Hadoop's Adolescence

An analysis of Hadoop usage in scientific workloads

Kai Ren[1], YongChul Kwon[2], Magdalena Balazinska[3], Bill Howe[3]
[1] Carnegie Mellon University, [2] Microsoft, [3] University of Washington
kair@cs.cmu.edu, ykwon@microsoft.com, {magda,billhowe}@cs.washington.edu

ABSTRACT

We analyze Hadoop workloads from three different research clusters from a user-centric perspective. The goal is to better understand data scientists' use of the system and how well the use of the system matches its design. Our analysis suggests that Hadoop usage is still in its adolescence. We see underuse of Hadoop features, extensions, and tools. We see significant diversity in resource usage and application styles, including some interactive and iterative workloads, motivating new tools in the ecosystem. We also observe significant opportunities for optimizations of these workloads. We find that job customization and configuration are used in a narrow scope, suggesting the future pursuit of automatic tuning systems. Overall, we present the first *user-centered* measurement study of Hadoop and find significant opportunities for improving its efficient use for data scientists.

1. INTRODUCTION

Hadoop [4], the open-source implementation of Google's MapReduce [12], has become a commonly used tool for Big Data analytics. Due to Hadoop's popularity, it is natural to ask the question: How well does Hadoop actually work? Many papers partly answer this question either by performing direct comparisons to alternate tools [24] or by carrying out measurement studies of production clusters [2, 9, 16, 26]. Prior work, however, analyzes primarily the performance of individual queries and the efficiency of entire Hadoop clusters, focusing on performance metrics at the cluster or application levels such as job resource utilization.

In this paper, we provide a complementary answer to the question of how well Hadoop works. We study what users actually do with their Hadoop system: we study *behaviors and application patterns from a user-centric perspective*. The users that we focus on are called "data scientists" in the sense that they have large datasets and need to process them to extract information. Our goal is to assess how well Hadoop works for data scientists in terms of what they need to do to write Hadoop applications, execute them, tune them, and use them to extract knowledge from their data.

Proceedings of the VLDB Endowment, Vol. 6, No. 10

Our analysis is based on Hadoop workloads collected over periods of five to 20 months in three different clusters. Our traces comprise a total of more than 100,000 Hadoop jobs. The clusters that we study come from academic institutions. Our data scientists are 113 domain experts from various disciplines as we describe in more detail in Section 2. The dataset of one cluster is made publicly available for further study (at this URL: `www.pdl.cmu.edu/HLA/`).

The goal of our work is to better understand user behavior in a Hadoop cluster: Are users submitting workloads consistent with what Hadoop has been designed to handle? Does the MapReduce abstraction work well for the types of data transformations that users need to perform? Are users capable of tuning their jobs as needed? To analyze these higher-level, user-oriented issues, we make use of information that is typically too sensitive for commercial Hadoop users to share, including user information and job specifications (*e.g.*, user identifier, names of Java classes, input/output directories) and high-resolution application-level statistics (*i.e.*, HDFS [8] and MapReduce counters). We also report on feedback from a few of the most active users.

We study three aspects of how users leverage their Hadoop clusters:

- **Application Workload** (Section 3). We first analyze how users author Hadoop applications, the resulting structure of these applications, and how users interact with their Hadoop jobs over time.
- **Tunings** (Section 4). We then study when and how users change configuration parameters and otherwise tune their Hadoop jobs.
- **Resource Usage and Sharing** (Section 5). Finally, we analyze the diverse, aggregate resource-consumption profiles of users.

In this study, we identify both good uses of Hadoop and mismatches between the system design and users' needs that call for various future improvements to the system. To the best of our knowledge, this is the first user-centric Hadoop measurement study, and the first measurement study to compare three research clusters and do so over periods of multiple months.

Overall, our analysis suggests that Hadoop usage is still in its adolescence. We do see good use of Hadoop: All workloads are dominated by data transformations that Hadoop handles well; users leverage Hadoop's ability to process

massive-scale datasets; customizations are used in a visible fraction of jobs for correctness or performance reasons. However, we also find uses that go beyond what Hadoop has been designed to handle:

- There are a significant number of independent, small-scale jobs that may be amenable to simpler, non-Hadoop solutions.
- Surprisingly few users are using higher-level declarative frameworks, though some are constructing "manual" workflows.
- Workloads are highly diverse, suggesting the need to effectively author and efficiently execute a large variety of applications.
- We see significant performance penalties arising from over-reliance on default configuration parameters.

In summary, we thus find that users today make good use of their Hadoop clusters but there is also significant room for improvement in how users interact with their Hadoop clusters: improving the effectiveness of authoring and the efficiency of executing diverse workloads, improving interactivity, better user education, and automatic optimization and tuning of complex applications.

2. DATA SETS OVERVIEW

We analyze the execution logs from three Hadoop MapReduce clusters used for research: OpenCloud, M45, and Web Mining. The three clusters have different hardware and software configurations and range in size from 9 nodes (Web Mining), to 64 nodes (OpenCloud), and 400 nodes (M45). In addition to the log analysis, we also report on feedback from a few of the most active users in the OpenCloud and Web Mining clusters.

OpenCloud. OpenCloud is a research cluster at CMU managed by the CMU Parallel Data Lab. It is open to researchers (including faculty, post-docs, and graduate students) from all departments on campus. In the trace that we collected, the cluster was used by groups in areas that include computational astrophysics, computational biology, computational neurolinguistics, information retrieval and information classification, machine learning from the contents of the web, natural language processing, image and video analysis, malware analysis, social networking analysis, cloud computing systems development, cluster failure diagnosis, and several class projects related to information retrieval, data mining and distributed systems.

The 64 nodes in this cluster each have a 2.8 GHz two quad core CPU, 16 GB RAM, 10 Gbps Ethernet NIC, and four Seagate 7200 RPM SATA disk drives. The cluster ran Hadoop 0.20.1 during the data collection.

M45. M45 is a production cluster made available by Yahoo! to support scientific research [1]. The research groups that used this cluster during the collection of the trace covered areas that include large-scale graph mining, text and web mining, large-scale computer graphics, natural language processing, machine translation problems, and data-intensive file system applications [16]. The 400 nodes in this cluster each contain two quad-core 1.86 GHz Xeon processors, 6 GB RAM, and four 7200 rpm SATA 750 GB disks. The cluster ran Hadoop 0.18 with Pig during the data collection. The first four months of the trace overlap with the end of the trace analyzed by Kavulya *et al.* [16].

Web Mining. Web Mining is a small cluster owned and administered by an anonymized research group. The group comprises faculty, post-docs, and students. Most users run web text mining jobs. The 9 nodes in the cluster each have four quad-core Xeon processors, 32 GB RAM, four 1.8 TB HDD. The cluster runs Hadoop 0.20.203 with Pig.

Log. Table 1 summarizes the duration of each collected log and the number of Hadoop jobs that it covers. For each cluster, our log comprises two types of information for each executed job. All the data was collected automatically by standard Hadoop tools requiring no additional tracing tools.

- *Job configuration files*: Hadoop archives a job configuration file for each submitted job, which is an XML file that carries the specification of the MapReduce job (*e.g.*, the class names of the mapper and reducer, the types of the keys and values, etc.).
- *Job history files*: Hadoop also archives a job history file for each submitted job, which includes for each task the initialization time, start time, completion time, and the time of each phase transition. In addition, this file includes a variety of counters, including the number of bytes and records read and written by each task.

