Comparing software stacks for Big Data batch processing

João Manuel Policarpo Moreira
Instituto Superior Técnico, Universidade de Lisboa
Lisboa, Portugal
joao.m.p.moreira@tecnico.ulisboa.pt

***Abstract*—The recent Big Data trend leads companies to produce large volumes and multiple varieties of data. Companies require to access data and process it in order to obtain the information required to make the best business decisions. The amount of data available continues to increase as most of the data will start to be produced automatically by devices, and transmitted directly machine-to-machine. Some examples of open-source systems that are used for processing data are Hadoop and Hive. These systems rely on complex software stacks and cannot run in computers with low hardware specifications, which are usually placed close to sensors that are spread out around the world. Having processing closer to the data sources is one of the envisioned ways to achieve better performance for data processing. An approach to solve this problem is to remove layers of software and seek to provide the same functionality with a leaner system without relying on complex software stacks. This is the value proposition of Unicage, a commercial system based on Unix shell scripting, that promises better performance by having a vast set of efficient commands that directly use the Operating System mechanisms for process execution and inter-process communication. The goal of this work was to analyse and evaluate the performance of Unicage when compared to other data processing systems, such as Hadoop and Hive. LeanBench, a benchmark that covers relevant workloads composed by multiple operations has been produced, allowing to perform benchmarking operations on the various processing systems in a comparable way. Multiple tests have been performed using this benchmark that help to clarify if the complexity of the software stacks is indeed a significant bottleneck in data processing. Guidelines for processing systems are presented.**

***Keywords*-Big Data, Benchmarking, Data Processing, Software Stacks**

I. Introduction

Data production continues to increase with the progression of technology. Companies produce large amounts of data, not only in volume but also in a wide variety of types. At the same time, in order to make the best business decisions in a timely manner, companies have a need to increase the velocity at which data is accessed and processed. Nowadays, these developments are designated by the term Big Data. In addition to companies, the Internet of Things will also further increase the amount of data produced [1]. This is due to the fact that most of the data will start to be gathered automatically by devices and their sensors, and at the same time, transmitted directly from machine-to-machine in the network. Open-source systems that are used for Big Data batch processing include Apache Hadoop and Apache Hive. However, these systems rely on external complex software stacks. As a consequence, it is very difficult to run these applications directly on smaller devices with low system specifications. A solution to this problem would be to remove most of these dependencies and produce tools that provide the same functionality but rely on simpler software stacks. One example of a tool like this is the Unicage system. Unicage removes dependencies on complex software by using individual Unix shell commands that provide a single processing function. These commands are configured in an shell script which run as Unix processes, use system calls directly and use pipes for communicating and transmitting information. The main objective of this work is to assess how Unicage, a system that uses the Operating System process execution and inter-process communication mechanisms directly, compares to Big Data processing tools that rely on complex software stacks. This document is structured as follows: I - Introduction, II - Background, III - LeanBench, IV - Evaluation and V - Conclusion.

II. Background

This Section presents the relevant Big Data processing tools, describing Unicage, the MapReduce programming model and systems based on MapReduce, such as Apache Hadoop and Apache Hive. In addition, multiple benchmarking tools for processing systems are presented.

A. Unicage

Unicage is a toolset that provides Unix shell commands which enable users to compose data processing systems. In order to make maintenance easy, Unicage promotes a distributed development approach, where data, programs and hardware are all separated from each other [2].

The Unicage development approach consists on writing a script using a standard Unix shell programming language. The script uses the default Unix commands, such as cp, $find$ and $sort$ in conjunction with commands of the Unicage toolset.

The Unicage toolset is composed by a large number of individual commands which provide individual functionalities required to process data. These include string formatting functions, system functions for input/output of files, common operations used in relational queries (such as joins, selections), mathematical and statistical functions.

Each individual command is designed to be simple to use and only provide a single function. Each command is written directly in the C programming language in order to take advantage of the input/output buffers and memory manipulation techniques of the language [3]. In addition, Unicage programs use the Unix shell, which internally uses kernel functions directly, avoiding other middleware software or any other unnecessary processing that could possibly slow down the processing.

B. MapReduce

MapReduce is a programming model designed to process and manage large datasets. The MapReduce programming model idea was firstly introduced at Google [4]. Developers observed that most of their data processing was composed by two types of operations, one for inputting data records and computing a set of intermediate key/value pairs, and a second operation to combine all of the data generated by the first operation. MapReduce applies this technique and uses two distinct functions: the `map` function processes a pair of key/values of data and generates an additional intermediate pair of key/values of data; the `reduce` function merges all the intermediate values generated by the `map` function. Both of these functions are written by the developer according to the type of the data and also to the type of processing that the data requires. Solutions to real world problems such as distributed grep and URL access frequency count can be implemented using this programming model.

Furthermore, programs written using this model are automatically able to be parallelized as map and reduce tasks can be assigned to multiple machines. Moreover, by adding more commodity machines, the cluster is able to be expanded. Each machine in the cluster is assigned as a worker for map and reduce tasks, executing the processing operations specified by the user-defined `map` and `reduce` functions.

C. Apache Hadoop

Apache Hadoop is an open-source framework designed for processing large datasets across clusters of computers. Hadoop uses the MapReduce programming model, which enables it to process the data in a distributed computing approach. The framework consists on three modules:

- **Hadoop Yet Another Resource Negotiator (YARN):** A job scheduling framework and cluster resource management module [5];
- **Hadoop MapReduce:** A YARN-based system for parallel processing of large datasets, using an implementation of the MapReduce programming model;
- **Hadoop Distributed File System (HDFS):** A distributed file system module that enables the storage of data across cluster machines [6].

