An experimental and comparative benchmark study examining resource utilization in managed Hadoop context

Uluer Emre Özdil¹ ² · Serkan Ayvaz³

Received: 23 December 2021 / Revised: 15 August 2022 / Accepted: 25 August 2022 / Published online: 5 September 2022

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract

Transitioning cloud-based Hadoop frameworks from IaaS to PaaS, which are commercially conceptualized as pay-as-you-go or pay-per-use, often reduces the associated system costs. However, the managed Hadoop systems obscure the inner performance dynamics of the platform and present a black-box behavior to the end-users. The aim of this study was to investigate the resource utilization of current managed Hadoop platforms. Thus, we explored three prominent Hadoop-on-PaaS proposals as they come out-of-the-box and conducted Hadoop-specific workloads using the HiBench Benchmark Suite. During the benchmark executions, the system resource utilization data from the worker nodes were collected and analyzed. The results indicated that the same property specifications among cloud services neither do guarantee similar performance outputs, nor produce consistent results based on different workloads within themselves. We anticipate that the managed systems’ architectures and pre-configurations play a crucial role in the performance outcomes.

Keywords  Big Data · Managed Hadoop · Hadoop-on-PaaS · HiBench · Performance evaluation

1 Introduction

As an emerging field, Big Data has potential to deliver major benefits to organizations by transforming data processing architectures. The term, Big Data refers to datasets associated with massive size, ever-growing speed, and a variety of data structures. By using complex data-driven models, Big Data technologies can help extract key insights from a vast amount of data. Recently, new systems and approaches have emerged to help facilitate the maintenance of Big Data environments.

Distributed storage and computation frameworks owe a great deal to web search engines for inheriting the motivation and foundational work previously done by the open-source community. These collective efforts have spawned what is known today as the Hadoop framework [1] and its ecosystem. Hadoop offers a modern analytical solution to the complex issues of processing big data. By providing seamless scalability and fault-tolerance to affordable commodity computers, it helps reduce operational and administrative efforts and associated costs for handling big data.

The commercialization of Cloud Computing in the midst 2000s [2] delivered high availability of storage and computing resources to end-users at affordable prices. This offered organizations a cost-effective alternative to the upfront infrastructure investment in purchasing hardware technology that would soon become obsolete and expensive to maintain. As part of ongoing process of cloud migration, many platform providers have also began serving IaaS systems that include virtual machine implementations of Hadoop instances, enabling Hadoop to be deployed from on-premise residence to the cloud.

Moreover, the Cloud Service Providers (CSP) have embraced the need to eliminate the complex implementation process of Hadoop on multi-node VMs by providing managed Hadoop systems that are commercially available as PaaS. These platforms allowed Hadoop clusters to be
created in a matter of minutes with hundreds of pre-installed and pre-configured computer nodes by specifying only a few configurations such as hardware specifications and the number of nodes prior to installation. By leaving the rigorous implementation work unrelated to the core purpose of the analysis to a service provider, end users can save a great deal of time and effort.

Nevertheless, this requires a trade-off. By definition, managed systems are prepackaged solutions delivered in a black-box manner. It means that CSPs apply behind-the-scenes tweaks to achieve better performance on selected approaches including memory-intensive or compute-intensive applications. However, the utilization dynamics of a managed system remains unclear from the end-user’s perspective.

Setting up and running a Hadoop cluster consists of intricate processes that require a large amount of upfront investment in hardware, highly skilled technical staff, and most importantly time. Utilizing tools that measure the performance of a big data framework against predetermined evaluation criteria can stem from a variety of motivations. First, enterprises tend to assess benchmark reports before approving costly investments. From the providers’ perspective, benchmark reports are strong marketing arguments if they perform satisfactorily. Otherwise, they become road maps for improvement. As for researchers, benchmarks are useful tools for testing different sets of hardware or software and validating optimizations made on respective systems to provide more confident results.

To provide reliable outputs, benchmarks should be representative and generate realistic workloads to be implemented in widely accepted use cases. Big Data frameworks such as Hadoop are complex architectures that provide various components including SQL-on-Hadoop, Machine Learning, graph engines, etc. The challenge for benchmarking frameworks remains in meeting these requirements. Since the invention of relational databases, there are organizations such as TPC [3] and SPEC [4] that have focused on providing industry standard benchmarks and updating their efforts according to cloud computing requirements. As extensively reviewed in the study by Han et al. [5], benchmarks are work in progress and require constant effort to stay up to date with developments in big data frameworks.

The study is an attempt to understand the system resource utilization behaviors within pre-configured managed Hadoop services during benchmark execution, to address shortcomings of systems’ structures and to discuss potential causes. Hence, the main contributions of the study can be listed as follows:

- We investigated resource utilization behaviors and inner-workings of selected pre-configured managed Hadoop services in their default configurations.
- To the best of our knowledge, this is the first study attempting to perform a comprehensive benchmark evaluation on a diverse set of pre-configured managed Hadoop services consisting of Google Cloud Dataproc, Azure HDInsight and Alibaba Cloud E-MapReduce.
- From the end-users perspective, we assessed the strengths and drawbacks of the systems and further explored possible reasons by performing a comparative benchmark study in an experimental environment.

### 2 Architectural concepts and background

Since Big Data emerged as a new field of research, novel approaches have been proposed to facilitate distributed storage and computing paradigms as an ongoing development process to overcome operational and analytical challenges. In this context, Hadoop has been embraced by a wide spectrum of beneficiaries from industry and academia since its launch in 2005. In its core functionality, Hadoop centres around the following three concepts:

- HDFS filesystem; for storing a massive amount of data across a cluster of nodes,
- The MapReduce framework; for distributed computation, and
- YARN; for allocating available resources for the requested tasks.

#### 2.1 HDFS

Hadoop Distributed File System, which stands for HDFS; was developed with inspiration from the guidelines described in the 2003 whitepaper [6] on Google’s File System (GFS). GFS is a distributed storage paradigm that can handle petabytes and larger-scale data volumes in a fault-tolerant cluster.

HDFS stores large volumes of data by splitting them into fixed-size data blocks, typically 128 MB each. In HDFS, each block is replicated to different nodes by a factor of 3. These values are the default configurations and can be customized. When the data file is requested, corresponding blocks are retrieved from the nodes across the cluster. The redundancy of the blocks guarantees high availability. If one or more nodes goes out of service, the requested data blocks are collected from the existing redundant copies on other nodes.

HDFS is a co-existing file system on the nodes it is installed on. It provides a globally distributed view of the
files across the cluster, thus it is possible to list an HDFS directory from all nodes. Files in HDFS are listed as they would exist on a local file system, but the physical partitions of the files are located in other physical locations.

Figure 1 depicts an overview of HDFS. The architecture of HDFS comprises a Namenode; a dedicated machine to keep track of files and folders and corresponding metadata such as data block locations in the cluster, and the data nodes where the data blocks are located [7]. Namenode is the single-point-of-failure. This means that if the namenode is down, the entire Hadoop system is out of service.

To overcome this issue, starting with version 2, Hadoop evolved to the concept of High Availability. As shown in Fig. 2, there are two namenodes, an active namenode, and a standby namenode, which communicate with the data nodes and store edit logs in a shared folder. As the name suggests, the active namenode is in charge, while the standby node acts more like a shadow system. Whenever the active namenode goes down, the standby namenode is activated so that the services remain available to end-users.

### 2.2 MapReduce

Similar to HDFS, Hadoop’s MapReduce (MR) is an open-source implementation of the MapReduce framework described in a Google whitepaper [8] published in 2004. As illustrated in Fig. 3, an MR flow starts with assigning blocks from HDFS as input splits to mappers. The computed intermediate results are then shuffled and transmitted to the reducers where outputs are sent back to the client node. Nodes running mappers and reducers provide parallel processing. The framework enables seamless scaling of processing power by simply adding new commodity computers to the cluster.

The computation is done locality in the nodes where corresponding data blocks reside. Executing the computation in local data blocks eliminates the need to move data between nodes for processing. In other words, MapReduce moves the code to the local data nodes for processing instead of moving data to separate computational nodes. This process is much more efficient and scalable than the reverse.

