Characterization of a Big Data Storage Workload in the Cloud

Sacheendra Talluri*
Delft University of Technology
Delft, the Netherlands
s.talluri@atlarge-research.com

Alicja Łuszczak
Databricks B.V.
Amsterdam, the Netherlands
ala@databricks.com

Cristina L. Abad
Escuela Superior Politecnica del Litoral
Guayaquil, Ecuador
cabad@fiec.espol.edu.ec

Alexandru Iosup
Vrije Universiteit
Amsterdam, the Netherlands
a.iosup@vu.nl

ABSTRACT

The proliferation of big data processing platforms has led to radically different system designs, such as MapReduce and the newer Spark. Understanding the workloads of such systems facilitates tuning and could foster new designs. However, whereas MapReduce workloads have been characterized extensively, relatively little public knowledge exists about the characteristics of Spark workloads in representative environments. To address this problem, in this work we collect and analyze a 6-month Spark workload from a major provider of big data processing services, Databricks. Our analysis focuses on a number of key features, such as the long-term trends of reads and modifications, the statistical properties of reads, and the popularity of clusters and of file formats. Overall, we present numerous findings that could form the basis of new systems studies and designs. Our quantitative evidence and its analysis suggest the existence of daily and weekly load imbalances, of heavy-tailed and bursty behaviour, of the relative rarity of modifications, and of proliferation of big data specific formats.

CCS CONCEPTS

• **General and reference → Measurement**; • **Information systems → Cloud based storage**; *Data management systems*;

ACM Reference Format:
Sacheendra Talluri, Alicja Łuszczak, Cristina L. Abad, and Alexandru Iosup. 2019. Characterization of a Big Data Storage Workload in the Cloud. In *Tenth ACM/SPEC International Conference on Performance Engineering (ICPE '19), April 7–11, 2019, Mumbai, India.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3297663.3310302

1 INTRODUCTION

Big data is at the core of many different applications relevant to our society, for example, in healthcare [19] [22], finance [24], and gaming [7]. To address more diverse and sophisticated uses of big data, system designers are creating radically different systems designs. For example, Spark [27] emerged at the beginning of the 2010s, as a response to changes in the needs previously addressed by MapReduce [8] and related systems in the 2000s. Understanding the workload of Spark-based ecosystems is important not only for computer systems research, but also for ecosystem tuning, for exploring new system designs [26], for tuning techniques [12], and for taking business operations decisions [14]. However, relevant Spark workloads are difficult to find, and so far relatively little is known about them outside the companies deploying such ecosystems. To address this problem, in this work we collect and characterize a detailed trace of Spark running in a prominent deployment, at Databricks.

The difference between designs is significant, even radical. For example, Spark addresses the current need of many users to run their big data workloads on-demand, by building an ecosystem that runs commonly in small clusters of virtual machines (VMs), provisioned from clouds, and attached to remote object storage systems. In contrast, the MapReduce and GFS [10] ecosystem champions statically deployed [11], tightly coupled physical clusters, with integrated storage.

The radical change in system designs is not arbitrary. Until the mid-2000s, large organizations alone could afford to operate large compute clusters containing tens of thousands of computers, shared between the multiple organizational units; small- and medium-scale organizations could not access easily such resources. Through the advent of cloud computing, individual organizational units in both large and small organizations can lease resources on-demand from a cloud provider. Cloud computing has reduced the barrier to computation by enabling many to access significant compute resources for only a short period of time, at accessible cost. After overcoming initial performance-related [16] and technical challenges, cloud computing resources are now used in many fields, including for big data processing [15].

For big data processing, the use of clouds introduces many new parts, in contrast to traditional data processing. A key difference resulting from the transition between self-hosting and cloud computing infrastructure is the architecture used for *persistent storage*, where data gets stored and from which it is retrieved. In the cloud, a common storage medium for large amounts of data is the *object store* provided by the cloud vendor. Examples of storage sub-systems operating as object stores and available in the cloud include Amazon

*Much of this work was done while the author was interning at Databricks.

ICPE '19, April 7–11, 2019, Mumbai, India

ACM ISBN 978-1-4503-6239-9/19/04…$15.00
https://doi.org/10.1145/3297663.3310302

Web Services (AWS) S3, the Microsoft Azure Blob storage, and the Google Cloud Storage.

As revealed by our systematic survey [18] of related work (details in Section 9), no characterization of Spark-based storage workloads currently exists. This hampers optimization and innovation related to Spark-based operations in the cloud, because workload characterization is a key tool in improving systems and their design, and later in benchmarking them for comparison and business decisions. Characterization studies can sometimes be replaced by the presence of representative workload traces. Although many Spark deployments exist, as indicated by the activity of the Apache Spark repository, and by the large audience and the technical talks at the Spark Summit and many scientific venues, workload traces and other operational data from relevant Spark deployments have so far been kept private.

Motivated by the importance of Spark-based systems processing big data in the cloud, by the novelty introduced by their cloud storage sub-systems, and by the scarcity of information publicly available about Spark storage workloads, we endeavor to answer the following research question: *What are the characteristics of big data storage workloads in the cloud?* Toward answering this question, our contribution is four-fold:

(1) We collect and process long-term workload traces from a relevant Spark deployment, at Databricks (Section 3). Our data spans over 6 months of operation, resulting in over 600 TB of log data. We devise a method for pre-processing and for the statistical analysis of these traces.
(2) We analyze the long-term trends (Section 4). We focus on two key I/O operations, reads and modifications. We investigate if diurnal and weekly patterns occur, if long-term patterns emerge, and if reads and modifications occur with relatively similar frequency.
(3) We analyze statistically the read operations (Section 5). (We focus on reads because we find that modifications are relatively rare.) We study if heavy-tails and burstiness appear in the distributions of the number and size of reads.
(4) We analyze statistically the popularity of clusters and of file types (Sections 6 and 7). We investigate if the clusters deployed on-demand are similarly used, and if the big data file formats and compression schemes are similarly popular.

As further conceptual contribution, we analyze several threats to the validity of this study (Section 8), and conduct a systematic survey [18] to compare our study with related work (Section 9).

This work aligns with our long-term vision, of Massivizing Computer Systems [17]. It aligns with the principle of creating a science of computer systems (principle P8 of the vision) by providing not only new knowledge, but reaffirming existing results about the behavior of storage workloads. It also aligns with the principle of increased awareness about the emerging properties of ecosystems (P9), by informing ourselves and the community about this workload. We specifically tackle in this work the challenge of understanding this emerging workload (challenge C19, "understanding the New World").

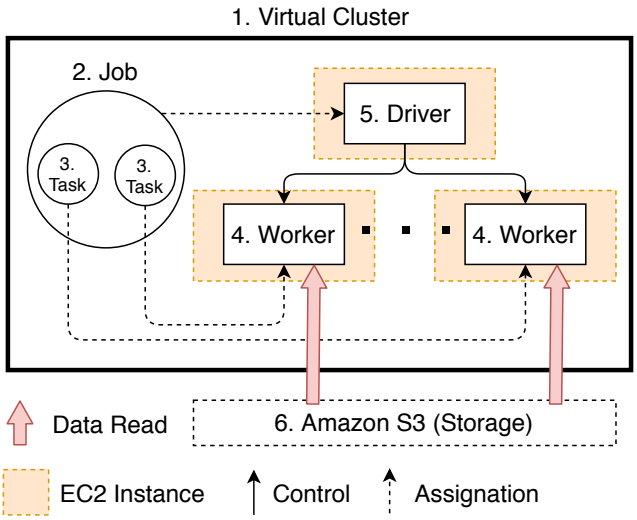


Figure 1: System architecture of a virtual cluster running on AWS. The workload analyzed in this paper comes from virtual clusters operated by Databricks.

