] generated data for collaborative filtering by randomly filling in a matrix with numbers. The generator took no input. The generator had three parameters. The sparsity, number of users (number of rows) and number of products (number of columns) could be controlled. The output of the generator was one text file containing a matrix. Each row represented a user and each column represented a product. Each cell represented a user's rating of a product.

**SQL**

There was only one generator for generating data used in SQL workloads. The generator generated SQL-based data according to [48]. The generator had two different parameters. The number of pages and user visits (entries per page) could be specified. The output was one file containing multiple tables depending on the number of pages.

**Websearch**

To generate data for the Websearch workloads there was one generator. The generator used the Nutch web crawler to generate web data with hyperlinks and random text. Text was generated using Hadoop RandomTextGenerator [25] and the Linux dictionary file. The generator took no inputs. The generator took four parameters: The number of mappers, number of reducers, number of pages and number of slot pages.

**Graph**

There was one generator for graph workloads in HiBench [32]. The generator randomly constructed a graph given a configuration file. The generator took no external inputs. There were three available parameters for the data-generator: a path to MatrixFactorizationModel,
output follow and the number of edges in the graph. The first parameter, the factorization model, was used to determine the weights of the edges. The degree of vertices and number of edges could also be controlled. The generator wrote one file which contained a graph in the form of an adjacency matrix.

**Streaming**

For streaming workloads in HiBench, there was one component acting as a data generator for establishing a stream of data. As input, the streaming generator took the output from one of the available four data generators: Hive, Pagerank, Bayes and Nutch [34]. A power-law distribution was then used to select data from the input data. The data was then streamed directly to a Kafka cluster. The interval span between each burst of data to send to Kafka could be controlled. The total amount of records, record length and number of rounds could also be controlled. The number of producers could also be controlled.

### 5.2.3 Workload Implementation

This section covers our findings on how the workloads in HiBench were implemented. An overview of available workloads in HiBench 7.0 is illustrated in Table 5.2. The workload name provides a convenient label which indicates the main algorithm or operation used in the workload. The workload type represents the approximate application domain of the workloads. The operation type is used to classify the majority of the operations in the workload. The workload submission policy indicates the amount of user control a workload has. The software stack column indicates the compatibility of the workload with different software stacks. It should be noted that the terms AI and Machine Learning were used interchangeably. This also applied to the terms Graph and Graph Analytics.
Table 5.2: Overview of the Workload Type, Operation Type, Workload Submission Policy and Software Stacks for each workload in HiBench 7.0