### 2.3 YARN

Hadoop version 2.x includes significant architectural improvements in terms of resource management. Version 1.x of Hadoop suffered from shortcomings due to the overload of its resource management tasks. These tasks were handled in MRv1, where job tracker node and task tracker nodes were running the organizational load of MapReduce executions. YARN, which stands for “Yet Another Resource Negotiator”, emerged as a middleware between HDFS and MapReduce, inheriting some of the overhead previously performed by MRv1. As demonstrated in Fig. 4, YARN has become a gateway for batch and streaming operations, interactive queries, and graph processing engines based on HDFS file system.

After YARN, MRv2 became more efficient at processing the intended tasks as it was freed from resource management overhead compared to the previous version. Also, YARN brought innovation to Hadoop by introducing new architectural elements such as the resource manager, a node dedicated to tracking available resources in the cluster, and the node managers, which are located on each worker node that launch containers and monitor their resource usage.

### 2.4 Selection of managed Hadoop services

Service-level agreement (SLA) comprising configuration and pricing optimizations are of dynamic nature. This means that improvements in performance or cost can be
implemented at any time. Thus, favoring some of the services over others in terms of cost or performance can quickly lose ground and become meaningless unless intended by a company to determine which services to choose from.

The main focus of the comparison in the study is not the performance differences between various managed Hadoop proposals, but the way their worker nodes utilize system resources when processing data based on their pre-configured settings. We refrain from commercially favoring any of the providers over the others, we do not evaluate their business value. Simply put, the objective of the study is to examine the managed Hadoop context in an experimental environment and comparatively observe the system utilization behaviors in each cluster based on the default configurations determined by the provider. We think that this approach makes our findings longer lasting, relevant, and free from potential impacts of overnight SLA updates. The benchmark results should not be interpreted as a competition between the service providers as it is not our goal to determine the efficiency of the providers relative to the others.

According to Gartner’s 2020 Magic Quadrant for Cloud Infrastructure and Platform Services [10], Amazon Web Services was the market leader, followed by Microsoft Azure and Google Cloud Platform in the leaders section. Alibaba Cloud, Oracle, IBM, and Tencent Cloud reserved their places in the Niche section. Recently, Gartner’s 2021 Magic Quadrant for Cloud Infrastructure and Platform Services, revised Alibaba Cloud’s rank from Niche to the Visionaries section. As for the experimental environment, we focused on the mid-level range in the study. This type of environment appears to be targeted by Microsoft Azure, Google Cloud Platform, and Alibaba Cloud.

Since our objective was to evaluate the resource utilization of managed Hadoop systems rather than their market positioning strategies, the current market position of the respective benchmark subjects did not play a dominant role in our selection process. Due to the resource limitations in the study, we were required to consider three CSPs only for the evaluations and choose between the options of AWS-EMR and Alibaba Cloud E-MapReduce. Cloud Computing is a rapidly growing and disruptive technology; we think new players should be taken into account. As there are studies evaluating GCP Dataproc, Azure HDInsight and AWS-EMR as presented in the subsequent Related Work section, and considering its transition potential to move to a higher level, we decided to include Alibaba Cloud in the evaluations.

2.4.1 Microsoft Azure (HDInsight)

HDInsight [11], Azure’s solution to manage Hadoop and Spark platform; is a product of the collaboration with Hortonworks that brings the Hortonworks Data Platform (HDP) to a cloud platform [12]. HDInsight differs in architecture from its counterparts by not having a Hadoop cluster installed on the cloud VM service layer, i.e. Azure Virtual Machines. Instead, it is a cloud-optimized HDP platform.

Another difference is that HDInsight mandates WASB, the Azure blob system, as the storage system, including the option to leverage Data Lake Storage by excluding Hadoop’s native file system HDFS from the options. At the time of the study, Azure required Hadoop to use High Availability mode with two master nodes but leave the hardware choice to end-users.
2.4.2 Google Cloud Platform (Dataproc)

Google’s managed Spark and Hadoop solution, namely Dataproc [13]; is a pre-configured Hadoop on PaaS service built on pre-installed VM instances on Compute Engine [14], another service of GCP. A number of operating systems including Debian and/or Ubuntu, which were proposed as pre-installed images during the Dataproc installation process, were offered at the date of the study.

Dataproc framework includes Hadoop core elements HDFS, YARN, and MapReduce by default, while various components related to the Hadoop ecosystem are made available as well. The end-user is also offered the choice to utilize Google’s proprietary cloud storage service, Cloud Storage, to store data for the long term as the Dataproc cluster must be terminated after use. By using Web UI or local Command Line Interface (CLI) via API, the end-users can access the Dataproc cluster and single VM instances within Compute Engine. Google offers a large number of data centers around the world.

2.4.3 Alibaba Cloud (E-MapReduce)

Alibaba Cloud’s managed Hadoop service E-MapReduce [15] utilizing Apache Hadoop and Apache Spark is placed as a service layer on its Elastic Compute Service (ECS) [16], a similar approach to the one of GCP’s. Although the number of data center locations Alibaba provides outside of Mainland China is not as large as its peers, there are competitive regions in the United States, Europe, Middle East, and the Asia Pacific. Alibaba Cloud’s managed service differs in its pre-installed Operating System. Aliyun Linux 2, a Linux distribution based on CentOS, is the open-source version of RedHat Linux developed by Alibaba Cloud.

AliyunOS proclaims to provide a stable and reliable environment optimized for the Alibaba Cloud infrastructure and is available as open source in GitHub repository [17]. E-MapReduce offers a wide variety of machine types with different specifications for specific purposes such as CPU intensive or memory-intensive tasks. As with GCP, Alibaba Cloud E-MapReduce allows the end-users to specify the storage type between the options of HDFS and Alibaba’s cloud storage, Object Storage Service (OSS).

2.4.4 Architectural divergences and similarities among selected providers

Since it is the architecture what determines resource utilization patterns and overall performances, it is worth taking a look at the architectures of selected services. It’s clear at first sight how Azure HDInsight differs by infrastructure choice: an all-in-one big data framework, originally called HDP, configured to handle out-of-the-box big data workloads, and optimized for Azure Cloud. The HDInsight way of accessing the instances of the cluster was not from a virtual machine service layer, but from a command line access to the instances.

GCP Dataproc and Alibaba Cloud E-MapReduce architectures showed both similarities and differences. Both CSPs leveraged virtual machine services (Compute Engine and Elastic Compute Service, respectively) for the cluster. But this is also where the similarity ends. While GCP provided Debian and Ubuntu images for the instances, Alibaba Cloud made use of its own operating system Aliyun Linux 2, which was derived from the CentOS operating system. It is claimed to be open source and optimized for Alibaba Cloud. Single nodes are accessible via service user interfaces or command lines on both providers.

Another architectural aspect to keep in mind when evaluating the results might be the providers’ approach in storage. GCP and Alibaba Cloud allow the end users choose whether they want to store data on local disk and HDFS of the cluster’s instances or prefer data isolation by selecting the provider’s cloud storage service (Google Cloud Storage and Object Storage Service, respectively). By choosing HDFS on the local disk, end-users can also specify the size of the local disk. In Azure, this option was not available to end users for some reason. Azure blob storage was the only storing option. While it’s petabyte limits supports scalability, considering that the local disk size still matters (intermediate Mapper data are written to local disk during MapReduce execution), giving this option could have resolved some issues before they occur during the benchmark executions, as described in the next sections.

2.5 HiBench benchmark suite

HiBench is a comprehensive and representative benchmark for Hadoop [18]. It was introduced to the open-source community in 2010. It helps to evaluate different big data frameworks in terms of speed, throughput, and system utilization. At the time of the study, the current version of HiBench was 7.1 and contained 29 benchmarks categorized in 6 sections (Micro, Machine Learning, SQL, Websearch, Graph, and Streaming) [19]. Table 1 lists the Hadoop-related workloads we used in HiBench 7.1.

The benchmarks in Micro category were taken from Hadoop’s native benchmark tools; where Dfsioe is the enhanced version of its originator, Dfsio. It calculates the average I/O rate, average throughput of each map task, and the total throughput of the cluster. The “Engine” column refers to the dependencies of HiBench that were
downloaded when Apache Maven compiling the benchmark suite, and exploited during benchmark executions.

Hive engine enables running HiveQL queries for Scan, Join, and Aggregation benchmarks. The Hadoop based ML benchmarks are run by Mahout. Nutch Engine is utilized for Nutchindexing benchmark. Pegasus, a peta-scale graph mining system, computes PageRank algorithm. The dependencies execute benchmark tasks by translating respective jobs into MapReduce.