3. APPLICATION WORKLOAD

In order to understand how data scientists today utilize the Hadoop ecosystem of tools, we start by analyzing three aspects of this usage. First, we examine the interfaces most used to author Hadoop applications (Section 3.1). We ask the question: **Do users primarily write raw Hadoop applications or do they rely mostly on higher-level declarative interfaces such as Pig Latin?** Second, we study in more detail the resulting structures of users' data transformations (Section 3.2). We ask the question: **Is the MapReduce abstraction good enough for most data transformations or do users need to combine multiple MapReduce jobs into complex graphs in order to get their desired result?** Finally, we analyze how users manage their data analysis tasks over time (Section 3.3). We ask the question: **Is the batch-oriented nature of Hadoop well suited for the way users seek to interact with their data?**

3.1 Application Types

The Hadoop ecosystem of tools today is rich. In this section, we analyze how users prepare their MapReduce jobs. We ask the following key question: **Do users primarily write raw Hadoop applications (using the native Hadoop Java API) or do they rely mostly on higher-level declarative interfaces such as Pig Latin?**

Method: To answer the above question, we examine all the jobs executed in each of the three Hadoop clusters any time during the collected traces. If the same job was executed multiple times, we consider each execution as a separate job (we study distinct jobs in later sections). We classify each job into one of the following categories:

- **Low-level API:** The simplest way to write a MapReduce application is to provide map and reduce functions through the native *MapReduce* Java API.

Cluster	Duration	Start Date	End Date	Successful/Failed/Killed Jobs	Users
OpenCloud	20 months	2010 May	2011 Dec	51975/4614/1762	78
M45	9 months	2009 Jan	2009 Oct	42825/462/138	30
Web Mining	5 months	2011 Nov	2012 Apr	822/196/211	5

Table 1: *Summary of Analyzed Workloads*

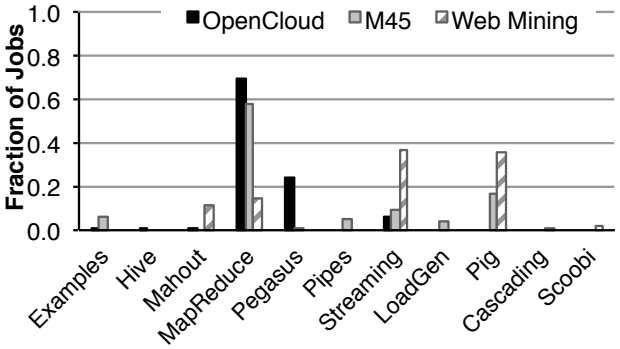


Figure 1: *Fraction of jobs per application type.*

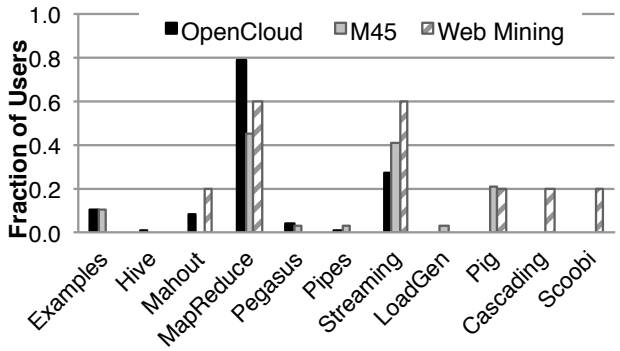


Figure 2: *Fraction of users per application type.*

Streaming allows users to specify map and reduce functions in any programming language. *Pipes* allows users to specify the functions in C/C++.

- **High-level API:** There are a few high-level API frameworks that help users write more complex applications by providing APIs with richer semantics than map and reduce. *Cascading* and *Scoobi* provide high-level APIs for users to easily construct workflows of map and reduce functions [11, 25]. *Pegasus* provides higher-level APIs for graph analysis [15].
- **High-level Language:** *Pig* and *Hive* fall into this category [22, 6]. A user writes a script in a high-level language then the runtime system compiles the script into a directed acyclic graph (DAG) of MapReduce jobs. Users can include user-defined functions.
- **Canned MapReduce Jobs:** There are a few prepackaged sets of MapReduce applications. In this category, the user only changes parameters and/or input datasets. All map and reduce functions are provided by the package. *Mahout* is a package that implements various data-mining algorithms in Hadoop. *Examples* are examples such as wordcount and terasort that are part of the Hadoop software distribution. *LoadGen* is also part of the Hadoop software distribution. It generates workloads for testing and benchmarking purposes.

Results: Figure 1 shows the fraction of jobs per application type in the three clusters. In both the OpenCloud and M45 clusters, the native MapReduce interface is the most popular way to program a MapReduce job while Streaming and Pig are popular in the Web Mining cluster.

Figure 2 shows how many users have used each application type. In all three clusters, most users tried the low level MapReduce API as well as Streaming. In the Web Mining cluster, one user tried different high-level APIs (*i.e.*, Cascading, Scoobi) and one user uses Pig.

Finding: Improvement Opportunity These results have several important implications: First, even though more powerful tools already exist today, users still prefer to write their MapReduce jobs in plain Java (or other language through the streaming interface). This is the case even as users need to transform their data through a sequence or even a DAG of MapReduce jobs (as shown later in Table 2). More complex tools thus do not have the expected uptake in any of the three clusters. Part of the reason could be user education since different high-level tools are popular in different clusters: Pig for M45 and Web Mining. Another reason is failure recovery. Some users from the Web Mining cluster complained that high-level tools that fail in the middle of a complex graph of jobs make it difficult to restart and resume from the location where the failure occurred. Second, legacy code (*e.g.*, unix commands and machine learning packages such as libsvm) and language familiarity (*e.g.*, python, perl, and ruby) also play an important role as shown by the prevalence of Streaming. Third, from talking with OpenCloud users, we find that some users consider these existing tools cumbersome to implement their algorithms and some users complain about the steep learning curve of these tools. Some Web Mining users indicated that certain uses of Hadoop do not focus on the MapReduce model but rather use Hadoop simply as a task scheduler.

3.2 Application Structures

Given that users write primarily MapReduce applications using the native interface, we study the structure of the resulting data transformations. We ask the following key question: **Is the MapReduce abstraction good enough for most data transformations or do users need to combine multiple MapReduce jobs into complex graphs in order to get their desired result?**

Method: We call a *pipeline* a set of related MapReduce jobs that take one or more datasets as input and produce one or more datasets as output. We study the structure of the data transformation pipelines that users execute.

We estimate data dependencies between jobs by looking at their input and output data paths[1]. We assume that input and output data paths are not renamed nor manually

[1]This approach can not capture map-side joins, for example, which may read another input passed through the job configuration. Thus, we may be missing some dependencies.

Cluster	Map Only	Single Job	Short Chain	Long Chain	Iteration	DAG
OpenCloud	21%	31 %	8%	3%	23%	14%
M45	11%	35 %	17%	9%	23%	2%
Web Mining	8%	70 %	14%	8%	0%	0%

Table 2: *Distribution of job structures.*

modified by users. If a job A uses another job B's output as its input, then we say that job A depends on B. The dependency relation forms a directed acyclic graph of jobs. Each connected component forms a pipeline.

Additionally, according to the feedback we received from some of the users in the OpenCloud cluster, several implemented iterative applications. To discover these iterative applications, we examined the names of the map and reduce functions for all applications executed more than 10 times. If the name was indicative of an iterative application such as "kmeans", "clustering", or "iteration", we classified the application into the "iteration" category. These structures were not captured by the previous data-dependency analysis, because users indicated that they renamed the input and output data paths.

Results: Table 2 shows the classification results. Approximately 46% to 78% of pipelines consist of a single job. This fraction is highest in the smallest Web Mining cluster. Additionally, 8% to 21% of single-job pipelines consist of map-only jobs. Another large group of pipelines are iterative applications, which correspond to 23% of pipelines in OpenCloud and M45. Chains of jobs are also popular. "Short chain" are chains consisting of two or three jobs. "Long Chain" pipelines consist of more than three jobs and include possibly additional iterative applications. Finally, some pipelines take the form of more complex directed acyclic graphs (DAGs) of jobs. By examining examples, those DAGs include applications that perform join operations between multiple datasets or run multi-stage machine learning algorithms.

