A Workload Model for MapReduce

Thomas A. de Ruiter

Delft University of Technology

A Workload Model for MapReduce

Master's Thesis in Computer Science

Parallel and Distributed Systems Group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Thomas A. de Ruiter

2nd June 2012

Author
Thomas A. de Ruiter <thomas@de-ruiter.cx>

Title
A Workload Model for MapReduce

MSc Presentation
11th June 2012

Graduation Committee

Dr. ir. D. H. J. Epema	Delft University of Technology
Dr. ir. A. Iosup	Delft University of Technology
Dr. ir. F. A. Kuipers	Delft University of Technology

Abstract

MapReduce is a parallel programming model used by Cloud service providers for data mining. To be able to enhance existing and to develop new MapReduce systems, we need to evaluate the performance of these systems. To this end we introduce in this work the Cloud Workloads Archive Toolbox. This toolbox facilitates the analysis of MapReduce workload traces, generation of realistic synthetic workloads, and the evaluation of MapReduce systems in simulation. We present an overview and analysis of real world MapReduce workload traces, we propose a model for MapReduce workloads, we describe the development of the toolbox, and we present an experiment in which we use our toolbox to evaluate two MapReduce schedulers.

Preface

This thesis is the final result of a graduation project and completes the master's degree programme Computer Science – with specialization in Parallel and Distributed Systems – of the Faculty of Electrical Engineering, Mathematics and Computer Science at Delft University of Technology.

I would like to use this preface to thank the graduation committee for their guidance, advises, and critics – especially Alexandru, for always being optimistic, and for spending huge amounts of red ink to enhance the quality of my work. Leonie, my parents, family, and friends, for their support, friendship, and their patience. My fellow students, for the conversations and coffee-breaks. The owners of the real-world workload traces used in this thesis, for making the traces available to science. The UC Berkeley AMP Lab people, for providing access their collection of workload traces. And finally Boxun, for sharing his Matlab skills with me.

Thomas de Ruiter

Delft, The Netherlands
1st May 2012

Contents

Chapter 1

Introduction

Many Cloud service providers have a need to analyze large amounts of data, for example to evaluate advertisement campaigns. The MapReduce programming model is widely used as a solution for these data mining problems. We would like to be able to evaluate and compare existing, enhanced, and new MapReduce systems. To this end we introduce in this work a toolbox that facilitates these analyses.

1.1 MapReduce

MapReduce is a programming model for parallel computing developed by Google [1]. The need to perform analyses on large amounts of data is not specific to Cloud service providers, but the MapReduce programming model was developed with this specific audience in mind, apart from Google it is known to be used by for example Yahoo!, MySpace, Facebook, and Twitter. Facebook [2] uses MapReduce for, among other, business intelligence, spam detection, and advertisement optimization.

The name MapReduce originates from the higher-order[1] map and reduce functions originally, found in functional programming languages. A MapReduce program is in fact the combination of a map and a reduce function, the map function is applied on the input data and the reduce function is applied on the output of the map function. Figure 1.1 gives an overview of how MapReduce works:

1. A MapReduce job consists of a map function, a reduce function, and input data (on a distributed file system).

2. First, the input data are partitioned into smaller chunks of data.

3. Then, for each chunk of input data, a "map task" runs which applies the map function to the chunk of input data. The resulting output of each map task is a collection of key-value pairs.

[1] Higher-order functions are functions that have a function as argument.

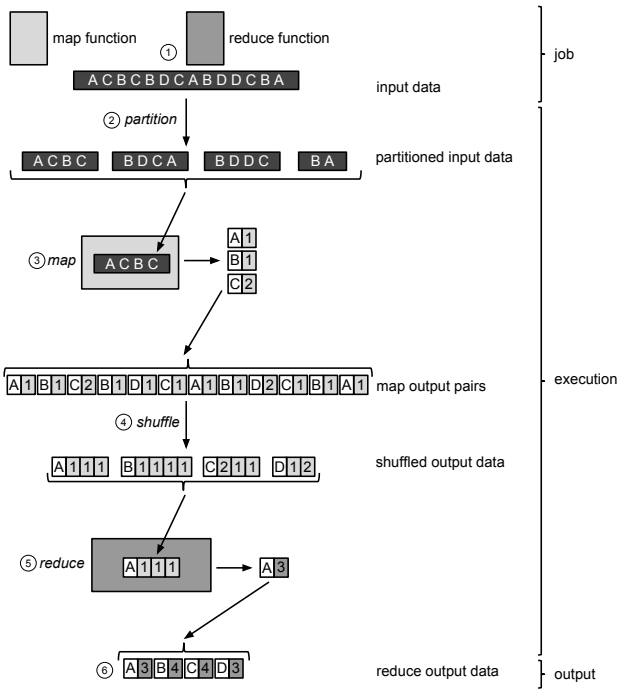


Figure 1.1: Overview of MapReduce.

4. The output of all map tasks is shuffled, that is, for each distinct key in the map output, a collection is created containing all corresponding values from the map output.

5. Then, for each key-collection resulting from the shuffle phase, a "reduce task" runs which applies the reduce function to the collection of values. The resulting output is a single key-value pair.

6. The collection of all key-value pairs resulting from the reduce step is the output of the MapReduce job. (In Hadoop the reduce outputs are merged after the the job has finished, when the user uses the "getmerge" command to get the output from the distributed file system.)

The main place where parallelization is exploited in MapReduce is during the running of the map and reduce tasks, depicted as respectively steps three and five in the above description. Although in the above overview the steps two to six seem to be distinct phases, the more advanced MapReduce implementations run these phases in parallel, a reduce task can for example start working as soon as the first key-value pair is emitted by a map task.

Map tasks and reduce tasks can be easily parallelized since the individual map and reduce tasks run in isolation. Because of this isolation, MapReduce is also fault tolerant, as failed tasks can be rescheduled without any problem.

Although in principle any problem can be formulated as a MapReduce job, it is not suitable for every problem. Since every task runs in isolation, the only usual way to communicate is using the task input and output. This makes a Poisson solver, that can be implemented in a few lines of C code for an MPI application on a cluster, complex to implement in MapReduce. On the other hand, the classical MapReduce word count example, is easier to implement in MapReduce than by using MPI.

There exist multiple implementations of MapReduce. Google has developed a private implementation, but there also exist various open-source implementations, of which the probably best known implementation is provided by the Apache Hadoop[2] project. There are also tiny implementations of MapReduce, like for example mincemeat.py[3], which can be useful for simple ad-hoc experiments, and there exists even an implementation of MapReduce in bash script, called bashreduce[4].

MapReduce runs generally on dedicated clusters, but you can also run MapReduce on virtual "clusters" in a cloud. Amazon offers for example Amazon Elastic MapReduce[5], a service which automatically configures a virtual MapReduce cluster on top of their cloud resources. Running MapReduce jobs in grid environments is also being researched, see Section 2.1.5.

1.2 Real-World MapReduce Workloads

As basis for this research we look at the execution of MapReduce jobs in real world MapReduce clusters. Information about this execution (like job arrival, start, and finish times, network usage, disk usage, etc.) is obtained from known cloud service providers. The obtained workloads are collected in the Cloud Workloads Archive; we leave the publication of this archive for future work.

Because the workload information is received in the form of database dumps and log files, all in different formats and with various levels of detail, the information is converted into a standard format, the Data Format for the Cloud Workloads Archive (see Appendix B). The data might already have been anonymized by the source of the data, otherwise it is anonymized during the conversion into the Data Format for the Cloud Workloads Archive. All tools in our toolbox make use of this data format.

1.3 Goals

Workload modeling is instrumental in the evaluation of existing MapReduce systems, and to developing and comparing of new and enhanced MapReduce systems.

[2] `http://hadoop.apache.org/`
[3] `http://remembersaurus.com/mincemeatpy/`
[4] `http://blog.last.fm/2009/04/06/mapreduce-bash-script`
[5] `http://aws.amazon.com/elasticmapreduce/`

However, few comprehensive workload models exist for MapReduce systems.

Our goal is to develop a comprehensive and realistic workload model for MapReduce systems. To this end we introduce the Cloud Workload Archive Toolbox. This toolbox is able to perform analyses on MapReduce workloads, it extracts models from MapReduce workloads, it generates realistic synthetic MapReduce workloads based on these workload models, and finally it is able to simulate the execution of these synthetic MapReduce workloads.

1.3.1 Research Questions

The main research question for this thesis is:

> *"Is the MapReduce scheduler X better than MapReduce scheduler Y?"*

This question leads to the following sub-questions:

Q1 What are the characteristics of MapReduce workloads?

Q2 How can we model MapReduce workloads?

Q3 How can we generate realistic synthetic MapReduce workloads?

Q4 Which MapReduce scheduler performs best in scheduling a certain workload?

1.3.2 Technical Objectives

The research questions lead to the following technical objectives:

T1 Automate MapReduce workload trace analysis.

T2 Automate MapReduce workload model parameter fitting.

T3 Automate synthetic MapReduce workload generation.

T4 Automate synthetic MapReduce workload simulation.

1.4 Our Approach

Our approach to the problem stated in Section 1.3 is depicted in Figure 1.2. This drawing shows the various steps that are being performed on workload traces by our toolbox:

1. As input we have a trace of a MapReduce workload from (preferably) a real production cluster.
2. This workload trace is converted into the Data Format for the Cloud Workloads Archive.

3. We perform analyses on the workload trace.

4. We fit the parameters of our model to the workload trace.

5. From our model and the fitted model parameters, we generate a realistic synthetic MapReduce workload.

6. We simulate the execution of the generated workload on a MapReduce cluster.

7. We take the trace of the simulation, and use it again as input for step 2.

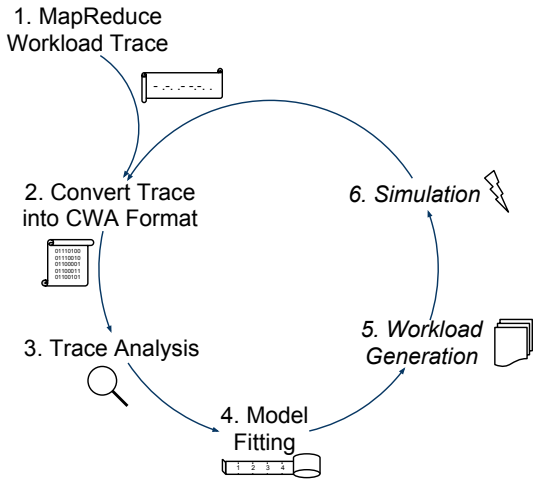


Figure 1.2: The circle of life for a workload trace.

We could of course stay in this circle infinitely, but that will not be very useful. What we actually do depends on the goal. If we want to evaluate schedulers, we generate multiple workloads with increasing load levels in step 5 and we repeat the simulation in step 6 for the different schedulers and the different workloads. We can then analyze and compare the results of the simulations by applying steps 2, 3, and possible 4 on the simulation traces.

1.5 What Has Been Done Before?

In other research, MapReduce workloads have already been analyzed, modeled, generated, and simulated. Publications describing this prior art are surveyed in Chapter 2. We have found that there already have been many attempts at creating MapReduce simulators and schedulers, but that there exist very few publications of in-depth analyses of MapReduce workloads, and that none of the MapReduce workload modeling attempts in the surveyed literature capture as much features as the model we present in Chapter 5.

1.6 Thesis Outline

The remainder of this thesis is organized as follows. In Chapter 2 we provide an overview of the state of the art, i.e., we survey current literature on the subject. In Chapter 3 we present the toolbox we developed, which fulfills the technical objectives set in Section 1.3: automating MapReduce workload trace analysis, modeling, generation, and simulation. In Chapter 4 we take a look at how MapReduce workloads perform in real world cluster settings, by showing the analyses of these traces. In Chapter 5 we present a model for MapReduce workloads. In Chapter 6 we show how our tools can be used to build better systems. And finally, we present our conclusions in Chapter 7.

Chapter 2

State of the Art

In this chapter we present a survey of related literature, to help the reader place this work into context, and to help show what the contribution of this work is.

First in Section 2.1 we survey MapReduce studies. Second, in Section 2.2 we show other non-MapReduce workload modeling studies. And finally in Section 2.3 we survey other non-MapReduce trace archives.

2.1 MapReduce Studies

In this section we survey MapReduce studies. We survey performance evaluation in Section 2.1.1, which at the same time gives some insight in the behavior of MapReduce workloads. We survey models in Section 2.1.2, workload generation in Section 2.1.3, simulators in Section 2.1.4, and finally schedulers in Section 2.1.5.

2.1.1 MapReduce Performance Evaluation

In order to get an idea of the properties of real production MapReduce workloads, we take a look at workload analyses by [2, 3, 4, 5, 6, 7, 8]. The most comprehensive study of a MapReduce workload we have found is the study by Kavulya et al. [3], who analyze a ten months workload trace from the Yahoo! M45 super computing cluster. Kim et al. [7] study the workload of their own MapReduce benchmark instead of a real production workload. Ganapathi et al. [5] and Wang et al. [6] call the sources of their traces respectively a “major web service,” and a “medium-scale Hadoop cluster,” we have our reservations for the quality of these traces, especially we doubt how “medium-scale” the second of the trace sources really is.

Job and Task Run Times

The mean duration of a MapReduce job is around or below 20 minutes by [3, 4, 5, 6, 7], with a maximum observed job duration of seven days (jobs were being killed by a weekly maintenance script) by [3]. The mean duration for tasks is around or below 25 seconds by [4], when looking only at reduce tasks, a mean duration of

around five minutes is shown by [2], with a maximum observed task duration of around one day by [4]. Kavulya et al. [3] show that 95% of jobs complete within 30 minutes, and that completion times follow a long-tailed distribution.

Map vs. Reduce Tasks

MapReduce jobs consist of map and reduce tasks, the ratio between map and reduce tasks is application specific.

The analysis of Yahoo! traces by Kavulya et al. [3] show an average of 154 (std. 558) map and 19 (std. 145) reduce tasks per job, and that 93% of the jobs consist almost entirely out of map jobs. The analysis of Facebook traces by Zaharia et al. [2] show a distribution of job sizes with 39% of the jobs having only one map task, with 30% of the jobs having 2-20 map tasks, with 29% of the jobs having 21-1500 map tasks, and 3% of the jobs having more than 1500 map tasks; the largest job had over 25,000 map tasks. The only job in the simulation by Wang et al. [6] has 480 map and 16 reduce tasks. These three publications all show in all situations less (or none at all) reduce tasks than map tasks.

CPU and Memory Demand

Ghodsi et al. [9] show that the bulk of tasks demand three or less CPUs and two or less gigabytes of memory. The tasks with high memory demand (of up to 9 gigabytes) are mostly reduce tasks.

I/O and Data Locality

Map tasks read input data from the distributed file system, their output data is "shuffled" to reduce tasks, and reduce tasks write their output to the distributed file system.

Chen et al. [8] show the cumulative distribution function for the input, shuffle, and output sizes in a six months Facebook trace. The mean data sizes are surprisingly low in the ranges of hundreds of kilobytes, hundreds of bytes, and megabytes for respectively input, shuffle and output. All these sizes go quickly up into gigabytes in the top 20% jobs, and to terabytes for the top 10% jobs.

Each node in a MapReduce cluster may serve as both compute node and data node, because of this, tasks could be scheduled on (or in the same rack as) the node containing the task's input data. We have "data locality" if the task is executed close to the data.

Wang et al. [6] show for their reference job 98% of the tasks running on a node containing the data, 1% of the tasks running in the same rack as a node containing the data, and 1% of the tasks running farther away than the data. Zaharia et al. [2] plot data locality as a function of the number of maps per job. Same rack locality reaches 90% at about 100 maps per job, and goes up to about 98% as the number of maps per jobs increases. Same node locality reaches 90% at about 7,500 maps

per job and goes up to about 92% as the number of maps per job increases. Zaharia et al. [2] also show data locality as function of file replication level and number of task slots per node.

Cluster Utilization, Failures, and Energy Consumption

There are three more interesting subjects, cluster utilization, failures, and energy consumption, which have each been studied in only one of these publications.

The utilization of a cluster is studied by Kavulya et al. [3]. The studied cluster seems to be underutilized even at peak moments, with a maximum monthly-average node and CPU utilization of respectively about 40% and 10%. Because of this low utilization values, the authors see an opportunity to reduce power consumption if energy-aware scheduling would be applied.

Failures were studied byKavulya et al. [3], the highlights of the failures study are, that 90% of the jobs failed within 150 seconds after the first aborted task, and that most failures occur in map tasks.

Energy consumption is studied by Chen et al. [8], with the goal of reducing power consumption by employing data compression. Based on power consumption measurements of a single node in an experimental setup, they present an algorithm to decide if compression of data would be beneficial.

2.1.2 MapReduce Workload Models

Models for MapReduce workloads are presented by [3, 5, 6, 10]. Unlike the model we present in Chapter 5, these works model only job completion times.

Kavulya et al. [3] use fitting of probability distributions (like in our model), for goodness of fit test they only use the Kolmogorov-Smirnov test. They suggest a run time prediction algorithm which focuses more on jobs in the near past.

Ganapathi et al. [5] use Kernel Canonical Correlation Analysis which maps MapReduce job configuration onto job performance. Job inter-arrival times, input sizes, and data ratios are captured in empirical distribution functions specified by five percentiles. The work is primarily targeted at Hive workloads, and could in principle also be used for generic MapReduce workloads, although in their experiments the prediction performance for generic MapReduce workloads is not as good as it is for Hive-only workloads.

The MapReduce workload models used by [6, 10] describe specific jobs and allow only for a “replay” of the workload, and not the generation of workloads. The model by Wang et al. [6] uses a description of input data as basis and jobs are modeled in a number of CPU cycles as function of the input size.

In Rumen [10] the workload model is essentially just the measured values in the trace, with as exception that for failing jobs, the chance of failure and the run time are captured in empirical distribution functions.

Cardona et al. [11] present a model for the distributed file system in a grid environment, which is not applicable for our work.

2.1.3 MapReduce Workload Generation

Procedures for generating MapReduce workloads are given by [5, 10]. Ganapathi et al. [5] sample values for job inter-arrival time, input size, etc., from the distributions defined by five percentiles by applying linear extrapolation.

Rumen [10] does not really generate synthetic workloads, as we would like to see it. They essentially replay a trace, with the exception that chances for failures and in case of a failure the corresponding run time are sampled from a distribution specified by percentiles using linear extrapolation.

2.1.4 MapReduce Simulators

For our work we need a MapReduce simulator, in order to simulate the execution of our generated synthetic workloads. We have found three publications [6, 11, 12] that present a MapReduce simulator.

Wang et al. [6] present MRPerf[1] a simulator build on ns-2. Features of this simulator include simulation of network traffic at packet level, CPU usage, and disk I/O time. Limitations of this simulator are that it does not support multiple replicas of chunk data, and that it sees disk I/O and computation as distinct phases that do not have any overlap.

Two publications [11, 12] describe simulators built on top of GridSim, which itself has been build on SimJava, a discrete event based simulation package.

Cardona et al. [11] has been developed to evaluate the influence of new scheduling algorithms for the distributed file system on MapReduce in grid environments. It is the only scheduler we found that explicitly takes node availability, and storage space into account.

Hammoud et al. [12] present MRSim[2]. Hammoud et al. state as need for a new simulator, that they were unable to get accurate results with MRPerf [6] and the simulator of Cardona et al. [11], and that Mumak is not able to estimate completion times. Their implementation does simulate multi-core CPUs, and other configuration settings that have an important impact on the performance like merge, copy, and sort parameters. A limitation of MRSim is that it can only simulate single rack clusters.