HiBench’s predefined default data scales i.e. Tiny, Small, Large, Huge, Gigantic, and Bigdata, are ranging from 32 kB up to 300 GB for Sort, 6 GB for Terasort, and 1.6 TB for Wordcount. User-defined data scales can also be implemented in HiBench configurations.

3 Related work

Big data has emerged as an active area of research recently and has attracted researchers from various perspectives from the open-source community, commercial enterprises and academia. Due to the high demand for big data tools and solutions, the number of Big data technologies available is increasing rapidly [20]. As the scales and requirements of institutions differ greatly, the task of choosing the most suitable big data systems has become even more challenging.

Big data benchmarking plays a key role in evaluating the performance of big data systems for various use cases at different levels [21, 22]. Numerous studies from the big data and cloud computing research community have recently explored the performance of cloud-based Hadoop clusters in different contexts, including effective distributed task scheduling [23], energy consumption efficiency [24], optimization of resource allocation [25, 26], and data-driven performance tuning [27]. In the review of related work, we focused on the state-of-art Hadoop benchmarking approaches, with an emphasis on managed Hadoop systems.

Poggi et al. [28] conducted a comprehensive study for benchmarking Hadoop PaaS services provided by Amazon (EMR), Google (Dataproc), and Azure (HDInsight). The benchmark tool they used, namely BigBench (TPCx-BB), is an industry-standard benchmarking framework developed by the Transaction Processing Performance Council (TPC). TPCx-BB comprises 30 business use cases and differs from SQL-only benchmarks by also requiring other frameworks such as MapReduce, User Defined Functions (UDF), Natural Language Processing, and Machine Learning.

The objective of their study was twofold: First, to characterize BigBench queries and out-of-the-box performances of Hive and Spark in the cloud, and second, to compare the cloud vendors for reliability on a data scale ranging from 1 GB to 10 TB. The medium-size clusters under the test consisted of 16 cores and 60+GB memory for the master node and 16 data nodes with a total of 128 cores. Executing TPCx-BB with data scales showed that Hive outperforms in scales up to 1 TB, while at 10 TB data scale, Spark performs up to twice as fast as Hive.

Wang et al. [29] utilized HiBench for assessing the effectiveness of their optimization solution on Hadoop’s MapReduce framework. The problem definition was targeting the current MapReduce framework’s inability to handle intermediate data efficiently. The output of maps was physically stored on the disk. Subsequently, it read from the disk and transmit to the reduce slots. As the number of mappers and reducers increases, expensive disk seeks occur, resulting in high execution times. The proposal of researchers, Aggregation, Partition, and Allocation (APA), aggregated intermediate data in each rack into a single file, and a file host pushes the data to reduce tasks. Benchmarking the experimental setup consisting of 50 data nodes across 10 racks and 40 Gbps rack interconnectivity yielded 40–50% performance improvement.

Hwang et al. [30] used HiBench and four other benchmarks (YCSB, CloudSuite, BenchClouds, and TPC-W) on Amazon EC2 instances introducing new performance metrics applicable to IaaS, PaaS, SaaS, and hybrid clouds. The cloud scaling strategies of vertical scaling (scale-up), horizontal scaling (scale-out), and mixed approach were discussed and evaluated in the study. The researchers conducted the aforementioned benchmarks on cluster scale strategies. The study has reached multiple conclusions about proposing new performance metrics, differentiating between scaling-up and scaling-out strategies and applicable use cases, the performance of scale-out vs. cost-effectiveness of scale-up approaches, and the close relation between cloud productivity and system elasticity, efficiency, and scalability.
Ahn et al. [31] put Spark on YARN’s performance on the test with HiBench to handle a flood of data generated by IoT devices. The experiment was performed on a cluster with one master and three worker nodes, each node possessing an Intel® Xeon® processor with 20 cores and 128 GB main memory with a total of 60 cores and 384 GB memory. HiBench workloads Micro (comprising Sort, TeraSort, and Wordcount), SQL (comprising Aggregation, Join, and Scan), and Machine Learning (comprising Bayes, Logistic Regression, Gradient Boosting Tree, Random Forest, and Linear Regression) were considered with the scale of 30 GB data. Spark occupied memory during the entire job execution, reducing the negative impact of I/O on processor performance.

The authors modified YARN’s minimum memory allocation and Spark executor settings to optimize resource usage so that the overall loads of Spark executors remain below the total system memory. In addition to HiBench’s time and throughput report, CPU/memory utilization and disk throughput were profiled as well. The findings of this study indicated that Spark guarantees performance when sufficient memory provided.

Han et al. [32] examined the impact of memory size on big data processing through Hadoop and Spark performance comparison, using HiBench’s k-Means workload as the sole benchmark. A k-Means benchmark was conducted to compare Hadoop and Spark for memory sizes of 4, 8, and 12 GB, and data scales ranging from 1 to 8 GB. The results demonstrated that Spark outperformed Hadoop as long as the total input data size was smaller than 33.5% of the total memory size of worker nodes. After reaching that ratio, Spark suffered from insufficient memory resources. This led to its interoperability with HDFS which caused a sharp drop in performance and put Hadoop ahead in throughput and duration performance.

They carried out a second experiment to determine whether Spark’s performance could be improved by changing the allocation settings for storage memory and shuffle memory while remaining within the specified memory limitations of 4, 8, and 12 GB. The authors interpreted the benchmark report as an improvement from 5 to 15% in Spark’s processing time.

Samadi et al. [33] conducted an experimental comparison between Spark and Hadoop installed on virtual machines on Amazon EC2 using nine of the provided HiBench workloads. For the reliability of the comparison results, the authors evaluated the workloads three times with input data scales of 1, 3, and 5 GB, respectively. Based on the outputs comprising duration, throughput, speed up, and CPU/memory consumption, it was concluded that Spark consumed less CPU resources and performed better across all workload results than Hadoop.

More recently, Ahmed et al. [34] investigated the predictability of execution times of generic Big Data workloads of Spark applications in Hadoop clusters. They developed a model to estimate the parallelization performance of Spark jobs in terms of runtime based on the number of executors in the cluster. The authors evaluated the performance of the proposed statistical model using five different workloads of HiBench Benchmark suite: WordCount, PageRank, Graph, SVM, and Kmeans. While they observed some predictable patterns in the experimental benchmark evaluations, they also noticed that adding more executors to the cluster did not guarantee a shorter runtime. Contrarily, it can potentially increase the duration of job execution for some workloads.

In another related work, Shih et al. [35] investigated the performance of virtualization technologies in data centers. They compared the capabilities of three distributed data processing environments: Docker containers, virtual machines and bare-metal servers. The authors evaluated the performance and the key features of each environment using two Hadoop workloads (WordCount and TeraSort) based on HiBench. They noticed that the container-based environment outperformed virtual machines in terms of execution time and had comparable performance to physical servers. Moreover, they integrated Docker containers with OpenStack to assess the maintenance and deployment process of the containers. They noted that the containerized cluster provides faster task execution and a less complex deployment process than traditional virtualization.

4 Materials and methods

We conducted the benchmark study in two use cases. Use Case 1 aimed to cover the overall resource utilization within HiBench’s predefined data scales, namely Huge and Gigantic. Workloads from the Hadoop specific benchmark categories (Micro, SQL, ML, and Websearch) were executed. In Use Case 2, we selected two workloads: I/O intensive Sort and CPU intensive Wordcount. The data were scaled up by using HiBench’s predefined data scales: Tiny, Small, Large, Huge, and Gigantic.

4.1 Experimental setup

This section covers cluster installations, building and running HiBench on installed clusters, capturing system resource utilization data across worker nodes, and visualizing the results of evaluations in the study. The files and codes used in the evaluations along with detailed documentation for the experimental environment setup are available to interested researchers in our GitHub repository [36].
4.1.1 Cluster installations

For the purpose of benchmarking managed Hadoop systems with out-of-the-box configurations, we adhered to three principles, given the availability of options:

- The geographic locations of data centers shall be the same or close,
- The computation power, memory and storage capacity of the clusters shall be same or as close as possible, and
- The Hadoop versions among the offerings shall be same or as close as possible to each other and remain within version limits supported by HiBench 7.1.