2 SYSTEM MODEL

We introduce in this section a system model for the operation of (Spark-based) big data workloads in the cloud. This model is inspired by deployments we have observed in practice across many organizations, in particular, in big data operations at Databricks. Figure 1 depicts the system model. We focus on the workload of requests issued to the storage layer, between the system workers (component 4 in the figure) and the system storage (6).

2.1 Workload Model

In our model, the workload consists of jobs arriving in the system as a *stream*. Jobs are either interactive and non-interactive. For interactive jobs, the arrival time is decided by the data analyst. For non-interactive jobs, e.g., batch or periodic, the system itself schedules when the job should run.

Each *job* (component 2 in Figure 1) is a unit of work that reads some input data from persistent storage, processes it, and produces an output. The output can be stored in persistent storage or directly displayed to the user.

A job is composed of at least two *tasks* (3), data generation (*read*) and data processing; other tasks, including tasks that produce data (*modification*), may also appear. Jobs have a directed acyclic graph (DAG) structure, with nodes representing tasks, and directed relationships between nodes representing dependencies (Ⓐ→Ⓑ means that task B cannot start before task A completes).

2.2 System Architecture

We model the class of system analyzed in this work after a common architecture used by organizations around the world when using Spark or Hadoop in the cloud. This architecture is comprised of one or more **virtual clusters** running as sets of virtual machines (VMs) leased from a cloud provider, such as Amazon Web Services, and organized into a logical group using a virtual network. Virtual clusters get deployed on-demand—when needed, for as long as

needed. Further, each virtual cluster is comprised of a set of core components, running on VMs obtained on-demand.

Figure 1 depicts the architecture of *one* virtual cluster. The virtual cluster corresponds to a single deployment of the data processing software, for example, the Databricks Runtime (components 4 and 5, explained later in this section). The virtual cluster is connected at runtime to a source of incoming data, from which it reads. In our model, a typical source of data is a cloud-based, persistent, *object-store* (component 6), for example, Amazon S3. If the processing results in data modifications, the virtual cluster is further connected at runtime to a data sink, for example, also Amazon S3.

A typical instance of this model appears at Databricks, whose Runtime system extends Apache Spark [27] with techniques to achieve high performance and usability. With the Databricks Runtime, VMs are occupied by one or several Spark *workers* (component 4), and a *driver*. The single driver schedules jobs onto each worker. Workers process tasks, by performing the computation and I/O related to the task. This involves reading data from a persistent source or another worker, running computations such as maps and filters, and storing the data locally or to persistent storage.

2.3 Storage Workload Model

In our model, workers read data from the object store. All the requests from all jobs to the object store form the *workload* under study.

The object store is eventually consistent and has a key-value interface. It supports a hierarchy of files like a traditional file system by means of having a path as a key to an object. A directory structure can be achieved by having all the files in the directory have the directory name followed by a '/' as a prefix. A file can have several prefixes, simulating a hierarchy of directories. The files stored in an object store are much larger than the size of the key. It is possible to read only part of a file specifying a range of bytes to read. AWS S3 and Azure Blob Storage are examples of object stores. S3 is the store accessed in this workload.

Files are stored in several popular formats. Parquet, JSON, CSV and Avro are some examples. Some file types such as parquet make use of metadata to store column statistics and other index information. The Databricks runtime takes advantage of the S3 ability to read specific byte ranges to read this metadata for efficient filtering and index traversal.

Files are compressed using popular compression schemes such as Snappy or Gzip. Some files, those formatted in CSV and such formats, are compressed as a whole. Other file formats, such as Parquet, have a higher granularity than whole file. For example, data in parquet is stored in row-groups in a file. In these cases, compression is applied at the row-group level. This makes row-groups self contained and readable without reading the whole file.

3 PROCESS FOR DATA COLLECTION, PROCESSING, AND ANALYSIS

The goal of our analysis is to gain a statistical understanding of the series of accesses to the object store by various Spark-based applications deployed in the cloud. The operational data corresponding to Spark-based storage workloads presents many collection, processing, and analysis challenges. Monitoring many small virtual clusters across organizational boundaries is challenging, which raises a complex data collection problem. Pre-processing the data needs to balance preserving meaningful information, with the need to protect the anonymity and corporate information. Analyzing the rare information raises the same challenge facing basic research—finding a balance between the breadth and depth of explored concepts. Addressing these issues, in this section we introduce a process to collect, process, and analyze storage-workload data from cloud-based Spark operations.

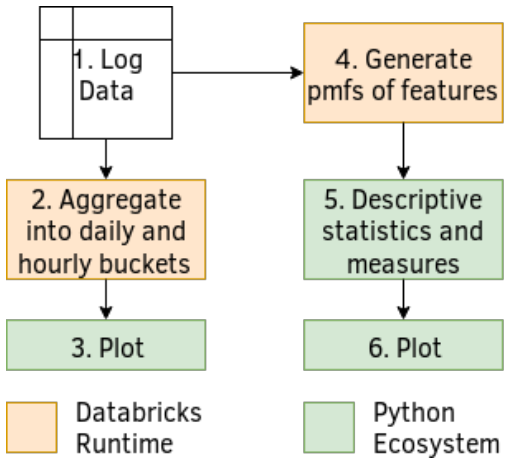


Figure 2: Characterization workflow.

3.1 Overview of the Process

Our process for cloud-based Spark operations consists of large-scale data monitoring, done concurrently with data pre-processing and preliminary analysis, followed by in-depth analysis.

Figure 2 depicts the core of our process. Log Data (component 1 in the figure) performs *log collection*. It continuously collects monitoring data and stores it into logs ready to be parsed (queried). To achieve this goal, this component must be highly reliable and redundant, stream monitoring data in and process it rapidly, use batch processing to compact and store data, and generate indexes to make querying tractable. Although for this component we rely on the Databricks Runtime system out-of-the-box, such features are provided by many of the monitoring systems based on the Apache ecosystem, e.g., using Kafka as communication pipeline, a streaming platform such as Apache Storm or Spark, and a big data platform such as MapReduce or Spark; the indexing, high availability, and redundancy features require domain-specific engineering.

For this work, the operations (*accesses*) recorded in the log are *reads* and *modifications*. Modifications are namespace modifications performed by the runtime, i.e., create, delete, bulk delete, create directory, and rename operations. Overall, the analysis focuses on *features*. A feature is a property of the access that can be computed by virtue of the item being accessed, such as popularity, or by virtue of its relative position to another (previous) access, such as interarrival time. These features can take on different values called *variates*.

Following log collection, the process branches into the analysis of long-term trends and the analysis of current features, both of which are conducted periodically in parallel with the log collection

Table 1: Format of a pre-processed log line.

hashed file path	hashed cluster id	hashed worker id	
size	timestamp	operation	

Table 2: Information about the analyzed logs.

Name	Period	Log Data Size
F	6 months	600TB
W1	1 week	15TB
W2	1 week	30TB

process, but with frequency under the control of the system analyst. (This assumes the system operator collects monitoring data at much higher frequency than the analyst needs updated results, which is typically the case; in practice, a re-analysis of monitoring data could occur every day for business decision.)