Figure 1 presents the architecture and modules of Hadoop.

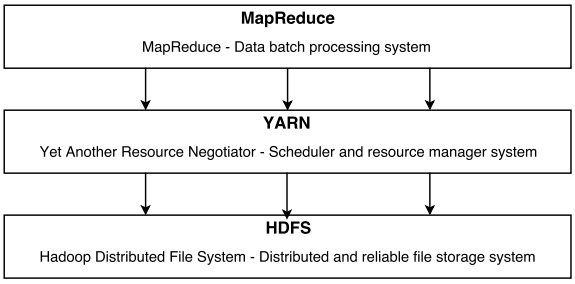

Figure 1. Hadoop Architecture and Modules.

Hadoop allows to store, manage and process large datasets in a reliable way as HDFS was designed to be fault-tolerant to hardware failures of cluster machines. Unlike traditional relational database management systems, Hadoop can process unstructured, semi-structured and structured data as the processing is not performed automatically but instead specified by the user.

D. Apache Hive

Apache Hive is a Data Warehousing system built on top of the Apache Hadoop framework [7]. Hive enables users to query large datasets that reside in a distributed storage form, such as the Hadoop HDFS system, without relying on defining complex low level `map` and `reduce` functions.

Hive provides *HiveQL*, an *SQL*-like interface that enables the user to create queries in a language similar to *SQL*. Internally, the queries written in *HiveQL* are compiled and directly translated to MapReduce jobs, which are executed by the Hadoop framework.

Hive manages and organises the stored data in three different containers, as shown in Figure 2.

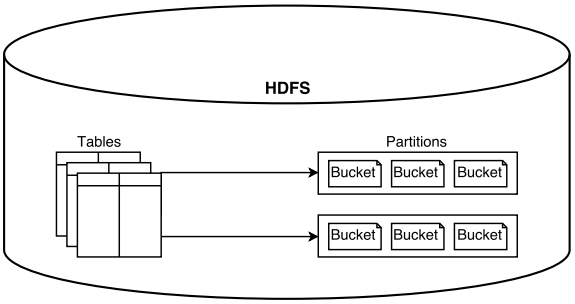

Figure 2. Hive Data Model.

- **Tables:** Similar to the tables in relational database systems, allowing filter, projection, join and union operations. These tables are stored in HDFS;
- **Partitions:** The tables may be composed of one or more partitions. Partition keys allow to identify the location of the data in the file system;

- **Buckets:** Inside each partition the data can be divided further into buckets. Each bucket is stored as a file in the partition.

Hive is composed by two main logical blocks: Hive and Hadoop. The Hive block is mainly responsible for taking the user input and turning it into something that is compatible with the Hadoop framework. Hadoop is then responsible to carry out the processing task.

E. Benchmarking Tools

There are multiple benchmarking libraries and tools available to evaluate the performance of Big Data processing systems. Nevertheless, a proper benchmarking method has to picked due to the differences in methods and paradigms of processing systems. Some examples of such benchmarking tools are presented in what follows.

1) Hadoop Benchmarking Tools: The Apache Hadoop distribution includes tools designed to test and benchmark various aspects of the Hadoop framework, such as the storage component (HDFS) and the processing component (MapReduce). These tools can be used for evaluating the execution of tasks in Apache Hive as well, due to the fact that Hive mainly relies on the Hadoop MapReduce and HDFS systems for the processing and storage capabilities respectively.

2) BigBench: BigBench [8] is a proposal for a Big Data benchmarking tool as an attempt to create an industry standard benchmark for Big Data. BigBench was designed for MapReduce based systems, such as Hadoop and Hive. The BigBench proposal suggests workloads that use a set of multiple different queries in order to evaluate the performance of the system, covering the three Big Data characteristics: *i)* Volume of data, *ii)* Variety of data and *iii)* Velocity of data processing.

For evaluating the Volume of data, BigBench proposes on having a Big Data generator able to be configured accordingly to a scaling factor in order to produce various volumes of data. When it comes to the variety of data, the proposal suggests that the Big Data generator should be able to produce structured and unstructured data. BigBench proposes having a continuous input of data into the storage system to evaluate the Velocity characteristic. The continuous input of data varies in volume, depending on the queries being performed and also on the defined scale factor. BigBench proposes supporting multiple distinct metrics according to what is being evaluated, in addition, it suggests creation of custom composite metrics in order to evaluate more than two relevant aspects at once.

3) BigDataBench: BigDataBench [9] is an open-source implementation and extension of the BigBench proposal, including additional different types of data sets with more extensive workloads that support multiple systems.

BigDataBench uses unstructured, semi-structured and structured datasets. The different datasets are representative of real world applications, such as offline analytics, real-time analytics and online services. BigDataBench uses real world datasets instead of using randomly generated data. However, to allow scalability of dataset sizes, BigDataBench introduced the *Big Data Generator Suite* (BDGS). This tool is capable of producing very large sets of synthetic data based on real datasets while keeping the three Big Data characteristics: Volume, Variety and Velocity. BigDataBench supports workloads for multiple Big Data processing systems.