Consequently, Frankfurt was selected as the data center location for all providers. Within the given options, we selected 8 CPUs/64 GB RAM for the master node and 4 CPUs/32 GB RAM per worker node with a total of 12 CPUs and 96 GB of computing power in the cluster.

Although the selection process with compatible hardware specifications was straightforward, the software selection step involved difficulties in finding identical versions of Hadoop. A wide variety of software configurations were provided by all vendors. An exact version alignment, including minor release levels of Hadoop and related software was not possible. As mentioned in Sect. 2.3, Hadoop received a major update with version 2, hence our solution was to select the Hadoop versions with minor differences. After making sure that the releases of 2.7.3, 2.8.5, and 2.9.0 were compatible, they were selected for the evaluations. The specified cluster installation options are given in detail in Table 2.

4.1.2 Building and running HiBench

Without applying any performance tweaks to the respective configurations, we immediately executed the benchmarks. We only modified the default configuration in one case, where benchmark execution was blocked as reported in Sect. 5.2.

Hadoop related benchmarks of HiBench were executed in the categories Micro (Sort, Terasort, Dfsioe, and Wordcount), SQL (Scan, Join, and Aggregation), ML (Bayes and Kmeans), and Websearch (Pagerank).

The build and run process in HiBench was straightforward: Downloading the source files, building HiBench with Apache Maven, setting HiBench configuration files, and running the selected benchmarks. The benchmark results comprising data such as duration, throughput, and associated MapReduce execution plans were logged. HiBench was implemented only on the master node of the cluster, meaning no additional work was required on the worker nodes.

4.1.3 Capturing resource utilization data

By using a low-footprint bash script based on sar commands, we collected the system utilization logs addressing CPU, Memory, and I/O utilization on each worker node of the cluster during the benchmark execution. The captured data enabled us to visualize the system utilization on each worker node within the cluster.

Due to the small size of the experimental cluster limited to three worker nodes, we excluded the network activity track. Our intuition was that data locality was either provided, or data blocks were handled within the rack locality, which was expected to have the least impact on network performance.

To capture utilization data, we modified a sar script according to our requirements (the original source code was provided by its developer on GitHub [37]). The data capture script was uploaded to each worker node and started manually before the benchmark execution. Upon the completion of benchmark, the scripts running on the worker nodes were immediately terminated (Ctrl+C), resulting in raw data being stored on disk.

Benchmark execution and the collection of utilization data are two separate processes. To ensure the reliability of utilization data, we manually removed redundant log lines from the utilization data files. In other words, the logged lines were deleted from the utilization data records, if they were caught after the benchmark completion time. The reference datetime for this action was the completion time of the benchmark in the benchmark report.

4.1.4 Visualization

The plots were generated based on the benchmark results to demonstrate cluster utilization data. We created two types of plots to observe the overall average utilization and how utilization behaves over the execution time:

**Spiderplots—for average system utilization** Spider (radar) plots are useful tools when displaying multiple variables within the same graph, its use cases are mostly for comparative reasons. With these plots, we aimed to display the average system utilization based on the benchmark result. A plot comprises six axes that represent the normalized average values for processor (CPU), Memory (MEM), IO-read and IO-write (IO-R and IO-W, respectively) utilization in the cluster. Benchmark results are represented as duration (DUR) and throughput (TPT). The application of a coloring convention enabled to differentiate between the utilization results of managed services.

The normalization scale was set between 0 and 1, where higher values indicate the better. For the sake of proper
The results of the benchmark evaluations are provided in two parts: Sect. 5.1 presents the overall benchmark results in tabular format in Tables 3 and 4, and average resource utilization in form of spider/radar charts in Figs. 5, 6, and 7. The Slowdown Estimate (SE) values, which make it easy to interpret the benchmark results in a comparative way, are stated in Tables 5 and 6.

The performance of Hadoop relies on the number of allocated mappers and reducers specified by the Hadoop configurations of a cluster. Therefore, as another important aspect, the number of mapper and reducer slots allocated for each cluster is also given in Tables 7 and 8.

Moreover, we assessed three cases in Use Case 2 (Terasort, Bayes, and Kmeans) in Sect. 5.2 for both Huge and Gigantic data scales. In these benchmarks, a cluster’s resource utilization was either terminated due to a failure as shown in Fig. 9, or a prolongation in utilization impacted the benchmark duration as illustrated in Fig. 11.

A contradictory case occurred in HDInsight for two benchmarks of Bayes and Kmeans using the same engine.
of Mahout. While the latter workload produced outstanding results (Figs. 12, 13), the first one was unexpectedly lagged behind others as shown in Figs. 10 and 11. To understand if it was due to an architectural issue or a misconfiguration, we explored MapReduce execution logs stored by HiBench.

### Table 3 Use Case 1 benchmark results

| Category | Benchmark   | IDS | Dataproc | HDInsight | E-MapReduce |
|----------|-------------|-----|----------|-----------|-------------|
| Data scale: huge | | | | | |
| Micro | Sort | 3.28 | 70 | 47.11 | 131 | 25.08 | 111 | 29.42 |
| | Terasort | 32.00 | 667 | 47.99 | 858 | 37.28 | 1054 | 30.37 |
| | Wordcount | 32.85 | 978 | 33.60 | 1470 | 22.34 | 889 | 36.95 |
| | Dfsioe-r | 26.99 | 294 | 91.77 | 662 | 40.79 | 245 | 110.21 |
| | Dfsioe-w | 27.16 | 379 | 71.73 | 658 | 41.30 | 281 | 96.49 |
| SQL | Scan | 2.01 | 73 | 27.63 | 157 | 12.83 | 74 (*) | 27.19 (*) |
| | Join | 1.92 | 181 | 10.61 | 356 | 5.39 | 175 (*) | 10.95 (*) |
| | Aggregation | 0.37 | 97 | 3.86 | 215 | 1.73 | 97 (*) | 3.85 (*) |
| ML | Bayes | 1.88 | 2604 | 0.72 | 6120 | 37.28 | 3017 | 0.62 |
| | Kmeans | 20.08 | 2321 | 8.65 | 2313 | 8.68 | 2070 | 9.70 |
| Websearch | Pagerank | 2.99 | 1544 | 1.94 | 3334 | 0.90 | 2458 | 1.22 |
| Data scale: gigantic | | | | | |
| Micro | Sort | 32.85 | 715 | 45.94 | 787 | 41.72 | 896 | 36.68 |
| | Terasort | 320.00 | 9821 | 32.58 | 1886 | 114.54 | 660 | 327.29 |
| | Wordcount | 328.49 | 10,131 | 32.24 | 216.03 | 236.11 | 114.54 | 660 |
| | Dfsioe-r | 216.03 | 915 | 13,596 | 217.33 | 1347 | 114.54 | 660 |
| | Dfsioe-w | 217.33 | 1347 | 114.54 | 1914 | 113.57 | 1060 | 205.12 |
| SQL | Scan | 20.10 | 457 | 43.96 | 514 | 39.09 | 407 (*) | 49.38 (*) |
| | Join | 19.19 | 595 | 32.27 | 761 | 25.24 | 594 (*) | 32.32 (*) |
| | Aggregation | 3.69 | 523 | 7.05 | 594 | 6.20 | 565 (*) | 6.52 (*) |
| ML | Bayes | 3.77 | 5350 | 0.70 | 12,589 | 0.30 | 6363 | 0.60 |
| | Kmeans | 40.16 | 4541 | 8.84 | 4042 | 9.94 | 4034 | 9.96 |
| Websearch | Pagerank | 19.93 | 8371 | 2.38 | 11,779 | 1.70 | 13,893 | 1.43 |

