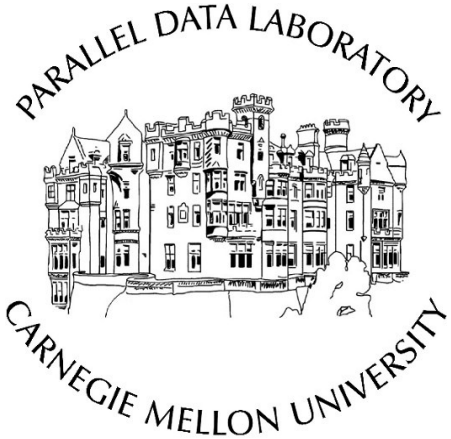


Hadoop's Adolescence: A Comparative Workload Analysis from Three Research Clusters

Kai Ren[1], YongChul Kwon[2], Magdalena Balazinska[2], Bill Howe[2]
[1] *Carnegie Mellon University,* [2] *University of Washington*
(kair@cs.cmu.edu, {yongchul,magda,billhowe}@cs.washington.edu)

CMU-PDL-12-106
June 2012

Parallel Data Laboratory
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

ABSTRACT

We analyze Hadoop workloads from three different research clusters from an application-level perspective, with two goals: (1) explore new issues in application patterns and user behavior and (2) understand key performance challenges related to IO and load balance. Our analysis suggests that Hadoop usage is still in its adolescence. We see underuse of Hadoop features, extensions, and tools as well as significant opportunities for optimization. We see significant diversity in application styles, including some "interactive" workloads, motivating new tools in the ecosystem. We find that some conventional approaches to improving performance are not especially effective and suggest some alternatives. Overall, we find significant opportunity for simplifying the use and optimization of Hadoop, and make recommendations for future research.

Acknowledgements: This research is supported in part by The Gordon and Betty Moore Foundation, the University of Washington eScience Institute, NSF under award, SCI-0430781 and 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 the Intel Science and Technology Center for Cloud Computing (ISTC-CC), by gifts from Yahoo!, APC, EMC, Facebook, Fusion-IO, Google, Hewlett-Packard, Hitachi, Huawei, IBM, Intel, Microsoft, NEC, NetApp, Oracle, Panasas, Riverbed, Samsung, Seagate, STEC, Symantec, and VMware. We thank the member companies of the PDL Consortium for their interest, insights, feedback, and support.

Keywords: Hadoop, Workload Analysis, User Behavior, Storage, Load Balance

1 Introduction

Hadoop [6], the open-source implementation of Google's MapReduce [17], has become enormously popular for big data analytics, especially among researchers. Due to Hadoop's popularity, it is natural to ask the question: How well is it working? To answer this question, we need to move beyond the conventional "cluster-centric" analysis that models each job as an independent black box and focuses on coarse global metrics such as resource utilization, usually in the context of a single cluster [5, 13, 24, 33]. Instead, we need to recognize that not all clusters are used in the same way, and that there is a rich ecosystem of tools, extensions, and libraries related to Hadoop that influence application characteristics, user behavior, and workflow structure.

In this paper, we analyze Hadoop workloads from three different research clusters, where the main users are from academic institutions. We have two goals with this work: To better understand application patterns and user behavior in a Hadoop cluster and to assess key performance properties across a set of research-oriented clusters. To analyze the higher-level, user-oriented issues, we make use of information that is typically too sensitive for commercial Hadoop users to share, including job specifications (*e.g.*, names of Java classes, user identifier, input/output directories) and high-resolution application-level statistics (*i.e.*, HDFS [10] and MapReduce counters). For the performance study, we compare and contrast our findings to those reported in prior work [3, 5, 11, 24]. Our study is broken into four parts: application workload, user behavior, IO performance, and load balance.

Overall, our analysis suggests that Hadoop usage is still in its adolescence. We see a significant number of independent, single-step, small-scale jobs that may be amenable to simpler, non-Hadoop solutions. There is evidence that users are working "interactively," submitting jobs sequentially after a short delay to digest the results. Relatively few are using higher-level declarative frameworks, though some are constructing "manual" workflows. We see significant performance penalties arising from over-reliance on default configuration parameters and default Hadoop customization options. Workloads are highly diverse, suggesting that simple, "one-size-fits-all," application-oblivious optimizations will not be broadly effective. Indeed, speculative execution does not seem to provide much benefit, and conventional scheduling that only tries to improve locality ignores huge opportunities for caching "hot" files. We find that the trend toward centralization of resources is effective: smaller, local clusters are underutilized. Overall, we find that there is enormous room for improvement in raising the level of abstraction for interacting with the cluster: improving interactivity, automatically optimizing complex tasks, and automatically applying application-aware tuning.

More precisely, our study makes the following contributions.

1) First, we comparatively analyze *three* Hadoop cluster workloads. All three correspond to clusters used for *academic research*. Because big-data analytics is an important facet of academic research, studying the use of big-data clusters in this setting is a critical component of understanding the requirements and performance of these systems.

2) Second, we study characteristics of applications that users execute in the three clusters at a greater level of detail than previous Hadoop-cluster [5, 13, 11, 12, 24] (Section 3). Some of the key questions that our study asks is whether applications take the form of complex workflows or one-off jobs. How much data is being moved around by these applications — are they looking for needles in haystacks or searching for high-level trends? Are these applications expressed in higher-level interfaces such as Pig Latin [30] or HiveQL [9], or are they written directly in Java? Surprisingly, we find that many, but not all, applications are small, single step operations implemented directly in Java, but that their data processing characteristics are highly varied.

3) Third, to the best of our knowledge, we are the first to study the user behavior in big-data clusters (Section 4). We study how users interact with the cluster: Are the applications submitted automatically or manually? Do users submit jobs sequentially and "interactively," waiting for the results of one before submitting another? Or do they submit batches of jobs all at once, (*e.g.*, parameter sweeps or monte carlo

simulations)? What is the distribution of resource utilization across users? How much of the work is attributable to "heavy hitters" that could benefit from dedicated resources or other specialized solutions? We find that approximately 10% of all jobs are submitted as part of a batch together with other jobs. Another 30% of jobs are submitted automatically in sequence. This leaves, however, 60% of the jobs where a user waits for one job to complete before firing the next one manually, most often within 2 minutes — users are working interactively. We also find that, when given the option, a visible fraction of users configure their jobs by changing important parameters such as enabling the use of a combiner or changing the HDFS [10] block size. The vast majority of users, however, simply execute their jobs on the out-of-the-box settings of the underlying Hadoop engine.

4) Fourth, we study the three cluster workloads from the perspective of their IO requirements and compare our findings to prior studies (Section 5). IO is known to be a bottleneck in high-performance data processing [18, 38]. We focus both on the overhead of reading data from disk, writing it to disk and also shuffling data over the network. We ask what IO workload users impose on the cluster and what can be done to reduce that load. We find that, as reported in prior studies [11], reading the input data to all analytics tasks dominates the IO workload. Surprisingly, we also see that Hadoop fails at scheduling map tasks in a way that ensures the input data is read locally. The vast majority of map tasks read their data over the network. Even more interesting, while we find a huge variance in input data sizes for different jobs, we also show that using modest-size per-node caches would enable a large fraction of the jobs to be served entirely from memory. In terms of managing the result of various analytics tasks, we find that 50% of jobs have their output data overwritten within 5 minutes, and 90% of jobs' output data are overwritten within 1 hour. This result calls for efficient methods to manage large numbers of ephemeral and diverse-in-size files in big-data clusters.

5) Finally, we study load imbalance challenges in these clusters: their prevalence, their impact, their causes, and the performance of the well-known speculative execution method in mitigating these imbalances. (Section 6). Our analysis reveals skew to remain a significant issue. More than 40% of long running jobs in the three clusters have outliers, defined as tasks running at least 50% longer than the median task runtime [5]. This result confirms that the magnitude of the outlier problem is the same as that seen in industrial clusters [5, 25]. Hash partitioning is a known methods to evenly partition load and is also the default approach used in Hadoop. We find, however, that while hash-partitioning redistributes reduce keys among reduce tasks evenly for more than 92% of jobs in all clusters, the resulting task runtimes are unbalanced for as many as 22% to 38% of jobs. We also find that speculative execution is not effective at addressing the problem and that users take little action to manually balance the computation by changing how work is allocated to tasks.

In summary, our study reveals that the use of Hadoop clusters in academic research environments is still in its adolescence: significant improvements in terms of performance and user support remain possible and are critically needed.

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).

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

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

analysis, security 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 dual quad core CPU, 16GB 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 [24]. The 400 nodes in this cluster each contain two quad-core 1.86GHz Xeon processors, 6GB of memory, and four 7200 rpm SATA 750 GB disk. 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 previously analyzed by Kavulya *et al.* [24].