4) AMP Big Data Benchmark: The U.C. Berkeley AMP Big Data Benchmark a standalone open-source tool for evaluating the performance of Data Warehousing systems based on MapReduce like Apache Hive. The main metric used for evaluation in the AMP Benchmark is the query execution time. When it comes to datasets, the AMP Benchmark includes support for both structured and unstructured datasets.

The AMP Benchmark workload is based on an existing proposed workload [10]. Some operations of the workload include simple scans, aggregations, joins and also other more complex user-defined functions.

5) HiBench: HiBench is an open-source Big Data benchmarking tool for MapReduce-based systems such as Hadoop. HiBench was firstly presented in 2011 [11] as a Hadoop benchmark suite. It includes many workload categories, composed by a wide set of Hadoop MapReduce jobs for many smaller benchmarking tasks, as well as larger benchmarking tasks such as real world applications and tasks.

HiBench evaluates the performance of Hadoop in three aspects: *i)* MapReduce job execution time, *ii)* Usage of system resources and *iii)* HDFS bandwidth and throughput.

To evaluate job execution times, throughput and usage of system resources, HiBench mainly uses smaller benchmarking tasks, such as sorting and word counting. For larger workloads HiBench uses tasks that are normally used in the real world such as PageRank.

When it comes to testing the HDFS component of Hadoop, HiBench provides a revised DFSIO test, which was modified to evaluate the aggregated bandwidth of HDFS. After the introduction of HiBench in 2010, it has been updated in 2012 [12] introducing workloads for benchmarking Data Warehousing systems.

III. LEANBENCH

The Big Data processing systems analysed in Section II use different paradigms. Unicage provides commands that focus both on batch processing and database querying operations. Hadoop is focused on batch processing while Hive is focused on providing data querying facilities that are not supported natively by Hadoop.

As some systems focus on batch processing and others on database querying operations, the LeanBench benchmark will be split into two different benchmarking modules.

The first module focuses on batch processing operations such as Sort, Grep and Wordcount and allows to make a comparison between Unicage and Hadoop. The second module focuses on database querying operations such as Selection, Aggregation and Join operations and allows to perform a comparison between Unicage and Hive.

Table I shows the benchmark operations for each system.

Table I
BENCHMARK OPERATIONS FOR EACH SYSTEM.

	Unicage	Apache Hadoop	Apache Hive
Sort	✓	✓	
Grep	✓	✓	
Wordcount	✓	✓	
Select Query	✓		✓
Aggregation Query	✓		✓
Join Query	✓		✓

A. Datasets

LeanBench uses two different categories of datasets: *i)* unstructured and *ii)* structured.

The unstructured datasets are used on the first part of the benchmark which focuses on processing operations in batch processing. The structured datasets are used on the second part of the benchmark, which is focused on data querying operations.

All of the datasets for the proposed benchmark are provided by the *Big Data Generator Suite* (BDGS) [9], a scalable data generation tool introduced by BigDataBench. This tool enables the generation of large amounts of data based on real datasets. For generating unstructured data, the tool uses two real datasets: *i) English Wikipedia* entries and *ii) Amazon movie review* entries.

For generating structured data, the tool uses an additional real dataset, which is based on anonymous *E-commerce transaction data*. The structured dataset is composed by two relational tables described below. The primary key of each table is represented with an underline.

OS_Order(Order_ID, Buyer_ID, Create_Date)

OS_Order_Item(Item_ID, Order_ID, Goods_ID, Goods_Number, Goods_Price, Goods_Amount)

B. Operation Implementation

In order to support the operations of LeanBench on the various systems, we have to implement several versions of the same operation as each system functions differently and provides different languages for users to implement operations. The operations will be implemented in *Unix Shell Script* for Unicage, in *Java* for Hadoop, and in *HiveQL* for Hive. The Wordcount and Join operation implementations are presented as an example. Wordcount is implemented in Unicage using a sequence of two commands, $sort$ and $count$, as shown in Listing 1. The $sort$ command is responsible for sorting the words of the input dataset and the $count$ command performs the counting operations. Line 1 specifies the interpreter to be used, which is ush, the Unicage command shell script interpreter. All Unicage scripts must use this interpreter.

Listing 1. Wordcount - Unicage (Unix Shell Script)

```
1 #!/home/TOOL/ush
2
3 sort -d ../input/dataset.dat |
4 count 1 1 > results/Wordcount_result
```

Wordcount is implemented in Hadoop using two `map` and `reduce` functions, as exemplified in Listing 2. The `map` function reads every word from the input dataset, and for each word found, an intermediate $< key, value >$ pair is written, consisting of the word found (the key), and the value, which corresponds to the integer value 1. After the `map` task finishes, the `reduce` task goes through all the intermediate $< key, value >$ pairs and reduces them: the values are summed for all the words that share the same key, and are written as output.

Listing 2. Wordcount - Apache Hadoop (Java)

```
1  private final static IntWritable one
2          = new IntWritable(1);
3  private Text word = new Text();
4  public void map(Object key, Text value
5          , Context context){
6    StringTokenizer itr =
7    new StringTokenizer(value.toString());
8    while (itr.hasMoreTokens()) {
9      word.set(itr.nextToken());
10     context.write(word, one);
11   }
12 }
13 public void reduce(Text key,
14   Iterable<IntWritable> values,
15   Context context){
16   int sum = 0;
17   for (IntWritable val : values) {
18     sum += val.get();
19   }
20   result.set(sum);
21   context.write(key, result);
22 }
```

Multiple commands are used to implement the Join operation in Unicage, as presented in Listing 3.