For the *analysis of long-term trends*, the raw monitoring data is both too large (big data) and includes sensitive information. We thus first pre-process it into a *compact format* and *apply a normalization step*, which we describe in Section 3.3. Then, we perform a typical long-term analysis, observing the evolution of the number of reads and modifications, and of the sizes of each of these operations (component 2 in Figure 2).

For the *feature analysis*, we use a diverse set of statistical tools on selected features: we compute the empirical probability density function (*EPDF*) and the empirical cumulative distribution function (*ECDF*), descriptive statistics, the Hurst parameter for long-term dependence, etc. We describe the key elements of this analysis in Section 3.4.

A key feature of our process is that it is primarily aimed at human analysis, for which it always concludes its operational branches with visualization (*plotting*)—components Plot (3 and 6 in Figure 2).

3.2 Log Collection

Table 1 summarizes the logs collected for this work. All accesses are timestamped, with timestamps were recorded at the time of logging as time since Unix Epoch, in milliseconds. Overall, the traces correspond to the combined workloads of organizations spanning healthcare, manufacturing, web services, and advertising. (We address the bias inherent to these traces in Section 8.)

Trace `Full` (F) collects all data accesses, both reads and modifications, over the entire 6-month period. We use F to analyze long-term trends. Collecting this data results in a massive log; the F trace exceeds 600 TB. In addition to the full workload, we collect over shorter periods of time more detailed data, useful for computing the histograms of features such as popularity and interarrival time. We thus collect one-week traces `Week 1` (W1) and `Week 2` (W2). The size of these smaller traces was of 15 TB and of 30 TB, respectively.

3.3 Pre-Processing

The collected logs were pre-processed into a compact format for analysis, which we depict in Table 1. Any data fields in the logs that are unrelated to our analysis were removed. The cluster and worker identifiers were originally strings, which both go contrary to the privacy needs of Databricks, and increased the storage, memory and computation costs. They were hashed using Murmur3 hash, and stored the most significant 64 bits of the 128-bit numbers of the hash. Thus, we had no access to any identifiable information during the analysis. Murmur3 was used for its performance and uniform distribution over the key space.

The magnitudes of variates of a features were normalized. This was done to keep the popularity and costs of the company a secret. However, the relative difference between magnitudes of variates and across features is still true and remains applicable. For long-term trends, the normalization was performed by dividing the count of an event by a normalizing factor. For example, the number of reads per day plotted is not the real number of reads that were performed on that day. For the statistical properties of individual features, data is presented in the form of a empirical cumulative distribution function which is normalized by definition. The variates themselves are not normalized. For example, when considering the size of reads, the sizes themselves are real.

3.4 Statistical Analysis

For ECDF plots, in some cases, we use a *symlog* axis instead of a pure logarithmic axis. We do this when the 0 variate is important to the plot. A symlog axis is a logarithmic axis where the are regions close to 0 are on a linear scale. Thus, we avoid the issue of $log(0)$ being undefined.

In many plots, such as Figure 7, we depict two similarly shaped curves. Each such curve corresponds to one the two different one-week-long periods in Table 2. For these figures, the *purple curve represents the data from W1; the green one, the data from W2.*

We observe that even though there is a large increase in the accesses from January to May, the general distribution does not change significantly. To quantify this claim, we use two tests to measure the similarity between two empirical distributions: Kolmogorov-Smirnov (KS) test [6] and Pearson's χ^2 test [23]. The KS test measures the maximum difference between the two cumulative distribution functions. We use the two-sample KS test, which measures the maximum difference between two empirical cumulative distribution functions. We chose this because it is easy to understand and the reader can visually see the quantitative distance output by this test in the graphs. Another reason is that it is distribution independent, unlike the Anderson-Darling test. Thus, we do not require critical value tables to measure goodness of fit. This is useful as we do not know the underlying distributions of the samples we have. The result of the KS test is used as an indicator of divergence. The χ^2 test measures the difference between histograms of two empirical distributions. i.e., it measures the difference in the probability at each variate. This was also chosen to be easy to understand and gives a complete view of the distribution instead of just at the location where the difference is maximum. This helps identify minor differences.

In some cases, we use more than one trace in the same part of the analysis. Because the variates for these cases may be different, we rebin the data into 100 logarithmic bins so that the distributions can be compared. Where rebinning without interpolation is not

Table 3: Descriptive statistics about number of reads and data read *per day*.

	total	mean	median	std. dev.
num. reads	5.57×10^5	3.06×10^3	2.83×10^3	1.11×10^3
bytes read	1.84×10^{12}	1.00×10^{10}	9.60×10^9	3.18×10^9

possible, we use 10 logarithmic bins or linear bins. The KS statistic is not sensitive to the number of bins. The χ^2 statistic is very sensitive to the number of bins chosen. Increasing the number of bins by an order of magnitude increases the χ^2 statistic by an order of magnitude, especially when it is small. However, the p-value doesn't change, making it a valid test. We use the implementation of χ^2 from the SciPy scientific computing library, version 1.1.0.

For each empirical distribution presented, we also present descriptive statistics. This includes the median, mean and standard deviation which are widely known. We also provide additional descriptive statistics. To quantify the dispersion of the data, we use the Coefficient of Variation (CV), which is the ratio of the standard deviation to the mean, and the Inter Quartile Range (IQR), which is the difference between the value at 75th percentile and 25th percentile. **Tail weight** is the sum of magnitudes of those elements which are in the 99th percentile as a fraction of the sum of magnitudes of all the elements.

For features where it is relevant and informative, the Hurst parameter is used to estimate long-range dependence [9]. It specifies if the value at a certain time would be higher, lower or randomly distributed based on the previous values. A Hurst parameter below 0.5 indicates a tendency of the series to move in the opposite direction of the previous values. i.e., highs are followed by lows. Thus, appearing to have high jitter. A value of 0.5 indicates random Brownian motion and above 0.5 indicates a tendency towards well defined peaks. We use the rescaled range (R/S) method to estimate the Hurst parameter.

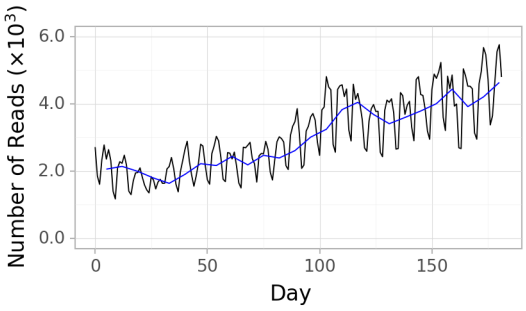
4 LONG-TERM TRENDS

We highlight the following long-term trends:

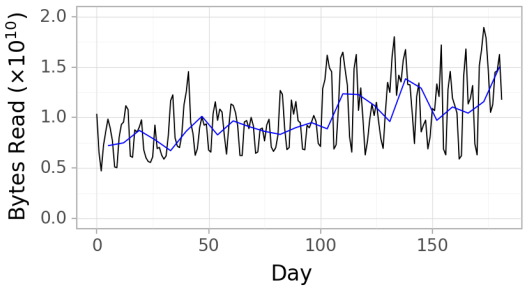
- **MF4.1** The number of reads and bytes read per day have doubled over 6 months.
- **MF4.2** The number of modifications per day has remained at the same level throughout the analysis period.
- **MF4.3** Both reads and modifications follow a diurnal pattern.
- **MF4.4** Large imbalance in number of reads and bytes read per hour occur on *daily* and *weekly* basis.
- **MF4.5** There are 2 orders of magnitude less modifications happening than reads.
- **MF4.6** Most modifications are file creations.