Mumak [13] is a MapReduce simulator that comes bundled with Hadoop since version 0.21. It does not perform simulation of low-level resources but just replays tasks with the run times specified in the input trace. Its main advantage is that it uses the native Hadoop schedulers.

Gridmix3 [14] is not a MapReduce simulator, it executes synthetic workloads on a real Hadoop cluster. It comes bundled with Hadoop, and has the same input (except for the cluster topology of course) and output formats as Mumak.

[1] `http://research.cs.vt.edu/dssl/mrperf/`
[2] `http://code.google.com/p/mrsim/`

2.1.5 MapReduce Schedulers

Schedulers in Hadoop allocate tasks to slots on the worker nodes. Hadoop comes by default with a FIFO scheduler, and since version 0.19[3] it has support for pluggable schedulers. In Hadoop version 0.21, there are three additional bundled schedulers, namely the Fair scheduler [2] developed by Facebook, the Capacity scheduler [15] developed by Yahoo!, and the Dynamic Priority Scheduler [16] developed by Hewlett-Packard.

Hadoop Bundled Schedulers

The Fair scheduler and the capacity scheduler both have the same goal, which is to share a cluster among users in such a way that production jobs meet their deadlines, and "interactive" jobs have short response times.

The idea in the Fair scheduler proposed by Zaharia et al. [2] is that instead of allocating a tasks as soon as it is it first in line, it might be beneficial to postpone the allocation of the task until a slot with good locality becomes available.

Although we have not found an official publication describing the Capacity Scheduler [15], as one of the bundled schedulers it can not be omitted here. Queues in the system each have a guaranteed capacity, but they are allowed to consume capacity not claimed by others. In addition to this, individual job limitations prevent single jobs to hog a queue. Job-preemption is currently not supported.

Sandholm and Lai [16] propose the Dynamic Priority Scheduler, the idea of this scheduler is that every user in the system has a budget and pays for the use of the cluster. Per time unit each user is allocated a fraction of the cluster which is the same as the fraction of his bid to the total sum of all bids in that time unit.

Other Schedulers

Zaharia et al. [17] propose the LATE, scheduler which attempts to minimize response time for jobs. The scheduler looks at the run times of the tasks for a job, and identifies stragglers, i.e., tasks that are running significantly longer than other tasks for the job. The scheduler uses excess capacity of the cluster to launch duplicates of these stragglers, in the hope that the duplicate would finish earlier than the original straggling version of the task.

Ghodsi et al. [9] propose a scheduler which is based on the notion of Dominant Resource Fairness, it "simply applies max-min fairness across user's dominant shares." tries to fit tasks on worker nodes in such a way that the CPU or memory demand on on each node maximize consumption of a specific resource instead of a fixed number of slots.

Polo et al. [18] have developed a scheduler which predicts the run time of tasks based on the run times of previous ran tasks, and then allocates only that many resources as is needed to meet the jobs deadline. This applies to both map and reduce

[3] `https://issues.apache.org/jira/browse/HADOOP-3412`

tasks. In absence of any information on the duration of map tasks, the maximum allocation is initially set to the number of remaining map tasks. In absence of any information on the duration of reduce tasks, the scaled mean duration of the map tasks is used.

Distributed File system Schedulers

Two of the studied papers focus on the distributed file system.

Chen et al. [8] try to minimize energy consumption and improve performance by compressing data. Blindly compressing all data requires more energy than just storing all data, in all but the shuffle phases. As a solution to this problem, the authors present an algorithm for deciding whether or not to compress data during a phase. It is concluded that energy consumption savings of up to 60% can be archived for jobs that are heavy on reads, or for jobs with highly compressible data.

Cardona et al. [11] envision a MapReduce implementation for grids. The main problem is the low availability of nodes. The default replica placement strategies are targeted at a cluster environment, in a grid environment replicas of data chunks must be carefully placed so that data is likely to be always available. The authors propose a scheduling algorithm for the placement of data chunk replicas. The algorithm sort nodes by availability and attempt to store replicas on the highest available nodes with enough free space, optionally it checks if the nodes processing power is above a required level.

2.2 Other Workload Modeling Studies

A more thorough overview of workload modeling studies are given by Iosup [19], Sec . 4.5.2, p. 68, these include Leland and Ott [20], Calzarossa and Serazzi [21], Balter and Downey [22], Feitelson [23] (Feitelson and Nitzberg [24]), Jann et al. [25], Lublin and Feitelson [26], Li et al. [27], Medernach [28], Song et al. [29], Li and Muskulus [30], Iosup et al. [31], and Iosup [19], Sec . 4.4.

Feitelson [32] is writing a textbook on the modeling of computer system workloads for performance evaluation, which is freely available[4].

2.3 Other Trace Archives

We are not the only ones who are collecting workload traces. Other notable existing archives of traces are the Parallel Workloads Archive by Feitelson [33], Failure Trace Archive[5] by Kondo et al. [34], the Grid Workloads Archive[6] by Iosup et al.

[4] `http://www.cs.huji.ac.il/~feit/wlmod/`
[5] `http://fta.inria.fr`
[6] `http://gwa.ewi.tudelft.nl`

	Analyzing									Modeling							Generating		Simulating						Scheduling								
	Run Times	# Maps/ #Reduces	CPU/Memory Demand	I/O	Data Locality	Cluster Utilization	Failures	Energy	Trace Source	Fitting	KS Test	Empirical CDF	KCCA	Job Makespan	Other	HDFS	Not just Replay	Multiple Classes	Data Locality	Network Latencies	Key Distribution	HDFS	Node Availability	Build On	Dominant Resource	Budgeting	Speculative Execution	Preemption	Run Time Prediction	Delay Scheduling	Prefer Data Locality	HDFS	Compression
Kavulya et al. [3]	+	+	–	–	–	+	+	–	Yahoo!	+	+	–	–	+	–	+																	
Ghodsi et al. [9]	–	–	+	–	–	–	–	–	Facebook																+	–	–	–	–	–	–	–	–
Hindman et al. [4]	+	–	–	–	–	–	–	–	Facebook																								
Kim et al. [7]	+	–	–	–	–	–	–	–	(Benchmark)																								
Ganapathi et al. [5]	+	–	–	–	–	–	–	–	major web service	–	–	+	+	+	+	+	+	+														
Wang et al. [6]	+	+	–	+	+	–	–	–	medium cluster	–	–	–	–	+	–	+			+	+	–	+	–	ns-2								
Cardona et al. [11]									SETI@Home	–	–	–	–	–	–	+			–	+	–	+	+	GridSim	–	–	–	–	–	–	–	+	–
Hammoud et al. [12]																			+	+	+	+	–	GridSim									
Zaharia et al. [2]	+	±	–	–	+	–	–	–	Facebook																–	–	–	–	–	+	+	–	–
Zaharia et al. [17]																									–	–	+	–	+	–	–	–	–
Polo et al. [18]																									–	–	–	±	+	–	–	–	–
Chen et al. [8]	–	–	–	+	–	–	–	+	Facebook																–	–	–	–	–	–	–	+	+
Sandholm and Lai [16]																									–	+	–	+	–	–	–	–	–
RUMEN [10]										±	–	+	–	+	+	–	–	–															
MUMAK [13]																			+	±	–	–	–	Hadoop									
Gridmix3 [14]																			±	±	±	±	±	Hadoop									
Hadoop Default Scheduler																									–	–	+	–	–	–	+	–	–
Capacity Scheduler [15]																									–	–	–	–	–	–	+	–	–

Table 2.1: Comparison of studied MapReduce Publications. The symbols " ", "+", "–", and "±" denote respectively "not covered", "treated", "not treated", and "partially treated".

[35], and the Peer-to-Peer Trace Archive[7] by Zhang et al. [36]. A more thorough overview of archives is given by Iosup [19], Sec . 3.5, p. 33.

[7] http://p2pta.ewi.tudelft.nl/

Chapter 3

MapReduce Analysis Toolbox

We developed a toolbox for analyzing and modeling MapReduce workloads, and for generating and simulating the execution of synthetic MapReduce workloads. This toolbox can be regarded as the fulfillment of the technical objectives T1, T2, and T3 stated in Section 1.3. In Chapter 6 we introduce a separate toolbox for MapReduce simulation using super-computers. We depict the work-flows of the MapReduce Analysis and the Simulation toolboxes in Figure 3.1.

This chapter describes the abilities of this toolbox, including how it allows the user to convert traces of MapReduce executions into a standardized format (see Section 3.1), to perform analysis on workload traces (see Section 3.2), to model the workload in these traces (see Section 3.3), to generate realistic synthetic workloads (see Section 3.4), and to simulate the execution of these synthetic workloads (see Section 3.5).

3.1 Trace Import

To enable our toolkit to work on workload traces from various sources, we need a default format to store this information. Even though many clusters use the same MapReduce implementation, MapReduce trace data still comes in a large variety of formats. This variety of formats is caused by the different ways cluster administrators have invented to extract data from the cluster, and by the need to anonymize and censor the data in order to protect company secrets. In order to cope with this large variety of trace file formats, we use the Data Format for the Cloud Workloads Archive throughout the toolbox.

3.1.1 Data Format for the Cloud Workloads Archive

The Data Format for the Cloud Workloads Archive [37] (CWA format), is a data format which captures anonymized MapReduce workloads with as much detail as possible, while also supporting non-MapReduce cloud workloads. Anonymization is a side-effect of a recommendation in the CWA format, namely that strings iden-

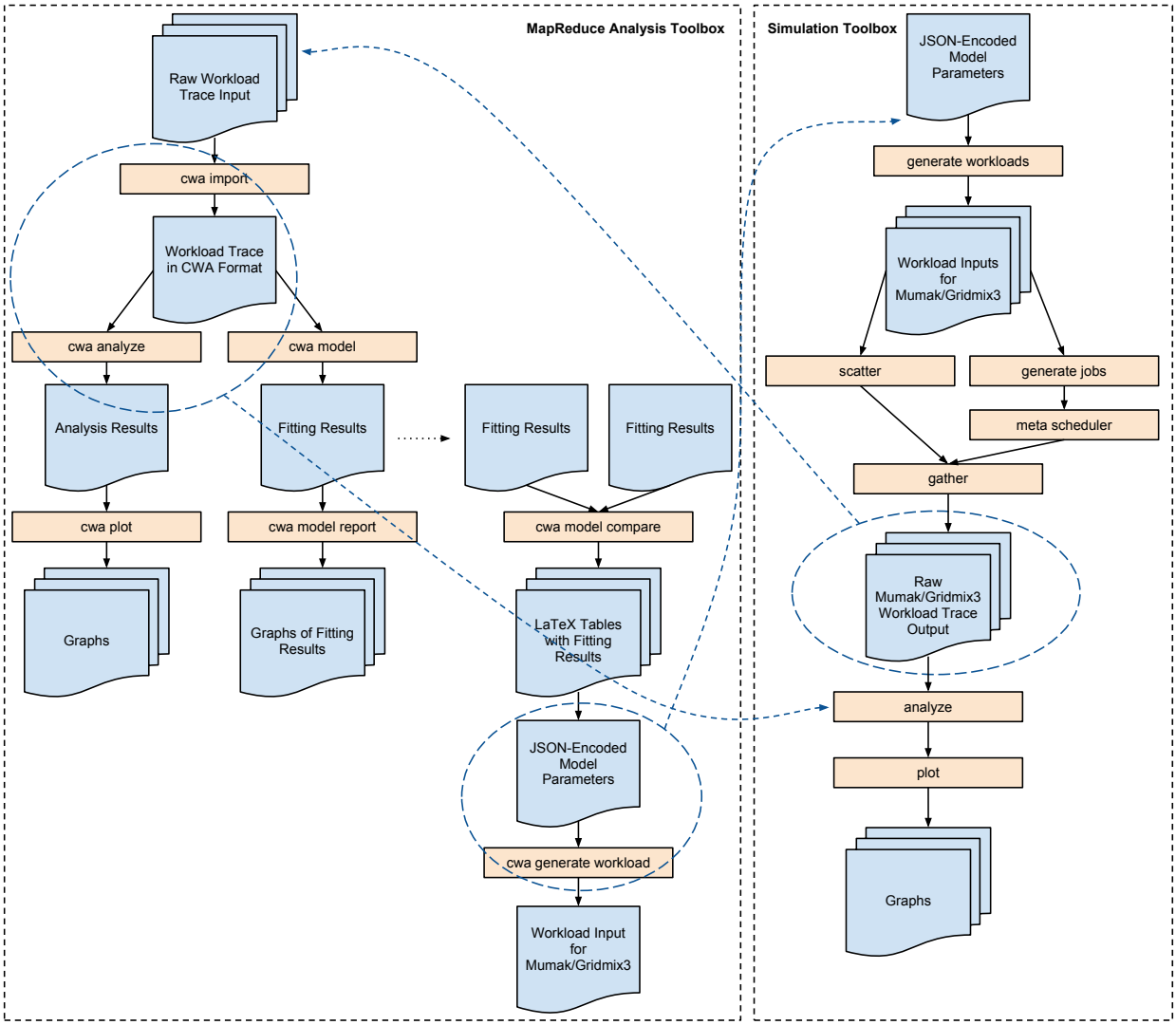


Figure 3.1: The MapReduce Analysis and Simulation Toolboxes.

tifying users, applications, etc., should me mapped to integer values to reduce file sizes. The CWA format is based on the proven Grid Workloads Archive[38] and Parallel Workloads Archive[39] data formats. The definition of the CWA format does specify the fields, it does not specify the how the data should be stored. The toolbox assumes that all CWA format data will be stored in tab separated files (see Appendix B). With the exception of this single subsection, all references to the CWA format in this document refer to these tab separated values files.

3.1.2 Import Scripts

In order for a trace to be used with the tools in the toolbox, we first need to convert the trace into the CWA data format. The toolbox itself only includes an importer for the Hadoop log. As most of the traces come in a custom format, we write a custom conversion Python script for each of these traces. The toolbox provides some tools, such as a writer for the CWA data format, to aid the development of the conversion scripts.

Depending on the format and contents of the original trace data, writing a conversion script can be non-trivial. An example of this non-triviality can be found in the conversion of the SN1 traces (see Section 4.2.1), the SN1 traces contain a textual description of the application which differs for each execution of the application. The CWA data format requires an executable identifier which is unique for each application but stays the same for the various executions of the same application. In Section 3.1.3 we present our solution to this classification problem.

3.1.3 Executable Identification

Although the SN1 traces (see Section 4.2.1) did not contain an executable identifier, these traces did contain a job description field which could be exploited to classify jobs to distinct executable identifiers. In order to classify the executables based on the job description field we wrote a few rules, in the form of regular expressions, to match the job description to an executable identifier. While choosing these rules, we had in mind the trade-off between precision and the number of rules. A smaller number of rules gives a more concise picture of the behavior of the applications, while more rules give a more precise result. The question remains if this perceived precision is really justified by the quality of the rules, we have therefore chosen the first of these two options. In the next three subsections we present the tools we developed to enable us to write these rules: `count_unique`, `applications`, and `cdf`.

The `count_unique` Tool

We wanted to write rules that each match an as large as possible number of job descriptions that seem to be referring to the same application. To this end we first looked at counting occurrences of distinct values of job descriptions, so that we

could match the descriptions with the highest occurrence count first. The main problem we encountered in this process, was that many job descriptions include a serial number or a date, which turns the job descriptions into distinct values.

In order to cope with the problem of having serial numbers and dates in the job descriptions, we included a feature in the count_unique tool to replace all numbers by the number sign ("#"), and to replace all day and month names into three at-symbols ("@@@"). These options can be used to aggregate job descriptions only differing in date or (serial) number into distinct values. On top of that we also included an option to convert all characters to lower-case characters, so that the matching will in effect be case-insensitive.

The tool takes as input a tab separated values file and the number of the column containing the values of interest. The tool outputs a tab separated values file with in two columns the value of the (altered) job description and the count of occurrences in the trace. An example of this output file is shown in Listing 3.2.

The `applications` Tool

To help produce a list of regular expressions for matching executables in the inconcise job descriptions in the workload trace, this tool counts the number of matches for each of the rules in a list of regular expressions. Job descriptions are classified by the first rule that matches.

This script takes as input a file containing the rules in the form of regular expressions, and a tab separated values file containing job descriptions combined with the count each of these description occurs in the to be studied workload trace. Not entirely by coincidence, this input is exactly the same as the output of the `count_unique` tool. The output is the list of rules, with for each of the rules the total sum of the counts of the matching job descriptions.

Examples of files containing rules, input, and output for this tool are shown in respectively Listings 3.1, 3.2, and 3.3.

Listing 3.1: Example `applications` rules

```
1 ^.*o+.*$
2 ^.*a+.*$
```

Listing 3.2: Example `count_unique` output, `applications` input

```
1 aap␣␣␣␣␣1
2 noot␣␣␣␣2
3 mies␣␣␣␣1
4 boom␣␣␣␣1
5 roos␣␣␣␣1
6 vis␣␣␣␣␣1
```

Listing 3.3: Example `applications` output

```
1 ^.*o+.*$␣␣␣␣␣␣␣␣4
```

```
2 ^.*a+.*$ 1
3 Other 2
```

The `cdf` Tool

We would like to plot the cumulative distribution function (CDF) of the matched rules, so we can see how much of the jobs are matched by the rules. The `cdf` tool can be used to construct an input data file for plotting a CDF/PDF graph using Gnuplot. The tool expects a tab separated input file with in the first column an identifier and in the second column a count (like the output files of the `count_unique` and `applications` tools). The script outputs a tab separated file with four columns: rank, identifier, normalized count, and cumulative normalized count.

The normalized count n_i for the count c_i for all the counts $\mathbf{c}$ is calculated as $n_i = c_i / \sum_{j=0}^{|\mathbf{c}|-1} c_j$. After all the normalized counts are calculated, the list is sorted in descending order. As an exception on the sorting process, the count identified by the name "Other" is always outputted last. The resulting list is written to the output file, while this is being done, the cumulative value is calculated by summing all previous values.

3.2 Trace Analysis

The toolbox provides the `analyze` tool to analyze MapReduce workload traces. From the traces this analysis tool extracts information on various metrics both over-time, and all-time, optionally broken down by one or more properties, for example, the run time of tasks over time broken down per executable. This process is explained more in-depth in Chapter 4. Using the data resulting from this analysis, the toolbox can automatically plot graphs using Gnuplot.

3.2.1 The `analyze` Tool

The workload trace `analyze` tool needs a few things to work. First, of course, it needs an input workload trace in the CWA format. Second, we need to specify which metrics we want to analyze and which breakdowns we want to be computed, e.g., run time per executable. The metrics and breakdowns (see Section 4.1 and Appendix B) can be specified in a configuration file (see Section A.4.2) on a per-trace basis or otherwise default settings are used.

This tool relies heavily on the two utility classes `TimeLine` (see Section 3.2.2) and `CStats` (see Section 3.2.2). The time line class is not quite unlike a histogram, it discretizes time based on a given interval length, anything stored on the time line is being aggregated into the interval it belongs to. The stats class aggregates multiple numerical values, it is then able to compute statistics of these values, like for example the mean, the standard deviation, and arbitrary percentiles. The time

line class has the ability to either compute the sum of all values in an interval, or to use this stats class to aggregated the values.