Listing 3. Join - Unicage (Unix Shell Script)

```
1 #!/home/TOOL/ush
2 cat input/OS_ORDER_ITEM.txt |
3 tr '|' ' ' > input/ORDER_ITEM_FORMATTED.txt
4 cat input/OS_ORDER.txt |
5 tr '|' ' ' > input/ORDER_FORMATTED.txt
6 join1 key=2 input/ORDER_FORMATTED.txt
7   input/ORDER_ITEM_FORMATTED.txt |
8 self 3 8 |
9 sm2 1 1 2 2 > results/Join_result
```

The `join1` command allows to join two tables, the `self` allows to select columns, and the `sm2` command sums values by key. The Unix commands `cat` and `tr` are also used to modify the dataset columns to be separated by a space instead of being separated by the character "|". This extra processing step has to performed because Unicage commands expect the dataset columns to be separated by spaces.

The Join operation in Hive is implemented using HiveQL, a process very similar to writing a query using a *SQL* language, as shown in Listing 4.

Listing 4. Join - Apache Hive (HiveQL)

```
DROP TABLE tempJoin;
CREATE TABLE tempJoin AS SELECT
order_item.buyer_id, sum(item.goods_amount)
AS total FROM item JOIN order_item ON
item.order_id = order_item.order_id
GROUP BY order_item.buyer_id;
```

C. Metrics

LeanBench considers three metrics: the first metric is the *execution time* of an operation. This metric is used to measure the performance of each operation between the different systems. This metric will also be used for comparing other characteristics of the systems. For example, it can be used to compare the processing times as the dataset sizes increase.

The second metric is the *Data Processed per Second* (DPS), obtained by dividing the input dataset size by the total processing time, in seconds.

The third metric is the *usage of system resources*. This metric enables to make a comparison of how each system uses the available resources, such as CPU and memory usage for each operation, and also allows to identify processing performance bottlenecks.

D. Summary

The LeanBench benchmark is composed by two different categories of operations: *i)* batch processing operations and *ii)* data querying operations. The batch processing operations are performed on unstructured datasets, allowing to perform a comparison between Unicage and Hadoop. Tthe data querying operations are performed on structured datasets and allow to compare Unicage and Hive.

The batch processing operations have been chosen accordingly to the standards of other existing Big Data benchmarking tools, such as BigDataBench and HiBench. As shown previously in Section II, it is common practice in these benchmarks to use operations such as batch sorting and word counting operations applied to large volumes of data in order to benchmark batch processing systems.

The database querying operations have also been chosen to match the standards of other Big Data benchmarking tools for Data Warehousing. For example, the AMP Big Data Benchmark, BigDataBench and HiBench evaluate the performance of a system by running queries based on operations such as Selection, Aggregation, and Join queries.

IV. Evaluation

The LeanBench benchmark has been performed in a single machine scenario, carrying out all of the batch processing operations in Unicage and Hadoop and all data querying operations in Unicage and Hive.

Unicage, Hadoop and Hive have been deployed and configured in a standalone operation mode in order to work entirely from a single machine, not requiring any additional hardware.

The scenario served to make a comparison between Unicage and Hadoop for the batch processing operations and Unicage and Hive for the data querying operations using a single machine. The scenario also identifies the processing capabilities of a single machine. In addition to this, the scenario allows to identify bottlenecks, such as CPU bound, memory bound and I/O bound bottlenecks. Bottlenecks in this particular scenario might be common since a single machine may not have enough resources for processing large data sets entirely at once. For example, the data set size can exceed the machine memory resources, which may cause a memory bound bottleneck in certain operations such as sort.

A. Experiments Performed

Two sets of experiments have been performed in this work. The first set served the purpose of identifying the processing limitations of a machine with low hardware specificications. A second set of experiments has been performed on a different machine with higher hardware specifications, allowing to process datasets of larger volume. Unicage `uspTukubai`, Hadoop `2.7.3` and Hive `2.1.1` have been used for both sets of experiments. The unstructured dataset sizes are described in Megabytes (MB) or Gigabytes (GB). The structured dataset sizes are described as the total number of rows and each row takes up approximately 85 Bytes in size. The results of the benchmark present the relative performance between two systems, calculated according to the formula presented in equation 1. Variables s and f represent the slowest and fastest systems respectively, p represents the performance in percentage. For example, a value of $p = 10$ would represent a 10% performance increase.

$$p = \left(\frac{\text{Avg. Operation Execution Time}(s)}{\text{Avg. Operation Execution Time}(f)} \times 100 \right) - 100 \quad (1)$$

For Hadoop, each operation is a job that has to be read and submitted in the system. All of the samples obtained in this evaluation had a job submission time of less than one second. The job submission times for Hadoop are included in values presented in the graphs but are not identified as

they take a very small fraction of the total execution time. The same applies for Hive, as it uses Hadoop for processing. In Unicage the processing starts immediately as the script is executed.

The values of the Average Grep operation presented represent the average execution times of individual Grep operations performed using multiple regular expressions, presented in Table II.

Table II
REGULAR EXPRESSIONS USED FOR GREP OPERATIONS.