We also plot the distribution of pipeline structures in each month to look at the evolution of these structures over time. Figure 3 shows the result for the OpenCloud cluster (other clusters follow similar patterns, and therefore are omitted for brevity). The distribution changes significantly across months but does not show any particular evolution. Single-job pipelines and map-only pipelines are most popular during the period from 2010-10 to 2011-06, while jobs with complex structures comprised the bulk of the workload during other periods.

Finding: These results show a consistent pattern in the OpenCloud and M45 clusters. Both comprise a good mix of different types of applications from data transformations that require a single MapReduce job to applications with complex structures. The Web Mining cluster is more heavily skewed toward single-job applications. This finding is interesting as it shows that at least 50% of applications in all clusters can fit in the MapReduce paradigm. At the same time, the remaining applications, such as iterative applications and other applications with more complex structures, need to go beyond the single-job MapReduce abstraction. The need for abstractions and efficient implementations beyond the basic MapReduce API is thus strong.

3.3 Application Management over Time

In this section, we analyze how users manage their data

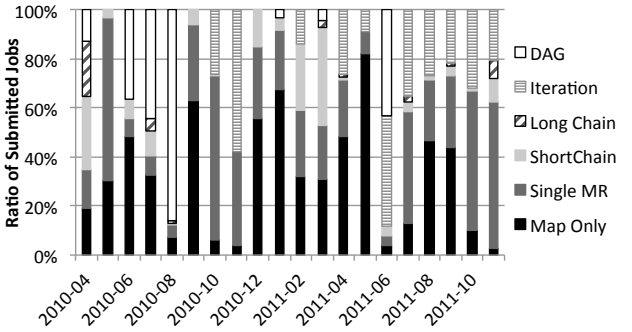


Figure 3: *Fraction of job structures in OpenCloud over time.*

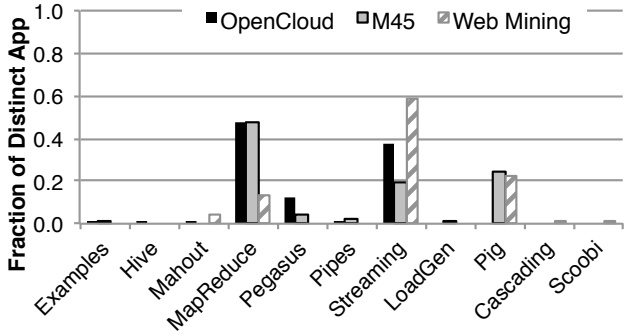


Figure 4: *Fraction of distinct applications per type.*

transformation pipelines. We ask the following key question: **Is the batch-oriented nature of Hadoop well suited to the way users seek to interact with their data?**

Method: We first analyze the workload by extracting application signatures. Our goal is to determine the frequency of different applications: Are most applications one-off ad-hoc analytics or rather repetitive tasks?

The application signature consists of the pair of names of map and reduce functions. We extract the names from the job configuration files and use different extraction mechanisms for each application type. For example, for a native MapReduce application, the Java class name is sufficient for a name. For a streaming application, we parse the command line and extract the name of the executable file or the name of the script. By only focusing on the function names, we can detect a MapReduce application that runs multiple times and processes different datasets in the same way. We do not know whether the functions have changed or not given our limited information (configuration and execution history). We assume that the same application signature is meant to process an input dataset in a certain way.

Results: Figure 4 shows the fraction of distinct applications submitted to the three clusters for each application type. Comparing with Figure 1, the changes in relative heights are striking. In all three clusters, streaming jobs form a large fraction of all distinct jobs (20% to 60%) while they result in a small fraction of actual job executions, especially for M45 and OpenCloud (5% and 10%). In contrast, MapReduce jobs written in native Java API also comprise many distinct jobs but are comparatively more repetitive. Manual inspection of a significant fraction of streaming jobs revealed that most were used for ad-hoc analytics, which are less repetitive. Interestingly, even Pig jobs are repetitive while one would expect Pig Latin to be the tool of

Cluster	1 Dataset	2 Datasets	$\geq$ 3 Datasets
OpenCloud	50%	20%	30%
M45	89%	3%	8%
Web Mining	87%	8%	5%

Table 3: *Fraction of distinct applications that were executed with different numbers of datasets.*

Cluster	1 Dataset	2 Datasets	$\geq$ 3 Datasets
OpenCloud	4%	6%	90%
M45	26%	9%	65%
Web Mining	30%	30%	40%

Table 4: *Fraction of job submissions for applications that were executed with different numbers of datasets.*

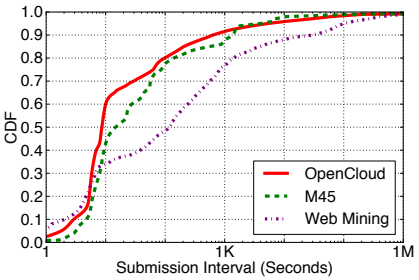
choice for exploratory analytics.

To further determine the fraction of repetitive tasks and one-off ad-hoc queries, we also examine, for each application, how many datasets it processed. Table 3 shows the fraction of distinct applications with different numbers of input datasets (distinct applications are differentiated by their application signatures as defined in the last section). We find that 50% to 89% of applications were only executed with one input dataset. We label those applications as one-off ad-hoc queries (or one-off ad-hoc data transformations). Interestingly, we find that a large fraction of these ad-hoc queries are written using the streaming interface: 58% of applications in OpenCloud, 91% in M45, 88% in Web Mining. Table 4 shows the fraction in terms of the number of job submissions. Although ad-hoc queries form a large fraction of all distinct applications, most of the actual job executions are due to repetitive tasks, especially in the OpenCloud and M45 clusters.

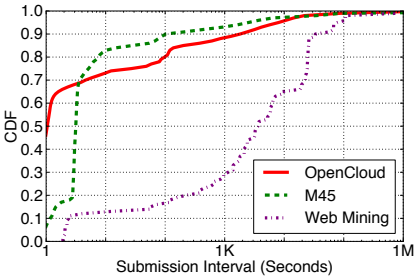
Finding: The above results show that the bulk of the use of Hadoop is indeed in repetitive transformations. However, there is also a large fraction of exploratory analytics. The implications of this result is that Hadoop schedulers must be ready to handle both types of workloads: a small number of exploratory (ideally interactive) analytics and a bulk of repetitive, batch-oriented jobs. For users performing exploratory analytics, native Hadoop Java interface may not be the ideal tool, since the cost of asking each question is high if each question is to be asked only once or only a small number of times. Such high development costs may partly explain the prevailing usage of the streaming interface with which users can rapidly implement map and reduce functions using a scripting language (*e.g.*, Python) or their favorite language.

Method: To better understand whether users run batch or interactive data transformations, we plot the distribution of the submission interval between separate pipelines and between jobs in the same pipeline. For two successive job pipelines A and B from the same user, we measure *submission time of* B $-$ *finishing time of* A. Within a pipeline, we measure the submission interval of a job A_i as *submission time of* A_i $-$ *finish time of* A_{i-1}, where A_{i-1} is the job immediately preceding A_i.

Results: Figure 5(a) shows that 35% to 60% of pipelines are submitted within 10 seconds of an earlier pipeline finishing. We posit that a user has no time to examine results within a 10 second threshold. The pipelines thus correspond most likely to independent and batch executions. At the other end of the spectrum, 10% to 20% of pipelines are submitted at least 15 min apart. These are either independent



(a) Between Pipelines



(b) Within Pipelines

Figure 5: *Distribution of the submission interval between successive pipelines for the same user, and submission interval between jobs within a pipeline. Jobs that share a data dependency (as defined in Section 3.2) are considered to be in the same pipeline.*

or a user needed a significant amount of time to look at the results between executions. Both types of workloads are good fits for the Hadoop batch-oriented scheduler. This leaves, however, 30% to 50% of pipelines that are executed between 10 seconds and 15 min one after the other. This pattern implies that a user was waiting for the results and took the time to reflect on the execution of one pipeline before executing another pipeline. This latter type of use may be better supported by more interactive data analysis engines.