The analyze tool uses an instance of the `TimeLine` class to store all information over time, and indeed, all values are aggregated by instances of the `CStats` class. As the workload trace file is processed, the values of the metrics of interest are stored on the time line, also for each breakdown-metric value combination separate values are stored on the time line, see Figure 3.2 for pseudo code describing this. Next to this, we count the number of running jobs and tasks over time, these counts we also calculate by breakdown. The counting is done by adding the value "1" to every time line interval, during which the job or task is actually running, see Figure 3.3 for pseudo code describing this.

```
1 for j in jobs:
2   for m in metrics:
3     timeline[m].add(j.start, j.get(m))
4     for b in breakdowns:
5       timeline[m,b,j.get(b)].add(j.start, j.get(m))
```

Figure 3.2: Pseudo-code for analyzing over time.

```
1 for j in jobs:
2   for moment in j.start to j.finish step interval:
3     timeline.add(moment, 1)
```

Figure 3.3: Pseudo-code for counting running jobs over time.

When the whole trace has been processed, the output is written to many files. (For all job and task metrics we output the all-time CDF/PDF data, we output per time interval the sums and percentiles. All this information is also outputted again for each breakdown value.)

3.2.2 Utilities

In this section we introduce the utility classes `TimeLine` and `CStats`, which we have used in, among other, the `analyze` tool.

The `TimeLine` Class

We have created the class `TimeLine` for the computation of statistics over time. Inside this class, time is discretized in intervals of a length specified by the `binwidth` parameter. Each interval is identified by an internal time index i which can be calculated from the time value t and the interval length w by performing a simple division $i = \lfloor t/w \rfloor$. The data are kept in a dictionary, using the index i as the key,

the choice for a dictionary was made to efficiently handle sparse time lines. The class can manage multiple variables of interest in the same instance, by using a key value while adding data. In each interval a dictionary is used to store data identified by these keys.

The main purpose of this class is to aggregate data per time interval. It has two separate ways to aggregate the data. First, the class performs a simple summation of all values in an interval. Second, it can use the `CStats` class to perform basic statistical analyses on the data in the interval. As final feature, the `TimeLine` class is able to provide these statistics for time intervals which are a multiple of the initial time interval, this is done by simply summing the values at the index positions covering the requested time interval.

The `CStats` Class

The `CStats` class is able to calculate various statistical properties for a variable of interest from a list of values. Every time a value is added to an instance, a couple of values are updated: the value count, the sum of the values, the sum of the squared values, the maximum and minimum value. These values are used to calculate, next to the minimum and the maximum value, also the mean, the standard deviation, and the coefficient of variation.

Optionally, if the `bKeepValues` parameter is set, all values are kept in an internal list $\mathbf{v}$. This list of values can then be used to calculate arbitrary percentiles, including the median. The p-th percentile is calculated by picking the $\lfloor |\mathbf{v}| \cdot 100/p \rfloor$-th element from the sorted list of values. The median, the 50th percentile, has a special treatment: If the total number of values is even, then the average of the two values in the middle of the list is calculated, e.g., $(\mathbf{v}_{|\mathbf{v}|/2} + \mathbf{v}_{|\mathbf{v}|/2-1})/2$.

3.3 Workload Model Parameter Fitting

The MapReduce workload models we present in Chapter 5 make use of probability distributions. In order to "model" a workload, we need to find distribution functions and their parameters that "fit" the data in the workload trace well. To this end we have developed a tool in Python, it uses maximum-likelihood functions for various probability distribution functions, and it selects the best fitting distribution for each modeled property, based on the values resulting from the goodness of fit tests and the D-statistic. This process is explained in-depth in Chapter 5. The result of this tool can be used to construct a JSON encoded file, containing the fitted distributions and their parameters.

In this tool we use the multiprocessing package for parallelization of the modeling work, and we use the pickle package to store the intermediate and final output. Pickle – also used for inter-process communication by the multiprocessing package – uses the cStringIO package which has a 2 GiB size limit. To cope with this size limit we have implemented a wrapper around pickle, which partitions a data

structures if needed. The modeling tool is the only tool in the toolbox that makes use of the external Python libraries NumPy and SciPy.

3.4 Realistic Synthetic Workload Generation

The toolbox contains a tool to generate realistic synthetic MapReduce workloads. This process is explained in-depth in Chapter 5. The tool takes as input the JSON encoded model file resulting from the model parameter fitting tool. As output it generates a workload file which can be used by Mumak and Gridmix3 as input.

The non-trivial part of this tool is that it is able to "brute-force" a specified load level. It generates a workload and calculates its load level, if the load level is not satisfying, the inter-arrival times will be multiplied by a factor. We perform a binary search for a factor resulting in a satisfying load level. To speed-up this process we reset the seed of the random number generator at the start of every iteration, unless we are no longer able to improve the load level by adapting the inter-arrival time multiplication factor.

3.5 Simulation

The CWA toolbox itself does not include a MapReduce simulator. We select an existing MapReduce simulator in Chapter 6, and we use this simulator to perform simulations of the execution of generated workloads. Our toolbox include tools to generate workloads for this simulator, and to convert the output of the simulations into the CWA data format for further analysis.

In Chapter 6 we perform a large number of simulations, and introduce to this end a separate toolbox for MapReduce simulation using super-computers.

3.6 Concluding Remarks

In this chapter we have introduced the Cloud Workloads Archive Toolbox, and explained its inner workings. This toolbox has been made available as open-source software. In Appendix A we provide instructions on how to obtain this toolbox, and we provide a short introduction to the usage of the toolbox.

Chapter 4

Workload Analysis

In this chapter we present the analysis of the real-world workload traces we obtained, this provides an answer to research question Q1: "What are the characteristics of MapReduce workloads?"

The remainder of this chapter is organized as follows. In Section 4.1 we present an overview of the metrics and breakdowns we use. In Section 4.2 we present the studied workload traces.

4.1 Metrics and Breakdowns

In the context of workload trace analysis we mean with metrics the properties we wish to study, like run time, wait time, inter arrival time, CPU usage, memory usage, disk usage, and network usage. With breakdowns we mean properties, like status, queue, user, and executable, for which we calculate separate statistics. For example an analysis of the run time broken down by user, gives the same statistics as the "global" statistics, for every single user in the system. The properties available in the CWA data format are shown in Appendix B.

The metrics and breakdowns that we want to analyze may differ for jobs and tasks. In tasks for example you may want to have a breakdown by task type (map or reduce), this specific property is not available for jobs. Using our toolbox it makes only sense to use properties with a small amount of discrete values as breakdown, as for every single value of the property the entire analysis will be performed.

4.1.1 Notable Metrics

We depict job and task time metrics in Figure 4.1. We give a short description of these and other notable metrics.

Inter-Arrival Time The inter-arrival time is, as depicted in Figure 4.1, the time interval between the arrivals of jobs or tasks in the system. This information is not as such available in the workload traces but can be extracted from the traces. It is easy to calculate the inter-arrival time – we only need to calculate

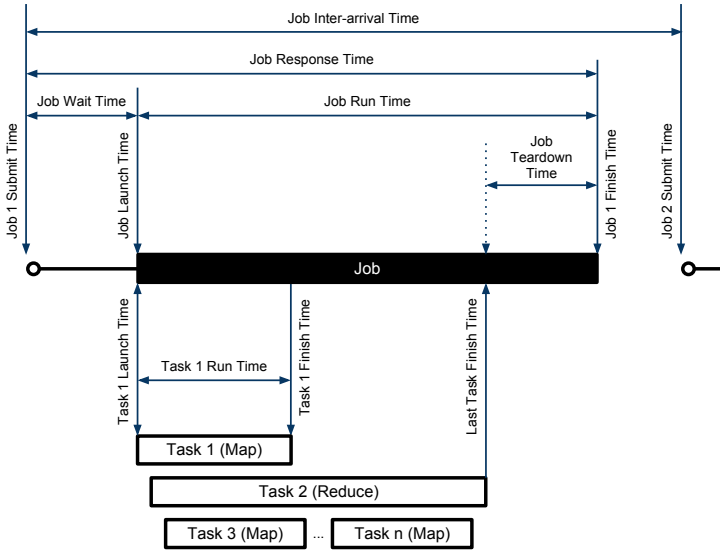


Figure 4.1: Job and task time metrics.

the difference between two subsequent submit times. This requires a trace sorted on submit times.

Wait Time The wait time is, as depicted in Figure 4.1, the time that a job has to wait after being submitted until it is started, i.e., the first task is started. As supported by Figure 4.22, in MapReduce systems the wait times are generally very low.

Run Time The run time is, as depicted in Figure 4.1, the wall clock time elapsed since the job or task is started, until it is finished. For jobs this means until the last task has been finished.

Response Time The response time is, as depicted in Figure 4.1, the wall clock time elapsed since the user submitted a job, until the job finished.

Slowdown The slowdown is a factor that indicates how much longer a job has run than in the most ideal scenario. For example, the absolute minimum time that a job needs to complete is the run time of its longest task; in this case the slowdown is the quotient of the response time and the run time of the longest task.

Number of running Jobs As shown in for example Figure 4.2 we calculate the number of running jobs over time; in this particular example a breakdown by application is made.

CPUs The property CPUs is the count of the number of CPU cores used in total for a job or task.

Total Wall Clock Time The name "Total Wall Clock Time" may be somewhat confusing; it means the sum of the wall clock times spent in every single processor core. So if a job would only have had two tasks – running at the same time – that each used four processor cores for one minute, the total wall clock time would be eight minutes, while the run time is likely to be about one minute.

4.1.2 Notable Breakdowns

Status The status of a job or tasks. Was the running of the job or task successful, a failure, or canceled?

Task Type The type of the tasks. In MapReduce there are two main task types, the Map tasks and the Reduce tasks. Depending on the MapReduce implementation there may be additional task types such as Setup and Cleanup.

Executable It may be hard (see Section 3.1.3) to identify executables in a workload trace, but if available, it could reveal interesting information, as different applications are likely to have different behavior.

Queue Queues are most likely to indicate different groups of users. Scheduled production jobs and ad-hoc interactive jobs could be in different queues, and can be scheduled appropriately. The Fair scheduler by [2], for example, attempts to provide an equal share of the cluster to each queue.

User Identifies users in the system, schedulers may use this instead of the queue in their scheduling algorithm.

4.2 Real-World Workload Traces

We have obtained sets of traces from real world MapReduce clusters. The level of detail in a trace differs per set of traces – a trace could have task information aggregated per job, or may contain only the successful jobs. A list of these sets is given in Table 4.1. In this section we describe each of these workload traces.

			Task Information		Failed	MapReduce	Number Of	
Workload Trace		Period	Aggregated per Job	For Each Task	Jobs	Only	Jobs	Tasks
SN1	(see §4.2.1)	6 months	+	–	–	+	1,129,193	?
SN2	(see §4.2.2)	9 days	+	–	+	+	60,978	9,365,863
Yahoo! M	(see §4.2.3)	2 weeks	+	+	+	+	28,248	27,317,243
Google	(see §4.2.4)	29 days	+	+	+	–	667,992	44,920,671

Table 4.1: Overview of the obtained real-world Workload Traces, the symbols "+", "–", and "?", depict respectively "available", "not available", and "unknown".

4.2.1 Social Network 1

We have obtained traces from a production MapReduce cluster of a large unnamed social network company. We refer to these traces as "Social Network 1" or SN1. Basic statistics of these traces are shown in Table 4.3. These traces cover a period of six months of a Hadoop cluster and a total of 1,129,193 jobs; we have no data to calculate the number of tasks. Although no individual task information is available, the traces do contain detailed task I/O information, aggregated per job. These traces have more peculiarities: they contain only successful jobs, and they do not contain application or user identifiers.

In order to identify the applications in the logs, we use the "JobName" field, which contains a textual description of each job. The values in the "JobName" field are matched against five regular expressions (see Table 4.2) and assigned an application identifier based on the matching regular expression. While choosing these rules, we had in mind the trade-off between precision and the number of rules, a smaller number of rules gives a more concise picture of the behavior of the applications, but more rules give a more precise result.

Rule	Application	Regular Expression	Matches
0	Copier	`(?i)^(recovery mode )?\S+\scopier\s.*$`	680212
1	Insert	`(?i)^insert.*$`	142985
2	From	`(?i)^from(\s|\().*$`	90111
3	Select	`(?i)^select\s.*$`	48542
4	Columnset Loader	`(?i)^(hourly\s)?columnset\sloader.*$`	42283
–	Others		125060

Table 4.2: Application identification rules.

Running Jobs

Figure 4.2 shows the cumulative counts of all running jobs per application type over time. In this graph we observe that the number of unmatched "others" jobs is quite small compared to the matched jobs; this shows that the five chosen rules for application classification cover a large part of the jobs.

We find a large amount of "copier" jobs in the first two months of the traces. We hypothesize that during that time the system was being loaded with data. As support for this hypothesis, we need to see a significant usage of I/O resources by these jobs during this period.

Job I/O

We have calculated the total amount of I/O by the jobs over time. This has been done by summation of all input and output for jobs for each time unit. Figure 4.3 shows the cumulative amounts of I/O for all jobs per application type over time. In this graph we observe that the amount of I/O used by the unmatched "others"

jobs is small compared to the total I/O, but not as small as expected from the small fraction of running unmatched jobs; the unmatched jobs seem to be relatively heavy on I/O.

The jobs matching rules 2 and 1, the “from” and “insert” jobs, are the largest I/O consumers. And contrary to what we hypothesized, the “copier” jobs do not seem to generate a significant amount of I/O during the first two months, they are actually the least I/O consuming of the classified jobs, therefore the hypothesis must be false. However, after the second month, the I/O consumption of the majority of the “copier” jobs is increased by about a factor 1000 as is visible in Figure 4.5a by the steep increase of the 50th percentile, and the I/O ratio changes resulting in more output per input as is visible in Figure 4.5b – it seems that a setting was changed to have the work done with fewer job runs.

Job Run Time

For the SN1 traces, the cumulative runtime split per application is shown in Figure 4.4. The total runtime of the “copier” jobs seems to increase after the two month period described above (also see Figure 4.5c), although a smaller number of “copier” jobs run. The “copier” jobs need to have an increase of runtime after the first two months, this is likely caused by the increase in I/O consumption.

The distribution of the jobs run times in Figure 4.23a shows that, although one job ran for little over one day, most of the jobs in this workload have very short run times: 50% of the jobs finish in under 35 seconds, 66% of the jobs finish in under 70 seconds, and 90% finishes in under 6 minutes.

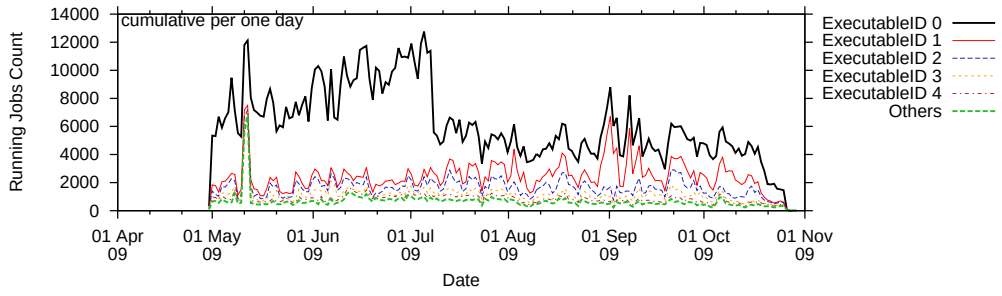


Figure 4.2: SN1, cumulative running jobs by application.

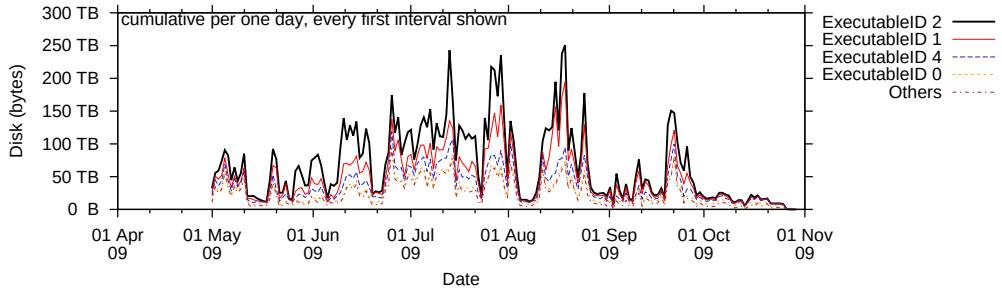


Figure 4.3: SN1, cumulative job I/O by application.

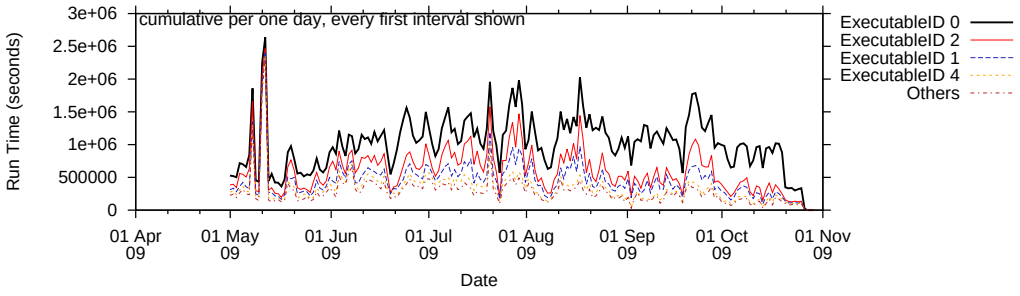
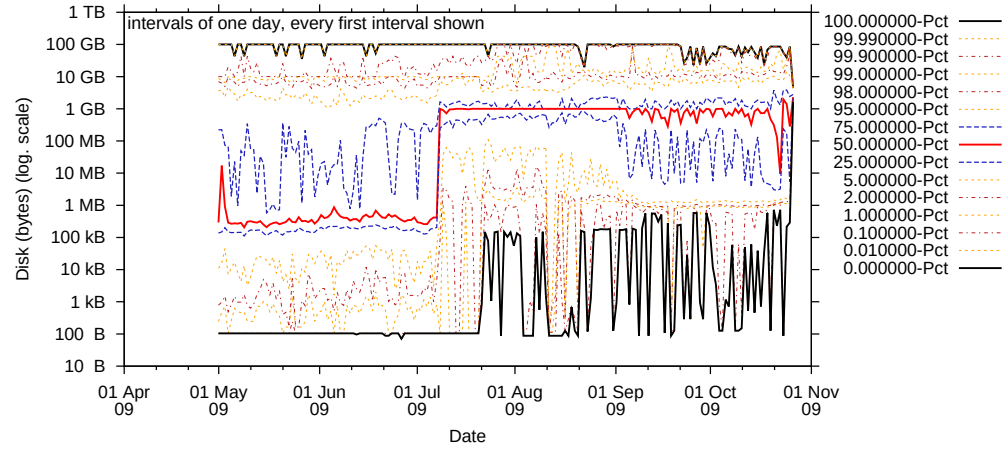
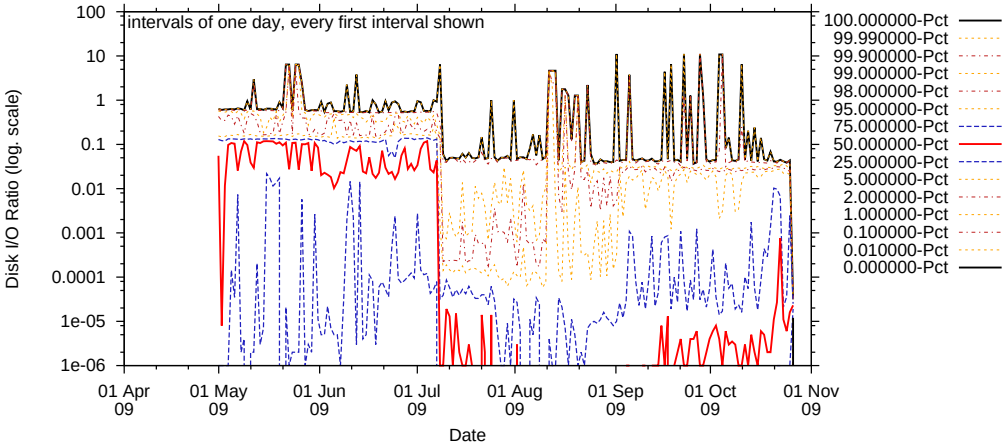


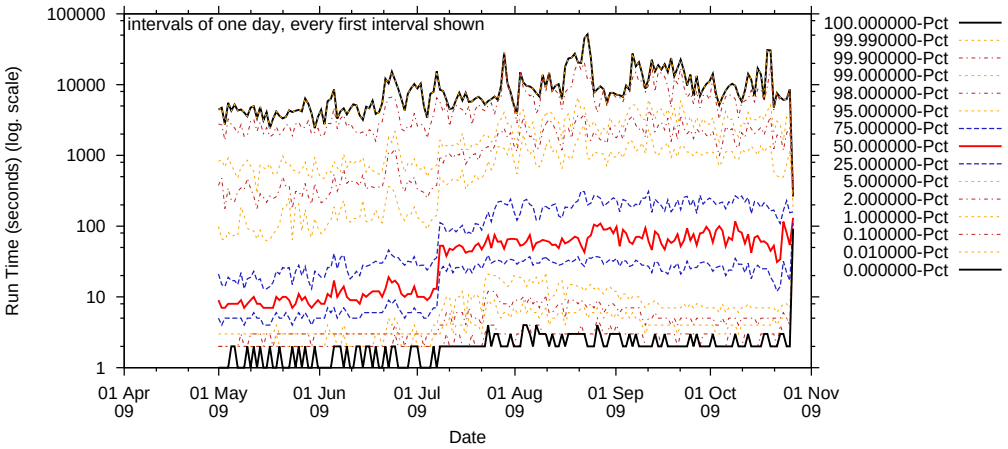
Figure 4.4: SN1, cumulative job runtime by application.