**IDS input data size (GB); \(D(t)\): duration (s); \(T(MB/s)\): throughput (MB/s)

(*)Benchmark succeeds after modifying preconfiguration, more on this in Sect. 6

(**)Workload failed to run within three attempts, discussed in Sect. 5.2

### Table 4 Use Case 2 benchmark results

| Work | Scale | IDS | Dataproc | HDInsight | E-MapReduce |
|------|-------|-----|----------|-----------|-------------|
| Sort | Tiny | 39.30 KB | 36 | 0.0012 | 69 | 0.0006 | 32 | 0.0012 |
| | Small | 3.28 MB | 36 | 0.09 | 70 | 0.0471 | 31 | 0.105 |
| | Large | 328.50 MB | 42 | 7.86 | 81 | 4.07 | 42 | 7.74 |
| | Huge | 3.28 GB | 70 | 47.08 | 141 | 23.36 | 107 | 30.69 |
| | Gig. | 32.85 GB | 694 | 47.30 | 699 | 47.00 | 883 | 37.20 |
| Wrdcnt | Tiny | 38.65 KB | 38 | 0.0001 | 68 | 0.0006 | 31 | 0.0012 |
| | Small | 348.29 MB | 50 | 6.51 | 98 | 3.34 | 47 | 7.06 |
| | Large | 3.28 GB | 129 | 25.45 | 269 | 12.20 | 120 | 27.27 |
| | Huge | 32.85 GB | 952 | 34.51 | 1487 | 22.10 | 888 | 37.00 |
| | Gig. | 328.49 GB | 9749 | 33.70 | 13,286 | 24.73 | 8622 | 38.10 |

**IDS input data size; \(D(t)\): duration (s); \(T(MB/s)\): throughput (MB/s)**
Due to space limitations, a small number of visual results are included in this paper. For interested readers, entire collection of high resolution plots demonstrating all experiment results are provided in the documentation repository of the study.¹

5.1 Overview and some remarkable cases

When HiBench completes running a workload, information regarding the benchmark results is stored in a file called “hibench.report” on the local disk in the master node of the cluster. Hence, Tables 3 and 4 present information based

¹ https://bit.ly/31IcS7F.
Fig. 6 Use Case 2—sort performances along with data scales

Fig. 7 Use Case 2—wordcount performances along with data scales
on these reports. The former table is constructed in two parts for the Huge and Gigantic data scales. The columns in the tables represent the category name, the name of the benchmark, and the following for each provider’s service: Input Data Size (IDS), duration in seconds ($D(s)$), and throughput in Megabytes per second ($T(MB/s)$). The second table displays HiBench report values for Use Case 2.

Similarly, the Sort and Wordcount benchmark results are presented for the Tiny, Small, Large, Huge, and Gigantic data scales.

The results indicated that the durations for Dataproc and E-MapReduce appeared closer to each other in many cases, whereas HDInsight showed progress in its own processing through the data scales. The Gigantic data scale for Terasort did not produce any results for HDInsight, which will be discussed in the following section. As for E-MapReduce, the SQL workload in each data scale revealed that these workloads required some modification to run in the respective provider’s service. These tables provide basic information about the benchmark duration and throughput, which is helpful in understanding benchmark comparisons in the subsequent sections.

To develop basis for the resource utilization comparisons, Slowdown Estimates (SE) are calculated based on the raw benchmark results for the second and third performances as follows: $SE = \frac{\text{Duration}}{\text{Shortest duration}}$. Table 5 demonstrates the results on the Huge and Gigantic data scales, respectively. The column “Benchmark” refers to the workload; “First”, contains the holder of the shortest duration; “Second” represents the service with the second

| Category | Benchmark | First | Second | SE | Third | SE |
|----------|-----------|-------|--------|----|-------|----|
| Data scale: huge | | | | | | |
| Micro | Sort | GCP | Alibaba | 1.59 | Azure | 1.87 |
| | Terasort | GCP | Azure | 1.29 | Alibaba | 1.58 |
| | Wordcount | Alibaba | GCP | 1.10 | Azure | 1.65 |
| | Dfsioe-r | Alibaba | GCP | 1.20 | Azure | 2.70 |
| | Dfsioe-w | Alibaba | GCP | 1.35 | Azure | 2.34 |
| SQL | Scan | GCP | Alibaba | 1.01 | Azure | 2.15 |
| | Join | Alibaba | GCP | 1.03 | Azure | 2.03 |
| | Aggregation | GCP-Alibaba | Azure | 2.22 | – | – |
| ML | Bayes | GCP | Alibaba | 1.16 | Azure | 2.35 |
| | Kmeans | Alibaba | Azure-GCP | 1.12 | – | – |
| Websearch | Pagerank | GCP | Alibaba | 1.59 | Azure | 2.16 |
| Data scale: gigantic | | | | | | |
| Micro | Sort | GCP | Azure | 1.10 | Alibaba | 1.25 |
| | Terasort | Alibaba | GCP | 1.02 | Azure | (*) |
| | Wordcount | Alibaba | GCP | 1.17 | Azure | 1.57 |
| | Dfsioe-r | Alibaba | GCP | 1.39 | Azure | 2.86 |
| | Dfsioe-w | Alibaba | GCP | 1.27 | Azure | 1.81 |
| SQL | Scan | Alibaba | GCP | 1.12 | Azure | 1.26 |
| | Join | Alibaba-GCP | Azure | 1.28 | – | – |
| | Aggregation | GCP | Alibaba | 1.08 | Azure | 1.14 |
| ML | Bayes | GCP | Alibaba | 1.19 | Azure | 2.35 |
| | Kmeans | Alibaba-GCP | Azure | 1.13 | – | – |
| Websearch | Pagerank | GCP | Azure | 1.41 | Alibaba | 1.66 |

(*)Workload failed to run within three attempts, discussed in Sect. 5.2

| Work | Scale | First | Second | SE | Third | SE |
|------|-------|-------|--------|----|-------|----|
| Sort | Tiny | Alibaba | GCP | 1.13 | Azure | 2.16 |
| | Small | Alibaba | GCP | 1.16 | Azure | 2.26 |
| | Large | GCP-Alibaba | Azure | 1.93 | – | – |
| | Huge | GCP | Alibaba | 1.53 | Azure | 2.01 |
| | Gig. | GCP | Azure | 1.01 | Alibaba | 1.27 |
| Wrdcnt | Tiny | Alibaba | GCP | 1.23 | Azure | 2.19 |
| | Small | Alibaba | GCP | 1.06 | Azure | 2.09 |
| | Large | Alibaba | GCP | 1.08 | Azure | 2.24 |
| | Huge | Alibaba | GCP | 1.07 | Azure | 1.67 |
| | Gig. | Alibaba | GCP | 1.13 | Azure | 1.54 |
shortest duration, “Third” lists the service provider with the longest duration for the workload. The “SE” column displays the slowdown estimate of the provider listed to their left. In Table 6, the first two columns presents the Workload and the Data Scale, while the rest of the columns represent the same attributes as Table 5.

GCP Dataproc and Alibaba Cloud E-MapReduce services appeared to show similar utilization patterns in the linecharts in most cases. Likewise, the respective SE values also indicated only minor differences as shown in Table 5. As mentioned previously, GCP and Alibaba have comparable architectures. Therefore, it is not surprising that similar resource utilization behaviors were observed in both clusters.

On the other hand, Azure’s HDInsight needs to be evaluated from a different perspective. There are many instances where the SE values of HDInsight are notably higher. However, considering this as an inefficiency would lead us to a trap, because HDInsight’s performance improvements over data scales would stay unnoticed. The architecture of HDInsight differs from Dataproc and E-MapReduce in that it is a cloud-optimized version of the Hortonworks Data Platform (HDP).

In Table 5, we observed that HDInsight significantly improved its SE values over the increasing data scales. For instance, in the SQL category, the SE values improved from a range between 2.03 and 2.22 on the Huge data scale, to a range between 1.14 and 1.28 on the Gigantic data scale. For the Kmeans workload in the ML category, the SE value of HDInsight reached to approximately 1.12 on the Huge data scale, and a negligibly small SE value on the Gigantic data scale. Additionally, its SE value for the Pagerank workload dropped from 2.16 on the Huge data scale to 1.41 on the Gigantic data scale.