4.1 Reads

We analyze the trends in number of reads and in bytes read, and find both increase significantly over the period we analyze (**MF4.1**). Figure 3 depicts the number of reads and bytes read per day over a 6 month period. Excluding the variation between days of the same week, the number of reads (Figure 3a) has increased from about 2.0×10^3 to over 4.0×10^3. We observe a similar phenomenon for number of bytes read (Figure 3b). We conjecture that this is due to



(a) Number of reads per day with weekly mean as the blue trend line



(b) Bytes read per day with weekly mean as the blue trend line

Figure 3: Number of reads and total amount of bytes read *per day* over a period of 6 months (normalized).

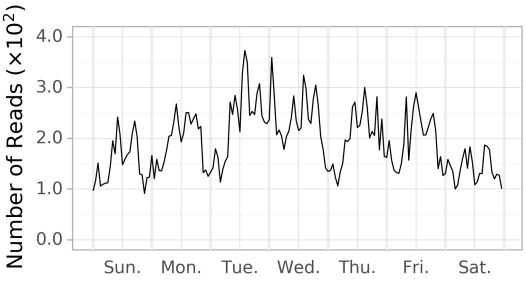
Table 4: Descriptive statistics about number of reads and data read *per hour* for a selected week.

	total	mean	median	std. dev.
num. reads	3.25×10^4	193.4	193.1	60.6
bytes read	9.36×10^{10}	5.57×10^8	5.57×10^8	1.95×10^8

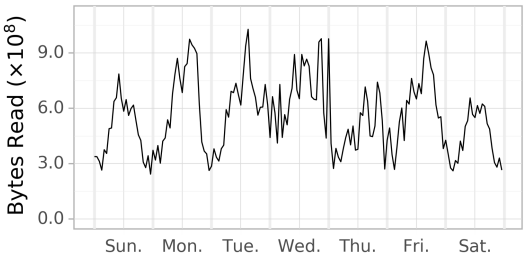
a combination of more users using the subset of ecosystem under study and existing users increasing their usage.

The descriptive statistics for the metrics "number of reads per day" and "bytes read per day" are summarized in Table 3. The high standard deviation indicates that days which experience significant higher or lower number of reads and bytes read occur. The similar values of mean and median indicate that the deviation from the mean is evenly distributed across low activity and high activity days.

We analyze the number of reads per hour and bytes read per hour over a period of one week, and find that significant load imbalances occur on a *weekly* and *daily* basis (**MF4.4**). Figure 4 depicts the number of reads and bytes read over a 1 week period. The number of reads (Figure 4a) changes from day to day. It peaks on Tuesday and bottoms out on Saturday. This is the same pattern that is also visible as variation in Figure 3. The number of reads also vary on a hourly basis, with the peak occurring during noon GMT and the period of least activity around midnight GMT (**MF4.3**). This is a diurnal pattern. Both the aforementioned variations (weekly and diurnal) also occur in number of bytes read (Figure 4b). A



(a) Number of reads per hour



(b) Bytes read per hour

Figure 4: Number of reads and total amount of bytes *per hour* over a period of one week (normalized).

Table 5: Descriptive statistics about number of modifications *per day* over 6 months and *per hour* for a selected week.

	total	mean	median	std. dev.
per day	1.37×10^4	75.03	64.23	19.19
per hour	594.3	3.5	3.4	0.8

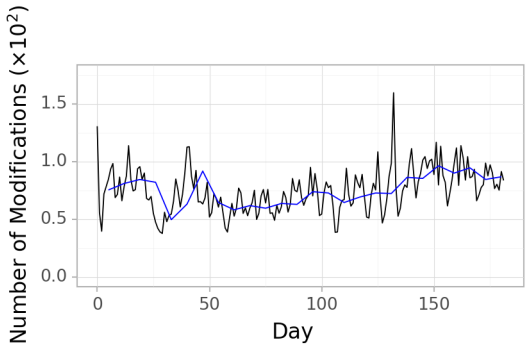
likely hypothesis is that a lot of jobs contributing to the peak are interactive jobs users submit during working hours.

The observation that there is an imbalance of usage during the week can prompt organizations to schedule their workload to fall on less busy days. In the same day, jobs can be scheduled during less busy hours of the night. This helps organizations take advantage of spot auction markets on EC2, to lower their compute costs, derived from the assumption that the spot market costs would be lower if there are fewer users bidding for the same compute resource.

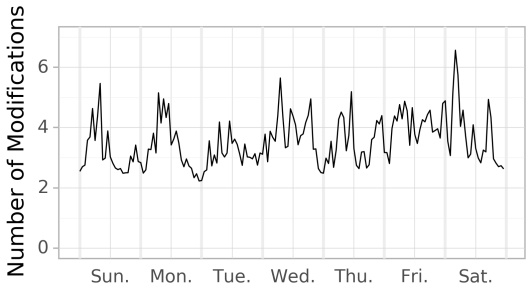
4.2 Modifications

We analyze the number of modifications per day, and find that the number of modifications has remained about the same throughout the analysis period (**MF4.2**). Figure 5a shows the number of modifications per day. Excluding the variation between days of the same week, the number of modifications remains at approximately the same level. Figure 5b depicts the number of modifications per hour. A diurnal pattern is readily apparent (**MF4.3**).

Table 5 presents the descriptive statistics characterizing the number of modifications per day and per hour. The total and mean number of modifications are two orders of magnitude lower than the number of reads in Tables 3 and 4 (**MF4.5**). Table 6 presents the distribution of number of modifications across operation types.



(a) Number of modifications *per day* over 6 months with weekly mean as the blue trend line



(b) Number of modifications *per hour* over 7 days

Figure 5: Modifications trend (normalized).

Table 6: Types of modifications over the 6-month period.

Create	Delete	DeleteMultiple	MkDirs	Rename
1.15×10^4	2.13×10^3	2.28	41.89	28.47

The number of creation operations is much higher than any other operation (**MF4.6**).

5 STATISTICAL ANALYSIS OF READS

Reads constitute a large majority of this workload and are the type of accesses that are frequently optimized through caching, tiering or other techniques to improve system performance. It can be seen from the long term trends in Section 4 that reads overwhelmingly dominate the workload. This can also be inferred analytically, as big data processing (this workload) is concerned with reading large amounts of data and processing it in different ways to gain valuable insight. In this section, we focus on the statistical properties of reads.

A single read has several features which we look at: the file that is being read, popularity of the file, size of the read, interarrival time and reuse time. Our major observations are:

MF5.1 The number of reads and bytes read exhibit negative long range time dependence.
MF5.2 All features (size, popularity, etc.) have a heavy tail.
MF5.3 Reads occur in bursts.
MF5.4 The distributions of features are stationary over long time periods.

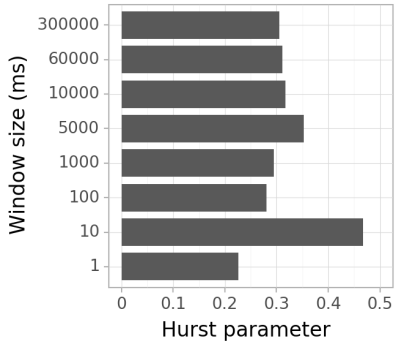


Figure 6: Hurst parameter estimation for number of reads in a time window.