(a) I/O consumption.



(b) I/O ratio.



(c) Run time.

Figure 4.5: SN1, “copier” jobs.

4.2.2 Social Network 2

We have obtained traces from a production MapReduce cluster of a large unnamed social network company. We refer to these traces as "Social Network 2" or SN2. Although, these traces do only contain information per job, they contain task information summarized per job – especially the number of tasks, failed tasks, and killed tasks. These traces cover a period of ten days of a Hadoop cluster and include a total of 60,978 jobs and 9,365,863 tasks; for the last three days there is no task-level information available. There is no information available that can help to identify applications in the traces.

Job and Task Status

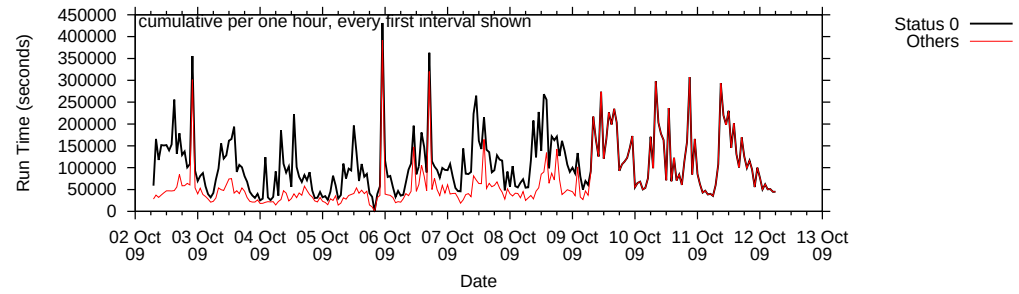
The trace does contain status information. This status information is available in two ways: for the jobs the status is available as a coded field, for the tasks the total number of failed and killed tasks is available next to the total number of tasks. The encoding of the status field in these traces is as follows: successful jobs have status 0, the "others"-jobs are the failed jobs.

The job counts in Figure 4.7a shows a very large amount of failing *jobs*. Most of these failing jobs do not seem to spawn any *tasks*, as can be seen in Figure 4.7b that shows that the large majority of the tasks comes from the successful jobs, and only a fraction from the failed jobs.

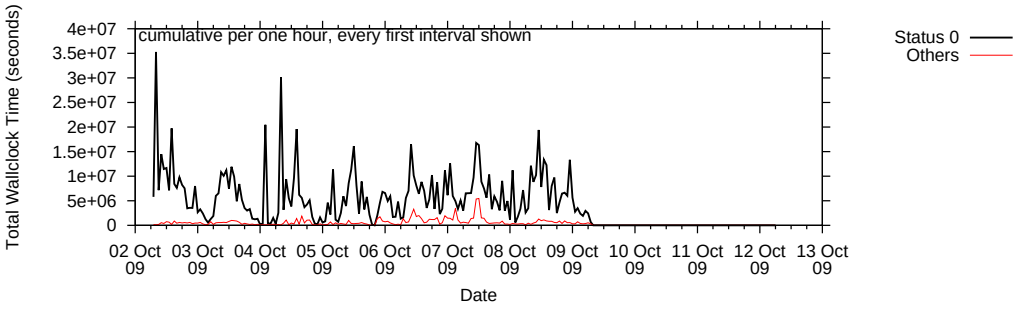
The amounts of failed and killed tasks, as shown in respectively Figures 4.7c and 4.7d, is very small compared to the total number of tasks, as shown in Figure 4.7b. Interesting to note is that only the successful jobs have failed or killed tasks.

Job Run Times

The distribution of the jobs run times in Figure 4.23b shows that, although one job ran for almost 4 days, most of the jobs in this workload have short run times: 50% of the jobs finish in under 1.5 minute, 66% of the jobs finish in under 3 minutes, and 90% finishes in under 12.5 minutes. Figure 4.6 shows that, although the failing jobs run fairly long (as indicated by the job run time), they consume only little of the cluster (as indicated by the total wall clock time).

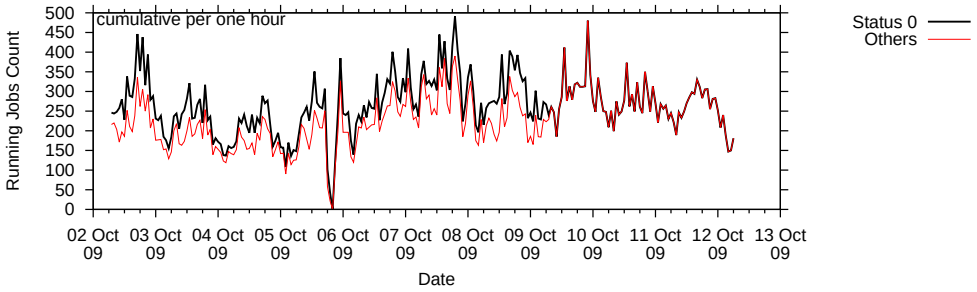


(a) Run time.

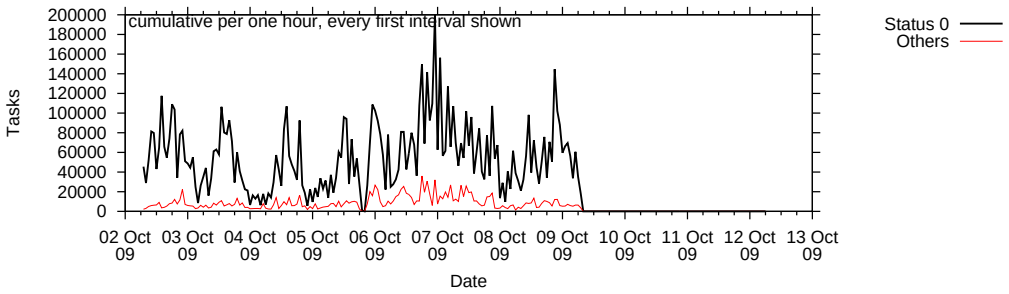


(b) Total wall clock time.

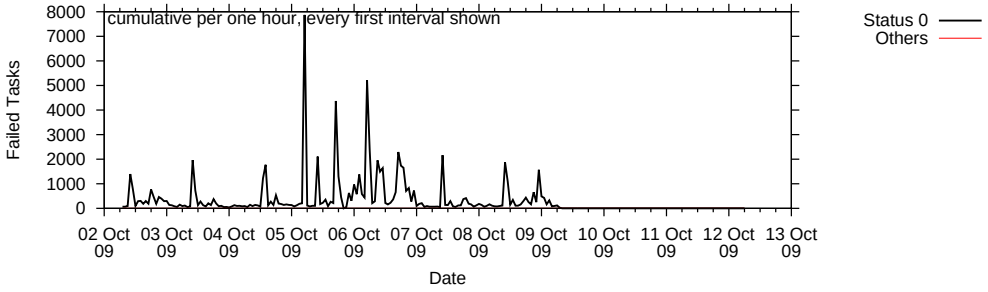
Figure 4.6: SN2, cumulative job times by status.



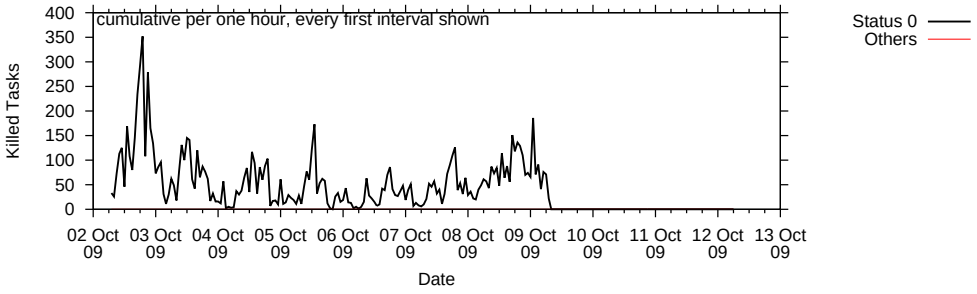
(a) All jobs.



(b) All tasks.



(c) Failed tasks.



(d) Killed tasks.

Figure 4.7: SN2, cumulative jobs and tasks by status.

4.2.3 Yahoo! M-Cluster

The Yahoo! M-Cluster traces we obtained cover a period of two weeks of a Hadoop cluster and include a total of 28,248 jobs and 27,317,243 tasks. These traces contain information at job level as well as for individual jobs. The traces identify for jobs both the executable and the user, in anonymized form. Basic statistics of these traces are shown in Table 4.6. This is the only workload trace where we can actually compare map and reduce tasks.

Jobs

We show the number of running jobs over time in Figures 4.8-4.10. These graphs seem to show that most of the time the system is underutilized. The spike of over 2000 running jobs at 14 March, shows that the capacity of the cluster is larger than the actual use, most of the time no more than 250 jobs are running during the 1 hour counts. This claim is however not very strong, as running many very short jobs rapidly after each other could cause a similar spike on a smaller cluster. In face, we see in Figure 4.11 that this spike did cause large cumulative run time, but not as much as the bursts around 9-11 March.

By Application For the Yahoo! M-Cluster traces, the cumulative number of running jobs per application is shown in Figure 4.8. The application identified as Rule 0, is the only application that runs a significant amount of jobs, except for some spikes of "other" jobs on 11 and 14 March.

By Status Figure 4.9 shows the number of running jobs by status. We see that almost all jobs succeed and only a small fraction of the jobs fails. The logs do not show any canceled jobs during this period.

By User Figure 4.10 shows that although the user with the largest consumption (User 0) is a large consumer compared to the other users, the cumulative usage of the less consuming users is not negligible.

Job Run Time

We have broken down the run time by application, by status and by user. We see bursts of high cumulative run times around 9-11 March.

By Application Figure 4.11 shows clearly that application 0 has by far the largest cumulative run time. A single application dominating the cluster looks like a scheduled production job, but the behavior is too irregular for a scheduled job.

By Status Figure 4.12 shows that although the number of failed jobs is negligible low, the run time consumed by failed jobs can be less, although still, negligible.

By User Figure 4.13 shows that two users generate around 9-11 March jobs that run significant long compared to the usual load – using the same application.

Tasks

We show the number of running tasks over time in Figure 4.14b.

Task Run Time We see in Figure 4.15 that the map tasks require cumulatively more time than reduce tasks, so either map tasks run on average longer than reduce tasks, or there are more map tasks than reduce tasks. In Figure 4.18 we see the run time distribution for both task types. We see that 50% of the map tasks run in less than 25 seconds, and that about 5% need more than 5 minutes to complete. We see that only 34% of the reduce tasks finish in less than 25 second, 50% finish in under 75 seconds, and 5% need more than 18 minutes to complete. Both the map and reduce tasks show a maximum value of about 44 hours. These observations show that reduce tasks generally run longer than map tasks. We have counted a total of 22,004,024 map tasks and a total of 5,313,212 reduce tasks, so, indeed, there are more than 4 times as much map tasks than reduce tasks in this trace.

Distributed File-system Usage We show in Figures 4.16 and 4.17 respectively the amounts of data read from and written to the HDFS, Hadoop's distributed file-system. We observe a few things in these graphs. First, we see that the tasks generally read much more than that they write. Second, we see that almost only map tasks read data from the HDFS, and that reduce tasks write more data to the HDFS than map tasks. Third, in the CDF we observe that about 40% of the tasks read almost exactly 128 MB data, this indicates a preference for data chunks of 128 MB.

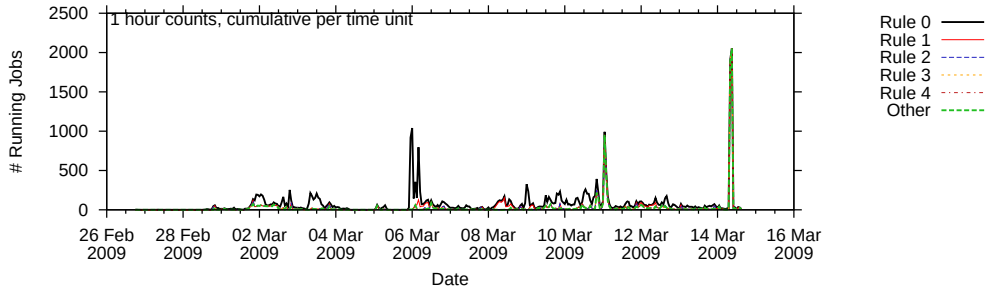


Figure 4.8: Yahoo! M-Cluster, cumulative running jobs by application.

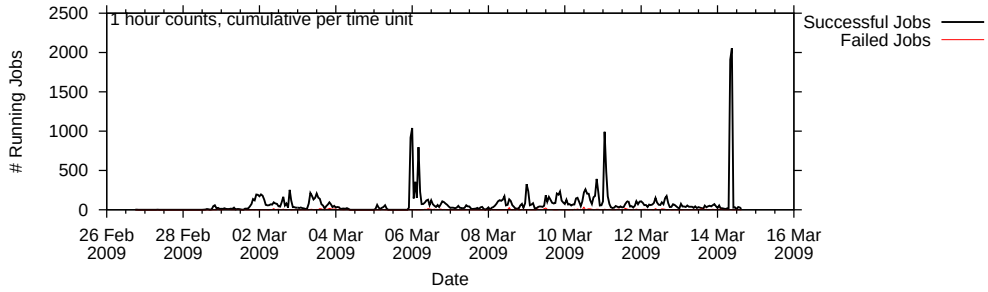


Figure 4.9: Yahoo! M-Cluster, cumulative running jobs by status.

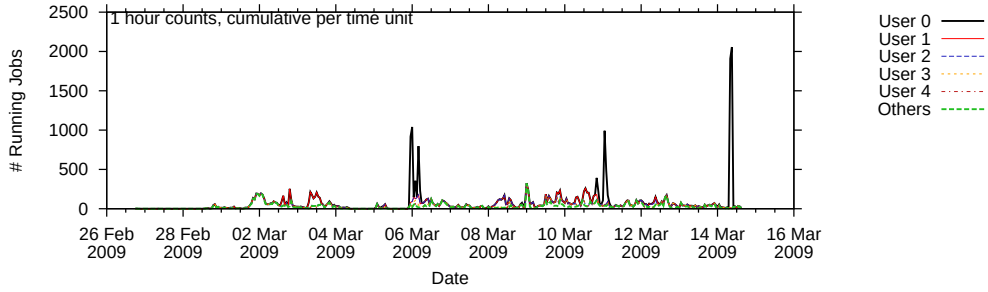


Figure 4.10: Yahoo! M-Cluster, cumulative running jobs by user.

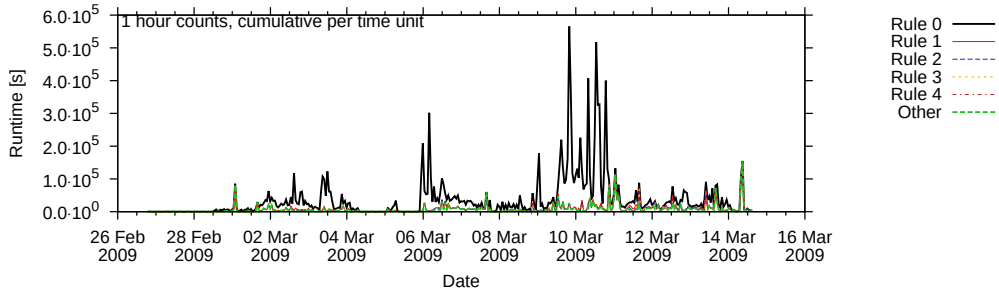


Figure 4.11: Yahoo! M-Cluster, cumulative job runtime by application.

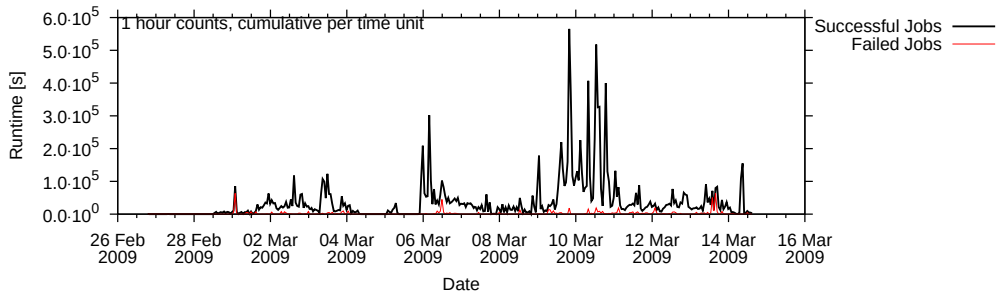


Figure 4.12: Yahoo! M-Cluster, cumulative job runtime by status.