Figure 5(b) shows the submission interval between jobs *within* a pipeline. In OpenCloud and M45, about 10% and 30% of jobs within a pipeline have a submission interval of less than 1 second. Those jobs are likely being managed by scripts that automatically execute the workflows. 10% to as much as 75% are submitted within 10 sec of each other. While these jobs may not be managed by a script, we posit again that a user does not have enough time to study the results. These jobs are submitted manually but the submissions are rather batch-oriented. At the other extreme, 15% to 80% of the jobs are submitted after at least a 15 min delay. This leaves only approximately 10% of the jobs submitted between 10 sec and 15 min, which is much lower than the same threshold for entire pipelines.

Finding: room for improvement Batches of jobs are well supported by Hadoop and our results indicate that a large fraction of jobs within pipelines and entire pipelines are executed in a non-interactive batch-oriented fashion, well-suited for Hadoop. There appears to be room for improvement in terms of helping users manage their jobs and pipelines since a large fraction of both jobs and pipelines are launched one after the other in a manner that appears to imply manual submissions that could perhaps be auto-

mated (whenever jobs or pipelines are submitted within a few tens of seconds of each other). There also appears, however, to be some need for more interactive analytics, which are not well supported by Hadoop today: a large fraction of pipelines are submitted quickly one after the other but with enough delay to let a user reflect on an earlier pipeline execution.

4. TUNINGS

A user can customize a Hadoop MapReduce job by supplying additional functions besides map and reduce. Additionally, there are many configuration parameters in Hadoop that are related to performance and fault-tolerance. In this section, we study how users are exploiting these two features, and whether these features are effective in practice. We ask three key questions: **When do users customize the execution of their jobs? How do users change the configuration parameters? How frequently do they make these changes?** We ask these questions in the context of basic job customizations (Section 4.1), customizations for the purpose of improving load balance (Section 4.2), and tuning configuration parameters (Section 4.3).

4.1 Job Customization

Most job customizations are related to ways of partitioning data and aggregating data from earlier stages. The first question that we ask is **When do users customize the execution of their Hadoop jobs?**

Method: In each cluster, we count the number of users who performed each type of customization at least once. Each customization requires that the user supply an extra *function* in addition to map and reduce.

Results: We consider each type of tuning separately.

Combiner: The Combiner performs partial aggregation during the local sort phase in a map task and a reduce task. In general, if the application semantics support it, a combiner is recommended. In OpenCloud, 62% of users have used this optimization at least once. In M45 and Web Mining clusters, 43% and 80% of users have used it respectively.

Secondary Sort: This function is applied during the reduce phase. By grouping different reduce keys for a single reduce call, secondary sort allows users to implement an optimized join algorithm as well as other complex operations. In the M45 cluster, no user applied a secondary sort. In the Web Mining cluster, only one user used secondary sort, and that was through the use of Pig, which implements certain join algorithms using secondary sort. In OpenCloud, 14% of users have used secondary sort, perhaps suggesting a higher level of sophistication or more stringent performance requirements.

Custom Partitioner: A user can also have full control over how to redistribute the map output to the reduce tasks using a custom partitioner. In the OpenCloud cluster, as many as 35% of users have used a custom partitioner. However, only two users in the M45 cluster and one user in the Web Mining cluster applied a custom partitioner.

Custom Input and Output Format: Hadoop provides an InputFormat and OutputFormat framework to simplify handling of custom data formats and non-native storage systems. In OpenCloud, 27% of users applied a custom input format at least once and 10% of users applied a custom output format. In M45, only 4 users applied a custom input format and only 1 user applied a custom output format. In Web Mining, only one user applied a custom input format and none applied a custom output format.

Finding: good use and room for improvement In general, job customizations help with performance (e.g. combiners) and correctness (e.g. input/output format), thus a visible fraction of users leverage them, especially the optional combiners. OpenCloud users tend to use more optimization techniques than users of the other two clusters, which implies that some additional user education could help a greater number of users get high performance from their Hadoop clusters.

4.2 Customization for Load Balancing

This section extends the previous analysis of job customizations (*e.g.*, custom input format and custom partitioner) to study its effectiveness in balancing load across map or reduce tasks and reducing the impact of stragglers, which are tasks that take significantly longer to complete than other tasks in the same phase. We ask two key questions: **Are load imbalances a significant problem and are users pro-active in mitigating these problems by tuning their jobs?**

Method: There are many causes of stragglers but one of the causes is *data skew*: some tasks take longer simply because they have been allocated more data. Ideally, if we assign the same amount of input data to tasks, then all tasks should take the same time to execute. We evaluate how close the workloads are to this ideal case by analyzing the relationship between input data size and task runtime.

For each phase of each job, we compute the ratio of the maximum task runtime in the phase to the average task runtime in that phase. We classify phases where this ratio is greater than 0.5 (meaning that at least one straggler took twice as long to process its data as the average) as *Unbalanced in Time (UT)*. Otherwise, the phase is said to be *Balanced in Time (BT)*. We compute the same ratio for the input data and classify phases as either balanced or unbalanced in their data (BD or UD).

Results: We look at load imbalance problems separately in the map and reduce phases of jobs in the three clusters.

Map Input Format: Figure 6 shows the result for the map phases. We break the results into four different types of applications based on whether the input data and/or the runtime are balanced or not (*i.e.*, (U)BD(U)BT as defined in the previous paragraph).

As expected, for the map phase, most jobs are balanced with respect to data allocated to tasks. However, a significant fraction of jobs (*e.g.*, more than 20% for all but Mahout in the OpenCloud cluster) remain unbalanced in runtime and are categorized as BDUT. These results agree with the result of the previous study in enterprise clusters [10]. Mahout is a machine learning algorithm package built on top of Hadoop [5]. Mahout is effective at reducing skew in the map phase, clearly demonstrating the advantage of specialized implementations. Interestingly, Pig produces unbalanced map phases similar to users' custom implementations. Overall, allocating data to compute nodes based on data size alone is insufficient to eliminate skew.

In Hadoop, the InputFormat mechanism is responsible for generating InputSplits which describe the input data processed by a map task. In our traces, fewer than 1% of jobs attempt to optimize the partitioning of the map input by using a custom input format. Additionally, users can man-

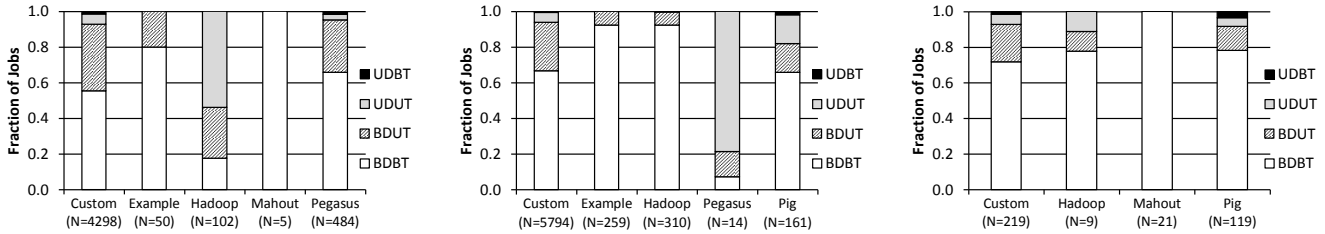


Figure 6: *Distribution of map runtime with respect to # of input records per application. N is the number of successful jobs that run map functions in each category. The labels indicate the category: (U)BD(U)BT stands for (un)balanced input data, (un)balanced runtime.*