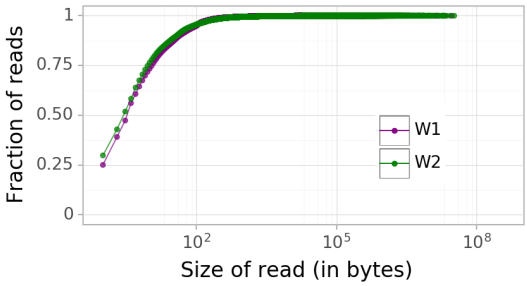


Figure 7: ECDF Read size vs. fraction of reads with logarithmic horizontal axis.

5.1 Count

We analyze the number of reads per time window, and find evidence of inverse long range time-dependence (**MF5.1**), and bursty behavior (**MF5.3**). We estimate the decay of this dependence using the Hurst Parameter for different window sizes in Figure 6. The Hurst parameter is particularly relevant to stationary time series, and we assume that the series of reads is stationary at small intervals.

The period we look at is one thousand times the window size for every window size; for example, if the window size is 10 milliseconds then we look at a period of 10 seconds. The Hurst parameter is lower than 0.4 for all window size, except one. This is evidence of inverse long term time-dependence. That means that over a specific period, the number of reads might increase or decrease but generally falls back towards the mean. This contrasts other studies of time dependence in big data workloads which present a positive long range time-dependence [1].

A burst is defined as a period of high number of reads (high activity) surrounded by a period of low number of reads (low activity). Negative long range time-dependence necessarily implies that periods of low activity are followed by high activity and vice versa. Thus, periods of high activity are surrounded by periods of low activity, satisfying the definition of a burst. Therefore, number of reads exhibits bursty behavior.

5.2 Read Sizes

The read size refers to the number bytes transferred from the cloud storage to the local machine per request. While checking for long term dependence, it refers to the number of bytes transferred in a time window. We analyze read sizes, and find evidence of inverse long range time-dependence (**MF5.1**), a heavy tail (**MF5.2**), bursty behaviour (**MF5.3**), and stationarity (**MF5.4**). Figure 7 depicts an empirical cumulative distribution of read sizes with logarithmic horizontal axis.

Visually and also from the descriptive statistics in Table 7, we observe that most requests read little data with the median request reading 3 or 4 bytes. Most of the bytes transferred are due to reads of large items. This is most visible in the "tail" item in the descriptive statistics. That shows that more than 80% of the data transferred in both periods is due the 99th percentile of requests sorted by size. We conjecture that this is a result of a large number of requests being *metadata reads*. They take advantage of the S3 feature which allows one to read certain range of bytes in a file instead of the whole file. These reads can be used to read file headers and assorted metadata and make decisions, such as to read the file or not. Thus, metadata reads account for most reads while data transfer for processing accounts for most of the data transferred.

The read size is small compared other studies with 90% reads smaller than 1KB. In other studies 90% of reads are smaller than 1KB in web caches [2], 100KB in HPC [3], 1MB in HPC [13], 4MB in consumer cloud [20], 10MB in video delivery [25] and 15MB in HBase [14].

Figure 7 depicts that distributions of the two traces W1 and W2 are not too different. The similarity has been quantified in Table 8. The variates were rebinned into 10 logarithmic bins for this purpose. We know from Figure 3 that the number of reads and bytes read has changed. This indicates that magnitude of number of reads or total data read has minor influence on the distribution of read sizes. Thus, the distributions remain stationary over long time periods.

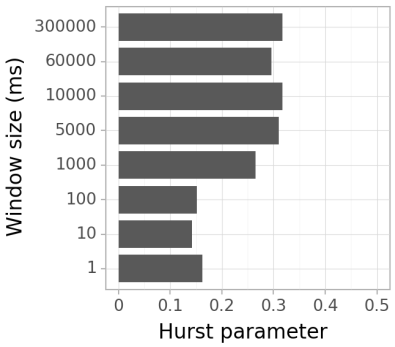


Figure 8: Hurst parameter estimation for bytes read in a time window.

We use the Hurst parameter to estimate the time-dependence of the series at different time scales, as done for counts. For every time window the sum of the bytes read by all reads in that window is considered as the value of the window. Figure 8 indicates that for all considered window sizes the Hurst parameter was estimated to be less than 0.35. The value is low, indicating an inverse correlation between subsequent values and a tendency of the series to fall back towards the mean—more so for small windows. This leads to periods of high activity surrounded by periods of low activity. The bursty behavior of the workload is characterized by the calculated Hurst parameter.

Table 7: Descriptive statistics about read size.

Period	median	mean	std. dev.	CV	IQR	tail weight
W1	4	100	31,535	316	11	0.84
W2	3	136	54,191	400	9	0.89

Table 8: Quantifying similarity of size distributions during two periods.

KS distance	χ^2 distance	χ^2 p-value
0.03	0.01	1.0

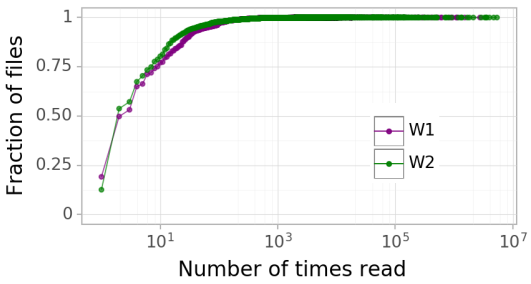


Figure 9: ECDF of Popularity vs. number of files with that popularity with logarithmic horizontal axis.

5.3 Popularity

The popularity of files is a measure of the number of times a file has been accessed. We analyze the popularity of files, and find a heavy tail (**MF5.2**) and stationarity (**MF5.4**). We compute popularity of files from hashed file paths in the anonymized data. Figure 9 depicts the empirical cumulative distribution function of popularity of files with a logarithmic horizontal axis. The descriptive statistics of popularity of files are presented in Table 9.

Visually and also from the descriptive statistics in Table 9, we observe that most files are not popular with a median and mean of 3 and 18. From the tail weight statistic, 0.37%-0.44% of reads are to the 99th percentile (1%) of files. But, 90% of reads are due to 70th percentile (30%) of files. This matches the observation in [1] that 90% of reads are due to 71st to 78th (29% to 22%) percentile. The tail of popularity of files is less heavy than read size, but it is heavy nonetheless.

Figure 9 depicts that distributions of the two traces W1 and W2 are not too different. Their similarity is quantified in Table 10. The variates were rebinned into 100 logarithmic bins for this purpose. We know from Figure 3 that the number of reads is higher during the second period. This is evidence towards the stationary nature of this distribution of popularity across a sufficiently large period, even if the magnitude of number of reads changes.

5.4 Interarrival Times

Interarrival time of a read is the time elapsed between the read and the previous read. We analyze interarrival times, and find near zero values at the ecosystem level.

Large ecosystems such as the one at Databricks do not operate as a single monolith; i.e., it is not just one large compute system making requests to a storage system. These ecosystems consist of numerous systems which themselves contain multiple sub-systems. As mentioned in Section 2, the ecosystem at Databricks consists of numerous Spark clusters, each of which is further composed of multiple workers. Storage related operations such as metadata management and caching operate in these layers and not globally. Storage accesses from different clusters and from different workers may not exhibit the same behavior as that of the whole system. Thus, we study interarrival time behavior at the whole ecosystem level and the level of individual virtual clusters.