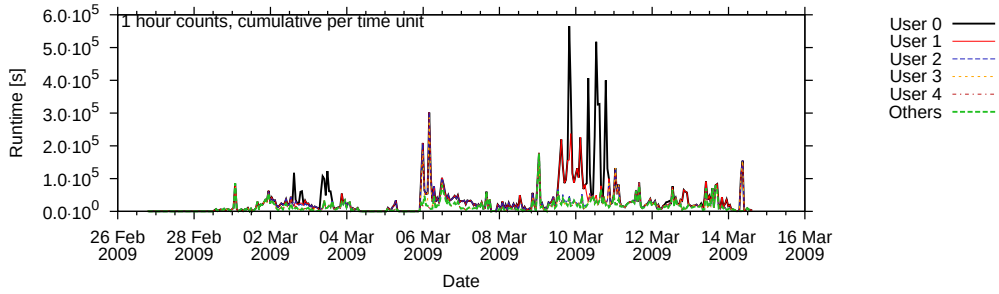
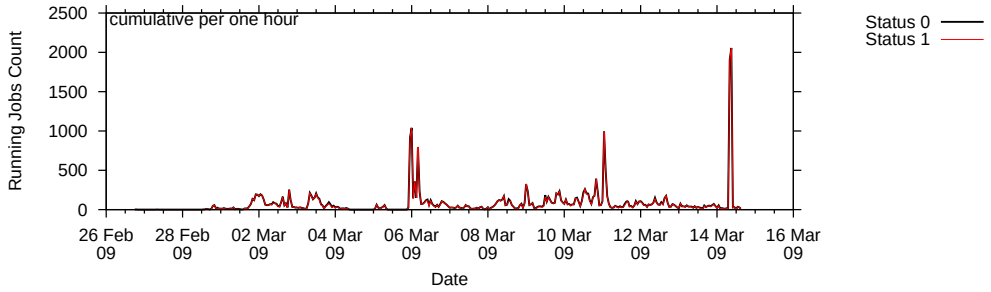
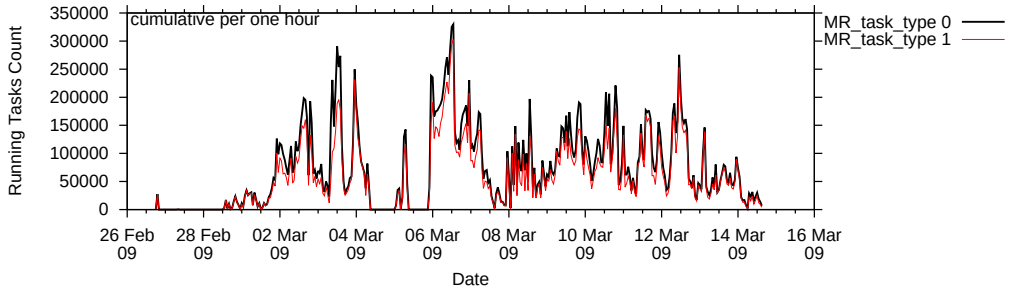


Figure 4.13: Yahoo! M-Cluster, cumulative job runtime by user.

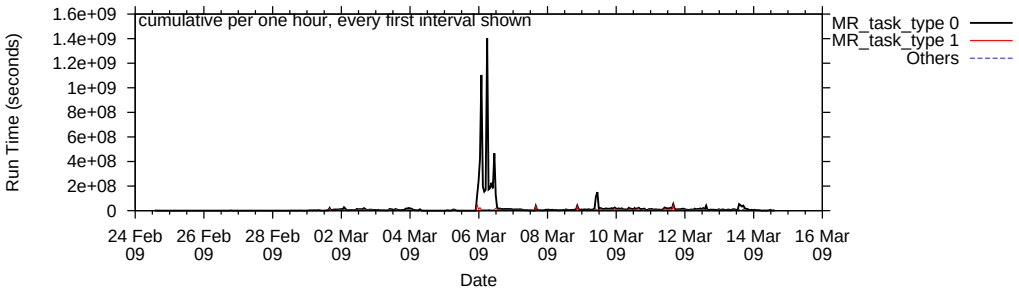


(a) Jobs per status (0: failed, 1: successful).

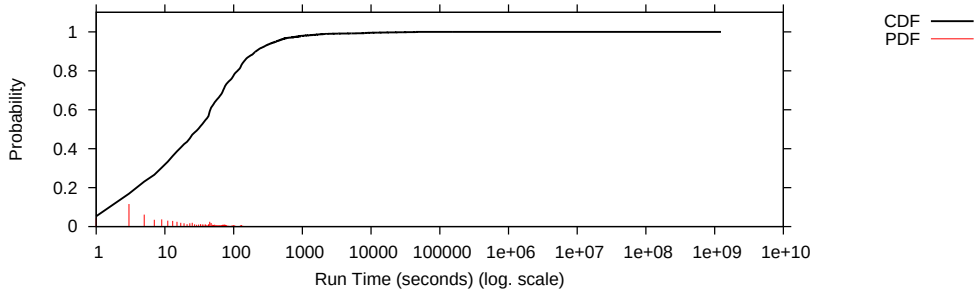


(b) Running tasks broken down by task type.

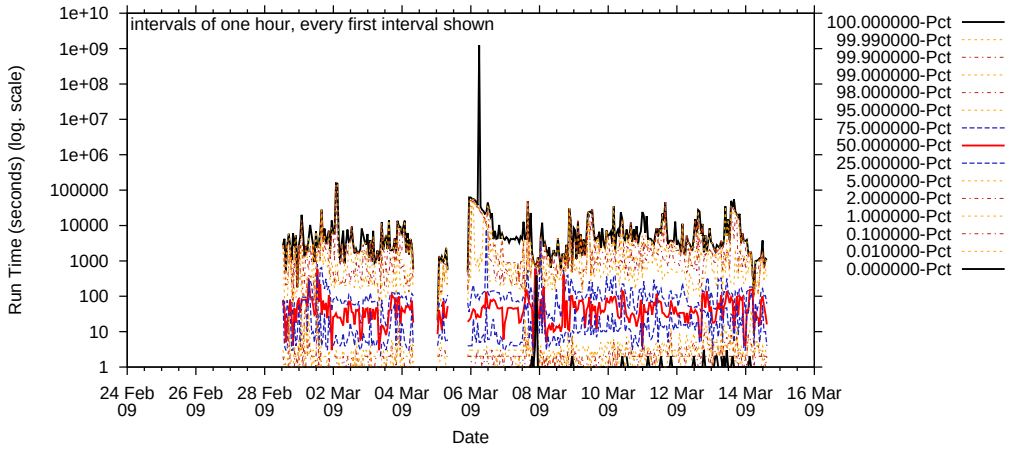
Figure 4.14: Yahoo! M-Cluster, Running jobs and tasks.



(a) Per task type (0: map, 1: reduce).

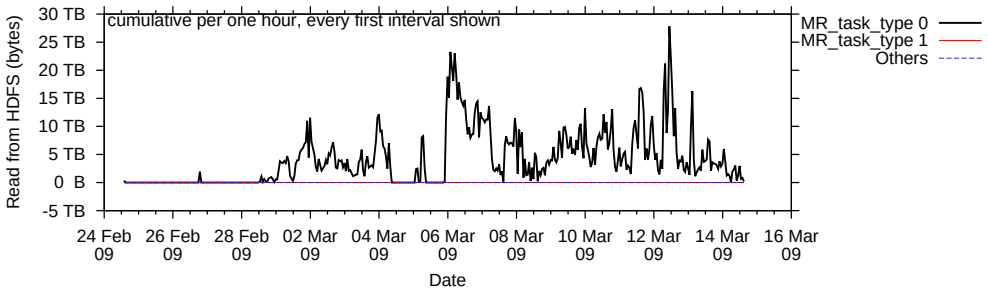


(b) Distribution.

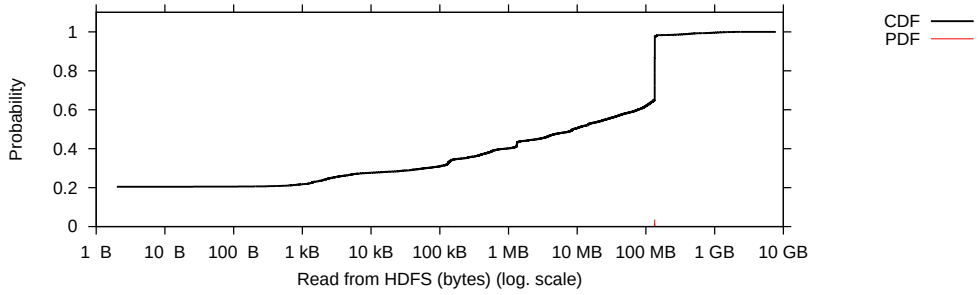


(c) Percentiles.

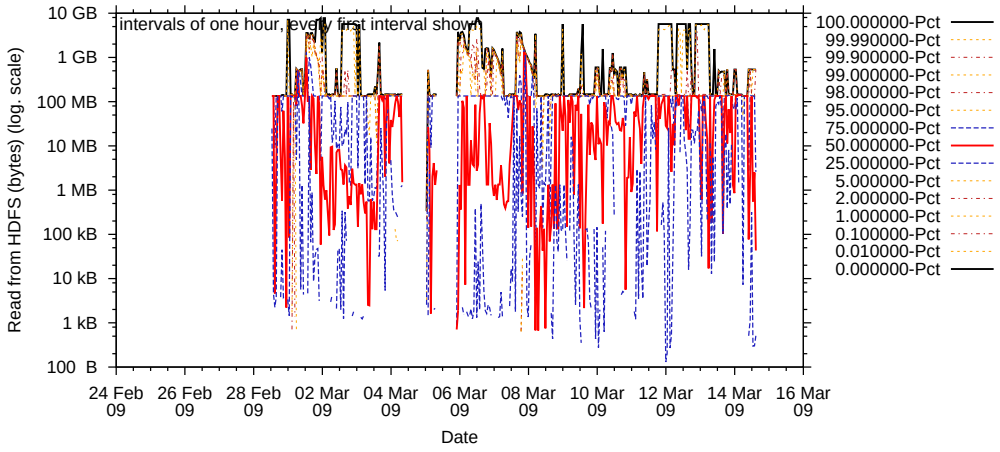
Figure 4.15: Yahoo! M-Cluster, Tasks run time.



(a) Per task type (0: map, 1: reduce). (Curves have different colors than for data written.)

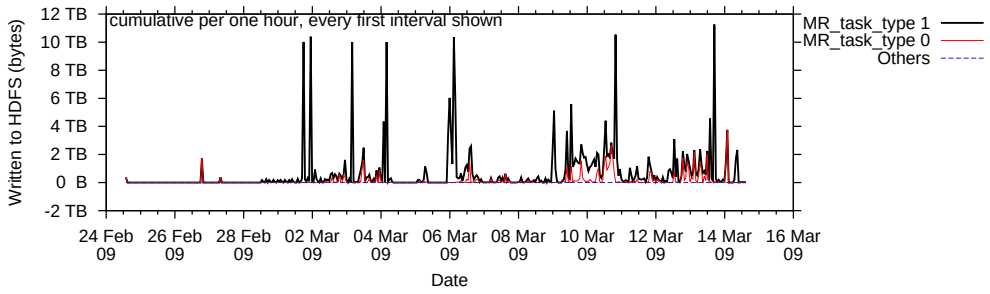


(b) Distribution.

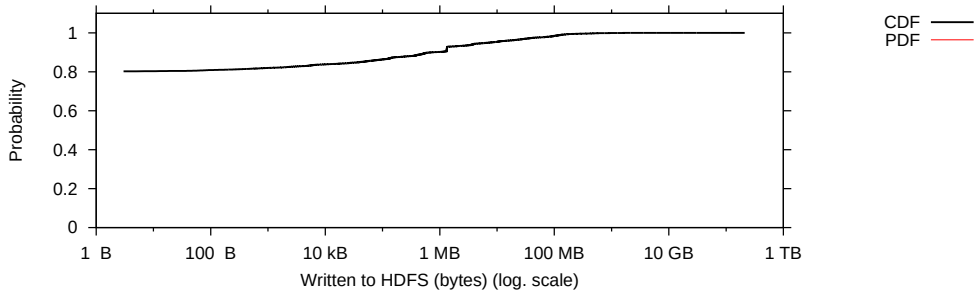


(c) Percentiles.

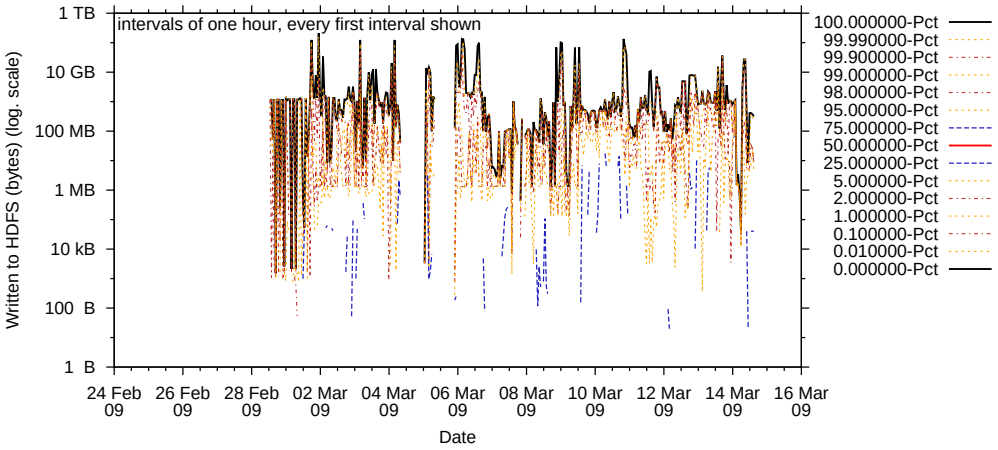
Figure 4.16: Yahoo! M-Cluster, HDFS data read.



(a) Per task type (0: map, 1: reduce). (Curves have different colors than for data read.)

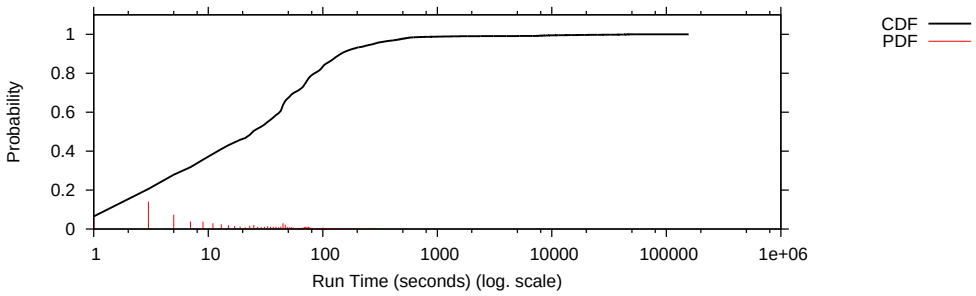


(b) Distribution.

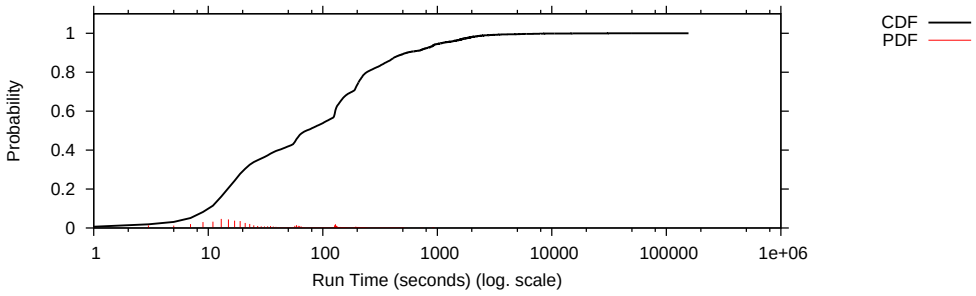


(c) Percentiles.

Figure 4.17: Yahoo! M-Cluster, HDFS data written.



(a) Map tasks.



(b) Reduce tasks.

Figure 4.18: Yahoo! M-Cluster, Task run time distribution, per task type.

4.2.4 Google

In November 2011, Google has released[1] traces of a cluster of about 11000 nodes for 29 days in may 2011, covering 667,992 jobs and 44,920,671 tasks. The trace consist of 1.6 GiB worth of job/task/machine events and 37 GiB worth of task usage information. Unfortunately for this work, these traces no not consist of only MapReduce jobs, and no MapReduce specific information is included. Given that Google introduced [1] MapReduce, there will be MapReduce jobs hidden in these traces. Possibly the MapReduce jobs in this trace could be identified and analyzed, we leave this for future work. The time-stamps in the trace have been re-based to 1 January 1970.

Job Wait Times

In Figure 4.19 we show the job wait times per status. In this graph we see that the cumulative wait times for the successful jobs is generally low compared to the canceled jobs. In fact, it is likely that jobs with long wait times are canceled just because of the long wait times. Failed jobs are almost invisible in this graph, so either there are only a few failed jobs, or they even fail to be scheduled. We find overlaps of the wait time spikes at “2 and 10 January” with a similar spike in the CPU consumption shown in Figure 4.21, likely the high CPU usage in these periods caused the increase in wait times.

Job Run Times

In Figure 4.20 we show the job run times per status. In this graph we see that the jobs causing large cumulative run times are the tasks that eventually get canceled. We see two spikes for the cumulative run times of canceled jobs around “3 and 11 January”, this is a strange one day difference with the spikes in Figure 4.21.

CPU Usage

In Figure 4.21 we show the usage of the CPUs per status over time. In the Google trace the amount of CPUs used by a single task is specified in a normalized way, we have assumed that the lowest value corresponded to a single CPU and calculated the “real” CPU count according to this assumption.

The Two Spikes

In most graphs we find two spikes, one on “2 or 3 January”, and one on “10 or 11 January”. As the dates of the start and end of the graphs are correct, we assume that the one-day differences in the spikes in Figures 4.20 and 4.21 are caused by two tries of the same set of jobs. First a set of jobs was tried and all failed, then a day later a similar set of jobs was tried and all were canceled. On the other hand, if

[1] `http://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1`

this would have been like this because of a mistake, we would not expect the same thing to happen again one week later.

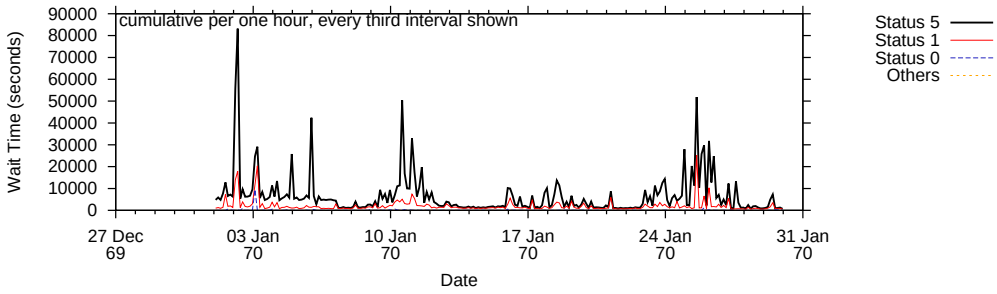


Figure 4.19: Google, job wait time per status (0: failed, 1: successful, 5: canceled).

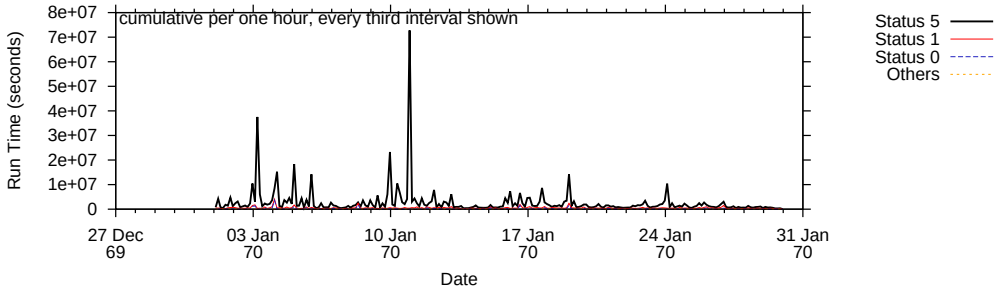


Figure 4.20: Google, job run time per status (0: failed, 1: successful, 5: canceled).