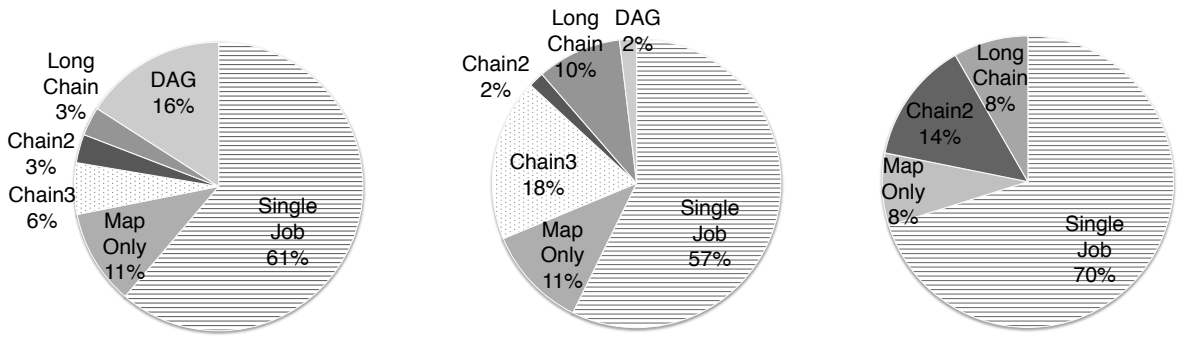
WEB MINING. WEB MINING is a small cluster owned and administered by an anonymized research group. The group comprises faculty, post-docs, and students. Most users run web text mining jobs. The 9 nodes in the cluster each have four quad-core CPU Xeon E5630, 32GB RAM, four 1.8TB HDD. The cluster runs Hadoop 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 three 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, the number of mappers, the number of reducers, 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/records read/written for each task.
- *HDFS Operations*: In version 0.20.x, the Hadoop file system (HDFS) records traces of all file system operations, which include the accessed path names, the client ID, and the operation type (*e.g.*, open, rename, etc.). This information is only available in OPENCLOUD.

3 Application Workload

To understand application workloads, we first study the structure of MapReduce job workflows (Sec. 3.1). We then investigate how users create them (Sec. 3.2). We finally study the type of data transformation that these jobs perform (Sec. 3.3). The analysis presented in this section provides insights into how users utilize the Hadoop MapReduce framework to perform parallel data processing and the type of processing they need to perform.



(a) OpenCloud (b) M45 (c) Web Mining

Figure 1: Distribution of job structures in three workloads.

3.1 Application Structures

We start by studying the structure of the data processing pipelines that users execute. A pipeline is a set of related MapReduce jobs that take one or more datasets as input and produces one or more datasets as output.

We estimate the data dependency 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 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. We group each job by the connected component it belongs to, and Figure 1 shows the grouping results.

Surprisingly, about 68% to 78% of jobs are single stage jobs whose output data are not utilized by other jobs. This fraction is highest in the smallest WEB MINING cluster. We further break down single jobs into map-only jobs and non map-only jobs. About 8% to 11% of all jobs are single jobs with only a map phase. Another large group of jobs are chain-like jobs, which correspond to 12% of jobs in OPENCLOUD and as much as 30% of jobs in the other two clusters. "Chain2 or Chain3" is a chain consisting of two or three jobs. "Long Chain" consists of more than three jobs and includes iterative applications. Finally, some jobs are part of more complicated directed acyclic graphs (DAGs) of jobs. By examining some examples, those DAGs include applications that perform join operations between multiple datasets or run multi-stage machine learning algorithms.

These results thus show a consistent pattern across the three clusters: data transformations that require a single MapReduce job, and sometimes even just a map phase, dominate the workload. Hence, any optimizations that focus on such jobs (*e.g.*, MapReduce Online [15]) will have a great impact on the overall workload. On the other hand, a significant fraction (up to 30%) of data transformations require a combination of jobs, most often a sequence. Users could thus also leverage tools and optimizations for these more complex scenarios.

3.2 Application Types

In a MapReduce system, users can run arbitrary map and reduce functions as long as the functions have compatible signatures. We analyze *how users prepare their MapReduce jobs*. We categorize the jobs as follows:

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

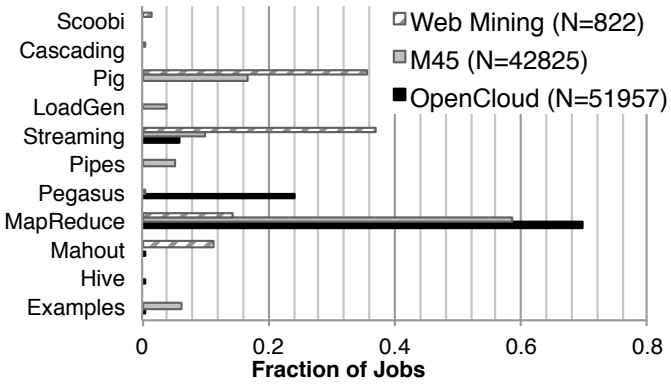


Figure 2: Fraction of jobs per application type.

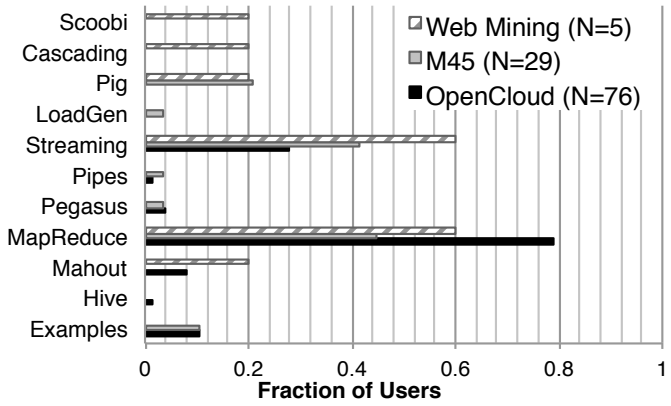


Figure 3: Fraction of users per application type.

- **Low-level API:** The simplest way to write a MapReduce application is to provide map and reduce functions through the native *MapReduce* Java API. *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 MapReduce 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 [14, 32]. *Pegasus* provides higher-level APIs for large scale graph analysis [23].
- **High-level Language:** *Pig* and *Hive* fall into this category [30, 9]. A user writes a script in a high-level language then the runtime system compiles the script into a DAG of MapReduce jobs. Users can include user-defined functions if necessary.
- **Canned MapReduce Jobs:** There are a few pre-packaged 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 MapReduce. *Examples* are examples such as wordcount and terasort that are part of the Hadoop software distribution. *LoadGen* is also a part of the Hadoop software distribution that generates workloads for testing and benchmarking purposes.

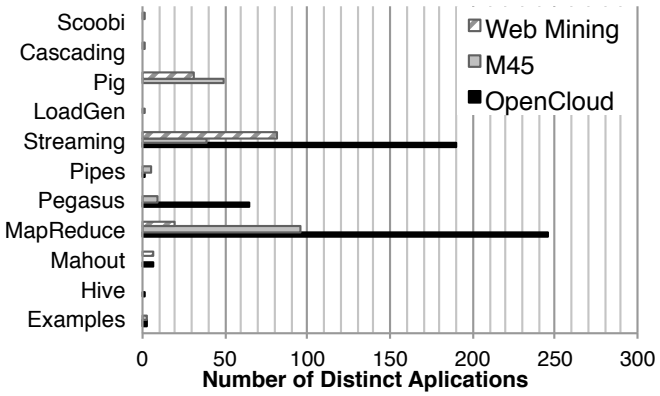


Figure 4: The number of distinct applications per application type.

Figure 2 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. From an informal conversation with authors of Pegasus, we confirmed that the Pegasus jobs in the OPENCLOUD cluster are mainly due to a class project.

Figure 3 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.

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 earlier in Figure 1). More complex tools thus do not have the expected uptake in these academic clusters. Part of the reason could simply be user education since the WEB MINING cluster shows a high utilization of one specific higher-level system, Pig, which is popular among the users of that cluster. Second, legacy code (*e.g.*, unix commands and machine learning package such as libsvm) and language familiarity (*e.g.*, python, perl, and ruby) also plays an important role as shown by the prevalence of the use of Streaming.

We further 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 binary code 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). So we assume that the same application signature is meant to process an input dataset in a certain way.

Figure 4 shows the number of distinct applications submitted to the three clusters for each application type. Comparing with Figure 2, the changes in relative heights are striking. In all three clusters, streaming jobs form a large fraction of all distinct jobs while they result in a small fraction of actual job executions, especially for M45 and OPENCLOUD. In contrast, Java Hadoop jobs also comprise many distinct jobs but are comparatively more repetitive. The latter is an expected result but the former is surprising. It is not clear why streaming jobs are less repetitive, since streaming is not an optimal programming interface for ad-hoc

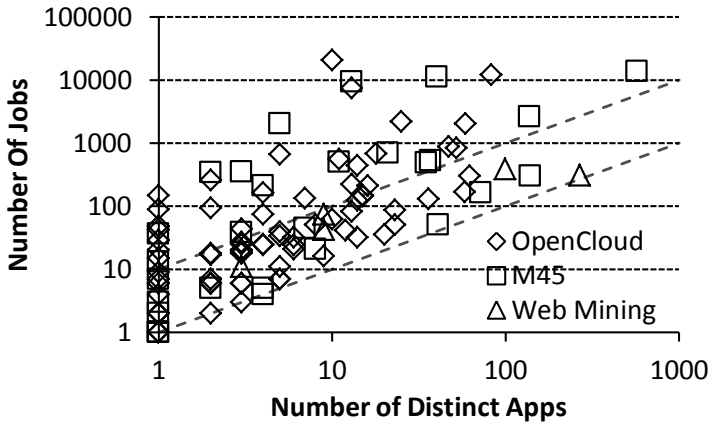


Figure 5: The number of distinct applications and the number of jobs per user. Each point corresponds to one user. The reference lines represent $y = x$ and $y = 10x$ respectively. Users above $y = 10x$ line run the same job more than 10 times. All three clusters have a good mix between exploratory and repetitive applications. Many users executed only a single application.

analytics. Interestingly, even Pig jobs are repetitive while one would expect Pig Latin to be the tool of choice for exploratory analytics. These results suggest that users may not be applying optimal tools for their needs or advanced tools are not optimal for these needs.