Figure 10 depicts the empirical cumulative distribution function of interarrival times with linear axes. From the figure and the descriptive statistics in Table 11, an overwhelming number reads have a 0 interarrival time, when measured with millisecond precision. The number of reads falls precipitously with increasing interarrival time, highlighting the high frequency of reads in the system.

Figure 10 depicts the ECDFs for W1 and W2 having a similar distribution. This similarity has been quantified in Table 12. The first six categories of the histograms were considered for this quantification. The reads are more frequent (smaller mean interarrival time) in May than they were in January. This is a consequence of a higher total reads by the system as presented in Section 4.

Table 9: Descriptive statistics about popularity.

Period	median	mean	std. dev.	CV	IQR	tail weight
W1	3	18	355	20	7	0.37
W2	2	16	615	40	6	0.44

Table 10: Quantifying similarity of popularity distributions during two periods.

KS distance	χ^2 distance	χ^2 p-value
0.07	0.2	1.0

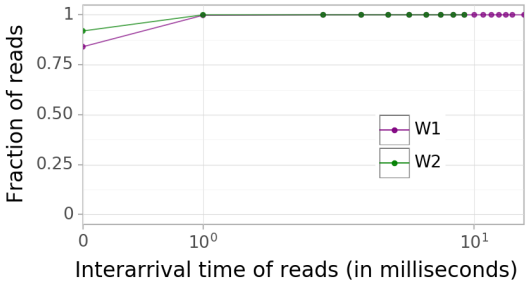


Figure 10: ECDF of interarrival time vs. number of reads.

Table 11: Descriptive statistics about interarrival of reads.

Period	median	mean	std. dev.	CV	IQR	tail weight
W1	0	0.16	0.38	2.32	0	1.0
W2	0	0.08	0.27	3.38	0	1.0

Table 12: Quantifying similarity of interarrival time distributions during two periods.

KS distance	χ^2 distance	χ^2 p-value
0.08	0.12	0.99

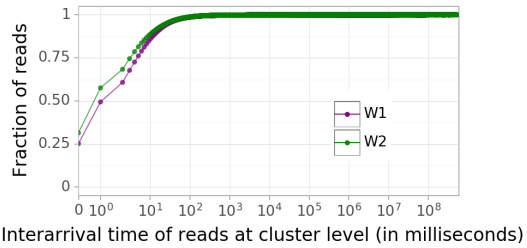


Figure 11: ECDF of interarrival time at cluster level of all clusters vs. number of reads with logarithmic horiz. axis.

Table 13: Descriptive statistics about interarrival times at cluster level.

Period	median	mean	std. dev.	CV	IQR	tail weight
W1	2	136	127,558	936	5	0.96
W2	1	95	104,626	1101	4	0.95

Table 14: Quantifying similarity of cluster level interarrival time distributions during two periods.

KS distance	χ^2 distance	χ^2 p-value
0.08	0.03	1.0

We define *interarrival time at cluster level* to be the time difference between a read and a previous read from the same cluster. We analyze interarrival times at the cluster level, and find a heavy tail (**MF5.2**), bursty behavior (**MF5.3**) and stationarity (**MF5.4**). Figure 11 depicts the empirical cumulative distribution function of this feature. Unlike interarrival times at the ecosystem level, where more than 75% reads had 0 interarrival time, the fraction of reads with 0 interarrival time is only around 25% at the cluster level.

The descriptive statistics for this feature are presented in Table 13. The median interarrivals of 2 and 1 indicate that reads are still frequent at this level of granularity. But, there are long periods with no reads as evidenced by the high mean value and the extremely high standard deviation, which is over 1000 times larger than the mean. So, there are reads which occur very close together and there are long periods of no activity. This is again evidence of burstiness.

Figure 11 depicts the similarity between distributions for W1 and W2. This similarity is quantified in Table 14. Thus, the distributions remain stationary over long time periods. The variates were rebinned into 100 logarithmic bins for this purpose.

5.5 Reuse Times

Reuse time refers to the time elapsed between two reads to the same file. We analyze the reuse times, and find a heavy tail (**MF5.2**), and stationarity (**MF5.4**). Figure 12 depicts the empirical cumulative

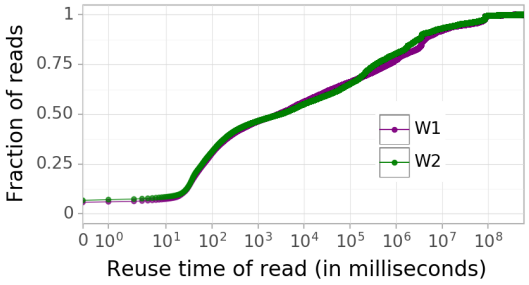


Figure 12: ECDF of reuse time vs. number of reads with logarithmic horizontal axis.

Table 15: Descriptive statistics about reuse times.

Period	median	mean	std. dev.	CV	IQR	tail weight
W1	2,720	5.77×10^6	2.46×10^7	4	6.77×10^5	0.30
W2	3,042	5.10×10^6	2.47×10^7	5	3.55×10^5	0.35

Table 16: Quantifying similarity of reuse time distributions during two periods.

KS distance	χ^2 distance	χ^2 p-value
0.05	0.008	1

distribution function of reuse times. The histogram of reuse times had over 90 million categories. This is interesting as other features such as size and popularity had histograms with multiple orders of magnitude fewer categories. This shows that reuse times are dispersed over the number line with little grouping behaviour when considered at the millisecond scale. There is also a steep increase in reads with reuse time between 10 and 100 milliseconds.

The descriptive statistics for reuse time are quantified in Table 15. The median reuse time is high at 2720ms for W1 and 3042ms at W2 compared to the median interarrival times of 0-4 observed. Therefore, reads of the same file are necessarily interspersed with reads of other files most of the time. The tail weight of 0.30 corresponds to a heavy tail.

Figure 12 depicts the similar distribtuion of reuse times for W1 and W2. This has been quantified in Table 16. Thus, the distributions remain stationary over long time periods.

6 DISTRIBUTION ACROSS CLUSTERS

The Databricks ecosystem is composed of many of systems and subsystems working in concert to deliver results. The distribution of reads across clusters refers to the number of reads contributed by each cluster to the total workload. We analyze the distribution of number of reads and bytes read across clusters, and our major finding is:

MF6.1 The distribution of number of reads and bytes read over clusters is heavy tailed.

Figure 13 depicts the empirical cumulative distribution function of the number of reads by each cluster. Most clusters perform a small number of reads, and the majority of the reads are by a very small number of clusters. The tail is very heavy which is apparent

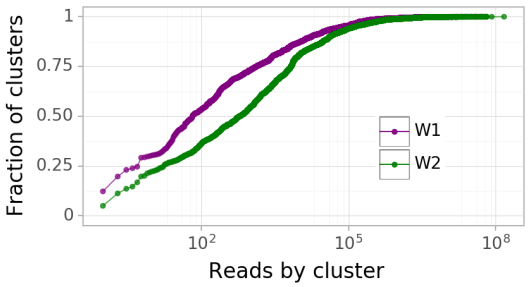


Figure 13: ECDF of number of reads by a cluster vs. fraction of clusters with logarithmic horizontal axis.

Table 17: Descriptive statistics about distribution of reads across clusters.