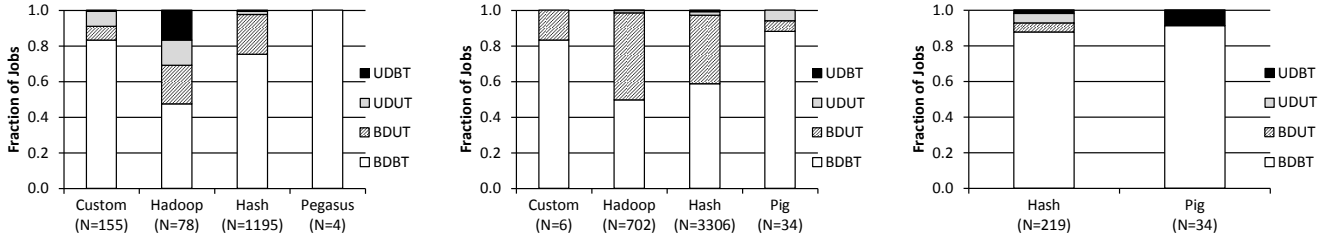


Figure 7: *Distribution of reduce runtime with respect to # of reduce keys per partition function. The labels indicate the category: (U)BD(U)BT stands for (un)balanced input data, (un)balanced runtime.*

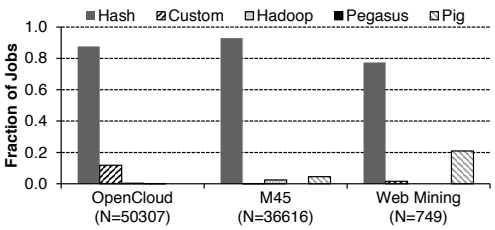


Figure 8: *Fraction of jobs per partition function. Hash partitioning is dominant in all three clusters. The fraction of jobs using customized partition function is relatively small, less than 10% in all clusters.*

ually specify the number of bytes per split if the input is stored in HDFS. In our traces, 12% of all jobs from 10 users in OpenCloud used this optimization. In M45 and Web Mining, fewer than 1% of all jobs from 1 and 3 users used the optimization, respectively. It is clear that users only rarely exploit these opportunities for optimizing the data allocation.

Reduce Key: We perform a similar analysis for the reduce phase using the number of reduce keys as the input measure. Instead of application types, we group the jobs by the partition function employed. The partition function is responsible for redistributing the reduce keys among the reduce tasks. Again, we find that users rely primarily on the default hash-partition function (Figure 8) rather than manually trying to optimize the data allocation to reducers.

Figure 7 shows the analysis per partition function[2]. Overall, Hash partitioning effectively redistributes reduce keys among reduce tasks for more than 92% of jobs in all clusters (*i.e.*, BDBT+BDUT). Interestingly, we observed that as many as 5% of all jobs in all three clusters experienced the *empty reduce* problem, where a job has reduce tasks that processed *no* data at all due to either a suboptimal partition function or because there were more reduce tasks than reduce keys (We observed the latter condition in 1% to 3% of all jobs).

For the jobs with a balanced data distribution, the runtime was still unbalanced (*i.e.*, BDUT jobs) for 22% and 38% of jobs in the OpenCloud and M45 clusters, respectively. In both the OpenCloud and M45 clusters, custom data partitioning is more effective than the default scheme in terms of balancing both the keys and the computation. Other partitioning schemes come with Hadoop distribution do not outperform hash partitioning in terms of balancing data and runtime. A noticeable portion of UDBT jobs (*i.e.*, data is not balanced but runtime is balanced) use the Total Order Partitioner, which tries to balance the keys in terms of the number of values. Pig, which uses multiple partitioners, consistently performs well in both M45 and Web Mining clusters.

Findings: room for improvement In summary, given (a) the prevalence of load imbalance problems, (b) the evidence that achieving load balance requires more than uniform data distribution, and (c) users' reluctance to use the manual tuning features provided by Hadoop to solve load imbalance problem, we recommend pursuing techniques that automatically reduce skew to achieve better performance [19, 27].

[2]In the M45 workload, the number of reduce keys was not recorded for the jobs that use the new MapReduce API due to a bug in the Hadoop API. The affected jobs are not included in the figure.

4.3 Configuration Tuning

Hadoop exposes a variety of configuration parameters for tuning performance and reliability. Here we discuss a few configuration parameters that are typically considered important for performance and fault-tolerance [12, 14]. We ask the same question as above, which is **How frequently do users change the configuration of their jobs to improve performance and reliability?**

Method: In each cluster, we count the number of users who performed each type of tuning at least once.

Results: We consider each type of tuning separately.

Failure Parameters: Users can control how failures are handled as well as erroneous inputs. In OpenCloud, 7 users explicitly specified a higher threshold to retry failed tasks, 6 users specified a higher "skip" to ignore bad input records, and 1 user specified a higher threshold in the number of tolerable failed tasks. In M45, 3 users set a higher threshold in the number of tolerable failed tasks. All Web Mining users stayed with cluster default values.

Java Virtual Machine (JVM) Option: The native Hadoop MapReduce interface is implemented in Java. If a map or reduce task requires a large memory footprint, the programmer must manually adjust the heap and stack sizes: 29 OpenCloud users, 11 M45 users and 3 Web Mining cluster users have changed default JVM option for their jobs.

Speculative Execution: Speculative execution is the default mechanism to handle straggler tasks. Only two users from OpenCloud and M45 have changed the cluster default value for their applications. By default, speculative execution is enabled in all three clusters. However, speculative execution was successful at most 21% of the time in improving the task completion time. Furthermore, map speculations did not run significantly faster than the original attempt (speculated too early). In OpenCloud and M45, many reduce speculations were late: they ran for less than 10% of the original task time before being killed.

Sort Parameters: Hadoop runs a merge sort at the end of the map phase and just before the reduce phase. There are four parameters that directly relate to those sorts. Two users of the Web Mining cluster adjusted `io.sort.mb` parameter to 200. Only one user of the M45 cluster adjusted `io.sort.mb` to 10. Other than that, all users used the cluster default values.

HDFS Parameters: The HDFS block size and replication factor affect how the final output of a MapReduce job is written. In OpenCloud, 11 users tried different values for the replication factor. In M45, two users adjusted the block size and only one user tried a different replication factor. Other than these, all users kept the cluster default values.

Figure 9 shows the percentiles and average number of configuration changes per-user each month. The trend is stable over time, but there are always users who tune many configuration parameters. By examining Table 5 in Section 5, we find that long-term (frequent) users know to tune more configuration parameters than less experienced users.

Findings: good use and room for improvement In summary, users tend to tune configuration parameters directly related to failures. By talking with administrators of OpenCloud, we learned that many of their users explicitly tuned these options in response to poor failure behaviors such as "out of memory" error. In contrast, users rarely tune parameters related to performance, perhaps because their performance requirements were generally being met, or perhaps because these parameters are more difficult to understand and manipulate.

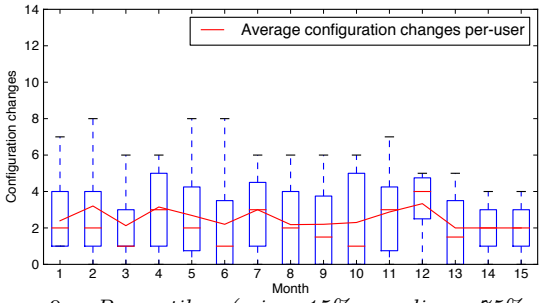


Figure 9: *Percentiles (min, 15%, median, 75%, max) and average number of configuration changes per-user each month.*

5. RESOURCE USAGE AND SHARING

The third part of our study focuses on how well Hadoop enables users to leverage their cluster resources.

We first start by looking at the magnitude of jobs executed in the various clusters in terms of the amount of data they need to process and the duration of their execution (Section 5.1). Hadoop is designed to process long-running jobs (since it has a high start-up overhead) over large input data files (since it runs across sharded datasets). We ask the following question: **Do users leverage Hadoop for long-duration jobs over large datasets?** Second, we look beyond single jobs and study how users leverage their clusters in aggregate over longer periods of time. We ask two key questions: **Do individual users consume large amounts of resources in aggregate? What are the resource consumption profiles of different types of users?** Finally, we focus on *data* reuse. We ask two questions **How frequently are the same input data files processed and how much sharing occurs between users? What is the lifespan of the result files?**

5.1 Big Jobs and Small Jobs

We first examine the distribution of I/O and runtime for jobs executed in the three Hadoop clusters. We ask the following key question: **Do users leverage Hadoop for long-duration jobs over large datasets?** We answer this question by looking at the distributions of I/O and runtimes within each month of each trace, which gives us both the data for each point in time but also the evolution of usage over time.