<table>
<thead>
<tr>
<th>Workload Name</th>
<th>Workload Type</th>
<th>Operation Type</th>
<th>Workload Submission Policy</th>
<th>Software Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort</td>
<td>Micro Benchmark</td>
<td>Algorithm</td>
<td>Pre-Specified Process</td>
<td>Hadoop, Spark</td>
</tr>
<tr>
<td>WordCount</td>
<td>Micro Benchmark</td>
<td>Algorithm</td>
<td>Pre-Specified Process</td>
<td>Hadoop, Spark</td>
</tr>
<tr>
<td>Terasort</td>
<td>Micro Benchmark</td>
<td>Algorithm</td>
<td>Pre-Specified Process</td>
<td>Hadoop, Spark</td>
</tr>
<tr>
<td>Sleep</td>
<td>Micro Benchmark</td>
<td>Algorithm</td>
<td>Pre-Specified Process</td>
<td>Hadoop, Spark</td>
</tr>
<tr>
<td>enhanced DFSIO</td>
<td>Micro Benchmark</td>
<td>IO</td>
<td>Parameter Control</td>
<td>Hadoop, Spark</td>
</tr>
<tr>
<td>Bayesian Classification</td>
<td>Machine Learning</td>
<td>Algorithm</td>
<td>Parameter Control</td>
<td>Spark</td>
</tr>
<tr>
<td>K-means Clustering</td>
<td>Machine Learning</td>
<td>Algorithm</td>
<td>Parameter Control</td>
<td>Spark</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>Machine Learning</td>
<td>Algorithm</td>
<td>Parameter Control</td>
<td>Spark</td>
</tr>
<tr>
<td>Alternating Least Squares (ALS)</td>
<td>Machine Learning</td>
<td>Algorithm</td>
<td>Parameter Control</td>
<td>Spark</td>
</tr>
<tr>
<td>Gradient Boosting Trees (GBT)</td>
<td>Machine Learning</td>
<td>Algorithm</td>
<td>Parameter Control</td>
<td>Spark</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>Machine Learning</td>
<td>Algorithm</td>
<td>Parameter Control</td>
<td>Spark</td>
</tr>
<tr>
<td>Latent Dirichlet Allocation</td>
<td>Machine Learning</td>
<td>Algorithm</td>
<td>Parameter Control</td>
<td>Spark</td>
</tr>
<tr>
<td>Principal Component Analysis (PCA)</td>
<td>Machine Learning</td>
<td>Algorithm</td>
<td>Parameter Control</td>
<td>Spark</td>
</tr>
<tr>
<td>Random Forest</td>
<td>Machine Learning</td>
<td>Algorithm</td>
<td>Parameter Control</td>
<td>Spark</td>
</tr>
<tr>
<td>Support Vector Machine (SVM)</td>
<td>Machine Learning</td>
<td>Algorithm</td>
<td>Parameter Control</td>
<td>Spark</td>
</tr>
<tr>
<td>Support Vector Machine (SVM)</td>
<td>Machine Learning</td>
<td>Algorithm</td>
<td>Parameter Control</td>
<td>Spark</td>
</tr>
<tr>
<td>Singular Value Decomposition</td>
<td>Machine Learning</td>
<td>Algorithm</td>
<td>Parameter Control</td>
<td>Spark</td>
</tr>
<tr>
<td>Scan, Join, Aggregate</td>
<td>SQL</td>
<td>Algorithm</td>
<td>Pre-Specified Process</td>
<td>Hadoop, Spark</td>
</tr>
<tr>
<td>SparkLine</td>
<td>Websearch</td>
<td>Algorithm</td>
<td>Parameter Control</td>
<td>Spark, Nutch</td>
</tr>
<tr>
<td>PageRank</td>
<td>Websearch</td>
<td>Algorithm</td>
<td>Parameter Control</td>
<td>Spark</td>
</tr>
<tr>
<td>Nutch indexing</td>
<td>Websearch</td>
<td>Algorithm</td>
<td>Parameter Control</td>
<td>Spark, Nutch</td>
</tr>
<tr>
<td>NWeight</td>
<td>Graph</td>
<td>Algorithm</td>
<td>Parameter Control</td>
<td>Spark(with GraphX or Pregel)</td>
</tr>
<tr>
<td>Identity</td>
<td>Streaming</td>
<td>Algorithm, IO</td>
<td>Parameter Control</td>
<td>Spark, Storm and Gearpump</td>
</tr>
<tr>
<td>Repartition</td>
<td>Streaming</td>
<td>Algorithm, IO</td>
<td>Parameter Control</td>
<td>Spark, Storm and Gearpump</td>
</tr>
<tr>
<td>Stateful Wordcount</td>
<td>Streaming</td>
<td>Algorithm, IO</td>
<td>Parameter Control</td>
<td>Spark, Storm and Gearpump</td>
</tr>
<tr>
<td>Fixwindow</td>
<td>Streaming</td>
<td>Algorithm, IO</td>
<td>Parameter Control</td>
<td>Spark, Storm and Gearpump</td>
</tr>
</tbody>
</table>
All HiBench workloads came with default configurations. The workloads also included configuration options for any underlying software layers, such as Spark running on Hadoop, when it was necessary to control external factors (seen from the target framework which was being tested). Individual workloads related to each workload type are explained in the subsections below.

**Micro Benchmarks**

Microbenchmark workloads were made to test a framework’s ability to process elemental operations i.e. stressing its raw processing power. There were in total five micro benchmarks available in HiBench. The Sort, Terasort, and Wordcount workloads were used to test MapReduce. The Sleep workload was made to test the Spark Job Scheduler. The enhanced DFSIO workload [29] was an adapted version of the DFSIO benchmark offered by the Hadoop Library. The purpose of Enhanced DFSIO was to measure the throughput of reads and writes in HDFS.