Description	Regular Expression
Match words ending in "es"	`\b\w*es\b`
Match words ending in "ies"	`\b\w*ies\b`
Match words starting with a vowel	`\b([aeiou]|[AEIOU])\w*\b`
Match words starting with a vowel followed by a consonant	`\b([aeiou]|[AEIOU])[^aeiou]\w*\b`
Match words starting with a vowel followed by another vowel	`\b([aeiou]|[AEIOU])[aeiou]\w*\b`
Match words starting with a vowel followed by another vowel, word ends with a vowel	`\b([aeiou]|[AEIOU])[aeiou]\w*[aeiou]\b`

1) Experiment Set 1: The first experiment set has been performed on a machine with low system specifications, described in Table III.

Table III
EXPERIMENT SET 1 - HARDWARE USED.

Machine Description
OS: Linux 4.4.0-66-generic # 87-Ubuntu SMP x86_64 GNU/Linux
CPU: Intel Core i3-2350M @ 2.30GHz
RAM: 2048MB DDR3-1333
HDD: 60GB 5400rpm SATA HDD

The initial tests consisted on performing batch and querying operations on small datasets, collecting 10 samples.

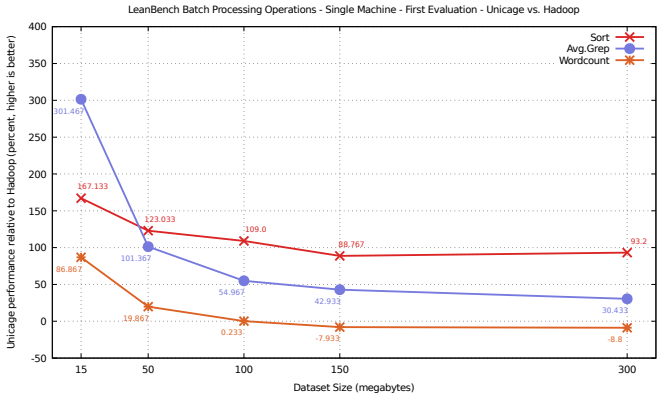

Figure 3. Unicage vs. Hadoop - Experiment Set 1.

Figure 3 describes the performance of Unicage relative to Hadoop in percentage for each batch processing operation.

The tests performed have allowed to conclude that Unicage performs better than Hadoop in all operations except Wordcount, for datasets above 100MB as shown in Figure 3. In addition, it is observed in all operations performed that the performance decreases as the volume of data to process increases.

When it comes to querying tasks, Unicage performed better than Hive in all operations. Similar to batch processing operations, a decrease in performance is observed as the volume of data to process is increased, however, it is significantly greater in comparison to batch processing, as can be observed in Figure 4.

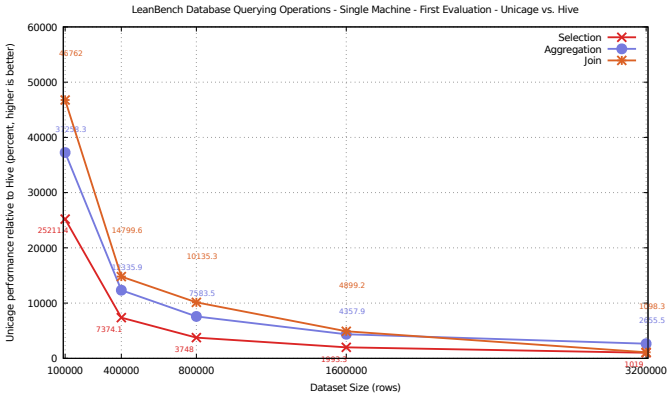

Figure 4. Unicage vs. Hive - Experiment Set 1.

After performing the tests presented above, additional tests have been carried out, using larger datasets described in Table IV.

Table IV
EXPERIMENT SET 1 - ADDITIONAL DATASET LIST.

Dataset Description	Volumes
Unstructured Datasets	2000 MB and 5000 MB
Structured Datasets	10M, 25M, 50M and 100M Rows

The additional benchmarking tests have served the purpose to verify the initial results obtained and also to identify how each processing system would respond to dataset sizes that exceed the machine hardware specifications. Batch processing operations have been performed twice for the 2000 MB dataset, and once for the 5000 MB dataset. The database querying operations have been performed 5 times each for all structured datasets.

The results obtained confirm that Unicage is faster than Hadoop for the Sort and Grep operations but there is a decrease in performance as the volume of data to process increases. For the Wordcount operation, the Unicage performance is lower and also continues to decline when compared to Hadoop.

Figure 5 presents the percentual performance of Unicage relative to Hadoop for batch processing operations of LeanBench.

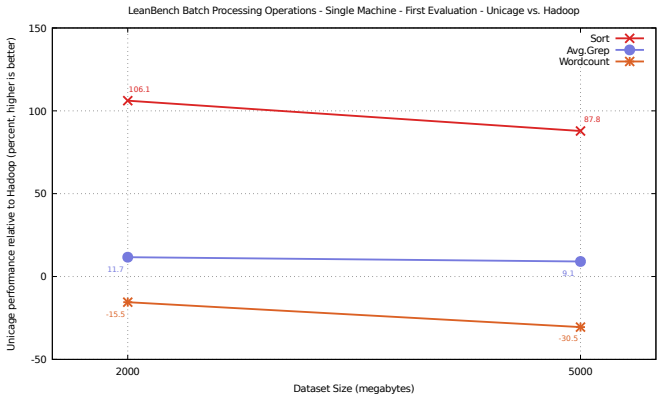

Figure 5. Unicage vs. Hadoop - Experiment Set 1 - Larger Datasets.