Method: We compute various percentiles of job durations and aggregate I/O size of jobs submitted during the data collection period. Figures 10 and 11 show the statistics collected from the OpenCloud cluster (M45 and Web Mining show similar trends, and therefore are omitted for brevity). Figure 10 shows the percentiles of job durations and Figure 11 shows the percentiles of aggregate I/O size by summing the amount of data processed during the map (input), shuffle and reduce (output) phases.

Results: As both figures show, small jobs dominate the workload in the OpenCloud cluster (workloads in the other two clusters are similar). For most months, more than 50% of jobs touch less than 10 GB of data, and run for less than 8

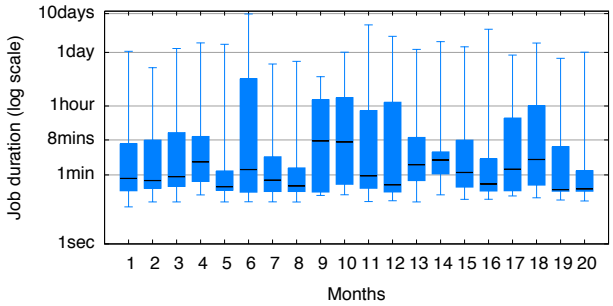


Figure 10: *Percentiles (min, 10%, median, 90%, max) of job durations in the OpenCloud cluster for each month.*

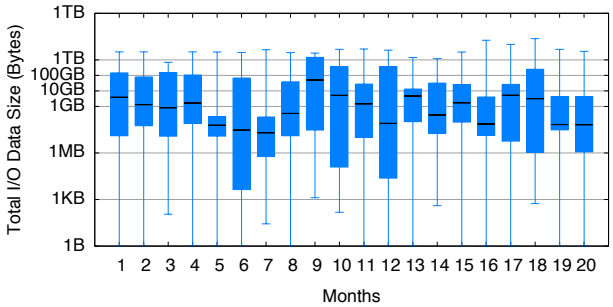


Figure 11: *Percentiles (min, 10%, median, 90%, max) of aggregate I/O activity (input+shuffle+output data size) of OpenCloud jobs each month.*

minutes. But outliers exist during the whole data collection period. At least 1% of jobs during most months process more than 100 GB and run longer than 5 hours. OpenCloud users indicated that the small jobs come from two sources. First, some small jobs are intrinsic to the application implementation and dataset. For example, in many iterative applications, each iteration is a short MapReduce job. As another example, some users use Hadoop simply as a scheduler to launch parallel short tasks that take very small datasets as input (*e.g.*, short video clips and executable binary files). Second, some short jobs are for debugging purposes (*e.g.*, finding bugs in programs and validating algorithms on small-scale datasets). We run a clustering algorithm to classify debugging and non-debugging jobs in the logs, by assuming that debugging jobs of the same application always have smaller I/O sizes, and are submitted before non-debugging jobs. We first group jobs by application signatures and submission times. For jobs having the same signature and submitted within the same time interval, we use k-means to cluster them into two groups based on I/O sizes. If the average I/O size of one group is 10× larger than the other, the first group is classified as debugging jobs. We find that approximately 15% of jobs in the OpenCloud cluster are debugging jobs (including failed and killed jobs). Workloads are also diverse across time: the distribution of job duration and aggregate I/O size changes significantly across months. Especially the 90th percentile and the maximum value vary significantly over time. In the other two clusters, small jobs are also prevalent and the workload also has high variation over time. The difference is in the median value of aggregate I/O size, which is less than 1 GB in M45, and between 10 GB and 100 GB in Web Mining.

Finding: good use and unexpected use: The key

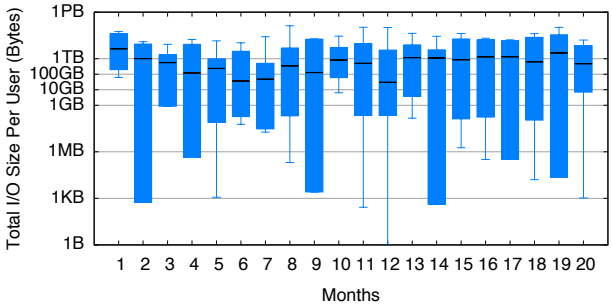


Figure 12: *Percentiles (min, 10%, median, 90%, max) of users' aggregate I/O activity (the sum of data bytes processed by all jobs of each user) over time in the OpenCloud cluster.*

implication of the above results is that users do leverage Hadoop to execute long-duration jobs that process massive-scale datasets (100's of GB to a few TB). However, users also try to use their Hadoop hammer to process short jobs over small datasets. In fact, short jobs over small datasets dominate the workload even when Hadoop is known to be a suboptimal tool for such workloads. Interestingly, some Web Mining users indicated that they often break their large and long-running jobs into smaller ones as a way to checkpoint intermediate results and ensure that less work is lost when an entire job fails.

5.2 Aggregate Resource Usage Profiles

Next, we examine the aggregate resource consumption per user. From the previous section, we know that users execute primarily short jobs over small datasets. The questions that we ask next are **Do individual users consume large amounts of resources in aggregate? What are the resource consumption profiles of different types of users?**

Method: We first compute three metrics of aggregate resource consumption for each user: *task time*, *data*, and *jobs*. The *task time* metric is the sum of all map and reduce task durations across all of a user's jobs. The *data* metric is the sum of the bytes processed across the *input*, *shuffle*, and *output* phases. The *jobs* metric is simply the number of total jobs submitted. We plot the distribution of these metrics in aggregate and also for each month of the trace.

To further characterize the patterns of user behaviors, we classify users by how heavily they use the cluster. For this, we use the K-means [20] clustering algorithm. Each user is represented as a vector of four features including *the number of submitted jobs, the number of active days, the total bytes of data processed by the user's jobs*, and *the total task time consumed by the user's jobs*. An *active day* is a day when the user submitted at least one job. Each dimension is normalized to the interval $[0, 1]$ by dividing all values by the maximum value across all jobs. The K-means clustering algorithm then groups together the vectors that are close to each other in the four-dimensional space.

Results: Figure 12 shows the percentiles of users' aggregated I/O activity for each month. Overall, users process significant amounts of data each month in spite of the fact that most of their jobs are short-lived. As the distributions show, the median user processes hundreds of GB of data to nearly 1 TB of data in a month. The distributions also show drastic differences between the top user and the bot-

tom users at 1,000X to 100,000X difference in total amount of data processed for most months, which indicates that different users do indeed have different resource consumption profiles. There is no visible trend over time. While the distributions change from one month to the next, there is no trend toward increased resource consumption over timescales of a few months.

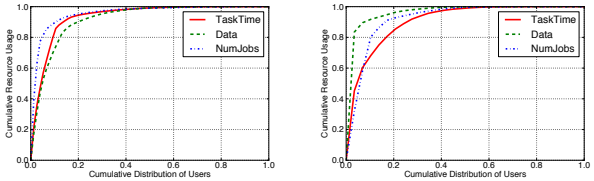
To characterize the differences in resources consumptions between users and start to identify different types of users, we plot the entire cumulative distributions of the three resource-consumption metrics (jobs, data, and time) over all users in two clusters OpenCloud and M45. We omit the Web Mining cluster since it has only 5 users. Figure 13 shows the results. As with many other human activities, the distribution of resource consumption is a long-tail distribution. All three metrics indicate that about 80% to 90% of resources are consumed by 20% of the users in both clusters. Hence, we can at least identify two types of users: the small fractions of heavy resource consumers and the majority of users who are light-weight resource consumers.