Finally, we look more generally at the relative frequency of repetitive and exploratory data transformations. Figure 5 shows the distribution of the number of distinct applications and the number of instances of these applications executed by each user in each cluster. The reference lines correspond to $y = x$ and $y = 10x$ respectively. Users above $y = 10x$ line run the same application more than 10 times. For both the number of distinct applications and the number of jobs submitted, there are orders of magnitude difference among users. We also see a good mix between exploratory applications executed only once and repetitive analytics executed hundreds or more times. Many users submit only a single application. 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.

3.3 Clustering by Running Behavior

We finally characterize the *types* of data transformations that users perform. For this, we use the K-means [27] clustering algorithm to group jobs by their data transformation patterns and running times following the approach by Chen et. al. [13]. Each job is represented as a vector of six features including *input size, shuffle size, output size, job duration, map task time* and *reduce task time*. 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 six-dimensional space.

Table 2 summarizes the clustering results. The table shows the centroid of each cluster, the size of each group and also the number of distinct users and applications in each group. To draw out the data transformation patterns in each cluster, we assign labels for the map phase and the reduce phase of each group based on how it modifies its input data: For the map phase, by comparing the input data size and the shuffle data size, we use three labels *extract*, *expand* and *transform* for three cases: input $\gg$ shuffle , input $\ll$ shuffle , and input $\approx$ shuffle. For the reduce phase, we define four labels *extract*, *expand*, *dump* and *none* for cases: shuffle $\gg$ output, shuffle $\ll$ output, shuffle $\approx$ output, and no reduce phase. We also add extra

Cluster	Input	Shuffle	Output	Duration	Map Time	Reduce Time	#Jobs	#Apps	#Users	Map Label	Reduce Label
OpenCloud	8GB	1GB	764MB	3 min	4K	2K	50859	631	75	Small	Small
	195GB	168GB	145GB	58 min	138K	64K	398	76	24	Transform	Dump
	860GB	22GB	16GB	4.5 hrs	940K	228K	395	63	23	Extract	Dump
	599GB	836GB	53GB	1.9 hrs	659K	331K	109	19	15	Expand	Aggregate
	355GB	50GB	162GB	16 hrs	6M	93K	45	9	8	Extract	Expand
	662GB	2.1TB	289GB	6 hrs	1M	733K	34	7	3	Expand	Aggregate
	799GB	199GB	1.4TB	6.5 hrs	729K	81K	33	16	6	Extract	Expand
	1.3TB	1.2TB	183GB	15 hrs	6M	4M	15	7	4	Transform	Aggregate
	13.0TB	2GB	262GB	21 hrs	9M	115K	11	2	4	Extract	Expand
	54GB	1GB	96GB	3.2 day	14M	264K	11	4	5	Computation	Expand
M45	3GB	2GB	1GB	2 min	3K	2K	40644	1095	29	Small	Small
	3GB	4GB	4GB	1 hrs	78K	6K	1350	98	18	Computation	Small
	104GB	120GB	166GB	55 min	105K	295K	270	34	14	Transform	Transform
	445GB	139GB	29GB	17 min	138K	71K	253	12	6	Extract	Aggregate
	4.9TB	4.9TB	433GB	2 hrs	1M	1M	111	5	2	Transform	Aggregate
	537GB	582GB	4.9TB	2 hrs	343K	2M	127	4	3	Transform	Expand
	402GB	1.8TB	399GB	4.3 hrs	1M	2M	30	7	5	Expand	Aggregate
	107GB	0B	107GB	2 day	7M	0	26	4	2	Transform	None
	121GB	170GB	8GB	20.5 hrs	2M	141K	14	7	6	Transform	Extract
Web Mining	15GB	10GB	8GB	8 min	8K	1K	718	370	4	Small	Small
	93GB	122GB	60GB	2.8 hrs	613K	45K	38	17	4	Expand	Aggregate
	31GB	187GB	74GB	6.5 hrs	1M	111K	18	3	2	Expand	Aggregate
	1.0TB	212GB	69GB	4.8 hrs	1M	69K	9	8	4	Extract	Aggregate
	44GB	307GB	111GB	1.2 day	2M	300K	7	1	1	Expand	Aggregate
	2.1TB	1.1TB	420GB	6.8 hrs	797K	270K	5	3	2	Transform	Aggregate
	45GB	5GB	2.5TB	7.2 hrs	1M	103K	5	3	2	Extract	Expand (Large)
	543GB	11GB	62GB	20.9 hrs	6M	40K	4	3	2	Extr/Comp	Expand
	3.5TB	7.1TB	273GB	22 hrs	2M	878K	3	2	1	Transform	Aggregate
	36GB	7GB	62GB	2.5 day	14M	57K	2	2	2	Computation	Expand

Table 2: Clustering jobs in each workload using k-means. Map and reduce times are the sum of running times of all tasks, *i.e.*, a job with 3 map tasks taking 15 seconds each has a map time of 45 task-seconds.

labels for small jobs and computation intensive jobs. Small jobs have very short duration, and computation intensive jobs have significantly long duration compared to their small input data size.

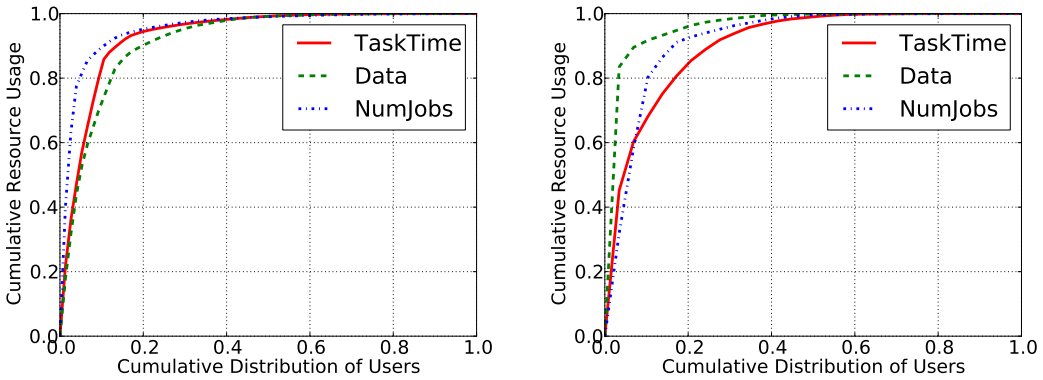
Small jobs dominate all the workloads. More than 90% of jobs touch less than 20GB, and run for less than 8 minutes. This result reflects what happens in production clusters [11] and points to a misuse of the Hadoop cluster: such short jobs on small datasets do not need a big cluster.

Workloads are also very diverse: no other category clearly dominates in all clusters. Interestingly, in each cluster, one type of data transformation is significantly more frequent than others with many users and many applications falling in that same category. The common category, however, changes from cluster to cluster.

The key implication of these results is that any good scheduler must be designed to handle a large number of very small jobs combined with a mix of different types of jobs.

4 User Behavior Analysis

How do users interact with the cluster? To capture the influence of user behavior on effectiveness of Hadoop, we consider the distribution of resource consumption among users (Sec. 4.1), how job submission patterns can be used to infer interactivity requirements (Sec. 4.2), and the degree to which users customize their jobs to obtain different performance characteristics (Sec. 4.3).



(a) OPENCLOUD (b) M45

Figure 6: Aggregated resource usage over users under three metrics. 20% of users consume about 90% of the resources.

4.1 Distribution of Resource Consumption Among Users

As with many other human activities, the distribution of resource consumption follows a power law. We measure the resource consumption of each user by three metrics: *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. Figure 6 gives the distribution of these three metrics over all users in two clusters OPENCLOUD and M45. We omit the WEB MINING cluster since it has only 5 users. All three metrics indicate that about 80% to 90% of resources are consumed by 20% of the users in both clusters.

4.2 Job Submission

Figure 7 shows hourly job submissions averaged by day of week during the monitored period. The shaded area represents the 8am to 6pm time window in each cluster local timezone. The WEB MINING cluster shows the strongest diurnal pattern: no jobs are submitted late at night. In OPENCLOUD, the workload is shifted diurnal toward evening and night. M45 does not particularly show any daily patterns partly because users come from multiple timezones and jobs are submitted by scripts. In all three clusters, late Saturday night to early Sunday morning see the fewest jobs on average. These results show that shared clouds with a greater variety of users can make better use of the physical cluster resources.

To better understand the users' submission behavior, we organize the jobs into "batches" and plot the distribution of the submission interval between these batches and the runtime distribution for the jobs within a batch. Jobs from each user are grouped by their dependency structure as defined in Section 3.1. Interdependent jobs are considered to be in the same batch. For two successive job batches A and B from the same user, we measure *submission time of* A $-$ *submission time of* B. Within a batch, we measure the running 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.