The performance of Hadoop is affected by the mapper and reducer slot allocation capacity specified by the pre-

### Table 7 Allocated map and reduce slots from use Case 1

| Work   | Scale | GCP Maps | GCP Reduces | Azure Maps | Azure Reduces | Alibaba Maps | Alibaba Reduces |
|--------|-------|----------|-------------|------------|--------------|--------------|---------------|
| Terasort | Huge  | 240      | 13          | 60         | 12           | 240          | 13            |
|        | Gigantic | 2388     | 13          | 600*       | X*           | 2388         | 13            |
| Scan** | Huge  | 24       | –           | 12         | –            | 24           | –             |
|        | Gigantic | 144      | –           | 36         | –            | 144          | –             |
| Join   | Huge  | 60       | 25          | 48         | 25           | 60           | 25            |
|        | Gigantic | 180     | 25          | 72         | 25           | 180          | 25            |
| Aggreg. | Huge  | 24       | 12          | 12         | 12           | 24           | 12            |
|        | Gigantic | 144    | 12          | 36         | 12           | 144          | 12            |
| Bayes  | Huge  | 13       | 12          | 12         | 1            | 13           | 7             |
|        | Gigantic | 13    | 12          | 12         | 1            | 13           | 8             |
| Kmeans | Huge  | 151      | 11          | 40         | 1            | 151          | 7             |
|        | Gigantic | 300    | 11          | 75         | 1            | 300          | 7             |

*Failed during execution
**Has no reduce operation

### Table 8 Allocated map and reduce slots in use Case 2

| Work   | Scale | GCP Maps | GCP Reduces | Azure Maps | Azure Reduces | Alibaba Maps | Alibaba Reduces |
|--------|-------|----------|-------------|------------|--------------|--------------|---------------|
| Sort   | Tiny  | 11       | 12          | 12         | 12           | 11           | 12            |
|        | Small | 11       | 12          | 12         | 12           | 11           | 12            |
|        | Large | 12       | 11          | 12         | 12           | 11           | 11            |
|        | Huge  | 24       | 12          | 12         | 12           | 23           | 12            |
|        | Gigantic | 251 | 12          | 60         | 12           | 252          | 12            |
| Wrdcnt | Tiny  | 11       | 12          | 12         | 12           | 11           | 12            |
|        | Small | 12       | 11          | 12         | 12           | 12           | 11            |
|        | Large | 24       | 11          | 12         | 12           | 23           | 12            |
|        | Huge  | 250      | 11          | 60         | 12           | 252          | 12            |
|        | Gigantic | 2447 | 12          | 612        | 12           | 2447         | 11            |
configuration of the managed system. Allocated mapper and reducer slots for various benchmarks are presented in Tables 7 and 8. We collected the values from the job counts within the MapReduce execution logs stored in the HiBench reports. The allocation tables support our assumption that similar architectures (Dataproc and E-MapReduce) produce a similar number of mapper slots during map phases.

In Fig. 5, the spider plots depict the main utilization behaviour for Use Case 1 Gigantic data scale, whether it is I/O-bound as in Sort, or CPU-bound as in Wordcount, or both. In most cases, the systems that can utilize more resources displayed shorter durations. Due to the failed workload execution in Terasort Gigantic data scale, Azure HDInsight’s duration and throughput results could not be provided in this figure.

The average cluster utilization plots for Sort benchmark in Fig. 6 indicate that HDInsight cluster is efficiently utilized when the data grows to Gigantic scale size. In the Wordcount benchmark (Fig. 7), the I/O averages determined throughput. Therefore, E-MapReduce with high CPU utilization among the clusters resulted in better duration.

5.1.1 Terasort

In the Terasort task with Huge data scale, GCP showed high processor utilization in the range between 80 and 100%. The memory consumption shifted between 80 and 100%, and I/O behavior displayed 500 transfers per second (tps) in overall and peaked at 1000 tps in I/O-read. This execution provided the highest throughput and 667 s of duration as shown in Fig. 8. Azure’s processor and memory utilization fluctuated in overall execution where memory performance incrementally reached 100% on a node. A stable I/O-write generally hovered around 70 tps and peaked at 250 tps in throughput, resulting in a duration of 858 s. With high memory and processor utilization and varying I/O tps in the overall process, Alibaba’s throughput performed 1054 s in duration.

When evaluating the Gigantic data scale, dramatic changes in resource utilization were observed, as displayed in Fig. 9. GCP demonstrated very heavy processor utilization over a wide range of 30–95%, memory consumption varying between 95 and 100%, and I/O read operations ranging from 800 to 1100 tps.

On the other hand, the benchmark execution failed for Azure. This can be seen in the graph where the activity lines are flattening, the I/O operations dropping to zero at one point. This has happened in all three attempts where we tried to run the benchmark. This was due to insufficient allocation on the local disk which required storage of intermediate data generated by map processes. HiBench couldn’t generate the corresponding benchmark results because the workload was incomplete. Consequently, the plot does not include the results for Azure. More details on this situation will be given in Sect. 5.2.

The resource utilization of Alibaba Cloud reached maximum levels where processor utilization was in a range between 80 and 90%. Memory utilization during execution was very high, the I/O reads and writes performed around 600 tps and peaked at 1600 tps. Alibaba achieved a slightly higher throughput at 9660 s, followed by GCP with 9821 s.

5.1.2 Bayes

At the Huge data scale, GCP performed around 70–100% in processor utilization. The memory consumption was about 30% with a short peak to 100%. I/O-write utilization was approximately 500 transfers per second. With these values, GCP achieved the highest throughput, hence the fastest response time with 2604 s as illustrated in Fig. 10.

Alibaba’s CPU utilization varied between worker nodes based on the load ranges. Memory utilization was heavy with a high level around 80%. The I/O transfers fluctuated from 100 to 450 tps, providing a response time of 3017 s.

When exploring Azure’s utilization plot, we observed an unusual event, an anomaly. Throughout the study, the lines in all plots presenting utilization across three worker nodes of a cluster were somewhat congruent, regardless of the provider. The lines for all three nodes were moving up or down in a more or less synchronized way.

However, a mismatch was evident here. For the processor utilization lines colored in blue, a very low utilization line (bottom of the plot) was representing a node, while for another node a higher utilization was observed. Also, the third node, represented by the line colored in light blue, was highly utilized at the beginning, but bottomed out after about 20 min. Similarly, memory consumption among worker nodes was also displaying incompatibilities with each other, as if the memory of each worker node was not utilized for the same task.

Disk utilization showed very low values compared to other services, around 100 tps. While the benchmark on Azure was completed without any errors, the results bar for Azure was outside the generally acceptable limits among services from the providers equipped with very similar features, as seen in the results plot at the bottom right. We think that this is potentially due to a structural issue that requires further investigation.

Figure 11 displays the comparative results for the cloud providers in Bayes task. In the results from the Gigantic data scale, we observed GCP’s processor utilization in the range of about 70–100%. The memory load increased from 40 to 70% and stabilized at 50%, and I/O-write transfers
Fig. 8 Use Case 1—Terasort (huge; 320 MB)
Fig. 9  Use Case 1—Terasort (Gigantic: 3.2 GB)
Fig. 10 Use Case 1—Bayes (image: pages: 500,000 classes: 0. Ng: 2)
Fig. 11 Use Case 1—Bayes (Gigantic; pages: 1,000,000 classes: 100 Ng: 2)
were close to 500 tps. This resulted in the highest throughput and produced a response time of 5350 s.

Alibaba provided varying utilization ranges among worker nodes. The CPU load exhibited a similar pattern to the Alibaba’s utilization in Huge data scale for the same benchmark, only more intense. The memory was slightly less loaded than the GCP’s in the same data scale. The I/O-write transfer for the worker nodes shifted around 200 tps, a relatively lower throughput. When compared to GCP, a relatively longer duration of 6363 s was performed.

Azure was the most awaited benchmark in the Huge data scale due to its utilization issues. Unfortunately, its utilization results for the Gigantic data scale of Bayes workload did not deviate much from those for the Huge data scale. These were the inconsistency in utilization between worker nodes, abnormal movements of representative lines, low disk utilization compared to values from other providers and a benchmark result that lagged far behind the others. This made no sense as all providers had almost identical features. At this point, it became clear that more research was necessary to determine the cause, which we examine in the next section.

5.1.3 Kmeans

GCP’s CPU utilization was observed in the range between 60 and 100%. The memory utilization was around 60% and increased towards the end at the reduce phase where I/O-write operations also raised to 500 transactions per second.

The utilization in Azure displayed a high CPU utilization between 80 and 90%, with a moderate memory consumption attained from 20 to over 50%. The I/O transfers were approximately 40 tps, which turned out to be the second shortest duration.

Alibaba utilized high levels of CPU and memory, including drops in processors on particular worker nodes. With an increased I/O-write transfer in the reduce phase and it reached the highest throughput and the shortest duration as illustrated in Fig. 12.