Period	median	mean	std. dev.	CV	IQR	tail weight
W1	67	44,694	764,354	17	1,332	0.78
W2	606	69,841	999,796	14	6,769	0.69

Table 18: Quantifying similarity of distribution of reads across clusters.

KS distance	χ^2 distance	χ^2 p-value
0.20	0.17	1.0

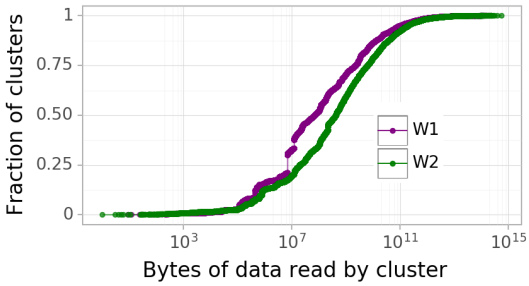


Figure 14: ECDF of bytes read by a cluster vs. fraction of clusters with logarithmic horizontal axis.

from the statistics in Table 17. Particularly from the tail weight statistic that 70% of the reads are from the 99th percentile of the most popular clusters.

The empirical cumulative distributions in Figure 13 depict that the distributions of W1 and W2 are similar. There is a slight difference and this is quantified in Table 18. The variates were rebinned into 10 logarithmic bins for this purpose.

6.1 Bytes Read by Clusters

Figure 14 depicts the empirical cumulative distribution function of bytes read by each cluster. Unlike all other distributions in this work, this one is also flat on the lower end (closer to 0) and not just the higher end of the horizontal axis. The steep incline in the middle indicates that the majority of the clusters read a moderate amount of data, between 10MB and 10GB. We hypothesize that the prevalence of cluster little data read is due to failed or newly started

Table 19: Descriptive statistics about distribution of bytes read across clusters.

Period	median	mean	std. dev.	CoV	IQR	tail weight
W1	7.36×10^7	1.87×10^{11}	3.61×10^{12}	19	2.36×10^9	0.90
W2	4.37×10^8	2.01×10^{11}	4.17×10^{12}	21	6.54×10^9	0.82

Table 20: Quantifying similarity of distribution of bytes read across clusters.

KS distance	χ^2 distance	χ^2 p-value
0.18	1.51	1.0

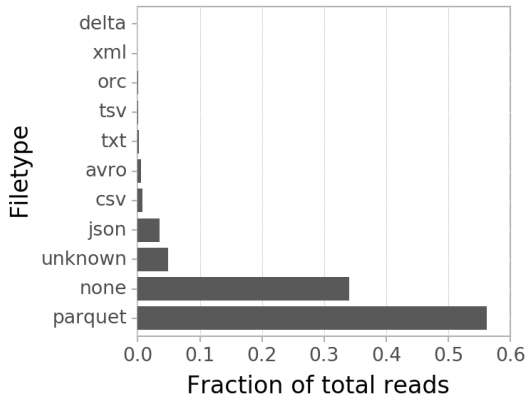


Figure 15: File format popularity for W2.

clusters. It is a very heavy tailed curve from descriptive statistics in Table 19. Particularly the tail weight statistic that 82%-90% of reads are from 99th percentile of most popular clusters.

The empirical cumulative distributions in Figure 14 depicts that the distributions during W1 and W2 are close. There is a slight difference and this is quantified in Table 18. The variates were rebinned into 100 logarithmic bins for this purpose.

7 DISTRIBUTION ACROSS FILE TYPES

The distribution of reads across file types helps understand what formats and file features are prevalent. This leads to informed decision making about what formats to optimize and support. In this section, we look at two features of files: storage format and compression. The storage format is the structure according to which bytes are stored on disk; examples include JSON, CSV and Parquet. The compression algorithm is the one used to compress stored data; examples include gzip and snappy. We obtain the information of the storage format and compression algorithms from the file extensions extracted from read requests and not from the data in files stored on the filesystem. Our major findings are:

MF7.1 Parquet is the most popular file format.

MF7.2 Snappy and Gzip are the most popular compression schemes.

7.1 File Format Popularity

Figure 15 depicts the distribution of file formats in trace W2. We see that Parquet is the most popular format. This is mainly due to its

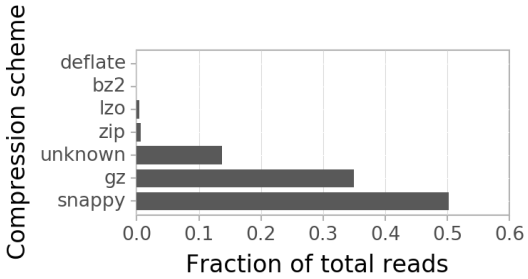


Figure 16: Popularity of compression schemes for W2.

suitability for big data analysis; in addition, it is the default format for storage in the Databricks ecosystem. A surprise to us was that this was followed by accesses to file without any file extension. This means that they did not have any string with a "." (dot) in them after the last "/" (slash). We conjecture that reads with no filetype are comprised of parquet or other files without any extension. This is followed by an almost equal fraction of accesses to JSON and files with unknown file format; JSON is a popular data format, particularly for web applications. The reads with the unknown extension are those whose paths were truncated or did not have an extension apart from the extension of the compression scheme used. The number of unknown files was a surprise; this is evidence that a significant fraction of the files are deeply nested, thus having long file paths, or are in compressed archives without any special file formatting such as Parquet. Finally, popular big data file formats like CSV, Avro, and ORC, also make an appearance.

7.2 Compression Scheme Distribution

Figure 16 depicts the distribution of compression schemes in trace W2. Gzip and Snappy are, by far, the most popular compression schemes, with an order of magnitude higher fraction of reads than anything else. The unknown fraction refers to files where the compression schema is not present in the extension or the file path was truncated.

8 THREATS TO VALIDITY

Our work has several limitations. We discuss in this section the three main limitations, the correlation between features, the bias inherent to our traces, and the lack of an example of direct use of the main findings.

Two features are said to be positively (negatively) correlated if an increase or decrease in one necessarily causes a corresponding (inverse) increase or decrease in the other. Not checking for correlations can lead to biased characterizations, because the reader is left with the impression that the variables under study are independent. Although not presented in this work, we have conducted a preliminary analysis of the linear correlation between the measured features. We have calculated the Pearson correlation coefficient between all possible pairs of features, at different levels of magnification. As expected, most correlations were between 0.1 and -0.1, indicating low correlation. Notably, among all the correlations we calculate, a few were above 0.9 (high correlation), but all others were between -0.17 and 0.25, which indicates these features are independently varying.

The bias inherent in trace use is that it is possible that our results, albeit valid for the traces used in the work, are not representative at-large. We argue that this trace is representative of remote storage used by collections of small clusters. The different clusters which are part of the trace are used by widely different organizations including healthcare, manufacturing, web services, and advertising. We also observed that the workloads from clusters from diverse organizations looked similarly across the different characteristics.

It has become common in recent characterization studies to include in the publication an example of direct use of the main findings, for example, for tuning a component of the system under study. To the proponents of this approach, providing such an example can reduce the threat that the findings may be useless. In our view, doing so is actually more of a threat to validity than not conducting such an experiment, because (i) the analyst has a direct incentive to select specific results over others, (ii) the construction of a useful example removes resources (and, ultimately, pages) from the characterization itself, and (iii) in the long-run, implying characterizations are not enough diminishes the scientific standing of our community.

9 RELATED WORK

We compare our work with previous studies on real-world, large-scale storage workloads, including big data. Overall, because of our focus on object stores, our work complements the body of work done on hardware-level storage for big data workloads, e.g., [21].