As shown in Figure 8, the distribution of the log value of per-user submission interval is close to a normal distribution. About 60% of the batches have a submission interval of less than 100 seconds, showing that workflow submissions are very bursty. In OPENCLOUD and M45, about 10% and 30% of jobs within

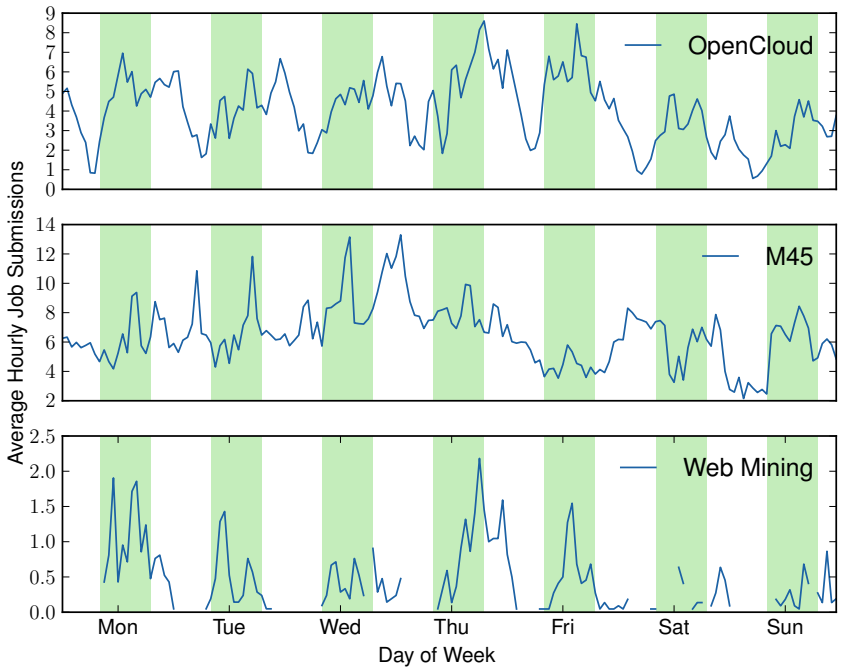


Figure 7: Hourly job submissions for all successful, failed, and killed jobs. Shaded area represents 8am to 6pm time window in local cluster time zone. The smaller local cluster suffers poor utilization due to a smaller user base, suggesting the trend toward centralization in public clouds is warranted.

a batch have a running interval of less than 1 second. Those jobs are likely being managed by scripts that automatically execute the workflows. However, there are still more than 20% of jobs with running intervals larger than 10 seconds in the three clusters, particularly in WEB MINING. Those jobs appear to be submitted manually by users, suggesting that they are using the cluster interactively for, say, debugging or result validation.

Batches of jobs are reasonably well supported by Hadoop, but interactive sessions are not. For each job, Hadoop incurs significant launch overhead. Hadoop results must be fetched from HDFS and inspected. Most importantly, the same data is accessed repeatedly. The ability to persist resource allocations across operations and respond to user input between them would add value. A system with the scalability of Hadoop but direct supports for interactive sessions appears to be a promising direction of research.

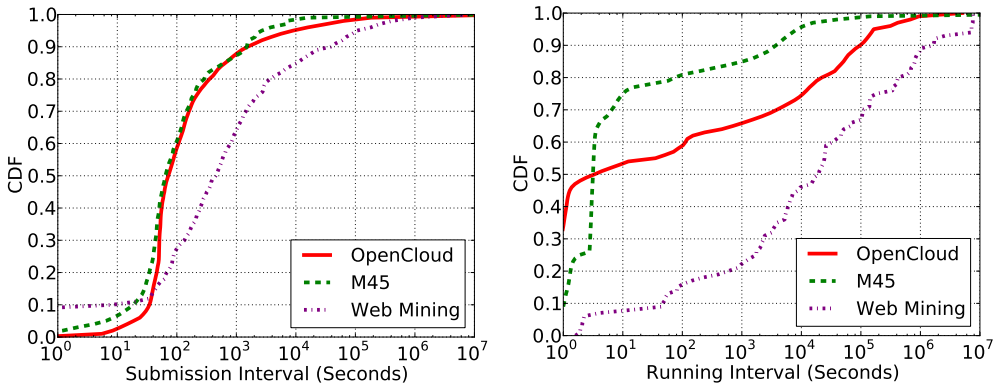
4.3 Hadoop Customization

A user can customize and optimize a Hadoop MapReduce job by supplying additional functions besides just map and reduce. In this section, we consider whether users are exploiting this feature in practice.

4.3.1 Job Customization

Most job customizations are related to ways of partitioning data and aggregating data from earlier stages:

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.



(a) Between Batches (b) Within Batches

Figure 8: Distribution of the submission interval between successive batches for the same user, and running interval between jobs within a batch. Jobs that share a data dependency (as defined in Sec.3.1) are considered to be in the same batch.

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 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.

In general, job customizations help with performance and 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.

4.3.2 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 [17, 22].

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. We discuss speculative execution in detail in Section 6.3.

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 related to those sorts. Two users of WEB MINING cluster have adjusted `io.sort.mb` parameter to 200. Only one user of M45 cluster have 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 the behavior of writing the final output of a MapReduce job. In OPENCLOUD, 11 users have tried different values for replication factor. In M45, two users have adjusted block size and only one user tried a different replication factor. Other than these, all users kept the cluster default values.

In summary, users tend to tune parameters directly related to failures. JVM options are used to prevent "Out of Memory" errors. By talking with administrators of OPENCLOUD, we learned that many of their users explicitly tuned these options in response to poor failure behaviors. 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.

5 Storage Workload Analysis

Hadoop clusters are designed to process large amounts of data, and therefore understanding workloads from the perspective of the storage system is crucial to understanding and improving the performance of these systems. In this section, we focus on analyzing storage workloads, including general data characteristics of MapReduce jobs, (Sec. 5.1), data access patterns (Sec. 5.2), and file system namespace statistics (Sec. 5.3).

5.1 Data Characteristics

There are three main stages in Hadoop MapReduce that involve disk traffic [6, 17]: (1) Map tasks read input data from HDFS; (2) Map tasks write intermediate results back to the local filesystem in preparation for the shuffle phase; (3) Reduce tasks write the final result back to HDFS. Figure 9 shows the aggregate map input, map output and reduce output bytes of all jobs in the three workloads. As found in many industry workloads [11], map input dominates the I/O traffic in all three workloads. An important optimization in MapReduce (and Hadoop) consists in scheduling map tasks on the physical machines that host a copy of their input data in order to maximize local data reads. Figure 10 (a) shows the cumulative distribution function (CDF) of jobs that read their input data locally. Interestingly, the Hadoop schedulers in the two large clusters were unsuccessful in placing map tasks close to their input data. For OPENCLOUD and M45, nearly 50% of jobs have less than 50% of data-local map tasks. This result confirms the potential importance of recent optimization that seeks to address this problem [4, 36, 37].

Figure 10(b-d) shows the distribution of input, shuffle and output data sizes for different jobs. Most jobs in all three workloads read, shuffle, and output less than 10GB. But the general distributions of data sizes in these three workloads differ a lot. For example, the median input data sizes in M45,OPENCLOUD,WEB MINING are 10MB, 100MB, 8GB respectively. Similar to the distribution of job durations, the data sizes of these jobs are small compared to the entire capacity of the cluster. The prevalence of small files are also found in various workloads such as big-data clusters in industry [11], scientific high performance clusters [16], and modern desktop traces [28]. This suggests that big-data storage systems should also be opti-

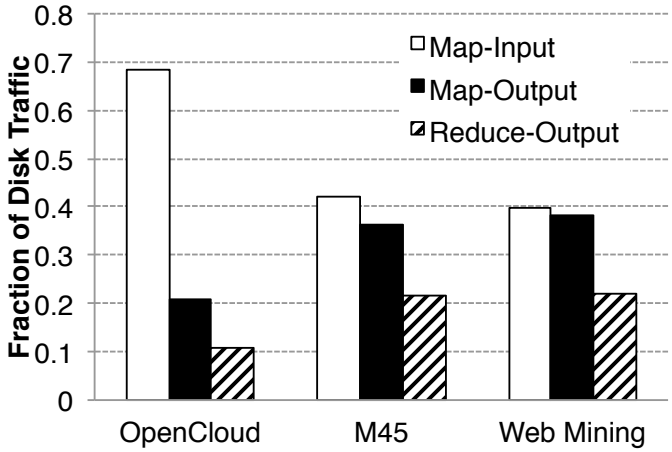


Figure 9: Fraction of data access that are aggregated by input, shuffle, and output of all jobs in each workload.

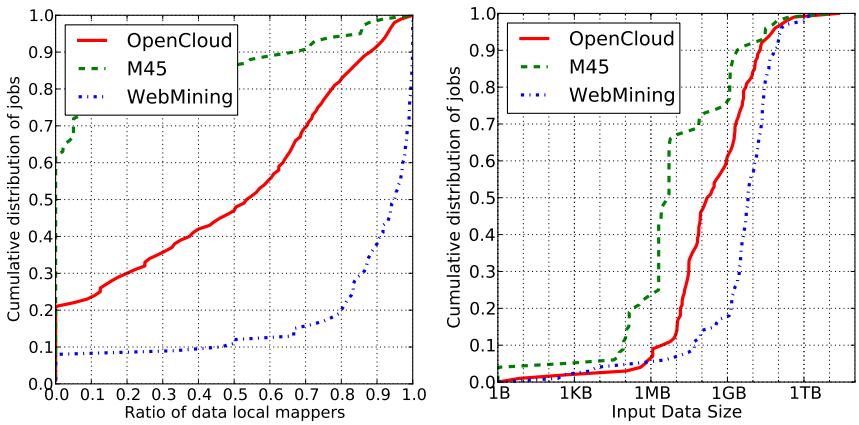
mized for small files. For example, 64MB default chunk size in original design [6, 17] may not be a good fit for diverse parallel computing workloads.