To further categorize users, Table 5 summarizes the results of the K-means clustering. The table shows the centroid of each cluster (the feature vector for the user at the center of each cluster), the size of each user group and also a list of statistics about each user group. The list of statistics includes the average number of distinct applications/types/structures used by each user, the median duration/data size of all jobs submitted by each group, and also the average number of configuration and customization changes (See details in Section 4) performed by each user.

Consistent with the CDFs from Figure 13, most users are clustered into one group since they use the cluster only lightly. The remaining frequent users are classified into several small groups. For example, in OpenCloud, there are four groups of frequent users. Two groups (G4 and G5) have the maximal centroid value in one of the dimensions, and the other two groups (G2 and G3) consume medium-level cluster resources. We notice that G5 with only two users submitted most of the small jobs (their median job duration is less than one minute). In contrast, the median job duration and I/O size of jobs submitted by users in G1 are significantly larger than that of frequent users. This behavior also happens in the other two clusters. Another observation is that frequent users tend to submit jobs with more complicated types and structures than infrequent users, and utilize significantly more customizations and configuration parameters.

Finding: Good use and room for improvement Overall, users thus process significant amounts of data in aggregate in their Hadoop clusters, but with highly uneven resource consumption between users. On the more negative side, large numbers of users do not leverage their Hadoop clusters much, which implies room for improvement in terms of getting these users to exploit the available resources. By clustering users into groups, we also found that some users have extreme behaviors (*e.g.*, submitting many small jobs). These behaviors might not cause problems to other users in under-utilized clusters, but definitely waste resources in busy clusters if the schedulers are not designed to tolerate such use cases.

5.3 Data Sharing and Reuse



(a) OpenCloud (b) M45

Figure 13: *Aggregated resource usage over users under three metrics. 20% of users consume about 90% of the resources.*

We finally focus on the data being processed in the Hadoop clusters and ask two key questions **How frequently are the same input data files processed and how much sharing occurs between users? What is the lifespan of the result files?** We study the implications of design enhancements such as caching and management of short-lived files.

Method: As in Section 3.2, this analysis assumes that input and output paths are not renamed nor manually modified by users. We compute the frequency with which various input data paths are accessed and output data paths are over-written.

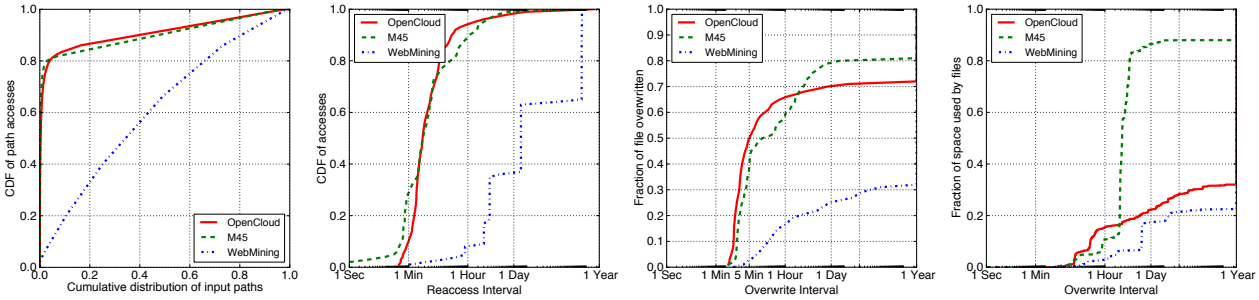
Results: Figure 14 (a) gives the distribution of access frequency of all paths in the three workloads. The output paths with the same path name are considered as different datasets when they are generated by different MapReduce jobs. For OpenCloud and M45, fewer than 10 percent of paths are responsible for more than 80% of all accesses. Further analysis in Figure 14 (b) indicates the 90% of data re-accesses happen within 1 hour in these two clusters. By examining the input and output file paths recorded in the configuration files of all MapReduce jobs, we found that only 1% datasets are shared across users. Therefore, the re-accesses shown in Figure 14 are due to users repeatedly processing the same data. However, the small fractions of "hot" datasets and their frequent and rapid re-accesses indicate that caching may help reduce I/O overheads.

Therefore a cache policy can possibly bring considerable benefit, which has also been observed in prior work on industry workloads [10, 3]. To analyze the effect of a global cache, we run a simulation to calculate the cache hit ratio. The simulation assumes that every node in the cluster uses a portion of its memory (with size α) to cache input and output data of MapReduce jobs. Thus the total size of the global cache is $\alpha \times$ *the number of nodes in the cluster*. The simulation only considers the input and output data as a whole instead of caching data at the block level, as suggested in previous work [3]. That means the global cache only keeps and evicts the entire input or output data of a MapReduce job. The cache policy we use is Least Frequently Used (LFU) with a sliding window. Items that are not re-accessed within a 1 hour window are evicted. Table 6 shows the global cache hit ratio in the three workloads, when tuning the cache size α at each node. For OpenCloud and M45 workloads, since small files are prevalent, their cache hit ratios are higher than 69% even when using only 512 MB memory at each node. Caching can thus effectively reduce disk I/O requests wasted by accesses to small files.

We also analyze how quickly a job's output data is overwritten by successive jobs. Figure 14 (c) shows the distribution of time intervals between data overwrites. For Open-

Cluster	Grp. Num.	Centroid of Feature Vector: #Jobs	Active Days	Total I/O (GB)	Total Task·Hour	Statistics of User Cluster: #App (avg.)	#Type (avg.)	#Struct (avg.)	Median Duration	Median I/O Size	#Config. &Custom.	#User
Open Cloud	G1	40	6	556	2653	5	1	1	26 mins	54 GB	5	59
	G2	373	30	3482	29716	31	2	4	5 mins	12 GB	12	8
	G3	2593	66	40907	59320	24	2	2	3 mins	4 GB	11	4
	G4	1105	89	79376	221486	38	2	4	3 mins	15 GB	11	3
	G5	16423	72	21465	103425	46	2	4	0.6 min	0.1 GB	9	2
M45	G1	325	10	8498	9998	30	1	2	111 mins	112 GB	8	26
	G2	11538	55	4281	64	10	2	2	2 mins	0.1 GB	23	2
	G3	11287	38	189340	1364524	40	2	6	0.6 min	2 GB	16	1
Web Mining	G1	140	19	14398	22818	8	2	2	65 mins	44 GB	11	3
	G2	388	47	3747	17759	118	3	5	3 mins	7 GB	10	1

Table 5: *Clustering users in each workload using K-means. Each user is a vector with four dimensions: the number of jobs, the number of active days, total data size processed by all jobs, and the total task time. Active day is the day when user submitted jobs. Total task time denotes the total running time of all tasks of all jobs.*



(a) Accesses (b) Re-access (c) Overwrite by file (d) Overwrite by space

Figure 14: *Distribution of access and overwrite patterns. (a) shows the distribution of access frequency of data paths; (b) shows the distribution of re-access frequency of input and output data. (c) shows the distribution of time intervals between output data with the same data paths; (d) shows the distribution of space used by overwritten data within different time intervals.*

CachePerNode (α)	OpenCloud	M45	Web Mining
512 MB	68%	78%	11%
1 GB	69%	81%	18%
2 GB	71%	84%	20%
4 GB	72%	86%	25%
8 GB	73%	88%	28%
16 GB	74%	89%	39%

Table 6: *Percentage of jobs whose entire inputs hit cache under different cache size α per node.*

Cloud and M45, 50% of jobs have their output data overwritten within 5 minutes, and 90% of jobs' output data are overwritten within 1 hour. Figure 14 (d) shows the space used by overwritten data within various time intervals. For OpenCloud, most overwritten output data are small files which only occupy a small portion of the storage space. For M45, large output data files are also overwritten within 1 hour. These results suggest that many reduce output files (especially files with small sizes) have a life span of less than an hour. Additionally, shuffle data generated at the map output stage are also short-lived objects in the local storage system, and they compose a considerable portion of disk traffic.