In the Gigantic data scale, all three services have retained their previous utilization patterns in a more condensed way as seen in Fig. 13. The performances of Alibaba and Azure were comparable in throughput. The result was 4034 s for Alibaba and 4042 s for Azure. GCP’s duration did not stand far apart at 4541 s.

5.2 Cluster utilization issues and bottlenecks

This section covers the results of detailed research on the issues and conditions encountered in the Terasort, Bayes and Kmeans benchmarks of Use Case 1 as mentioned in the previous section.

When running the Terasort benchmark in Gigantic data scale for HDInsight, the execution was terminated when approximately 20% of the map process was reached. This happened over the course of the three trials we did. Considering HDInsight’s competitive performance at the Huge data scale for Terasort, the unexpected termination in the Gigantic data scale appeared to be something structural, and required deeper inspection.

Before continuing, it is worth remembering that end-users are allowed to choose between HDFS or the cloud storage service of the respective provider as storage option on both Dataproc and E-MapReduce. The option to specify the size of the local disk for the worker nodes is also given.

However, end-users are only allowed to use WASB blob storage in HDInsight. At the time of the study, there was no option to specify a local disk size for worker nodes. When delving into the HiBench logs, we noticed that a DiskErrorException occurred during the shuffle phase. There was no sufficient space on the local disk of the worker node to write intermediate map results before being passed to the reducer.

This made sense, because once the map phase was 20% complete, the reducers would have barely started. We marked this shortcoming as a structural bottleneck since all end users running the Terasort process at this scale would encounter the same error. From Fig. 9, it is clearly visible where the error occurred for HDInsight and became a straight line. Duration and throughput results for HDInsight could not be provided in graphs due to incomplete execution. We made the error logs available for the interested readers.

The duration—throughput bar charts in Figs. 10 and 11 shows HDInsight’s throughput bar at half the size of other services in the Bayes workload. Accordingly, its duration bar is twice as long. While investigating the MapReduce execution logs, we observed that a Java Heap space error was causing the process to restart nine times on the Gigantic data scale. This issue occurred at about 83% completion rate for each map process. This prolonged the overall duration of the benchmark.

Considering that the mapper’s Java Virtual Machine (JVM) heap size is a property (mapreduce.map.java.opts) that could be set within the configurations, we see this as a fixable setting that needs further improvement in managed Hadoop’s default configurations. Raw MapReduce execution logs for Bayes workload are also publicly available.

Another case that seemed contradictory was HDInsight with Bayes and Kmeans. Both workloads belong to the ML category, and utilize the same Mahout engine. Therefore, it was surprising to see that the Bayes task performed poorly

---

2 https://bit.ly/3yoQFHS.
3 https://bit.ly/3m2ntA.
Fig. 12 Use Case 1—Kmeans (huge; clusters: 5 dimensions; 20 samples; 100,000,000 Samp Per Input: 20,000,000 Max It: 5 K: 10 Convergedist: 0.5)
Fig. 13  UCI—Kmeans (Gigantic; clusters: 5 dimensions; 20 samples; 200,000,000 Samp Per Input: 40,000,000 Max It: 5 K: 10 Convergedist: 0.5)
while the Kmeans evaluation achieved good performance, as shown in Figs. 12 and 13.

One reason might be the size of slots allocated for map and reduce operations on each service. Table 7 displays the mapper/reducer slot numbers allocated by the respective provider during the benchmark. These numbers are calculated by pre-configured settings of the providers prior to the benchmark execution.

We detected that HDInsight allocated 12/1 mapper/reducer slots in Bayes task, whereas these numbers were 13/12 and 13/8 in GCP and Alibaba, respectively. This means that GCP and Alibaba were able to pass their intermediate data of 13 map slots to the given 12 and 8 reduce slots. Considering that the error shown below re-occurred nine times in the execution logs and the reducer was eventually killed, it is likely that HDInsight required more local disk space to fill the intermediate results of 12 mappers to just 1 reducer slot.

### 6 Discussion

In the study, we investigated the utilization behavior of the managed Hadoop services by benchmarking the experimental environments of different providers. For a fair assessment, we activated the services at the same geographical data center locations with the same computation specifications. Minor differences occurred in Hadoop versions, and no configuration tweaks were applied to the clusters.

A common approach to benchmarking is to run a specific workload multiple (usually three) times and take the average result as the final value. This is crucial for performance-focused or commercially driven benchmarks. We strayed from this approach with one execution per benchmark. The rationale is drawn by our interest in being an observer of cluster resource utilization at a non-performative boundary. Rather than averaging across multiple runs, we collected utilization data for each worker node during runtime. This allowed us to inspect cluster utilization over a time span.

Performance results or slowdown estimates were often visited as secondary indicators to observe the impact of the respective utilization behaviors. The marginal benefit of timely-based average utilization data would be less important for the objective of the study. Our main concern was not to compare the performance of experimental environments, but to understand the internal dynamics of managed Hadoop services and help guide informed decisions when choosing the service providers.

HiBench comes with dependencies that must be downloaded when compiled by Apache Maven. The Hive engine is one of those dependencies used by HiBench to run SQL workloads of Scan, Join, and Aggregation. Alibaba Cloud E-MapReduce comprised a ready-made Hive hook that triggers a Java file to run post-executional transactions for other services within the package.

This configuration prevented HiBench from starting SQL benchmarks, as the HiBench-based Hive engine did not include the aforementioned jar file defined for E-MapReduce’s specific environment. A workaround attempt to copy the related jar file to an appropriate directory within HiBench made the jar file usable. However, this time, the Hive engine of an older version of HiBench did not support the hook “hive.exec.post.hooks” defined in Alibaba’s Hive configuration.

In this situation, disabling the corresponding Hive hook from Alibaba Cloud E-MapReduce’s UI management console apparently fixed this issue and enabled HiBench’s SQL workloads to run. Its impact on the respective system utilization remains unknown to us, so it needs to be clarified here. Such problems did not occur with GCP and Azure.

Managed Hadoop systems provide a comfort zone through automated implementation processes. However, this may manifest itself as a deficiency on the resource utilization dynamics of the respective system. Our study aimed to help end-users better understand the architectures of the managed Hadoop systems. Yet it is necessary to investigate the architecture of the respective managed system and implement performance enhancing processes before using them in production environments.

### 7 Conclusion and future work

In the study, we conducted HiBench benchmarks against Hadoop PaaS clusters in their recommended, ready-to-use form provided by respective CSPs. The purpose of the study was to inspect resource utilization in the context of managed Hadoop systems by using HiBench Benchmark Suite. We executed Hadoop related benchmarks across three managed services and captured system resource utilization data on worker nodes.
Based on the benchmark results, we presented slowdown estimates to exhibit performance differences from one another. For the selected benchmarks, we collected the number of the allocated map and reduce slots relative to the vendors’ managed Hadoop systems. We found out that there is a strong correlation between sufficiently allocated map slots and cluster utilization, consequently their performances. It appears that the default pre-configuration settings do not allow sufficient resources to be utilized and affect the performance.

The results indicated that managed Hadoop services do not necessarily use system resources in similar ways, even if they are defined by the same properties. It depends more on managed Hadoop architecture. We noticed that similar architectures follow a resembling pattern in utilization over time, but different architectural approaches deviate in performance and utilization behavior. Future work will focus on exploring different managed platforms such as Apache Spark and expanding the scope with additional cloud vendors.

Author contributions UEO: conceptualization, data collection, development of methodology, programming, writing-draft preparation, writing—review & editing. SA: conceptualization, development of methodology, supervision, writing—review & editing

Funding The authors did not receive support from any organization for the submitted work.

Data availability All data generated or analysed during this study are included in this published article: “Huang et al. [18]” The files and codes used in the evaluations along with detailed documentation for the experimental environment setup are made available to interested researchers in our GitHub repository entire results: https://github.com/emretto/benchmark-hadoop-on-paas.

Declarations

Conflict of interest Author Serkan Ayvaz and Author Uluer Emre Ozdil declare that they have no conflict of interest.

Ethical approval The authors consciously assure that this material is the authors’ own original work, which is not currently being considered for publication elsewhere. This article does not contain any studies with human participants or animals performed by any of the authors.

Informed consent This article does not contain any studies with human participants or animals performed by any of the authors. The consent is not a requirement for this study.