5.2 Access Patterns

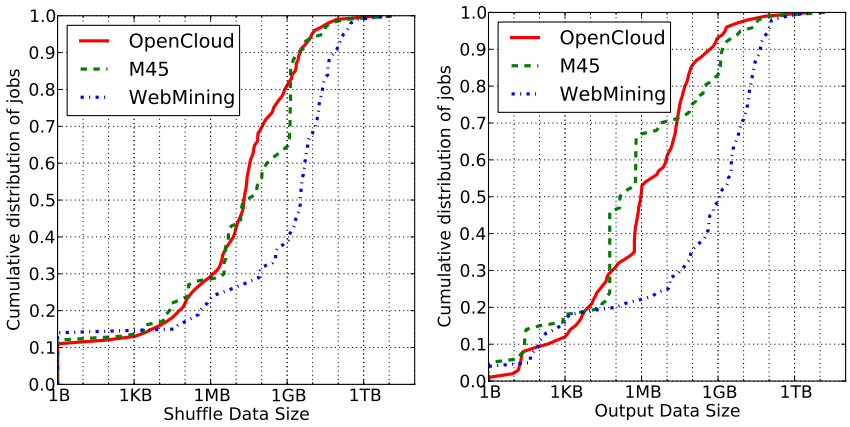
This section analyzes MapReduce jobs' access pattern to input and output files. The analysis is based on input and output paths recorded in the configuration file of each job. As in Section 3.1, this analysis assumes that input and output paths are not renamed nor manually modified by users. Figure 11 (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 paths. For OPENCLOUD and M45, about 10 percent of paths are responsible for 90% of all accessses. This observation indicates that caching may help reduce I/O overheads. Further analysis in Figure 11 (b) indicates the 90% of data re-accesses happen within 1 hour. Therefore a cache policy can possibly bring considerable benefit, which has also been observed in prior work on industry workloads [11, 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 3 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 512MB 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 11 (c) shows the distribution of time intervals between data overwrites. For OPENCLOUD and M45, 50% of jobs have their output data overwritten within 5 minutes, and 90% of jobs' output data are overwritten within 1 hour.



(a) Local Mapper Ratio (b) Input



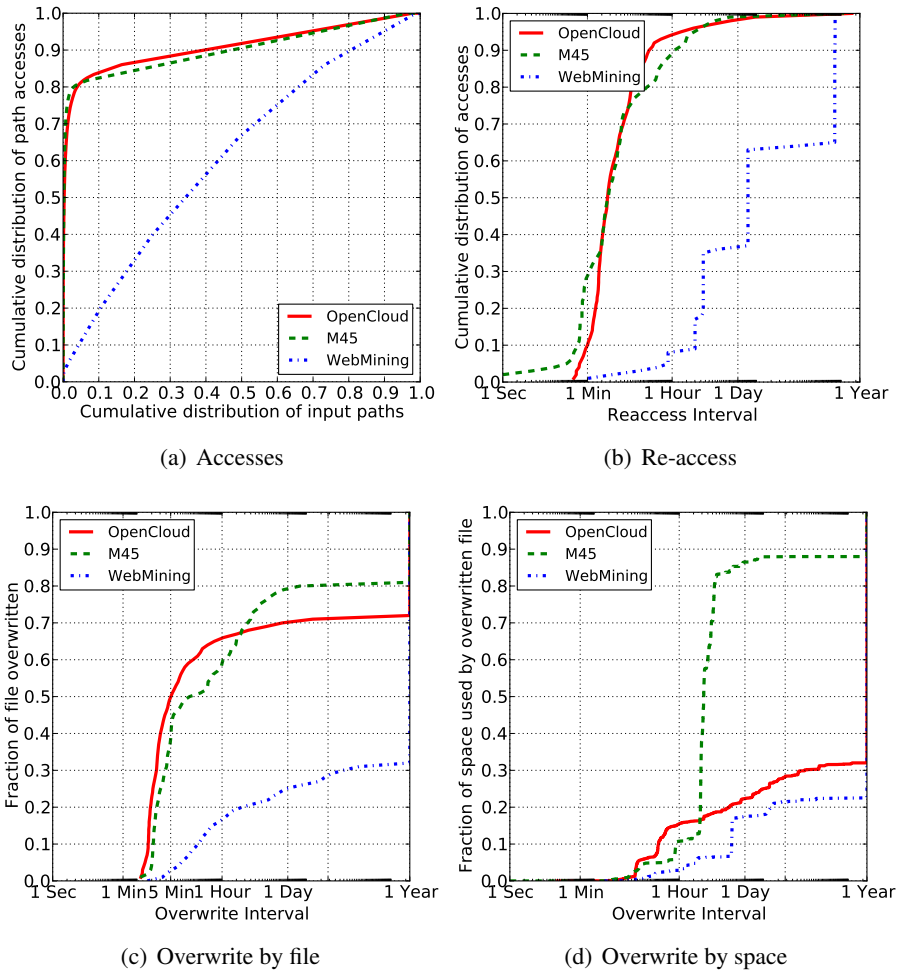
(c) Shuffle (d) Output

Figure 10: (a) Distribution of jobs with different ratios of data local mappers; (b)-(d) Distribution of per-job data size for the input, shuffle and output phase.

Figure 11 (d) shows the space used by overwritten data within various time intervals. For OPENCLOUD, most overwritten output data are small files which only occupy small portion of storage space. For M45, large output data 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 as indicated in Section 5.1. These observations suggest that there is potential room to improve defragmentation algorithms (cleaning algorithm in LFS [31]) of local file systems by explicitly grouping short-lived files. So short-lived files can be cleaned together without causing fragmentations on disks.

5.3 File System Namespace

Many distributed file systems that run on large clusters, including HDFS [10], GFS [21], PanFS [35] are limited by the speed of the single metadata server that manages an entire file system namespace. Next generation HDFS [7] and ColossusFS [20] will provide distributed metadata services. This motivates us to understand the statistics and workloads of filesystem namespaces. Different from previous sections, the



(a) Accesses (b) Re-access

(c) Overwrite by file (d) Overwrite by space

Figure 11: Distribution of various access 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.

logs we analyze here are from the Hadoop NameNode (metadata server of HDFS), and are only available in OPENCLOUD.

First, we analyze the frequency of each file system operation. There are 4.1 billion operations in total over the entire trace. The filesystem call *open* is the most frequent operation and is responsible for approximately 97% of all HDFS operations in OPENCLOUD. Figure 12 (a) shows the distribution of operations except "*open*". "*list*" is frequently used to list files in a directory or get status about a particular file to generate InputSplit data structure on job submission. "*rename*" is more frequent than could be expected because Hadoop tasks use "*rename*" to achieve a simple form of transactions called output commit. The final output files are first written to temporary directories. Once the tasks finish successfully, temporary directories are renamed to the final output file names.

The CDF in Figure 12 (b) shows the cumulative distribution of directories by the number of files they contain, and their weighted distribution. While more than 80% of directories contain fewer than 10 files, there are still 0.2% directories each of which contains more than 10,000 files. These 0.2% directories contain about 60% files in the whole file system.

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

Table 3: Percentage of jobs whose entire inputs hit cache under different cache size α per node.

Cluster	Total	>5 mins	$\|map\|>1$	$\|reduce\|>1$
OPENCLOUD	51957	5144 (9.9%)	4953 (9.5%)	3077 (5.9%)
M45	42825	6621 (15.5%)	6538 (15.3%)	4130 (9.6%)
WEB MINING	822	369 (44.9%)	368 (44.8%)	255 (31.0%)

Table 4: Overall Statistics for Straggler Analysis: total number of jobs in the log, number of jobs longer than 5 min, and number of jobs with more than one map or reduce task.

Figure 12 (c) shows the histogram and CDF of files by directory depth. A majority of files (about 80%) reside in directories with depth between 5 and 10. Figure 12 (d) further shows the depth of most pathnames accessed by each namespace operation. The depth is between 4 and 11. This result indicates that, in order to achieve high performance, it is necessary to reduce iterative lookups for each pathname component, especially when file namespace is partitioned across multiple metadata servers.

6 Skew in the workload

In parallel computing, balancing workload is important not only for resource utilization but also individual job completion times. In the ideal scenario, all tasks in a job are scheduled and complete at the same time. In practice, however, some tasks often run significantly longer than other tasks in the same job. Such long running tasks (*i.e.*, straggler tasks) can significantly delay the job completion time.

We first analyze the frequency and distribution of straggler tasks (Section 6.1), then analyze the relationship between skew and techniques to partition input data (Section 6.2). Finally, we evaluate the effectiveness of speculative execution in Hadoop, which is the default mechanism to address straggler tasks (Section 6.3).

Scope of Analysis: In this section, we limit our study to successfully completed jobs only, because counters and timing information are not captured in the job history for failed and killed jobs. Unless explicitly stated, we only consider jobs that run longer than 5 minutes because the impact of stragglers are insignificant for short jobs (*i.e.*, even jobs with straggler are fast). A task only straggles with respect to some other task, so we only consider map and reduce phases that have at least two tasks. Table 4 summarizes the number of jobs (and job phases) analyzed in this section. When calculating the task runtime, we only take the "map" phase and "reduce" phase into account and ignore the *shuffle* and *sort* phases.[2]

[2]Unfortunately, we can not exclude the local sort phase of map tasks because the version of Hadoop running in all three clusters does not separate the timing of the local sort phase from that of the map phase.