Unicage is faster than Hive in all of the benchmark queries, confirming the initial results obtained previously when using smaller datasets. However, while the performance of Unicage relative to Hive remains high, it continues to decrease as larger datasets are used.

Figure 6 shows the percentual performance of Unicage relative to Hive for querying operations.

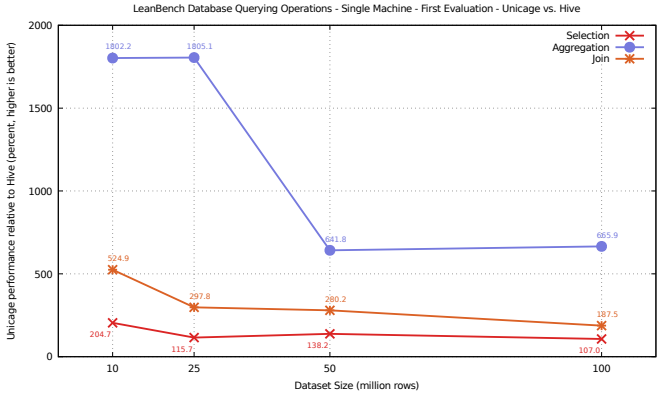

Figure 6. Unicage vs. Hive - Experiment Set 1 - Larger Datasets.

2) Experiment Set 2: After reaching the limitations of the machine used in the first experiment set, a second machine was used, with higher hardware specifications, described in Table V.

Table V
EXPERIMENT SET 2 - HARDWARE USED.

Machine Description
OS: Linux 3.13.0-117-generic #164-Ubuntu SMP x86_64 GNU/Linux
CPU: Intel Xeon E5506 @ 2.13GHz (Using two cores)
RAM: 10GB DDR3-1066
HDD: 100GB 7200rpm SATA HDD

The higher hardware specifications of this machine allowed to use larger dataset sizes for performing the benchmark operations. Table VI lists all of the datasets used in this experiment.

Table VI
EXPERIMENT SET 2 - DATASET LIST.

Dataset Description	Volumes
Unstructured Datasets	5GB, 10GB, 20GB, 40GB and 60GB
Structured Datasets	200M and 400M Rows

The batch processing operations have been performed 5 times for the 5 GB dataset, twice for the 10 and 20 GB datasets, and once for the 40 and 60 GB datasets. The database querying operations have been performed 10 times for each structured dataset.

Unicage proved to remain faster than Hadoop for the Sort and Grep operations, using the 5, 10 and 20 GB datasets, but there is a significant drop in performance as the dataset sizes increase, similar to previous results.

Figure 7 describes the percentual performance of Unicage relative to Hadoop.

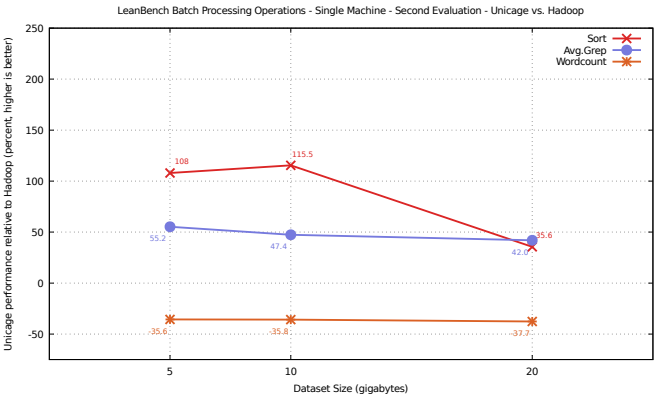

Figure 7. Unicage vs. Hadoop - Experiment Set 2.

The results allow to conclude that Unicage remains slower than Hadoop for the Wordcount operation. For the remaining operations Unicage remains faster than Hadoop but the performance continues to decrease slightly. Tests have been performed using the 40 and 60 GB datasets for Grep and Wordcount operations. Table VII presents the average operation execution times obtained in these tests.

Table VII
AVERAGE EXECUTION TIMES - GREP AND WORDCOUNT

System	40 GB Dataset	60 GB Dataset
Unicage Avg. Grep	4458.262 seconds	Failed execution
Hadoop Avg. Grep	6359.157 seconds	10669.153 seconds
Unicage Wordcount	23284.28 seconds	Failed execution
Hadoop Wordcount	15840 seconds	25108.92 seconds

Unicage terminated execution in the middle of processing the 60GB dataset, failing to produce a valid output. While

some of the Grep operations performed are not the most complex to match, the expressions match a large portion of the entries in the Dataset, resulting in a large volume of entries matched to count, which may explain why Unicage failed to process the data, similar to the Wordcount operation, which counts the entries of the entire dataset.

When it comes to the querying operations of LeanBench, Unicage produced faster results than Hive when processing datasets of 200 and 400 million rows in all the benchmark queries, similar to previous evaluations.

Figure 8 shows the percentual performance of Unicage relative to Hive.

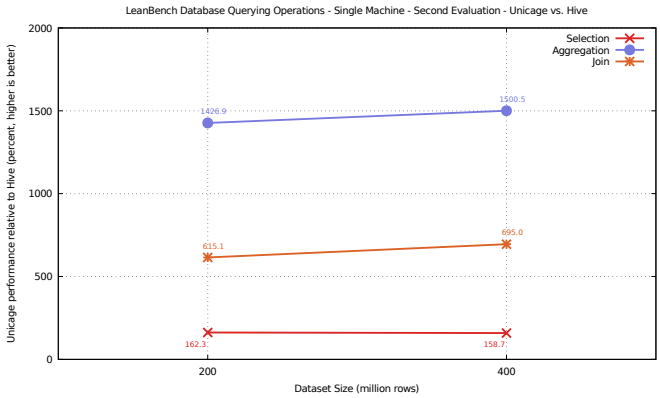

Figure 8. Unicage vs. Hive - Experiment Set 2.