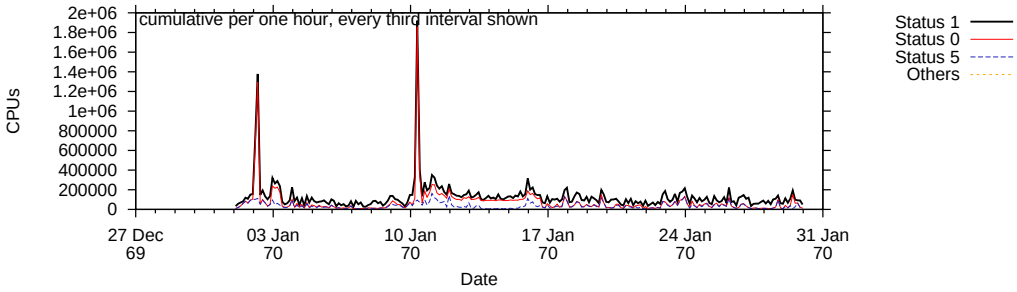


Figure 4.21: Google, CPUs per status (0: failed, 1: successful, 5: canceled).

4.2.5 Comparison all Workload Traces

In this subsection we compare the job wait times, the job run times, the task run times, the job I/O, and the number of tasks of the SN1, SN2, Yahoo! M-Cluster, and Google workload traces; unfortunately not all traces contain all this information.

Job Wait Times

We compare the distributions of the job wait times of all the workload traces in Figure 4.22. Unfortunately the SN2 trace does not contain wait time information. In these graphs we observe that the wait times in all workloads are generally extremely low, 80% of the jobs is started within a second. It looks like the cluster of the SN1 workload trace is over-provisioned, as almost 100% of the jobs is started within a second. The Google cluster has compared to the SN1 and Yahoo! MapReduce clusters a very long tail for the run times, of up to three days versus 15 minutes – this is also a sign that the Google cluster is not (just) a MapReduce cluster.

Job Run Times

We compare the distributions of the job run times of all the workload traces in Figure 4.23. In these graphs we observe that the shortest and longest jobs in the Google trace run longer than those in the other traces; the centers of the distributions do not differ significantly – 10-90% of the jobs run in 10-1000 seconds. It seems that Google's task provisioning method requires more time to start a task than Hadoop.

Task Run Times

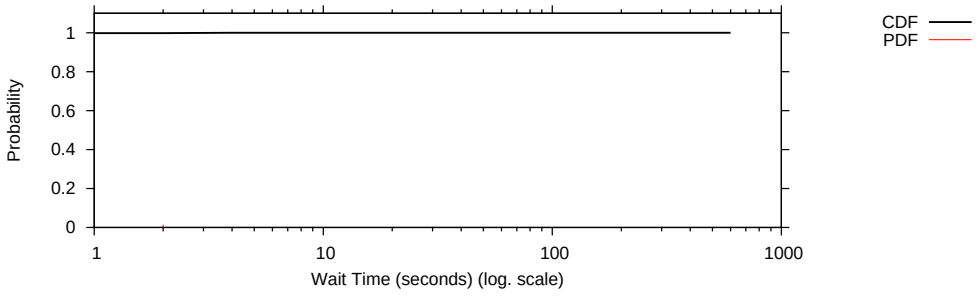
We compare the task run times in Figure 4.24. Unfortunately only the Yahoo! trace and the Google trace contain information on the task run times. We see a preference for a runtime of about 30 minutes for about 10% of the tasks in the Google trace. Tasks in the Google trace run longer than in the Yahoo! trace. In the Yahoo! trace about 25% of the tasks finish in under 10 seconds, where in the Google trace almost no task finishes in under 20 seconds; also the tail of the Google task run times is much longer.

Job I/O

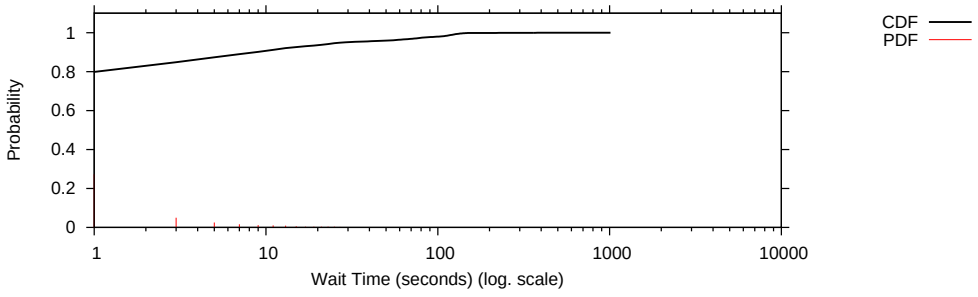
We compare the distributions of the job I/O of the workload traces in Figure 4.25. Unfortunately the SN2 trace and the Google trace (although it contains detailed information on the time spent on I/O) do have no information on the total amount read from and written to the file-system. In these graphs we see that the I/O usage of the SN1 and the Yahoo! M-Cluster workloads do not differ significantly.

Number of Tasks

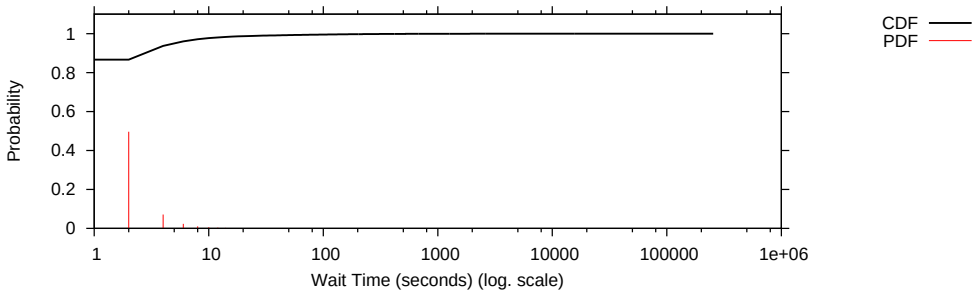
We compare the number of tasks per job in Figure 4.26. Unfortunately there is no information on the number of tasks available in the SN1 trace. In the SN2 workload we find that 52% of the jobs have no tasks, and 18% have only 2 tasks, this accounts already for 70% of all tasks. We assume that these numbers are caused by the failing jobs visible in Figure 4.7a, and by the fact that task information is missing for the last three days of the trace (resulting in 0 tasks per job); but three missing days in a ten day workload are not likely to contribute to 52% of all jobs.



(a) SN1

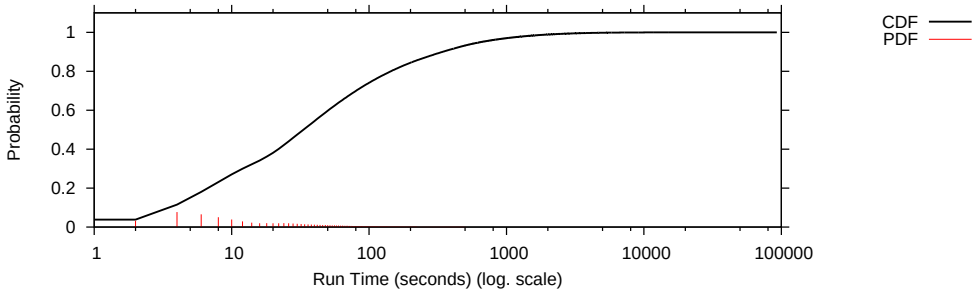


(b) Yahoo! M-Cluster

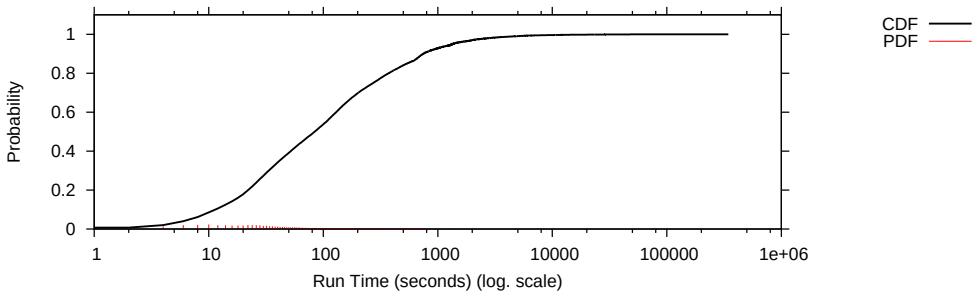


(c) Google

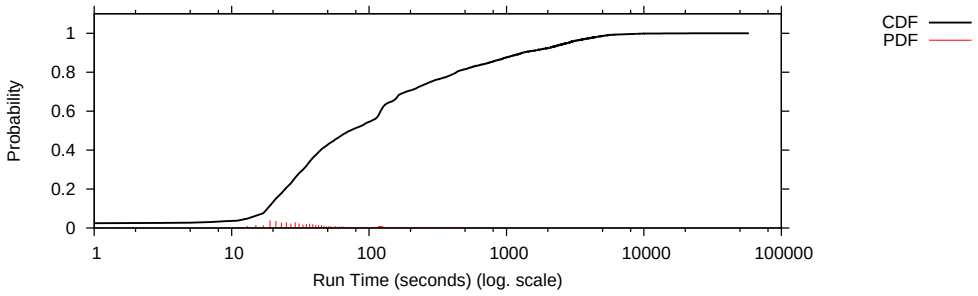
Figure 4.22: Job wait times.



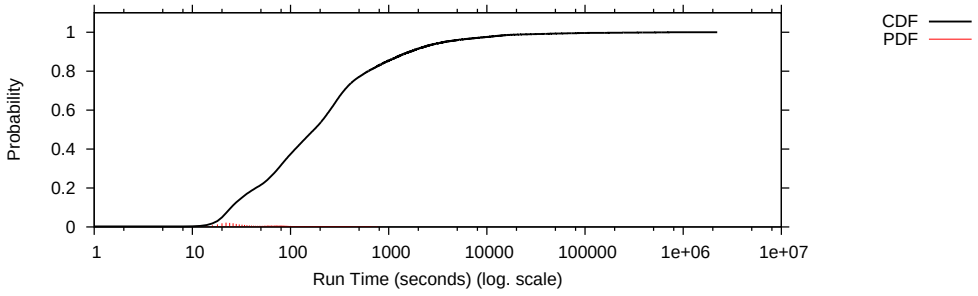
(a) SN1



(b) SN2

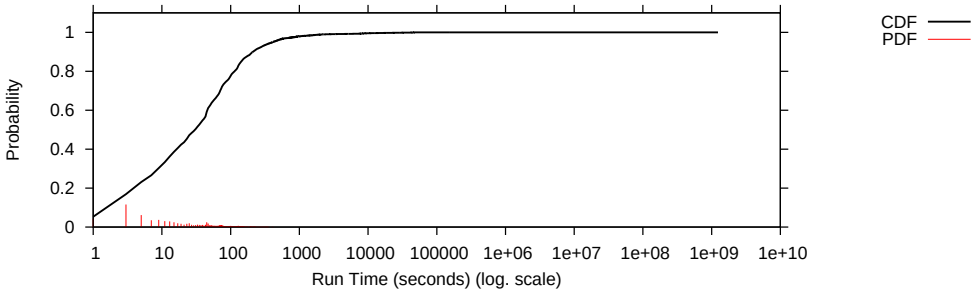


(c) Yahoo! M-Cluster

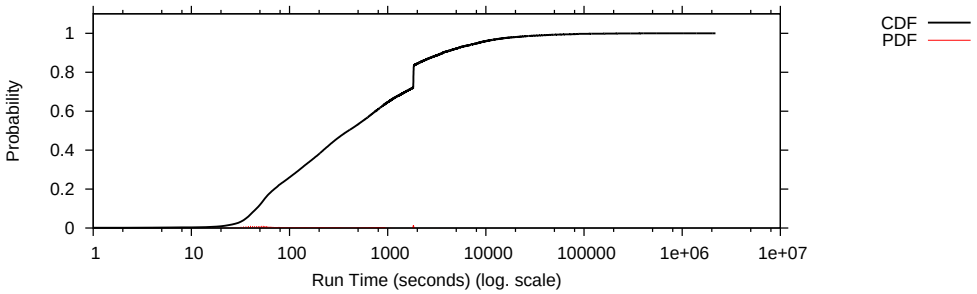


(d) Google

Figure 4.23: Job run times.

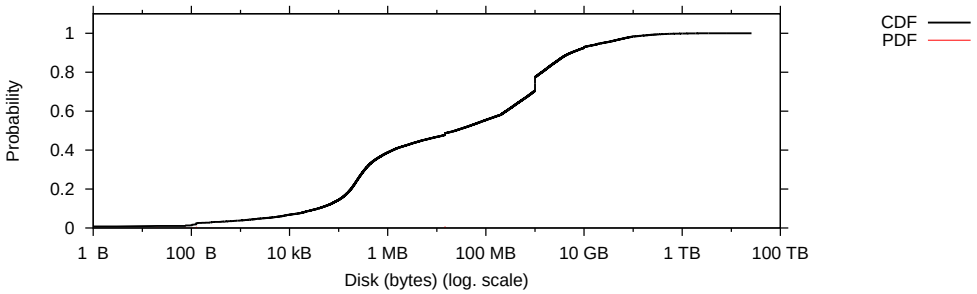


(a) Yahoo! M-Cluster

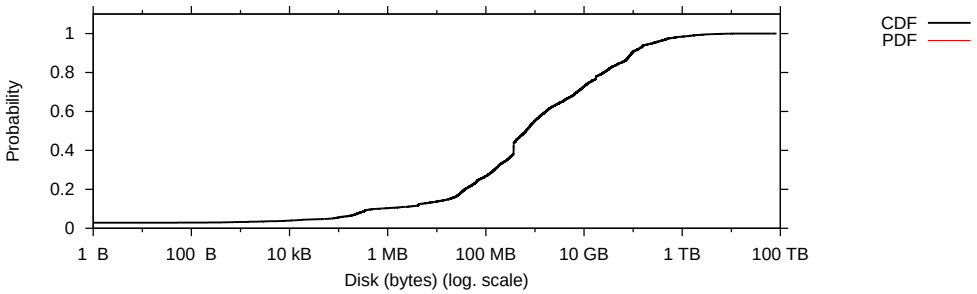


(b) Google

Figure 4.24: Task run times.

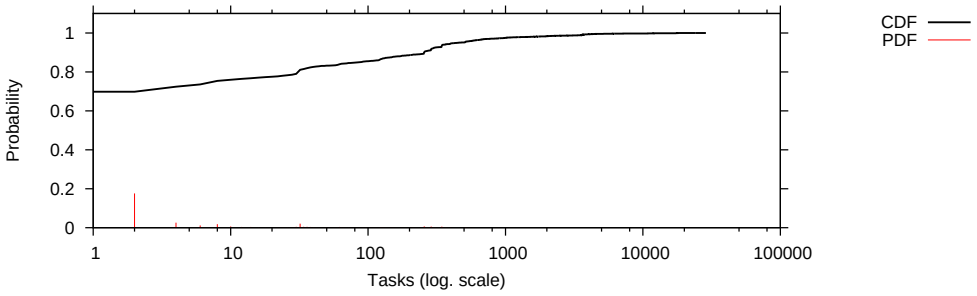


(a) SN1

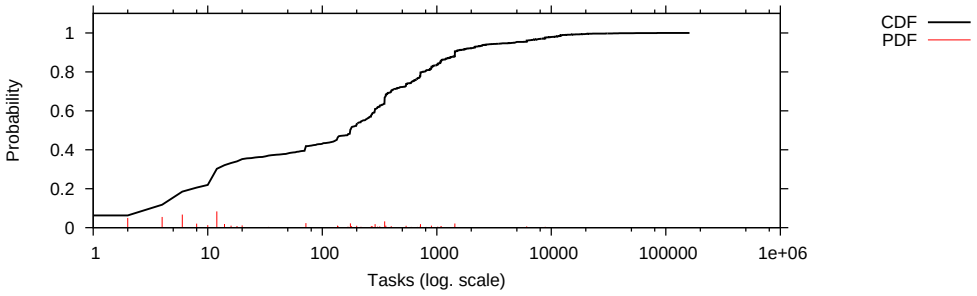


(b) Yahoo! M-Cluster

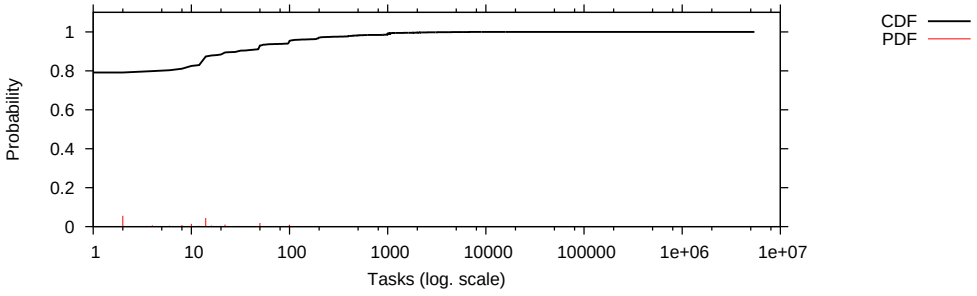
Figure 4.25: Job I/O.



(a) SN2



(b) Yahoo! M-Cluster



(c) Google

Figure 4.26: Number of tasks per job.

Table 4.3: Basic statistics for the SN1 trace.

	Min	1%-tile	10%-tile	25%-tile	Mean	Median	75%-tile	90%-tile	99%-tile	Max	Std	CoV
Job Wait Time (seconds)	0.000	0.000	0.000	0.000	0.457	0.000	1.00	1.00	2.00	601	2.62	5.74
Job Inter-arrival Time (seconds)	0.000	0.000	1.00	3.00	14.0	7.00	15.0	32.0	105	250181	239	17.1
Job Executable ID	0.000	0.000	0.000	0.000	0.635	0.000	1.00	2.00	4.00	4.00	1.10	1.73
Job Run Time (seconds)	0.000	2.00	5.00	10.0	165	35.0	107	345	2179	92671	654	3.97
Job Forced-quit Time (seconds)	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	0.000	-0.000
Task Inter-arrival Time (seconds)	–	–	–	–	–	–	–	–	–	–	–	–
Task Run Time (seconds)	–	–	–	–	–	–	–	–	–	–	–	–
Task CPUs	–	–	–	–	–	–	–	–	–	–	–	–
Task Disk IO Ratio	–	–	–	–	–	–	–	–	–	–	–	–
Task Memory	–	–	–	–	–	–	–	–	–	–	–	–

Table 4.4: Basic statistics for the SN2 trace.