Findings: room for improvement These observations suggest that there is potential room to improve cluster performance through simple data caching policies. Additionally, space allocation and reclamation algorithms in the local filesystems can also be improved by explicitly grouping short-lived files. So short-lived files can be cleaned together without causing fragmentations on storage devices, and this policy would be helpful for modern storage devices (such as flash device and shingled disks) which have high performance penalties with bad space reclamation policy.

6. FUTURE WORK

In this paper, we examine how more than 100 data scientists from a variety of domains and in three academic clusters use their Hadoop systems. Looking at the more than 100,000 jobs executed by these users, we identify important requirements for the next generation Big Data systems that want to go beyond what Hadoop offers today.

First, and most important, we find that Big Data analysis comes in a large variety of forms. A good fraction of applications follow the single-job, large input, long-duration tasks form for which MapReduce was designed. The needs, however, extend significantly beyond this pattern: many jobs are iterative with short-duration inner-loops, some applications require complex combinations of jobs to produce a desired result, others yet are embarrassingly parallel and can be spread almost arbitrarily across cluster resources. The next-generation Big-Data systems must thus cater to this extensive variety, rather than optimize for a single type of applications as some systems do today (*e.g.*, [21]).

Second, debugging and tuning Big-Data analysis applications is a great challenge. While there exist important research from automatic tuning [7] to correctness debugging [23] and even performance debugging [18], these tools still need to make their way into widespread use. An interesting observation is that users are willing to work with the engine to overcome performance challenges and avoid

failures, but the complexity of the settings is overwhelming and possibly not the most effective interface to seek input from users about their applications. Informative monitoring tools and novel interactive debugging tools are urgently needed by Hadoop users.

Finally, our analysis points to several possible optimizations to current Hadoop systems. One optimization is to use an in-memory cache to speed up repetitive tasks and iterative applications, since their data sets typically fit in memory. Another possible optimization is to improve the disk block allocation and defragmentation algorithm of the local filesystems for Hadoop workloads, by explicitly grouping small short-live files to reduce fragmentation.

7. RELATED WORK

The closest related work are the studies of the Facebook and Cloudera cluster traces by Chen et al [10, 9]. Their studies focus on cluster-level and job-level metrics, as well as the usage of SQL-like programmatic frameworks. They also find that a wide range of diversity exists in their workloads, and small jobs and temporal locality in data accesses are common. In contrast to our workloads, SQL-like programming frameworks are popular. Kavuly *et al.* [16] also study logs from the M45 cluster, but they focus on cluster-level workload characterization and performance prediction.

Work on optimizing MapReduce and Hadoop is also relevant. PACMan [3] studies the effects of different global cache policies for HDFS and finds that in some production cluster traces, the majority (96%) of active jobs can fit into a fraction of the cluster's total memory. DiskReduce [13] analyzes file systems workloads of several Hadoop and high-performance computing clusters. They find that small files are prevalent in large clusters. Ke *et al.* [17] find that load imbalance problems are prevalent in a variety of industrial applications. Mantri [2] studies the causes of outlier tasks and ways of mitigating outlier tasks caused by data imbalance or resource contentions. SkewTune [19] proposes run-time repartition to mitigate skew problems. Starfish [14] investigates automatic tuning configurations for Hadoop MapReduce programs. Our analysis shows users are typically only tuning configuration parameters for failures, suggesting that automatic tuning is an important feature.

8. CONCLUSION

We studied workloads from three Hadoop clusters used for academic research. These new Hadoop cluster traces contain richer information than previous studies by recording user behaviors and application specifications, both critical for understanding requirements of Big-Data systems. In these workloads, we find that applications are highly diverse in styles, structures, and performance characteristics and that higher-level tools (*e.g.*, Pig) do not satisfy users' needs. We also find that users tend to optimize their applications at the algorithmic level by using Hadoop customizations, and only tune configuration parameters related to failures. Default configurations result in some performance degradations (*e.g.*, load imbalance). Our conclusion is that the use of Hadoop for academic research is still in its adolescence. Easing that use, especially for sophisticated applications, and improving the system to tolerate workload diversity and facilitate tuning are important future research directions.

9. ACKNOWLEDGMENTS

We thank N. Balasubramanian and M. Schmitz for helpful comments and discussions. We also thank the owners of the logs from the three Hadoop clusters for graciously sharing these logs with us. This research is supported in part by The Gordon and Betty Moore Foundation, National Science Foundation under awards, SCI-0430781, CCF-1019104. Qatar National Research Foundation 09-1116-1-172, DOE/-Los Alamos National Laboratory, under contract number DE-AC52- 06NA25396/161465-1, by Intel as part of ISTC-CC. We thank the member companies of the Parallel Data Lab Consortium for their feedback and support.

10. REFERENCES

[1] Yahoo! reaches for the stars with M45 supercomputing project. `http://research.yahoo.com/node/1884`.
[2] G. Ananthanarayanan et al. Reining in the outliers in Map-Reduce clusters using Mantri. In *OSDI*, 2010.
[3] G. Ananthanarayanan et al. PACMan: Coordinated memory caching for parallel jobs. In *NSDI*, 2012.
[4] Apache Foundation. Hadoop. `http://hadoop.apache.org/`.
[5] Apache Foundation. Mahout: Scalable machine learning and data mining. `http://mahout.apache.org/`.
[6] Ashish Thusoo et. al. Hive: a petabyte scale data warehouse using Hadoop. In *ICDE*, pages 996–1005, 2010.
[7] S. Babu. Towards automatic optimization of mapreduce programs. In *SoCC*, pages 137–142, 2010.
[8] D. Borthakur. The Hadoop distributed file system: Architecture and design. `http://lucene.apache.org/hadoop/hdfs_design.pdf`, 2007.
[9] Y. Chen et al. The case for evaluating MapReduce performance using workload suites. In *MASCOTS*, pages 390–399.
[10] Y. Chen et al. Interactive query processing in big data systems: A cross-industry study of MapReduce workloads. *PVLDB*, 5(12):1802–1813, 2012.
[11] Concurrent, Inc. Cascading. `http://www.cascading.org/`, 2012.
[12] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, 2004.
[13] B. Fan et al. DiskReduce: RAID for data-intensive scalable computing. Technical Report CMU-PDL-11-112, PDL, Carnegie Mellon University, 2011.
[14] H. Herodotou and S. Babu. Profiling, what-if analysis, and cost-based optimization of MapReduce programs. *PVLDB*, 4(11):1111–1122, 2011.
[15] U. Kang et al. PEGASUS: A peta-scale graph mining system implementation and observations. In *ICDM*, pages 229–238, 2009.
[16] S. Kavulya et al. An analysis of traces from a production MapReduce cluster. In *CCGRID*, pages 94–103, 2010.
[17] Q. Ke et al. Optimizing data partitioning for data-parallel computing. In *HotOS*, 2011.
[18] N. Khoussainova et al. Perfxplain: Debugging mapreduce job performance. *PVLDB*, 5(7):598–609, 2012.
[19] Y. Kwon et al. SkewTune: mitigating skew in mapreduce applications. In *SIGMOD*, pages 25–36, 2012.
[20] S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.
[21] G. Malewicz et al. Pregel: a system for large-scale graph processing. In *SIGMOD*, pages 135–146, 2010.
[22] C. Olston et al. Pig Latin: a not-so-foreign language for data processing. In *SIGMOD*, pages 1099–1110, 2008.
[23] C. Olston et al. Generating example data for dataflow programs. In *SIGMOD*, pages 245–256, 2009.
[24] A. Pavlo et al. A comparison of approaches to large-scale data analysis. In *SIGMOD*, pages 165–178, 2009.
[25] Scoobi Team. A Scalar productivity framework for Hadoop. `https://github.com/NICTA/scoobi`, 2012.
[26] B. Sharma et al. Modeling and synthesizing task placement constraints in Google compute clusters. In *SoCC*, pages 3:1–3:14, 2011.
[27] R. Vernica et al. Adaptive MapReduce using situation-aware mappers. In *EDBT*, pages 420–431, 2012.