**Machine learning**

In total there were eleven workloads in HiBench to benchmark different machine learning implementations. All workloads were implemented using the Spark mllib machine learning library. The Bayesian Classification workload tested the performance of running a classification algorithm. K-means tested the k-means clustering algorithm. The logistic regression algorithm categorized items. Alternating Least Squares ran a collaborative filtering algorithm. Gradient Boosting Trees and Linear Regression performed a linear regression analysis. Latent Dirichlet Allocation was a topic model which inferred topics from documents. Principal Component Analysis ran a statistical model to find the relation between variables. Random Forest were ensembles of decision trees. Support Vector Machine ran a classification algorithm. Singular Value Decomposition was a factorization workload.
SQL

There was only one workload for testing SQL data processing. The workload executed a list of online analytical processing queries. The queries were HIVE queries consisting of scan, join and aggregate.

Websearch

Websearch workloads took a graph representing a network (commonly the Internet) and mined information from it. There were two workloads for running websearch workloads. The Pagerank workload was a ranking benchmark used by search engines and is implemented using Pegasus 2.0 [39], a graph mining system. Nutch Indexing was a web crawler implemented using Apache Nutch [10].

Graph

The only graph workload available from HiBench was called NWeight and was implemented using Spark and was therefore only available to be executed on systems running Spark. NWeight was implemented to calculate association between two vertices, given a weighted graph. The workload could be configured to run in a model based on GraphX [12] or in a model based on Pregel [42], which was up to the user to configure.

Streaming

The workloads for benchmarking stream-based systems include four different tests, as seen in Table 5.2. In order to emulate a setting where data was being streamed to the targeted system, a third party streaming platform was needed. HiBench solved this by implementing their workload with Kafka as a streaming platform. As data was generated, it was placed into a topic which the targeted framework subscribed to. The workload, therefore, required the additional component of Kafka during runtime.
Chapter 6

Discussion

The goal of the project was to investigate BigDataBench and HiBench with regards to interfacing, data generation, and workload implementation, gaining a deeper understanding of each tool. This was in large part achieved, especially with regards to data generation and workload implementation. A difficulty during the work was the application of the individual criteria to BigDataBench and HiBench.

A limitation of this study was the application of the criteria. Data generation and workload implementation as criteria were in large part adapted from a similar study by Han, John, and Zhan [26]. When trying to apply the criteria from the paper more thoroughly to individual data generators and workloads, the methodology was unclear. Part of this problem stemmed from the specialization and complexity of the individual parts of the tools. Each benchmarking tool/framework combination was so different in implementation that a separate methodology would need to be used for each combination.

Another difficulty faced when comparing BigDataBench and HiBench was the categorization methodology used by the different tools. BigDataBench divided its workloads according to the level of complexity while HiBench categorized workloads according to the type of computation. Since none of the tools adhered to a common standard for categorizing the workloads, workloads different enough to warrant a new category were placed together. This made a comparison of workload implementation with regards to the application domain or level of complexity difficult.
The interfacing of data generation and workload implementation of BigDataBench and HiBench are discussed in the sequential order in section 6.1, 6.2, and 6.3.

6.1 Interfacing

In general, the interfacing for both BigDataBench and HiBench is similar for all but the streaming workloads. Hibench uses a separate streaming framework while BigDataBench uses the framework itself to stream data for the workload. Both approaches have strengths and weaknesses.

Being dependent on the framework to stream data requires the developer to continually rewrite the code for each software stack as the API for the framework changes. Another issue is that the implementation has to be done for each applicable software stack, introducing more variability in the data stream and the results. Running the streaming and the workload on the same framework can also cause take CPU time away from the workload, skewing the results.

6.2 Data Generation

In general, the data generators of both BigDataBench and HiBench have several issues. The first issue is consistency. While most of the generators provide a way to control the volume of the output dataset, the number and properties of the parameters controlling the size are not the same across the generators. Without any assisting documentation to understand the impact of different options, changing the size of the output data set is harder than it needs to be.
The Offline Analytics data generators of BigDataBench were relatively simple while still taking volume, veracity, and variety into consideration. The text generator was the one most at fault by providing only a multinomial distribution of words with no real sentence structure. This lowered the reliability of the dataset since texts with different distribution could not be generated.