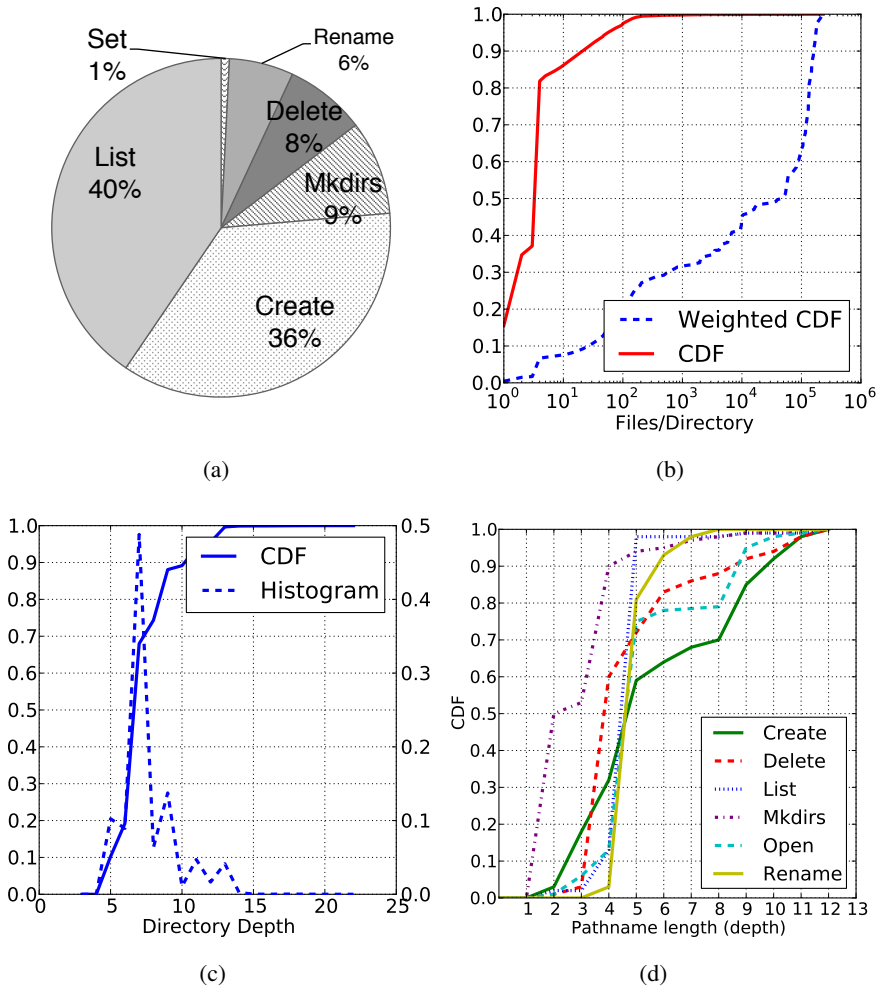


Figure 12: Namespace statistics in OPENCLOUD: (a) Distribution of HDFS NameNode operations except *open* in OPENCLOUD; (b) CDF and Weighted CDF of directories by count of files; (c) Distribution of files by directory depth; (d) Distribution of HDFS NameNode operations by length of accessed pathname.

6.1 Straggler Tasks

Dean *et al.* identified the straggler problem [17] and Ananthanarayanan *et al.* analyzed the problem in a production cluster [5]. In this subsection, we replicate the straggler analysis by Ananthanarayanan *et al.* [5][3]. Ananthanarayanan *et al.* categorizes a task as a straggler if the task is running at least 50% longer than the median task in the same phase of the same job. Using this definition, the three research clusters are similar to the enterprise clusters studied by Ananthanarayanan [5] and an earlier study of the M45 cluster [24]: there are many stragglers, and some are orders of magnitude slower than the median runtime.

Table 5 summarizes the number of jobs that have stragglers in the map phase, the reduce phase, and in both phases. Overall, more than 40% (and up to 75%) of jobs that run longer than 5 minutes have at least one straggler. We also find that even the map phase, which is usually regularized by assigning a fixed amount of bytes to each task, frequently experiences the straggler problem. This finding confirms that, in these three clusters, the input data size alone is not a good indicator of task runtime. The same finding was

[3]We use the term *straggler* instead of *outlier* for consistency with the original MapReduce paper [17].

Stragglers	Map	Reduce	Map ∧ Reduce	Map ∨ Reduce
OPENCLOUD	2967 (58%)	1940 (38%)	1109 (22%)	3798 (74%)
M45	3643 (55%)	2659 (40%)	1310 (20%)	4992 (75%)
WEB MINING	140 (38%)	35 (9%)	18 (5%)	157 (43%)

Table 5: Number and fraction of jobs that have stragglers in map, in reduce, and both phases. More than 40% of jobs running longer than 5 minutes have at least one straggler.

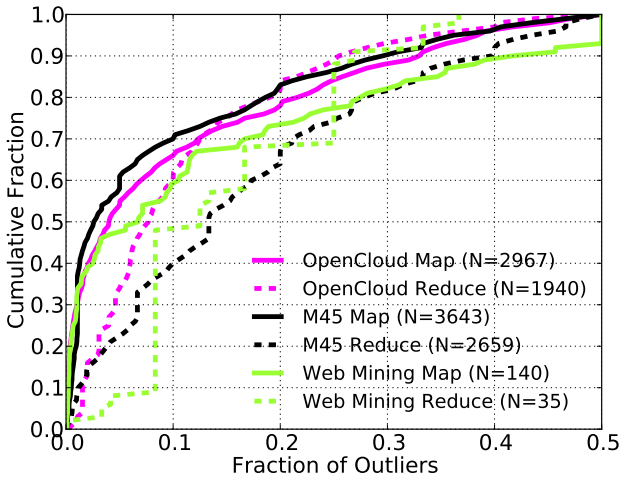


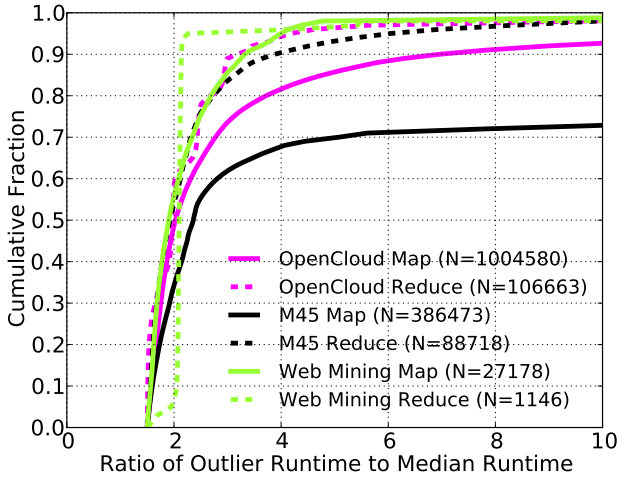
Figure 13: Cumulative fraction of straggler tasks in Map and Reduce phases. *N* is the number of jobs that run for longer than 5 minutes and have stragglers per cluster, per phase.

reported in the analysis of enterprise clusters [11].

Knowing that many jobs have stragglers, we can also ask how many tasks per job are stragglers. To answer this question, we plot the cumulative distribution function of stragglers in each cluster, and for each MapReduce phase. Figure 13 shows the results. In all three clusters, 25% of map and reduce phases that include any stragglers have more than 15% of tasks as stragglers. The M45 reduce distribution closely resembles the one reported in Ananthanarayanan *et al.* [5]. However, we can observe variations in the distributions across phases and clusters.

In Figure 14, we show the distribution of relative runtimes of straggler tasks with respect to the median task runtime of the same phase in the job. In all three clusters, 75% of reduce-side stragglers complete within 2.5x of the median runtime. Map stragglers have larger variations across clusters: 55% and 65% of map straggler tasks complete within 2.5x of median runtime in M45 and OPENCLOUD respectively. As observed in Ananthanarayanan *et al.* [5], the distribution is heavy-tailed. We also showed the values at 99%, 99.9%, and 100% percentiles in the table. Some straggler tasks run orders of magnitudes longer than the median runtime.

In summary, the straggler problem is prevalent in all clusters, and slow tasks frequently run more than 2.5x to orders of magnitude slower than the median.



Percentile		99	99.9	100
OpenCloud	Map	70.0	1106.0	23068.5
	Reduce	15.3	64.9	54692.5
M45	Map	15.3	153.0	1537.5
	Reduce	11.4	143.8	1267.4
Web Mining	Map	11.2	66.9	170.7
	Reduce	46.4	404.3	409.3

Figure 14: Cumulative fraction of ratio of straggler runtime to median task runtime. *N* is the number of straggler tasks per cluster, per phase. The table shows straggler runtime at specific percentiles.

6.2 Input Data Size and Task Runtime

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.

Method: 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).

Map Input Record: Figure 15 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 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 [11]. Mahout is a machine learning algorithm package built on top of Hadoop [8]. 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 simply based on data size alone is insufficient to eliminate skew.