	Min	1%-tile	10%-tile	25%-tile	Mean	Median	75%-tile	90%-tile	99%-tile	Max	Std	CoV
Job Queue ID	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	0.000	-0.000
Job Partition ID	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	0.000	-0.000
Job Run Time (seconds)	0.000	4.00	13.0	29.0	434	86.0	274	750	4813	344461	3154	7.27
Job Fail fraction (fraction of total tasks)	0.000	0.000	0.000	0.000	0.013	0.000	0.000	0.005	0.333	1.00	0.078	5.87
Job Reduce ratio (fraction of total tasks)	0.000	0.000	0.000	0.000	0.108	0.000	0.111	0.500	0.969	1.00	0.208	1.92
Job User ID	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	0.000	-0.000
Job Wait Time (seconds)	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	0.000	-0.000
Job Inter-arrival Time (seconds)	0.000	0.000	1.00	2.00	14.2	8.00	19.0	34.0	83.0	8034	37.4	2.64
Job Forced-quit Time (seconds)	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	0.000	-0.000
Job Executable ID	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	0.000	-0.000
Job Group ID	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	0.000	-0.000
Job Total number of tasks (count)	0.000	0.000	0.000	0.000	154	1.00	9.00	258	3620	28687	982	6.40
Task Inter-arrival Time (seconds)	–	–	–	–	–	–	–	–	–	–	–	–
Task Run Time (seconds)	–	–	–	–	–	–	–	–	–	–	–	–
Task CPUs	–	–	–	–	–	–	–	–	–	–	–	–
Task Disk IO Ratio	–	–	–	–	–	–	–	–	–	–	–	–
Task Memory	–	–	–	–	–	–	–	–	–	–	–	–

Table 4.5: Basic statistics for the Google trace.

	Min	1%-tile	10%-tile	25%-tile	Mean	Median	75%-tile	90%-tile	99%-tile	Max	Std	CoV
Job Queue ID	0.000	0.000	0.000	0.000	0.918	1.00	2.00	2.00	2.00	3.00	0.832	0.907
Job Run Time (seconds)	0.000	16.0	26.0	62.0	2856	176	440	1650	29767	2219070	35709	12.5
Job Fail fraction (fraction of total tasks)	0.000	0.000	0.000	0.000	0.242	0.000	0.000	1.00	1.00	1.00	0.428	1.77
Job Reduce ratio (fraction of total tasks)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	–
Job User ID	0.000	32.0	105	141	285	238	469	584	730	932	193	0.675
Job Wait Time (seconds)	0.000	0.000	1.00	1.00	8.18	2.00	3.00	4.00	33.0	253976	567	69.3
Job Inter-arrival Time (seconds)	0.000	0.000	0.000	0.000	3.76	1.00	4.00	13.0	31.0	764	6.92	1.84
Job Forced-quit Time (seconds)	-1.00	-1.00	-1.00	-1.00	2353	-1.00	198	756	19351	2179584	34819	14.8
Job Executable ID	34.0	3438	3602	3761	10252	5004	11890	28670	37943	39729	9616	0.938
Job Total number of tasks (count)	1.00	1.00	1.00	1.00	67.3	1.00	2.00	31.0	1004	5442378	8486	126
Task Inter-arrival Time (seconds)	0.000	0.000	0.000	0.000	26.8	0.000	0.000	0.000	35.0	1966315	2436	90.8
Task Run Time (seconds)	0.000	22.0	48.0	96.0	2815	400	1828	3656	38796	2179528	21252	7.55
Task CPUs	1.00	2.00	2.00	2.00	2.14	2.00	2.00	2.00	4.00	4.00	0.523	0.245
Task Disk IO Ratio	–	–	–	–	–	–	–	–	–	–	–	–
Task Memory	–	–	–	–	–	–	–	–	–	–	–	–

Table 4.6: Basic statistics for the Yahoo! trace.

	Min	1%-tile	10%-tile	25%-tile	Mean	Median	75%-tile	90%-tile	99%-tile	Max	Std	CoV
Job Run Time (seconds)	0.000	8.00	21.0	31.0	513	80.0	294	1344	5534	58050	1503	2.93
Job Fail fraction (fraction of total tasks)	0.000	0.000	0.000	0.000	0.010	0.000	0.000	0.000	0.000	1.00	0.099	9.98
Job Reduce ratio (fraction of total tasks)	0.000	0.000	0.000	0.000	0.002	0.000	0.000	0.000	0.000	1.00	0.040	24.9
Job Wait Time (seconds)	0.000	0.000	0.000	0.000	6.53	0.000	1.00	10.0	124	1012	26.3	4.03
Job Inter-arrival Time (seconds)	0.000	0.000	1.00	2.00	55.3	8.00	36.0	93.0	397	189881	1410	25.5
Job TaskRunTimes	0.000	1.00	3.00	8.00	305	33.0	90.0	214	2577	1e+09	237518	779
Job Forced-quit Time (seconds)	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	0.000	-0.000
Job Disk parameter a	-9e+09	-1e+09	-4e+07	5703	2e+08	5212436	1e+08	3e+08	2e+09	1e+12	6e+09	26.9
Job Disk parameter b	-1e+08	-2e+07	-4607026	-807801	199881	-4612	153050	4656541	3e+07	5e+07	7036996	35.2
Job Executable ID	0.000	0.000	0.000	0.000	755	0.000	1487	2226	3150	3329	986	1.31
Job Total number of tasks (count)	1.00	1.00	4.00	12.0	980	177	628	1440	12537	162917	3855	3.93
Task Inter-arrival Time (seconds)	0.000	0.000	0.000	0.000	0.220	0.000	0.000	1.00	2.00	34901	14.7	67.1
Task Run Time (seconds)	0.000	1.00	3.00	8.00	305	33.0	90.0	214	2577	1e+09	237518	779
Task CPUs	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.000	0.000
Task Disk IO Ratio	0.000	0.000	0.000	0.000	21580	1.00	2.22	27.4	10054	9e+08	1474602	68.3
Task Memory	–	–	–	–	–	–	–	–	–	–	–	–

Chapter 5

MapReduce Workload Modeling

In this chapter we propose a model for MapReduce workloads and a procedure to generate synthetic yet realistic MapReduce workloads. This provides respectively answers to questions Q2: "How can we model MapReduce workloads?" and Q3: "How can we generate realistic synthetic MapReduce workloads?"

First, we discuss why we model workloads in Section 5.1 and introduce the statistical tools we use in Section 5.2. Second, we propose a MapReduce model in Section 5.3. Third, we propose a procedure to generate synthetic MapReduce workloads in Section 5.4. Finally, we conclude in Section 5.5.

5.1 Why Model?

A real trace of a MapReduce workload is the most realistic workload which can be used for performance evaluation or other purposes, as it is, in fact, real. Still, it can be advantageous to model workloads while we have traces of real workloads available. From the model of a workload, we can generate synthetic workloads which we can use to drive simulations. Advantages of using workloads generated from a model, instead of using real workload traces, include:

Flexibility Generating synthetic workloads from a model gives the ability to change model parameter values.

- Changing model parameter values can be used to fit the workload to cluster configurations that are different – in particular, larger – for example, from the configuration of the cluster from which the traces were collected.
- Changing model parameter values can be used to fit the workload to different levels of cluster load.

Insight The model parameters reveal information about the original workload, and can help to get a better understanding of the workload. For example, it is

possible to calculate the average run time from the distribution and its parameters.

Size The size of a file containing model parameter values is very small compared to a normal workload trace.

Privacy Workload trace owners might have less objections to distributing model parameter values than to the distribution of a real workload trace.

Feitelson [32], Section 1.3.2, gives a more thorough explanation about why the use of workload models, instead of workload traces, can be beneficial.

The MapReduce workload models we propose are statistical models, another approach would be modeling using neural networks. Modeling using neural networks is undesirable for a number of reasons. First, the parameters of such models do provide little usable information about the workload. Second, it is difficult to sample from a neural network. Third, the training of a neural network is complex, may require large amounts of data, and may not converge or stabilize.

5.2 Statistical Modeling

We use a number of statistical tools for modeling MapReduce workloads. In the following subsections we give short introductions to these tools: Distributions, Goodness of Fit, Distribution Selection, and Correlation.

5.2.1 Distributions

We would like to capture the nature of workload properties using statistical probabilities. For example, we could calculate the probability for a task to be a reduce task. For properties with continuous value ranges, such as the task run time, we could create a histogram and calculate the probability for each of its bins. In this histogram-scenario we can adapt the granularity by changing the bin-widths; smaller bins lead to a higher granularity, at the cost of more model-parameters (the per-bin probabilities) and at the risk of over-fitting (fitting the empirical data so precisely that it does not generalize).

We would like the number of parameters to be small, so instead of taking the above histogram approach, we use well-known probability density functions (PDFs) with a small number of parameters; a PDF is similar to a histogram with infinitely small bins. For the empirical data it is easier to determine the integral of the PDF, that is, the cumulative distribution function (CDF). The CDF for empirical data can be calculated by determining, for each distinct sample value, the probability that a sample has a lower or equal value.

In our modeling process we use the Normal Distribution, the Log-Normal Distribution, the Weibull Distribution, the Generalized Pareto Distribution, and the Exponential Distribution. We have chosen these distributions because they have a small number of parameters, are available by default in Matlab, SciPy, and many

other statistical tools, and because they have been proven to be useful in practice for modeling computer science systems and workloads [19]. We have chosen not to use Hyper-distributions, because of the larger number of parameters and the risk of over-fitting.

We can use maximum-likelihood estimation to fit the parameters of a distribution to empirical data. In this work we have used the maximum-likelihood estimation functions that are built into SciPy. In Table 5.1 we present the chosen distributions.

Function	Notation	Parameters
Normal	$\mathcal{N}(\mu, \sigma^2)$	μ location, σ^2 squared scale
Log-Normal	$\ln \mathcal{N}(\mu, \sigma^2)$	μ log-scale, σ^2 shape
Exponential	$\mathrm{Exp}(\lambda)$	λ rate
Weibull	$\mathrm{Wei}(\lambda, k)$	λ scale, k shape
Generalized Pareto	$\mathrm{Par}(\mu, \sigma, \xi)$	μ location, σ scale, ξ shape
Gamma	$\mathrm{Gam}(\kappa, \theta)$	k shape, θ scale

Table 5.1: Distribution functions we use for statistical modeling.

5.2.2 Direct and Indirect Modeling

We use two modeling approaches, a direct-modeling approach and an indirect-modeling approach. In the direct-modeling approach, a property (the variable of interest) is modeled by a single probability distribution and its parameters are constants set through the MLE fitting process. In the case of the indirect-modeling approach, a property is modeled by a single probability distribution, and each of its parameters are also modeled by *second-level* distributions.

For example, we model all properties directly in Section 5.3.1, and we model the task-specific properties indirectly in Section 5.3.2.

The indirect-modeling approach gives the promise of a better fit with real data, albeit at the cost of additional parameters and the probability of over-fitting.

5.2.3 Goodness of Fit

Although we may have found maximum-likelihood estimates (see Section 5.2.1) for the parameters of a distribution, the distribution may still be unfit to represent the empirical data. Therefore, we use goodness of fit tests to reject those distributions that are not likely to fit the data.

A goodness of fit tests returns a p-value, this p-value is the probability that, under the assumption that the empirical data is sampled from a given distribution, we find samples at least as unfitting as the empirical data. We apply several goodness of fit tests, and we reject a distribution when the p-value returned by *any* of the selected goodness of fit tests is lower than the significance level α of 0.05.

We have selected two different goodness of fit tests: the Kolmogorov-Smirnov

test and the Anderson-Darling test. These two goodness of fit tests have been chosen because they focus on the center and the tail of the distribution, respectively.

Kolmogorov-Smirnov The Kolmogorov-Smirnov (KS) uses the D-statistic (see Section 5.2.4) as the basis for the test. Large values for the D-statistic are likely to occur at the "center" of a distribution, therefore is the KS test more focused at the "center" of a distribution.

Anderson-Darling The Anderson-Darling (AD) test is, because of a weight function, more focused at the "tail" of a distribution.

As the Kolmogorov-Smirnov test works better for small sample sizes, we run these tests each 1000 times with 30 samples each time. We return as p-value the average of all the p-values obtained by these 1000 tests.

5.2.4 Selection of the Best Fit

The goodness of fit tests do not provide a way to select the best fitting distribution, they only reject those distributions that do not fit well. After we have tested the goodness of the fit of the distributions, we might have no fitting distribution left; on the other hand, we might have multiple fitting distributions, from which we need to select one.

We use two different ways of selecting the best fit. If we have multiple fitting distributions for a single set of directly-modeled data, we use their computed D-Statistic to select the best fitting distribution. If we attempt to fit multiple sets of data by meta-distributions as in the indirect-modeling approach (see Section 5.2.2), we use the number of fits to select a fitting distribution.

D-Statistic To decide which of the fitting distributions to use, we compare the D-Statistics and select the distribution with the *lowest* D-Statistic. The D-Statistic value is defined as $D = \sup_x |F_n(x) - F_0(x)|$, where F_n and F_0 are respectively the empirical cumulative distribution function and the cumulative distribution function of the tested distribution. So, we select the distribution which has the lowest maximum-difference between the distribution's CDF and the empirical CDF.

Number of Fits We use the number of fits when we want to fit a meta-distribution – a set of distributions that model each of the parameters of another distribution. In this case the data is partitioned into sets which correspond to a process in the *real* system, e.g., for each job a set of tasks. For each set of data we calculate the maximum-likelihood estimates of the parameters for each distribution. Then, we use the goodness of fit tests to reject those distributions that do not fit well, and we use the D-Statistic to select the best of the remaining distributions. Finally, we count for each distribution the number of items in the set for which it was selected as the best fitting distribution. We select the distribution that matched the most items. For a fictitious example

see Table 5.2. In this example we show a three-job workload, the jobs consist of 1, 128, and 128 tasks. For the task run time we see that the Log-Normal distribution has the best fit for a total of 129 tasks and that the Weibull distribution has the best fit for 128 tasks; therefore we select the Log-Normal distribution to model the task run time.
We have chosen to select based on the number of matched items and not on the number of matched sets, to reduce the influence of small sets, as we have observed that small sets tend to fit more often than larger sets.

		Best-Fitting Distribution		
Job	**Task Count**	**Task Run Time**	**Task CPU**	**Task Memory**
1	1	Log-Normal	Weibull	Exponential
2	128	Weibull	Exponential	Weibull
3	128	Log-Normal	Weibull	Weibull
Overall Best Fit		*Log-Normal (129)*	*Weibull (129)*	*Weibull (256)*

Table 5.2: Fictitious three-job example finding the best distribution function.

5.2.5 Correlation

Some properties could influence each other; if this influence is consistent, we call this a correlation. For example, if a task that needs to read a large amount of data has a run time proportional to the amount of data, there exists a correlation between the disk usage and run time values. We would like to be able to detect correlations between various properties, so that these can be exploited in our model. The Matlab `corr`[1] function can be used to detect correlation between two properties; it takes two sets of property values as input and returns a single correlation value in the range 0 (no correlation) to 1 (strong correlation). We call properties modeled by a probability distribution function *primary* properties, and properties modeled by a correlation *secondary* or *derived* properties.

5.3 Our Statistical MapReduce Workload Models

We have explored four statistical models, shown in Figure 5.1 and Table 5.3. The models are in this table ordered in decreasing level of complexity. In the first two of these models we model the task-specific properties indirectly. The intuition behind this process is that the tasks of a single job might have similar behavior, but that the tasks of different jobs might have very different behavior.

The reason why we have explored multiple models is that we would like to build a comprehensive workload modeling framework and to explore alternatives in the large space of statistical models.

[1] `http://www.mathworks.nl/help/toolbox/stats/corr.html`

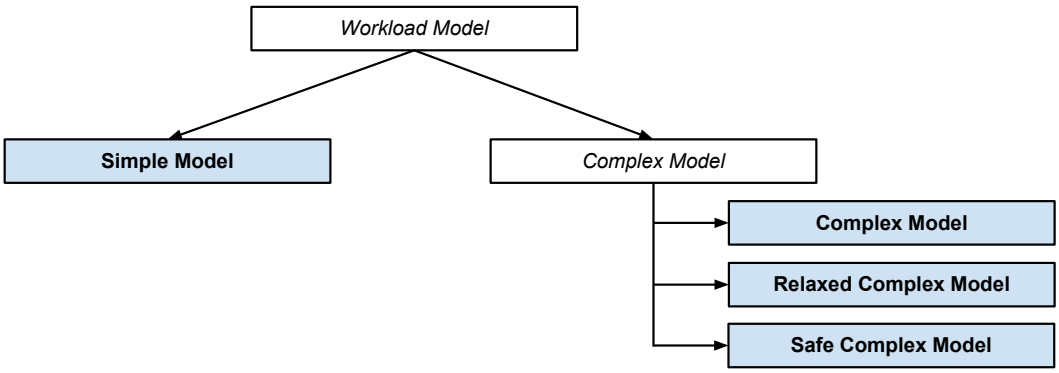


Figure 5.1: Taxonomy of our MapReduce workload models.

Model	Sect.	Properties			Map/Reduce Distinction	Significance Level	Indirect Distr. Selection
		Job-specific	Task-specific	Secondary			
Complex Model	5.3.2	Direct	Indirect	Disk I/O	Yes	0.05	Best Fit
Relaxed Complex Model	5.3.3	Direct	Indirect	Disk I/O	Yes	0.02	All Fits
Safe Complex Model	5.3.4	Direct	Direct	Disk I/O	Yes	0.05	–
Simple Model	5.3.1	Direct	Direct	–	No	0.05	–

Table 5.3: Overview of the Models

5.3.1 The Simple Model

We define a MapReduce workload as a collection of jobs, with for each individual job a collection of tasks. We propose a simple model. The simple model uses only four distributions, which can be hand-picked based on the results in Section 5.3.5.

This model is oversimplified, it makes for example no distinction between map and reduce tasks. On the other hand has it very few parameters, therefore it will not easily be over-fitted.

Inter-arrival time Captures the inter-arrival times of both jobs and tasks. This model does not make a distinction between the inter-arrival times of jobs and tasks.

Number of tasks Captures the number of tasks of a job.

Map/Reduce ratio Captures the ratio between the number of map and reduce tasks of a job.

Task run time Captures the task run times. This model does not make a distinction between the run-times of map and reduce tasks.

5.3.2 The Complex Model

We define a MapReduce workload as a collection of jobs, with for each individual job a collection of tasks. As explained in Section 1.1, a MapReduce job is written

in the form of a map function and a reduce function. When such a job runs, a partition function divides input data from a distributed file system, and a number of map tasks is started with each a part of the divided data as input. A number of reduce tasks is started, and a shuffle function assigns the output of the map tasks to these reduce tasks. The output of the reduce tasks is written to the distributed file system, and the job finishes.

We model for the map and reduce phases the properties listed in Table 5.4. We do not model the DFS usage and the partition and shuffle phases. We omit these to reduce the number of model parameters – as this information is also captured by the task's disk and network usage – and also because some of the necessary information is not yet available in the CWA data format (see Appendix B). As possible directions for future work, the CWA data format and the model could support the DFS, partition and shuffle phases, and other task types – for example, setup and cleanup.

All job properties are modeled directly. The task-specific properties are modeled indirectly, i.e., we use second-level models to model the model parameters (see Section 5.2.2). Since the map and reduce tasks generally behave very differently from each other, as shown in Chapter 4, we perform model fitting separately for the map tasks and the reduce tasks.

The main two features of the Complex Model over the Simple Model are:

1. Modeling of map tasks and reduce tasks separately.

   In Chapter 4 we have observed that map and reduce tasks behave very differently from each other. By modeling the two task types separately we capture the differences.