B. Additional Evaluations

Unicage provides a custom sorting command ($msort$) based on the merge sort algorithm. A separate evaluation of this command proved that while the command performs faster than the Unix $sort$ command for small datasets, it lacks support for external sorting, a technique that allows to sort very large datasets that do not fit in memory by using secondary storage such as the hard-drive to store intermediate processing information.

Due to the lack of support for external sorting, the command can easily use the entire amount of memory and swap space of the machine, leaving it in an unresponsive state. Tables VIII and IX describe the execution times and memory usages for the $sort$ command of Unix and the $msort$ of Unicage, allowing to make a comparison of both sorting commands.

Table VIII
UNIX $sort$ COMMAND.

Dataset Size	Execution Time (seconds)	Memory Usage
100 MB	92.82	1347 MB
200 MB	194.97	2694 MB
400 MB	411.78	5385 MB
800 MB	857.06	7499 MB

Table IX
UNICAGE $msort$ COMMAND.

Dataset Size	Execution Time (seconds)	Memory Usage
100 MB	16.53	2282 MB
200 MB	34.51	4565 MB
400 MB	71.18	10482 MB
800 MB	Did not terminate	13161 MB

The high memory consumption makes the command unfeasible to use when processing large datasets in a single machine. For the reasons above, it was decided that the batch processing operations of LeanBench implemented in Unicage would use the Unix $sort$ command, as it supports external sorting and allows to perform tests with large datasets without being limited by the system memory resources.

In addition, Unicage provides commands that allow to split datasets into multiple parts to facilitate the processing of large datasets. However, both splitting and joining processes have costs and a separate test was performed in order to assess the minimal costs. Table X summarises the results obtained.

Table X
UNICAGE - AVERAGE SPLITTING COSTS (SECONDS) - 10 SAMPLES.

Dataset	200MB Splits	500MB Splits	1000MB Splits
20GB	572.677	567.335	569.065
40GB	1182.247	1176.253	1182.171

The minimum average cost of joining back the 20 GB dataset (based on 10 samples) was 628.333 seconds, and 1200.103 seconds for joining a 40 GB dataset. The amount of splits to join did not affect the joining costs. It is also important to note that the join costs presented take only into account the cost of joining the splitted parts back into one file. In most operations, such as Sort and Wordcount, there are additional joining costs. For example, in the Sort operation there is the cost of merging the splitted parts back together without losing the sorted entries, and for the Wordcount operation there is an additional cost of calculating the total values (sum) of the entries that share the same key. This shows that splitting datasets introduces significant costs that have to be considered as part of processing times.

C. Experiment Overview

The batch processing benchmarking operations have shown that Unicage performs better than Hadoop in all of the operations for datasets sized below 100MB. However, Unicage performs worse than Hadoop when processing the Wordcount operation with datasets larger than 100MB. While Unicage remains faster than Hadoop for the remaining batch processing operations, the performance declines as the dataset sizes increase, indicating that there may be a point where Unicage starts performing worse than Hadoop.

In particular, this fact has been observed in the second set of experiments performed, where Unicage failed to produce a valid output when performing Grep and Wordcount operations on a 60GB dataset.

For data querying benchmark operations, Unicage performs better than Hive. Similar to batch processing results, the performance of Unicage when compared to Hive also declines as the dataset sizes increase, but there are some exceptions. In the second set of experiments, a performance increase has been observed in Unicage when performing the Aggregation and Joining queries for the 400 million row dataset in comparison to the 200 million row dataset, as shown in Section IV-A.

When it comes to bottlenecks, memory bound bottlenecks have been identified when using the Unicage $msort$ command as described in Section IV-B. The memory usage does not exceeds the total system memory, even when the dataset sizes surpasses the system memory capacities. This is explained due to the fact that Hadoop, and Unicage (when using the Unix $sort$ command) support external sorting techniques. System resource usage logs show that all operations of LeanBench are mainly limited by the CPU time in both systems, however, additional memory would allow to process larger datasets faster by not relying as much on external sorting, which requires secondary slower memory such as the hard-drive.

Nevertheless, in a single machine setup, having support for external sorting capabilities is essential in a processing system, as the dataset sizes can easily exceed the system memory capacity. In addition, introducing additional system memory is not always an option in this type of setup as there are physical and hardware limitations of how much memory can be added to a system.

V. Conclusion

LeanBench, a benchmark tool for evaluating the performance of Big Data processing systems has been proposed and created, allowing to perform a benchmark for two different categories of systems: batch processing systems such as Hadoop, and data warehousing systems that include data querying functionalities, such as Hive. In addition, LeanBench includes workloads that support benchmarking Unicage, a tool composed by multiple individual commands that in conjunction are able to perform both batch processing and database querying tasks.

A. Results Overview

For database querying tasks, Unicage has an advantage over Hive, which can be explained due to the reduced software stack. While Hive relies on the entire Hadoop stack for performing the processing, Unicage exclusively performs the individual processing commands specified in the querying script of the task. However, the querying language of Hive does present an advantage to Unicage as users can directly write queries in *HiveQL*, a process very similar to writing *SQL* queries. In Unicage, it is not possible to write queries directly in an *SQL*-like language, and queries must instead be written using individual low-level processing commands.