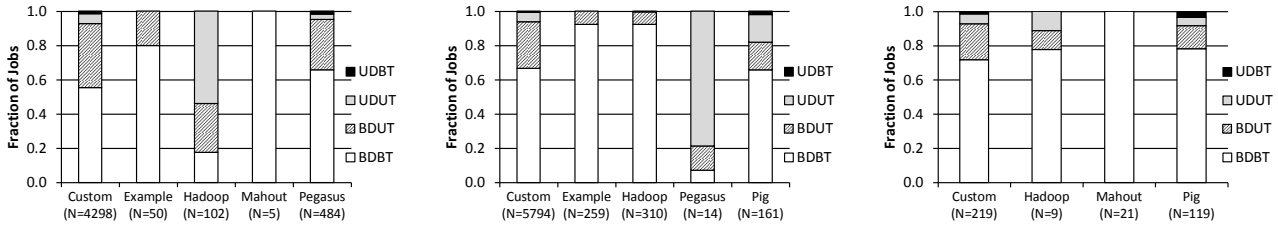


Figure 15: Distribution of map runtime with respect to # of input records per application. *N* is the number of successful jobs that run map functions in each category. The labels indicate the category: (U)BD(U)BT stands for (un)balanced input data, (un)balanced runtime.

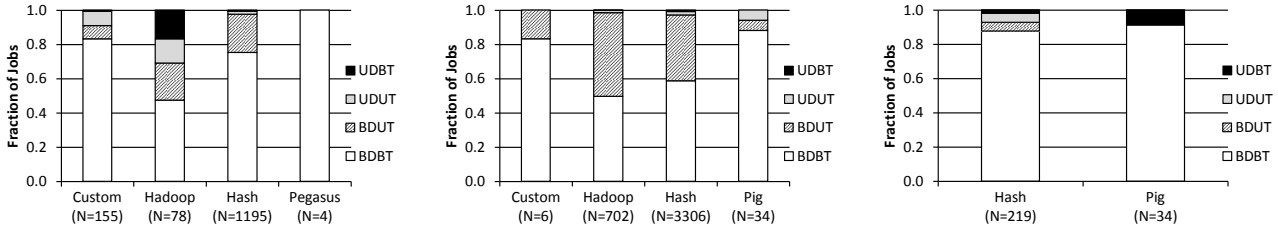


Figure 16: Distribution of reduce runtime with respect to # of reduce keys per partition function. The labels indicate the category: (U)BD(U)BT stands for (un)balanced input data, (un)balanced runtime.

In Hadoop, the InputFormat mechanism is responsible for generating InputSplits which describe the input data processed by a map task. In our traces, less than 1% of jobs attempt to optimize the partitioning of the map input by using a custom input format. In addition, the user can manually specify the number of bytes per split if the input is stored in HDFS. In our traces, 12% of total jobs from 10 users in OPENCLOUD used this optimization. In M45 and WEB MINING, less than 1% of total 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 found that users rely primarily on the default hash-partition function (Figure 17) rather than manually trying to optimize the data allocation to reducers.

Figure 16 shows the analysis per partition function[4]. 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).

[4]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.

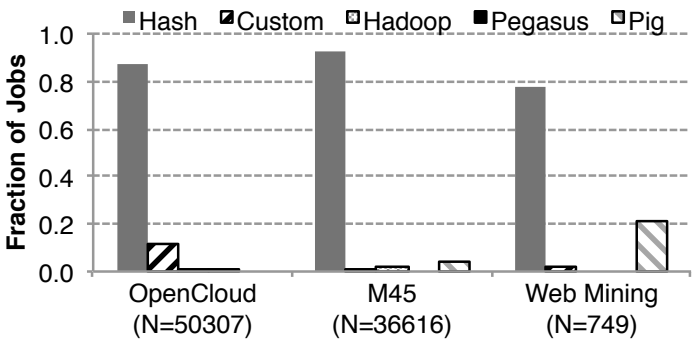


Figure 17: Fraction of jobs per partition function. Hash partitioning is dominant in all three clusters. The fraction of jobs using customized partition function is relatively small, less than 10% in all clusters.

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.

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, we recommend pursuing techniques that automatically reduce skew to achieve better overall performance [26, 34].

6.3 Speculative Execution

Finally, we analyze the effectiveness of speculative execution. Speculative execution is the default mechanism implemented in Hadoop to handle straggler tasks. The intuition is that the task scheduler will eagerly schedule a redundant copy of a task whenever there exists an idle slot in an attempt to improve job completion time. The detailed implementation varies from version to version in Hadoop; explaining this mechanism is beyond the scope of this paper.

Method: For all successful jobs, we extract tasks that involve multiple attempts and categorize the tasks according to the characteristics of the attempts. The *Recompute* label indicates that the attempts were necessary to recover from the failure of the original attempt. *Successful* label is that the attempts were part of a successful speculative execution (*i.e.*, the speculative task was initiated after the original, but completed first). The *Unsuccessful* label represents an unsuccessful speculation (*i.e.*, the original attempt completes faster than speculative attempts). Since the Hadoop does not keep track of precise timings for failed and killed attempts, we inferred the missing data using other information. In particular, the failure of an original task attempt may leave the start time of the failed attempt unrecorded in the attempt log. In this case, we recover the start time from the task log.

Overall Statistics: In Table 6, we showed the default values of speculative execution of the clusters and the number of jobs and speculated tasks. The OPENCLOUD and WEB MINING clusters enable the speculative execution by default while M45 does not. Thus, all speculative executions observed in M45 cluster were initiated by users who explicitly overrode the default value. The fraction of jobs and tasks that

Table 6: Total number of jobs and speculated tasks with default speculative execution value

		Jobs	Tasks	Default
OPENCLOUD	Map	16857 (32%)	548993 (3%)	On
	Reduce	6751 (13%)	124931 (5%)	
M45	Map	1195 (3%)	6025 (0.1%)	Off
	Reduce	2165 (5%)	38835 (2%)	
WEB MINING	Map	467 (57%)	6966 (1%)	On
	Reduce	278 (34%)	630 (5%)	

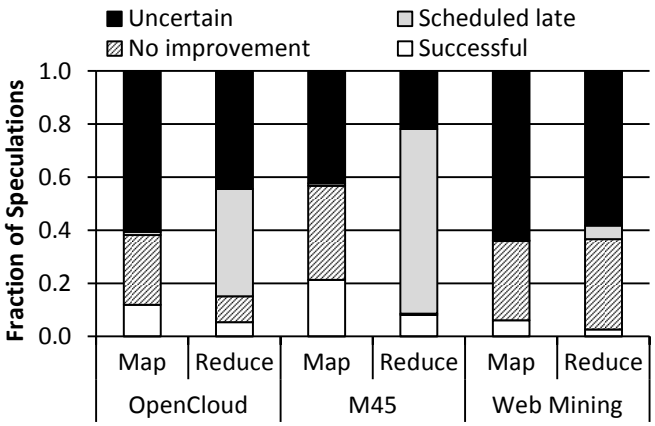


Figure 18: Fraction of Speculation Result per Phase per Cluster. In all clusters, a speculative execution completed faster than the original attempt (*successful*) at most 21%. In all clusters, map speculations did not run significantly faster than the original attempt (*no improvement*). In OPENCLOUD and M45, many reduce speculations were *scheduled late* (speculative attempts ran only less than 10% of original attempt time before killed).

have at least one task that was attempted multiple times varies from cluster to cluster and between map and reduce. The fraction of jobs in M45 is significantly less than those of OPENCLOUD and WEB MINING due to the default value.

In Figure 18, we summarized the result of speculations in all clusters. Speculative execution was not very successful in all three clusters: only 3 to 21% of speculations were *successful*. We further analyzed those *unsuccessful* speculations by classifying each unsuccessful attempt as *scheduled late*, *no improvement*, and *uncertain*. A speculation is *scheduled late* if all speculation attempts for a task ran less than 10% of the original attempt runtime before killed. Those killed attempts ran too briefly thus do not have enough time to run till the end. Also, a speculation may not improve runtime at all. We label such speculations as *no improvement* when the oldest speculation attempt ran more than 90% of the original attempt runtime. The tasks fall into this category are likely to be genuinely long running tasks thus re-executing the same tasks in different node does not improve runtime significantly. We label the remaining unsuccessful speculations as *uncertain*.

In all three clusters, most of map speculations do not run significantly faster than the original attempts. For reduce, many speculation attempts in both OPENCLOUD and M45 were scheduled too late.

By analyzing *recomputed* tasks, we found only one instance of lost map output (*i.e.*, re-execute previously completed map task of which output is lost due to a node failure) in OPENCLOUD. Ananthanarayanan *et al.* also reported that such recomputation due to loss of map output is rare [5].

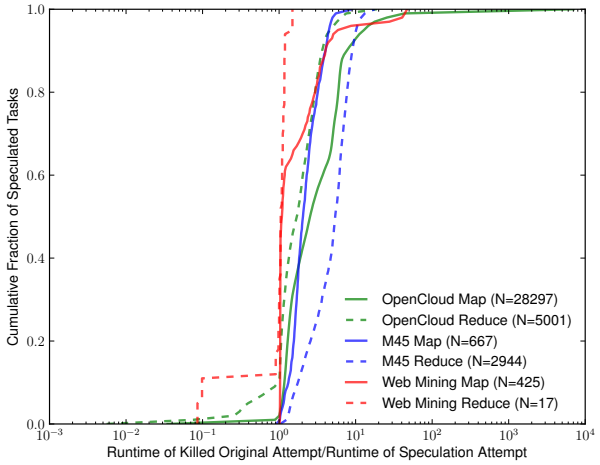


Figure 19: Successful Speculative Execution. Cumulative distribution of runtime ratio between the original attempt and successful speculation attempt. Median speedup of successful speculative execution varies from 1.05x to 5.4x and maximum speed up varies from 1.5x to 7000x.