References

1. Apache Hadoop. https://hadoop.apache.org/. Accessed 22 May 2022
2. Announcing Amazon Elastic Compute Cloud (Amazon EC2)—beta. https://aws.amazon.com/about-aws/whats-new/2006/08/24/announcing-amazon-elastic-compute-cloud-amazon-ec2-beta/. Accessed 22 May 2022
3. TPC-History. http://tpc.org/information/about/history5.asp. Accessed 22 May 2022
4. SPEC—Standard Performance Evaluation Corporation. https://www.spec.org/. Accessed 22 May 2022
5. Han, R., John, L.K., Zhan, J.: Benchmarking Big Data systems: a review. IEEE Trans. Serv. Comput. 11, 580–597 (2018). https://doi.org/10.1109/TSC.2017.2730882
6. Ghemawat, S., Gobioff, H., Leung, S.-T.: The Google file system. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, pp. 29–43 (2003). https://doi.org/10.1145/1165389.945450
7. White, T.: Hadoop: The Definitive Guide. O’Reilly, Beijing (2015)
8. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Presented at the OSDI 2004—6th Symposium on Operating Systems Design and Implementation (2004)
9. Schätzle, T.H., Przyjaciel-Zablocki, M., Alexander: Giant Data: MapReduce and Hadoop. ADMIN Magazine. http://www.adminmagazine.com/HPC/Articles/MapReduce-and-Hadoop/. Accessed 30 Oct 2020
10. Ramel, B.D.: 08/04/2021: what are Gartner’s “Cautions” about big 3 cloud providers? https://virtualizationreview.com/articles/2021/08/04/gartner-cloud-2021.aspx. Accessed 15 Apr 2021
11. Azure HDInsight—Hadoop, Spark, & Kafka Service—Microsoft Azure. https://azure.microsoft.com/en-us/services/hdinsight/. Accessed 8 Jan 2021
12. Announcing general availability of Azure HDInsight 3.6. https://azure.microsoft.com/en-us/blog/announcing-general-availability-of-azure-hdinsight-3-6/. Accessed 14 Jan 2021
13. Dataproc. https://cloud.google.com/dataproc. Accessed 8 Jan 2021
14. Compute Engine: Virtual Machines (VMs). https://cloud.google.com/compute. Accessed 8 Jan 2021
15. What is E-MapReduce?—Product Introduction—Alibaba Cloud Documentation Center. https://www.alibabacloud.com/help/doc-detail/28068.htm?spm=a2c63.l28256.b99.4.65e270b2YXyKDV. Accessed 14 Jan 2021
16. Elastic Compute Service (ECS): Elastic & Secure Cloud Servers—Alibaba Cloud. https://www.alibabacloud.com/product/ecs. Accessed 17 Jan 2021
17. Alibaba Cloud Linux OS. https://alibaba.github.io/cloud-kernel/os.html. Accessed 14 Jan 2021
18. Huang, S., Huang, J., Dai, J., Xie, T., Huang, B.: The HiBench benchmark suite: characterization of the MapReduce-based data analysis. In: 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010), pp. 41–51 (2010). https://doi.org/10.1109/ICDEW.2010.5452747
19. GitHub—Intel-bigdata/HiBench. HiBench is a big data benchmark suite. https://github.com/Intel-bigdata/HiBench. Accessed 8 Jan 2021
20. Guo, Q., Xie, Y., Li, Q., Zhu, Y.: XDataExplorer: a three-stage comprehensive self-tuning tool for Big Data platforms. Big Data Res. 29, 100329 (2022). https://doi.org/10.1016/j.bdr.2022.100329
21. Sfaxi, L., Aissa, M.M.B.: Babel: a generic benchmarking platform for Big Data architectures. Big Data Res. 24, 100186 (2021)
22. Prieto, P., Abad, P., Gregorio, J.A., Puente, V.: Fast, accurate processor evaluation through heterogeneous, sample-based benchmarking. IEEE Trans. Parallel Distrib. Syst. 32(12), 2983–2995 (2021)
23. Ghazali, R., Adabi, S., Down, D.G., Movaghar, A.: A classification of Hadoop job schedulers based on performance optimization approaches. Clust. Comput. 24(4), 3381–3403 (2021)
24. Ghafari, R., Kabutarkhani, F.H., Mansouri, N.: Task scheduling algorithms for energy optimization in cloud environment: a comprehensive review. Clust. Comput. 25, 1035–1093 (2022). https://doi.org/10.1007/s10586-021-03512-z
25. Cheng, D., Wang, Y., Dai, D.: Dynamic resource provisioning for iterative workloads on Apache Spark. IEEE Trans. Cloud Comput. (2021). https://doi.org/10.1109/TCC.2021.3108043
26. Li, C., Cai, Q., Luo, Y.: Dynamic data replacement and adaptive scheduling policies in spark. Clust. Comput. 25(2), 1421–1439 (2022). https://doi.org/10.1007/s10586-022-03541-2
27. Costa, R.L.D.C., Moreira, J., Pintor, P., dos Santos, V., Lifschitz, S.: A survey on data-driven performance tuning for big data analytics platforms. Big Data Res. 25, 100206 (2021)
28. Poggi, N., Montero, A., Carrera, D.: Characterizing BigBench queries, hive, and spark in multi-cloud environments. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 10661 LNCS, pp. 55–74 (2018). https://doi.org/10.1007/978-3-319-72401-0_5
29. Wang, H., Shen, H., Reiss, C., Jain, A., Zhang, Y.: Improved intermediate data management for MapReduce frameworks. Presented at the Proceedings—2020 IEEE 34th International Parallel and Distributed Processing Symposium, IPDPS 2020 (2020). https://doi.org/10.1109/IPDPS47924.2020.00062
30. Hwang, K., Bai, X., Shi, Y., Li, M., Chen, W.-G., Wu, Y.: Cloud performance modeling with benchmark evaluation of elastic scaling strategies. IEEE Trans. Parallel Distrib. Syst. 27, 130–143 (2016). https://doi.org/10.1109/TPDS.2015.2398438
31. Ahn, H., Kim, H., You, W.: Performance study of spark on YARN cluster using HiBench. Presented at the 2018 IEEE International Conference on Consumer Electronics—Asia, ICCE-Asia 2018 (2018). https://doi.org/10.1109/ICCE-ASIA.2018.8552137
32. Han, S., Choi, W., Muwaqif, R., Nah, Y.: Impact of memory size on big data processing based on Hadoop and Spark. Presented at the Proceedings of the 2017 Research in Adaptive and Convergent Systems, RACS 2017 (2017). https://doi.org/10.1145/3129676.3129688
33. Samadi, Y., Zbakh, M., Tadonki, C.: Performance comparison between Hadoop and spark frameworks using HiBench benchmark. Concurr. Comput. (2018). https://doi.org/10.1002/cpe.4367
34. Ahmed, N., Barczak, A.L., Rashid, M.A., Susnjak, T.: A parallelization model for performance characterization of Spark Big Data jobs on Hadoop clusters. J. Big Data 8(1), 1–28 (2021)
35. Shih, W.C., Yang, C.T., Ranjan, R., Chiang, C.I: Implementation and evaluation of a container management platform on Docker: Hadoop deployment as an example. Clust. Comput. 24(4), 3421–3430 (2021). https://doi.org/10.1007/s10586-021-03337-w
36. GitHub Repository of the study. https://github.com/emretto/benchmark-hadoop-on-paas. Accessed 24 May 2022
37. Jota juliojsb/sarviewer. https://github.com/juliojsb/sarviewer. Accessed 12 Dec 2020

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Uluer Emre Özdíl is a self taught programmer with specialization in data driven web development and SQL authoring. In 2018, his curiosity in big data concepts and technologies led him to participate at and earn Master’s degree from Bahçeşehir University’s “Big Data Analytics and Management Program”. His main interest lies in cloud based distributed computation paradigms. Currently, he is working as a data engineer at Capgemini Germany’s Insights & Data division in Cologne.

Serkan Ayvaz is an associate professor at the Department of Computer Engineering at Yıldız Technical University. Serkan Ayvaz received his Bachelor’s degree in Mathematics and Computer Sciences from Bahçeşehir University in 2006. He received his Master’s degree in Technology with specialization in Computer Technology from Kent State University in 2008. He completed his Ph.D. in Computer Science at Kent State University in 2015. His current research interests include big data analytics, parallel programming, cloud computing, and deep learning.