We conduct a systematic survey [18] of over 20 high quality venues in large-scale systems (HPDC, SC, etc.), systems and operating systems (NSDI, OSDI, etc.), and performance (SIGMETRICS, ICPE, etc.). Because the common type of storage studied in this work, object stores hosted in the cloud, is relatively new, we include in our systematic survey only work published since 2010. Table 21 summarizes the main results of our systematic survey. We observe that the studies we find differ in the type of system, the resources available, and the types of analysis included in the study. Ours is the first study to cover systematically all the dimensions covered in the table; in this sense, it complements each of the previous studies along at least one dimension.

Closest to our work we find the previous work by Abad et al. [1] and by Chen et al. [5] on big data workloads that read from a distributed storage system. Our work complements these studies with analysis of long-term trends, time-dependence, bursts, stationarity, and distribution across clusters. Our work also extends and supports the characterization of file popularity, interarrival time, reuse time, and file formats.

10 CONCLUSION

Characterization studies help system operators tune their systems and system designers create new techniques, based on observed behaviour of existing systems. Motivated by the emergence of cloud-based big data processing, and in particular by the emergence of Spark as a new class of big data processing systems, in this study we have conducted the first workload characterization of a representative cloud-based Spark deployment.

We have collected from Databricks a combined Spark and S3 workload trace spanning over 6 months of operation, and designed a method to process and analyze this trace. Our method focuses

Table 21: Comparison of our work with previous characterization studies, ordered chronologically.

	Type	Pop.	TimeDep.	Interarrival	Size	Levels	SysPop.	Ops.
Chen 2011 [4]	Enterprise						✓	✓
Carns 2011 [3]	HPC	✓			✓			
Abad 2012 [1]	MapReduce + HDFS	✓	✓	✓	✓			✓
Chen 2012 [5]	MapReduce + HDFS	✓	✓		✓			
Atikoglu 2012 [2]	Web Cache	✓		✓	✓			✓
Liu 2013 [20]	Consumer Cloud				✓		✓	✓
Harter 2014 [14]	Messaging + HDFS	✓		✓	✓			
Gunasekaran 2015 [13]	HPC	✓			✓			
Summers 2016 [25]	Video Delivery				✓		✓	
This Work	**Spark + S3**	✓	✓	✓	✓	✓	✓	✓

on investigating the long-term trends appearing in the read and modification operations, the statistical properties of the number and size of reads, and the relative popularity of clusters and of file formats.

Overall, our study contains several novel insights, but also corroborates well the findings of previous studies of big data storage workloads. This work also gives quantitative evidence that Spark and previous (e.g., MapReduce) storage workloads differ. Collected from Sections 4–7, our main findings are:

MF4.1 The number of reads and bytes read per day have doubled over 6 months.
MF4.2 The number modifications per day has remained at the same level throughout the analysis period.
MF4.3 Both reads and modifications follow a diurnal pattern.
MF4.4 Large imbalance in number of reads and bytes read per hour occur on *daily* and *weekly* basis.
MF4.5 There are 2 orders of magnitude less modifications happening than reads.
MF4.6 Most modifications are file creations.
MF5.1 The number of reads and bytes read exhibit negative long range time dependence.
MF5.2 All features (size, popularity, etc.) have a heavy tail.
MF5.3 Reads occur in bursts.
MF5.4 The distributions of features are stationary over long time periods.
MF6.1 The distribution of number of reads and bytes read over clusters is heavy tailed.
MF7.1 Parquet is the most popular file format.
MF7.2 Snappy and Gzip are the most popular compression schemes.

For the future, we plan to investigate the impact of our findings on the design and tuning of Spark-based ecosystems.

ACKNOWLEDGEMENTS

We thank Pieter Senster and Databricks Amsterdam; Laurens Versluis and Erwin van Eyk for proofreading. We thank ACM SIGSOFT (the CAPS program) for their travel grant. Projects Vidi Magna-Data and COMMIT/ co-support this work. The work of C. Abad is partially funded by a Google Faculty Research Award.

REFERENCES

[1] Abad et al. 2012. A storage-centric analysis of MapReduce workloads: File popularity, temporal locality and arrival patterns. In *IISWC*.
[2] Atikoglu et al. 2012. Workload analysis of a large-scale key-value store. In *SIGMETRICS*.
[3] H. Carns et al. 2011. Understanding and Improving Computational Science Storage Access through Continuous Characterization. *TOS* 7, 3 (2011).
[4] Chen et al. 2011. Design implications for enterprise storage systems via multi-dimensional trace analysis. In *SOSP*.
[5] Chen et al. 2012. Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads. *PVLDB* 5, 12 (2012).
[6] Conover. 1980. *Practical Nonparametric Statistics, Chapter 6.* Wiley New York.
[7] Microsoft Corp. 2013. 343 Industries Gets New User Insights from Big Data in the Cloud. https://azure.microsoft.com/en-us/case-studies/customer-stories-343industries/
[8] Dean et al. 2004. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*.
[9] Feder. 2013. *Fractals, Chapter 8.* Springer Science & Business Media.
[10] Ghemawat et al. 2003. The Google file system. In *SOSP*.
[11] Ghit et al. 2014. Balanced resource allocations across multiple dynamic MapReduce clusters. In *SIGMETRICS*. 329–341.
[12] Ghodsnia et al. 2014. Parallel I/O aware query optimization. In *SIGMOD*.
[13] Gunasekaran et al. 2015. Comparative I/O workload characterization of two leadership class storage clusters. In *PDSW*.
[14] Harter et al. 2014. Analysis of HDFS under HBase: a facebook messages case study. In *USENIX FAST*.
[15] Hashem et al. 2015. The rise of "big data" on cloud computing: Review and open research issues. *Inf. Syst.* 47 (2015).
[16] Iosup et al. 2011. Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. *TPDS* 22, 6 (2011), 931–945.
[17] Iosup et al. 2018. Massivizing Computer Systems: A Vision to Understand, Design, and Engineer Computer Ecosystems Through and Beyond Modern Distributed Systems. In *ICDCS*. 1224–37.
[18] Kitchenham and Charters. 2007. Guidelines for performing systematic literature reviews in software engineering. EBSE Technical Report EBSE-2007-01. updated, version 2.3.
[19] Lee. 2017. Innovation in big data analytics: Applications of mathematical programming in medicine and healthcare. In *BigData*.
[20] Liu et al. 2013. Understanding Data Characteristics and Access Patterns in a Cloud Storage System. In *CCGrid*.
[21] Pan et al. 2014. I/O Characterization of Big Data Workloads in Data Centers. In *BPOE*.
[22] Raghupathi et al. 2014. Big data analytics in healthcare: promise and potential. *Health information science and systems* 2, 1 (2014).
[23] Rice. 2003. *Mathematical statistics and data analysis, Chapter 13.* China machine press Beijing.
[24] Amazon Web Services. 2016. FINRA Adopts AWS to Perform 500 Billion Validation Checks Daily. https://aws.amazon.com/solutions/case-studies/finra-data-validation/
[25] Summers et al. 2016. Characterizing the workload of a Netflix streaming video server. In *IISWC*.
[26] Trivedi et al. 2018. Albis: High-Performance File Format for Big Data Systems. In *USENIX ATC*.
[27] Zaharia et al. 2010. Spark: Cluster Computing with Working Sets. In *USENIX HotCloud*.