Successful Speculative Execution: In Figure 19, we analyzed the speedup of successful speculative executions. For the runtime of the original reduce attempt, we adjust the start time of the attempt to the time when the last map attempt completes if the reduce attempt started before completion of all map tasks. Thus, ratio of some reduce attempts may be underestimated. Median speedup of speculative execution varies from 1.05x to 5.4x and 90% of the speculative execution runs up to 10 times faster than the initial attempt. A few speculations run orders of magnitude faster than their initial attempts and most of them completes under a minute. After closely investigating speculative executions that run orders of magnitude faster than their initial attempts, we suspect that those tasks are typically experiencing I/O problems and the speedup was more pronounced especially when there are large number of tasks, thus speculation was delayed at the end of each phase.

To sum up, speculative execution is very effective to handle I/O related problem, especially when there exists a large number of tasks. However, a large fraction of speculations suffers from errors by either speculating too early or too late. To improve accuracy of speculation, making more informed decision as well as stealing work rather than starting from the scratch are important [5, 29, 34, 26].

7 Related Work

Since the first time MapReduce was proposed and its open source implementation Hadoop was made available to the world, they are increasingly gaining popularity from both academic research and industry. Lots of works have been done to understand MapReduce workloads, and improve the performance of MapReduce framework. But to our knowledge, this paper presents the most complete study of academic clusters which incorporate rich information about application level statistics and user-related information. Perhaps the closest related works are studies of Facebook and Cloudera cluster traces [11, 13]. Their studies focus on cluster-level and job-level metrics such as overall resource utilization and data access patterns, as well as the usage of SQL-like programatic frameworks. They also found a wide range of diversity existing in their workloads, with exceptions that small jobs and temporal locality in data accessing are common. In contrast to our workloads, SQL-like programming frameworks are very popular. [24] also studies logs from M45 cluster, (overlapping datasets used in this paper), but its main focuses are cluster-level workload characterization, failure characterization and performance prediction.

There are also previous works on improving performance of one or more aspects of MapReduce and Hadoop systems that are related to our analysis. Quincy [36] and Delay scheduler [37] proposes scheduling algorithms that make trade-off between fairness and data-locality. PACMan [3] studies the effect of different global cache policy for HDFS, and found that in some production cluster traces, the vast majority (96%) of active jobs can fit into a fraction of the cluster's total memory. DiskReduce [19] studies erasure coding for distributed file systems under data insensitive workloads, and analyze file systems workloads of several Hadoop production and high-performance computing clusters. They found small files are prevalent even in large clusters. For data and computation imbalance problems, [25] also found these problems are prevalent in a variety of industrial applications. Mantri [5] studies the causes to outlier tasks and ways of mitigating outlier tasks caused by data imbalance or resource contensions. SkewTune [26] proposes run-time repartition to mitigate skew problems. RoPE [2] dynamically optimzes the execution plan of recurring jobs based on run-time statistics and historical information. Starfish [22] investigates the method of automatic tunning configurations for Hadoop MapReduce programs. Our analysis shows automatic tunning will benefit a lot since many users merely tune configurations for correctness instead of performance.

8 Conclusion

We studied job performance profiles, configurations and user history files from three different Hadoop clusters for academic research. These new Hadoop cluster traces contain richer information than previous studies by recording application specification and user behaviors, which are critical for understanding the requirements and performance of big-data systems.

We find that in the three workloads, a majority of users submitted many small single stage applications implemented in Java, although the rest of workloads are highly diverse in application styles and data processing characteristics. We see underuse of Hadoop features, extensions and optimization tools. We also find conventional techniques do not effectively increase data locality and mitigate load imbalance, and other alternatives such as caching may help with IO performance.

Our conclusion is that the use of Hadoop for academic research is still in its adolescence. Easing the use of Hadoop, and improving system designs subject to changing use cases are crucial research directions for future.

References

[1] Yahoo! reaches for the stars with M45 supercomputing project. http://research.yahoo.com/node/1884.

[2] Sameer Agarwal, Srikanth Kandula, Nico Bruno, Ming-Chuan Wu, Ion Stoica, and Jingren Zhou. Re-optimizing data parallel computing. In *NSDI*, 2012.

[3] G. Ananthanarayanan, A. Ghodsi, Andrew Wang, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica. PACMan: Coordinated memory caching for parallel jobs. In *NSDI*, 2012.

[4] Ganesh Ananthanarayanan, Sameer Agarwal, Srikanth Kandula, Albert G. Greenberg, Ion Stoica, Duke Harlan, and Ed Harris. Scarlett: coping with skewed content popularity in mapreduce clusters. In *EuroSys*, 2011.

[5] Ganesh Ananthanarayanan, Srikanth Kandula, Albert G. Greenberg, Ion Stoica, Yi Lu, Bikas Saha, and Edward Harris. Reining in the outliers in Map-Reduce clusters using Mantri. In *OSDI*, 2010.

[6] Apache Foundation. Hadoop. http://hadoop.apache.org/.

[7] Apache Foundation. HDFS Federation. http://hadoop.apache.org/common/docs/r0.23.1/hadoop-yarn/hadoop-yarn-site/Federation.html.

[8] Apache Foundation. Mahout: Scalable machine learning and data mining. http://mahout.apache.org/.

[9] Ashish Thusoo et. al. Hive: a petabyte scale data warehouse using Hadoop. In *ICDE*, 2010.

[10] Dhruba Borthakur. The Hadoop distributed file system: Architecture and design. http://lucene.apache.org/hadoop/hdfs_design.pdf, 2007.

[11] Yanpei Chen, Sara Alspaugh, and Randy H. Katz. Design insights for MapReduce from diverse production workloads. Technical Report UCB/EECS-2012-17, EECS Department, University of California, Berkeley, 2012.

[12] Yanpei Chen, Sara Alspaugh, and Randy H. Katz. Interactive query processing in big data systems: A cross-industry study of MapReduce workloads. In *VLDB*, 2012.

[13] Yanpei Chen, Archana Ganapathi, Rean Griffith, and Randy H. Katz. The case for evaluating MapReduce performance using workload suites. In *MASCOTS*, 2011.

[14] Concurrent, Inc. Cascading. http://www.cascading.org/, 2012.

[15] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, and Russell Sears. MapReduce online. In *NSDI*, 2010.

[16] Shobhit Dayal. Characterizing HEC storage systems at rest. Technical Report CMU-PDL-08-109, Parallel Data Lab, Carnegie Mellon University, July 2008.

[17] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[18] David J. DeWitt, Erik Paulson, Eric Robinson, Jeffrey Naughton, Joshua Royalty, Srinath Shankar, and Andrew Krioukov. Clustera: an integrated computation and data management system. In *VLDB*, 2008.

[19] Bin Fan, Wittawat Tantisiriroj, Lin Xiao, and Garth Gibson. DiskReduce: RAID for data-intensive scalable computing. Technical Report CMU-PDL-11-112, Parallel Data Lab, Carnegie Mellon University, 2011.

[20] A. Fikes. Storage architecture and challenges. presentation at the 2010 Google faculty summit. Talk slides at http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/university/relations/facultysummit2010/storage_architecture_and_challenges.pdf, 2010.

[21] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *SOSP*, 2003.

[22] Herodotos Herodotou and Shivnath Babu. Profiling, what-if analysis, and cost-based optimization of MapReduce programs. *VLDB*, 2011.

[23] U. Kang, Charalampos E. Tsourakakis, and Christos Faloutsos. PEGASUS: A peta-scale graph mining system implementation and observations. In *ICDM*, 2009.

[24] Soila Kavulya, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan. An analysis of traces from a production MapReduce cluster. In *CCGRID*, 2010.

[25] Qifa Ke, Vijayan Prabhakaran, Yinglian Xie, Yuan Yu, Jingyue Wu, and Junfeng Yang. Optimizing data partitioning for data-parallel computing. In *HotOS*, 2011.

[26] YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. SkewTune: mitigating skew in mapreduce applications. In *SIGMOD*, 2012.

[27] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 1982.

[28] Dutch T. Meyer and William J. Bolosky. A study of practical deduplication. In *FAST*, 2011.

[29] Kristi Morton, Abram Friesen, Magdalena Balazinska, and Dan Grossman. Estimating the progress of MapReduce pipelines. In *ICDE*, 2010.

[30] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig Latin: a not-so-foreign language for data processing. In *SIGMOD*, 2008.

[31] Mendel Rosenblum and John K. Ousterhout. The design and implementation of a Log-Structured file system. In *SOSP*, 1991.

[32] Scoobi Team. A Scalar productivity framework for Hadoop. https://github.com/NICTA/scoobi, 2012.

[33] Bikash Sharma, Victor Chudnovsky, Joseph L. Hellerstein, Rasekh Rifaat, and Chita R. Das. Modeling and synthesizing task placement constraints in Google compute clusters. In *SoCC*, 2011.

[34] Rares Vernica, Andrey Balmin, Kevin S. Beyer, and Vuk Ercegovac. Adaptive MapReduce using situation-aware mappers. In *EDBT*, 2012.

[35] Brent Welch, Marc Unangst, Zainul Abbasi, Garth A. Gibson, Brian Mueller, Jason Small, Jim Zelenka, and Bin Zhou. Scalable performance of the Panasas parallel file system. In *FAST*, 2008.

[36] Yuan Yu, Pradeep Kumar Gunda, and Michael Isard. Distributed aggregation for data-parallel computing: interfaces and implementations. In *SOSP*, 2009.

[37] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *EuroSys*, 2010.

[38] Jingren Zhou, Nicolas Bruno, and Wei Lin. Advanced partitioning techniques for massively distributed computation. In *SIGMOD*, 2012.