When it comes to batch processing tasks, Unicage performs better than Hadoop when processing volumes of data lower than 100MB but the lack of support for external sorting make Unicage perform worse than Hadoop for processing larger volumes of data. As described in Section IV-B, Unicage provides commands to split datasets into multiple individual parts to facilitate processing large volumes of data, but this process introduces significant costs that have to be taken into consideration.

In a single machine scenario, the small software stack of Unicage presents advantages and disadvantages. The small software stack of Unicage allows to process some tasks faster than other systems by avoiding unnecessary processing steps. However, Unicage is at a disadvantage when performing tasks that require sorting large volumes of data.

B. Guidelines

The best processing system to use depends largely on the processing task. In general, for batch processing Hadoop is the best choice if the processing task requires sorting large volumes of data that are larger than 100MB. If the processing task does not involve sorting large volumes of data then Unicage is a faster alternative to Hadoop. For processing tasks that involve querying structured datasets, Unicage is the best choice, at a cost of lacking a dedicated querying language. If a querying language similar to *SQL* is required then Hive is the best choice as it supports *HiveQL*, at a cost of performing significantly slower than Unicage.

Figure 9 presents a flowchart with the best system choice to some of the most common processing tasks and requirements.

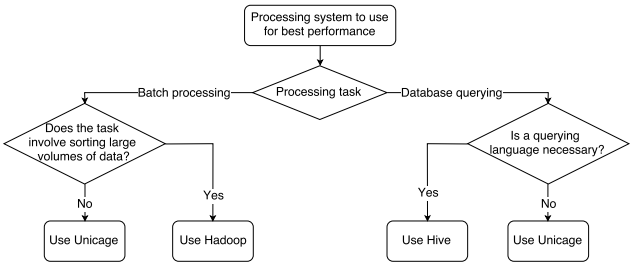

Figure 9. Best processing system according to different processing tasks and specific requirements.

In conclusion, there are multiple systems designed for each kind of processing task. There is not a single system that is best for every existing task, as each processing system presents multiple advantages and disadvantages. This work

has presented a quantification of the cost in processing performance by having large software stacks with additional layers.

Unicage would benefit by having additional abstraction layers allowing to handle the processing of large datasets. The additional software layers of Hadoop and Hive provide ease of use for developers by providing multiple features that are characteristic of Big Data processing systems, allowing to reduce development and maintenance times, at a cost of slower processing performance: a tradeoff between development and processing complexity.

C. Future Work

The experiments performed in this work have allowed to perform a single machine comparison between Unicage and Hadoop for batch processing tasks, and a comparison between Unicage and Hive for database querying tasks. However, the single machine experiments performed do not allow to make an assessment of how the different systems perform in a cluster configuration. Using a single machine restricts the volume of data that can be analysed in experiments and requires processing systems to work from a single machine, when they have been designed to work in a cluster setup, which is the case with Hadoop.

A cluster setup is the most commonly used when dealing with Big Data. The processing nodes of a cluster allow to scale-out the processing by distributing the work along nodes, enabling to process larger amounts of data significantly faster when compared to a single machine setup. In addition, the cluster setup can always be expanded by adding extra processing nodes according to the specific requirements of processing tasks. This is not possible with a single machine configuration as there are hardware limitations on how much a single machine can be upgraded.

A future cluster evaluation is necessary to make a performance assessment of each processing system. Cluster benchmarking experiments require the different systems to be deployed in a cluster. Hadoop supports cluster configurations natively, but Unicage is designed to work from a single machine, and requires additional software to be able to be deployed in a cluster. When it comes to the benchmarking tool, LeanBench has been designed to work in both single machine and cluster scenarios.

In conclusion, experiments performed in this work have allowed to make a single machine performance comparison of the systems but a future evaluation is necessary to assess cluster scenario performance.

References

[1] Dieter Uckelmann, Mark Harrison, and Florian Michahelles. *Architecting the Internet of Things*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[2] USP Lab. How to Analyze 50 Billion Records in Less than a Second without Hadoop or Big Iron. Technical report, July 2013. MIT-CRIBB, Universal Shell Programming Laboratory.

[3] USP Lab. Unicage FAQ. Technical report, 2016. Universal Shell Programming Laboratory.

[4] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI04: proceedings of the 6th Conference on Symposium on Operating Systems Design and Implementation*. USENIX Association, 2004.

[5] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM, 2013.

[6] Dhruba Borthakur. Hdfs architecture guide. Technical report, 2008.

[7] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive: A warehousing solution over a map-reduce framework. *Proc. VLDB Endow.*, 2(2):1626–1629, August 2009.

[8] Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. Bigbench: towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of data*, pages 1197–1208. ACM, 2013.

[9] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, et al. Bigdatabench: A big data benchmark suite from internet services. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 488–499. IEEE, 2014.

[10] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J Abadi, David J DeWitt, Samuel Madden, and Michael Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 165–178. ACM, 2009.

[11] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *New Frontiers in Information and Software as Services*, pages 209–228. Springer, 2011.

[12] Shengsheng Huang, Jie Huang, Yan Liu, Lan Yi, and Jinquan Dai. Hibench: A representative and comprehensive hadoop benchmark suite. In *Proc. IEEE International Conference on Data Engineering (ICDE)*, 2012.