The matrix and graph generators were superior with regards to reliability by using a real graph as input for the generators. The volume of the matrix could also be controlled by changing the number of iterations of the Kroenecker algorithm. A problem with this approach was that the exact size of the graph was unknown before the generation was complete.

Both of the Data Warehouse generators provided several ways to scale the volume of the dataset. Only the e-com generator provided a way to control the exact size of the output dataset, however. This made it easier to measure the impact of different datasets of the same size. A major weakness of both generators was that they used a random generator to fill the output datasets i.e. there was no way to control the veracity of the output datasets.

Since BigDataBench provided only ready-made datasets for the AI workloads the veracity could not be controlled at all. While the ready-made datasets were excellent for analyzing the performance of a specific system the results from using that dataset were not applicable to any other scenario.

There were no data generators for streaming workloads in BigDataBench. Instead, they provided for ways to control data production rate and sending rate. The approach that was used for this was separating the streaming component from the input dataset. This enabled the use of different datasets, making the variation of data easier.

The microbenchmark data generators in HiBench provided control over size i.e. the volume of the output dataset through the use of a parameter specifying the size in GB. All data was generated using the Java random generator, however, making the adjustment of veracity possible only through the random seed.
The machine learning data generators in HiBench provided control over both the volume, veracity, and velocity of the data through the use of parameters specifying the number of workers, number of rows and the epsilon factor. A problem with the machine learning data generators was that the connection between the parameters and the output dataset were hard to understand and required familiarity with machine learning systems.

The graph generator in HiBench provided control over both the veracity and volume of the output graph through the use of an input graph(distribution) and the number of edges.

The data generator for SQL data in HiBench only provided control of the volume of the output dataset. This makes it one of the weakest data generators since it can neither generate data according to different distributions nor specify the exact size of the output.

The streaming data generators of HiBench were similar to the one in BigDataBench but used data from generators instead of ready-made datasets. This allowed for more control over both the volume and veracity of the data.

### 6.3 Workload Implementation

In general, the workload implementation in BigDatabench and Hibench were different with regards to both the number of modifiable parameters and applicability to different software stacks. BigDatabench offers less control in its workloads but is compatible with many frameworks. Hibench offers control over the workload execution in many cases but is limited in most cases to Hadoop and Spark. The natural conclusion is that Hibench is well suited to benchmark Hadoop and Spark while BigDatabench is a more encompassing suite for testing a wide variety of frameworks.

Half of the Offline Analytics workloads in BigDataBench were controlled using a trace, meaning that the workloads modeled a real-world
scenario. The other half were executed according to a pre-defined script, making the results non-applicable to different scenarios. The NoSQL workloads work similarly, with no way of modifying the execution of the workload without rewriting the code. This was problematic for testing databases since the execution of commands could happen in different order and with a different velocity. All of the AI workloads in BigDataBench were designed consistently with a common problem. Since the workload was not based on any real data, the results of running the workload would not be applicable to real-world systems.

The workloads in HiBench were more simple in design, with no workloads being based on real-world workloads or input data. The majority of the micro workloads and the only SQL workload were also hard-coded. A major problem with all but the streaming workloads was that they could only be executed on Hadoop and Spark, rendering the capability of HiBench quite low. A strength of HiBench was the streaming workloads, which were parameter controlled applicable to a wide range of software stacks.
Chapter 7

Conclusion

In conclusion, neither BigDataBench or HiBench was superior with regards to the interfacing, data generation or workload implementation. Both were designed to function differently. BigDataBench was designed to be applied to a larger set of different frameworks while HiBench was focused on the Hadoop family. Another important difference was the data generation of each tool. BigDataBench had, at least in part, included real datasets and used real-world traces in the data generation. The majority of the datasets used in HiBench were generated synthetically. With regards to interfacing, both tools had different approaches to streaming, with HiBench using a dedicated component for streaming data for workloads.

To gain a deeper understanding of how BigDataBench and HiBench a case study could be made with companies using these suites. This would give insight into how the tools are used in practice, and what requirements potential users have.

In order to increase the understanding of the separate components of BigDataBench and HiBench, future work could focus on a specific criterion or a specific system. A large part that was left out of this thesis was an analysis of the report generation part of the benchmarking process, and appropriate metrics to measure. A separate study could also be performed to analyze the component benchmarks of BigDataBench.
References