2. Indirect-modeling of task properties.

   We assume that tasks (of the same type) within the same job behave more or less similar, while tasks of different jobs might behave very differently. By using the indirect-modeling approach, we capture this behavior. If we use a direct-modeling approach for task properties, we get uniform values over the tasks of all the jobs, which might significantly impact the results.

Directly Modeled Properties

For the jobs we have two kinds of properties, properties that are directly used for the job characteristics themselves, and properties that are used as distribution parameters to sample the characteristics of the tasks of the job. For the directly-modeled properties we perform three steps:

1. We determine the maximum-likelihood estimates for the parameters of each distribution, based on (real) data in a workload trace (Section 5.2.1).

2. We calculate the goodness of fit of these distributions compared to the workload (Section 5.2.3).

Property	Type	Job	Task	Value
Inter-arrival time	D	+	+	Seconds
Executable ID	D	+	±	Integer ID
Run time	I	±	+	Seconds
Number of tasks	D	+	−	Count
Map/Reduce ratio	D	+	−	Fraction: Maps/Reduces
Forced-quit time	D	+	−	Seconds
CPU's	I	±	+	Count
Disk IO	C	±	+	Bytes
I/O ratio	I	±	+	Fraction: Input/Output
Memory	I	±	+	Bytes
Job exit state	D	+	−	Integer Coded State
Task exit state	I	−	+	Integer Coded State

Table 5.4: Properties included in our model. The symbols "D", "I", and "C", indicate respectively "directly modeled", "indirectly modeled", and "modeled based on correlation". The symbols "+", "±", and "−" indicate respectively "modeled", "implicit", and "not applicable".

3. We choose which of the distributions we will use to model the property (Section 5.2.4).

Using the steps above, we model directly the following properties:

Inter-arrival time The inter-arrival time is defined as the time elapsed between two job arrivals in the system. This metric can be used to generate job arrival times for synthetic workloads. From a trace, we calculate the inter-arrival times as the difference between the submit times of subsequent jobs.

Executable identifier Shows which executable is running as the job. Although we do not really use this property in this work, we include it because we expect it to be correlated to the other properties and this should be exploited in future work.

Run time The time that a job runs, i.e., the wall clock time elapsed since the job was started until the job finishes. The job run time is influenced by the run times of the individual tasks and how the individual tasks are scheduled.

Number of tasks The sum of the number of map tasks and the number of reduce tasks for the job.

Map/reduce ratio The ratio between map and reduce tasks. Together with the total number of tasks, this ratio can be used to determine the number of map and reduce tasks.

Forced-quit time The number of seconds elapsed since the job submission until the job is forced-quitted. A job might for example be forced-quitted by the user, by an administrator, and by maintenance scripts. In our model we do not cover the scenarios where individual tasks are preempted.

Job exit state The exit state of the job, shows whether the job finished successfully, failed, or was forced-quitted.

Indirectly-Modeled Properties

For different jobs, the properties of the tasks of a job might follow different distributions. We would like to capture this behavior in our model, albeit at the cost of a larger number of parameters. For the tasks of each job, we fit each task-properties to a probability distribution, and then we fit the parameters for these distributions by a meta-distribution. Since the parameters of a distribution function are generally not compatible with those of other distribution functions, we have to settle on a single probability distributions function per task-property.

For the indirectly-modeled properties, we first perform the same three steps used for the directly modeled properties, i.e., for each job we determine for each indirectly-modeled property the best fitting distribution and its parameters. We model task properties for map and reduce tasks separately. Next, we determine the all-over best fitting distribution function and we model its parameters. For the indirectly-modeled properties we perform three steps:

1. For each job in a (real) workload trace, we perform same three steps used for the directly modeled properties on the map and the reduce tasks of the job, separately:
   (a) For each property, we determine the maximum-likelihood estimates for the parameters of each distribution (see Section 5.2.1).
   (b) We calculate the goodness of fit of these distributions compared to the workload (see Section 5.2.3).
   (c) We choose which of the distributions we will use to model the property (see Section 5.2.4).
2. For each task-property:
   (a) We select the distribution that fits the largest number of tasks. Table D.13 summarizes the actual counting results for the task run times.
   (b) We model each parameter of the selected distribution with a meta-model, i.e., we apply again the same three steps on the parameter values: determining the maximum-likelihood estimates, calculating the goodness of fit, and selecting the best distribution. In this step, we only use the parameter values of fits that passed the goodness of fit tests.

Using the steps above, we model indirectly the following properties:

Task CPU The CPU demand of a task. Hadoop does not measure the CPU usage. Although it depends on the cluster configuration, it is reasonable to assume that there will be one CPU available per task. The real CPU usage can be measured by instrumenting the worker nodes.

Task disk I/O ratio The ratio between the amount of data read from and written to the disk by a task.

Task memory The memory demand of a task.

Task network The sum of the network send and received traffic of a task.

Task run time The time that a task runs, i.e., the wall clock time elapsed since the task was started until it finishes.

Task exit state The exit state of the task, shows whether the task finished successfully, or failed.

Correlated Properties

We have observed that the total amount of task disk IO is strongly correlated with the task run time (See Table 5.6). We assume that the total amount of disk IO h can be calculated from task run time r using $h = \alpha r + \beta$. For each job in a trace, we estimate α and β based on the values of h and r using NumPy's `polyfit`[2] function. Finally, we model both α and β using the directly model approach explained earlier in this section.

5.3.3 The Relaxed Complex Model

The complex modeling approach, described in Section 5.3.2, may be too restrictive to find good fits for the studied real workloads. Therefore we have explored in this section a less restrictive model, the Relaxed Complex Model. This model is derived from the Complex Model, with the selection criteria relaxed:

1. For the indirect modeling part, we have lowered the significance level from 0.05 to 0.02, resulting in the acceptance of less-well fitting distributions.

2. We have adapted the process for the selection of the distribution in the indirect modeling process. In the complex model we select a distribution based on the number of tasks that each distribution fits *best*, that is for each job its number of tasks is counted for (if any) the distribution that has the lowest D-statistic of all distributions that pass the goodness of fit tests. In this relaxed model, we select based on the number of tasks that each distribution fits *well*, that is for each job its number of tasks is counted for all distributions that pass the goodness of fit tests.

5.3.4 The Safe Complex Model

As an alternative to the Complex Model and the Relaxed Complex Model we have explored in this section the Safe Complex Model. This model still models the

[2] `http://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html`

same properties as the Complex Model, we still make the distinction between map and reduce tasks, we still exploit the correlation between run time and the amount of disk I/O, but we do no longer attempt to use indirect modeling for the task properties.

For each task property we use direct modeling to fit the values of the property for all the map tasks, and separately for all the values of the property for all the reduce tasks.

5.3.5 Modeling Results

We have used the modeling tool in the toolbox to fit the distributions described in Section 5.2.1 to model the properties in the SN1, SN2, Yahoo!, and Google workload traces. We encounter a number of problems. First, the SN1 workload trace does not contain any task-specific information. Second, not all properties are present in all the workload traces. We summarize the quality of the fits in Table 5.5. Task CPU and memory demands do not fit at all , since the traces contain for these properties only a single or very few distinct values.

Model	Inter-arrival Time	Run Time	I/O Ratio
Complex Model	bad fit	good fit	bad fit
Relaxed Complex Model	bad fit	good fit	bad fit
Safe Complex Model	bad fit	good fit	very bad fit
Simple Model	n/a	good fit	n/a

Table 5.5: Quality of the fits.

Correlations

In Table 5.6 we show for both the SN1 and the Yahoo workload traces the correlations between the available job properties. Based on the observed high correlation value for the total wall clock time and the disk usage, we have decided to model the task runtime from the amount of disk usage of the task.

If we look at the correlation values for the Yahoo! workload in Table 5.6b, we see that the fail and cancel factions highly correlate with each other. This is caused by the fact that the cancel fraction is 0 for all jobs, and the fail fraction is 0 for all but one jobs.

Directly-Modeled Properties

We show the results of the directly-modeling attempts in Tables D.1 through D.10. We note some strange things, for example, we would have expected the inter-arrival times (see Figure 5.2 and Table D.1) to be modeled using an exponential distribution, as this distribution is suitable for modeling a Poisson process such as inter-arrival times. However, in these specific modeled workload traces, it seems that

the Weibull distribution is the best at modeling the inter-arrival times. And surprisingly, the executable identifiers are modeled fairly well for the Google trace (see Figure 5.4 and Table D.6), likely this is caused by the large number of sequentially numbered distinct executable identifiers – making it a continuous distribution.

The wait times are not modeled well by any distribution for any of the modeled workloads, as can be seen in Table D.2. This is most probably caused by the fact that the wait times are very short, but have some large outliers. The job run times are fitted well for all the workload traces, see Table D.3 and Figure 5.3.

Indirectly-Modeled Properties

The indirectly-modeled properties are used in the family of complex models. The Complex Model, the Relaxed Complex Model, but not in the Safe Complex model.

The Complex Model In Tables D.12 through D.15, we show per distribution the percentage of the tasks that it fitted best, these tables are used to select a distribution for the model. The task run time is the only property that is being fitted well, the other properties are fitted very poorly. Unfortunately, the properties "disk I/O ratio" and "memory" were not fitted at all. This is likely caused by the ranges of the values for these properties, e.g., the amount of memory for a task in the Google trace is the total amount of memory of the node it runs on – resulting in only a few distinct values.

In Tables D.16 through D.19 we show the fitting results of the parameters for the selected distributions. Unfortunately, distribution function parameters are not fitted well. Goodness of fit values of more than the confidence level 0.05 are the exceptions, because of this we adapt our selection policy to select primary based on the D-statistic, except when a goodness of fit value of 0.05 or higher is available.

The Relaxed Complex Model For the Relaxed Complex Model, we have almost the same results as for the Complex Model. The percentages of task fits are shown in Tables D.20 through D.24, and the actual indirect-modeling fits are shown in Tables D.25 through D.28.

The fits for the task run times in the Relaxed Complex Model seem slightly better fits than those of the Complex Model, when we compare Tables D.17 and D.26.

The Safe Complex Model The fits for the Safe Complex Model are shown in Tables D.29 through D.32. We expected much better fits for the Safe Complex Model, hence the name, but here also the task run time is the only property that was fitted good.

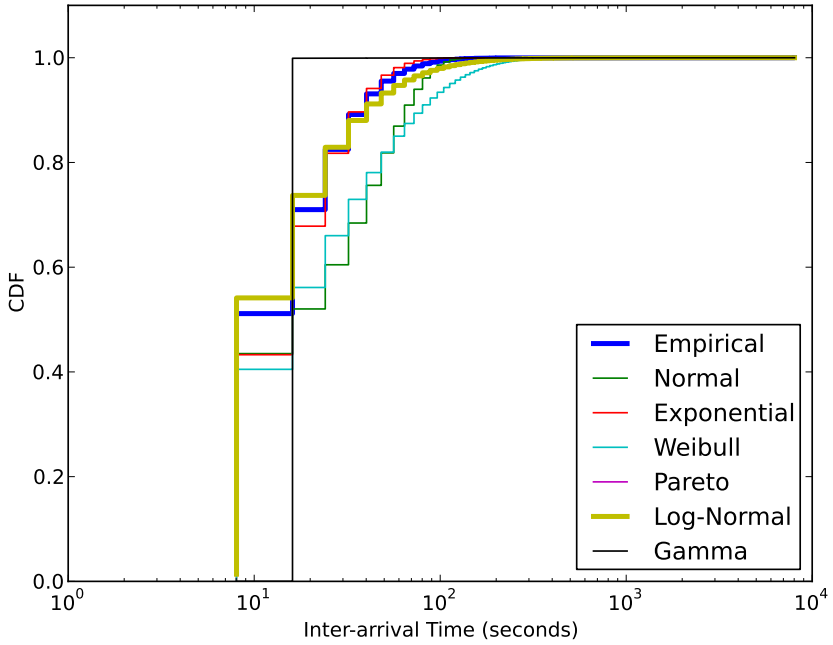


Figure 5.2: Job inter-arrival time fits (SN2 workload).

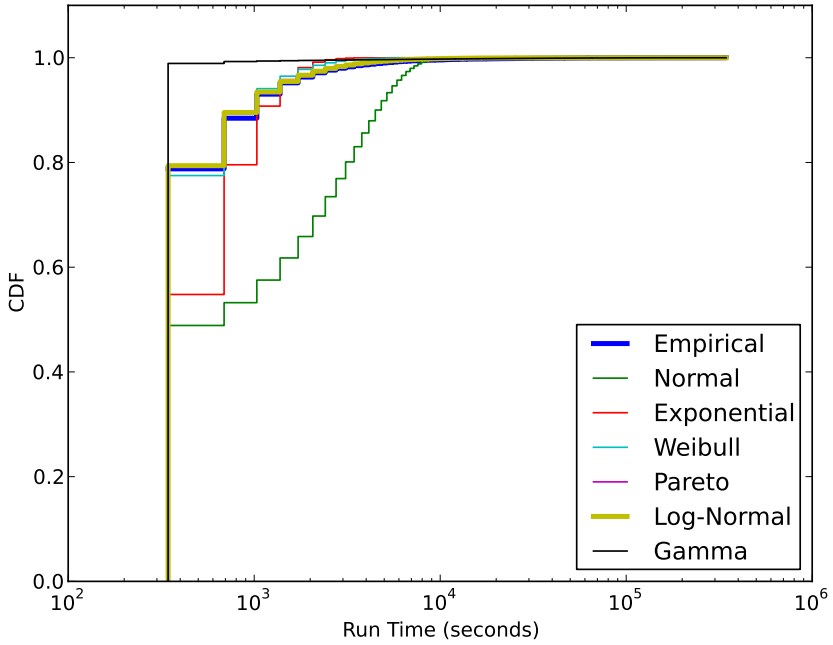


Figure 5.3: Job run time fits (SN2 workload).

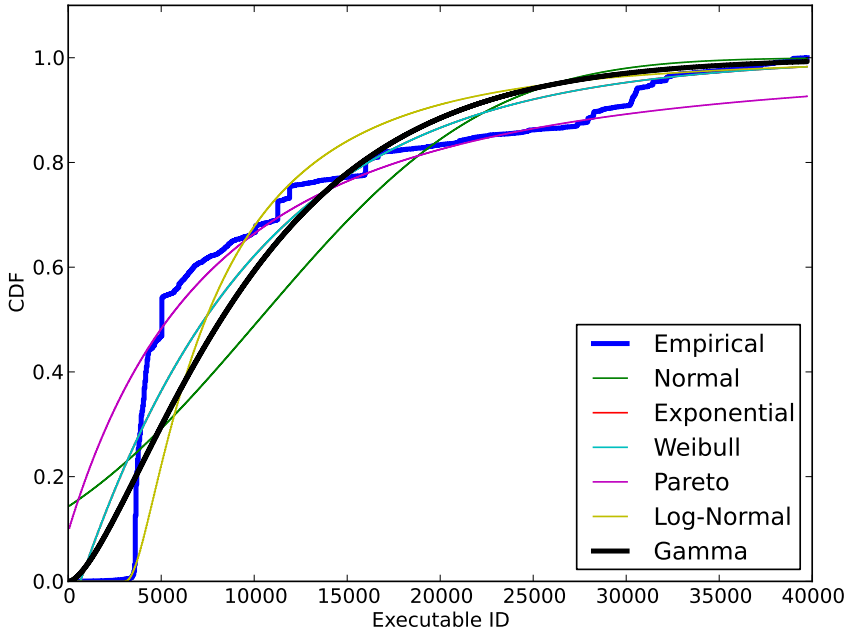


Figure 5.4: Job executable ID fits (Google workload).

property	InterArrivalTime	WaitTime	RunTime	CPUs	TotalWallClockTime	Memory	Network	Disk	Status	Tasks	Reduce Fraction	Fail Fraction	Cancel Fraction	Run Time Parameter 1
WaitTime	0.00													
RunTime	0.00	0.00												
CPUs	?	?	?											
TotalWallClockTime	0.00	0.01	**0.28**	?										
Memory	?	?	?	?	?									
Network	?	?	?	?	?	?								
Disk	0.00	0.00	**0.29**	?	**0.79**	?	?							
Status	?	?	?	?	?	?	?	?						
Tasks	?	?	?	?	?	?	?	?	?					
Reduce Fraction	?	?	?	?	?	?	?	?	?	?				
Fail Fraction	?	?	?	?	?	?	?	?	?	?	?			
Cancel Fraction	?	?	?	?	?	?	?	?	?	?	?	?		
Run Time Parameter 1	?	?	?	?	?	?	?	?	?	?	?	?	?	
Run Time Parameter 2	?	?	?	?	?	?	?	?	?	?	?	?	?	?

(a) SN1 trace.

property	InterArrivalTime	WaitTime	RunTime	CPUs	TotalWallClockTime	Memory	Network	Disk	Status	Tasks	Reduce Fraction	Fail Fraction	Cancel Fraction	Run Time Parameter 1
WaitTime	?													
RunTime	?	?												
CPUs	?	?	?											
TotalWallClockTime	-0.00	?	?	?										
Memory	?	?	?	?	?									
Network	?	?	?	?	?	?								
Disk	0.01	?	?	?	**0.52**	?	?							
Status	-0.05	?	?	?	-0.02	?	?	-0.04						
Tasks	0.01	?	?	?	**0.28**	?	?	**0.63**	-0.02					
Reduce Fraction	-0.02	?	?	?	-0.01	?	?	-0.09	**0.25**	-0.21				
Fail Fraction	0.00	?	?	?	0.05	?	?	0.02	-0.32	-0.01	-0.10			
Cancel Fraction	0.00	?	?	?	0.05	?	?	0.02	-0.32	-0.01	-0.10	**1.00**		
Run Time Parameter 1	?	?	?	?	?	?	?	?	?	?	?	?	?	
Run Time Parameter 2	?	?	?	?	?	?	?	?	?	?	?	?	?	?

(b) Yahoo! trace.

Table 5.6: Correlation between all the job properties. A question mark "?" indicates that a correlation value could not be calculated.

5.4 Synthetic MapReduce Workload Generator

We present a procedure for generating synthetic MapReduce workloads using the Simple Model in Section 5.4.1, and a procedure for generating synthetic MapReduce workloads using the family of Complex Models in Section 5.4.2.

5.4.1 Procedure using the Simple Model

We propose to generate synthetic workloads, from the Simple Model, using the procedure shown in pseudo-code in Algorithm 5.1. This procedure takes two kinds of inputs: a duration δ in seconds, and probability distributions D_{1-4}. Duration δ specifies the length of the period in seconds, during which job submissions are generated. Distributions D_{1-4} are used to sample values for various job- and task-characteristics. We show input distributions, the model parameters, in Table 5.7. Unfortunately, most of the MapReduce properties could be modeled for only a single workload trace. For the model parameters we have selected the best fitting distribution, regardless to which workload it belongs. Essentially we present the model parameters for a hypothetical system which borrows properties from all studied systems. Each of the properties corresponds to a real system.

The procedure generates a vector $\mathbf{a}$ and 2 two-dimensional vectors $\mathbf{y}$ and $\mathbf{r}$, which are all described in Section 5.4.2.

			Parameters			
Input	What	Function	Shape	Location	Scale	Remarks
D_1	Job Inter-Arrival Times	Weibull	1.224	-0.430	6.90	Table D.1, SN1
D_2	Number of Tasks	Log-Normal	2.54	0.941	93.2	Table D.7, Yahoo!
D_3	Reduce Task Ratio	Normal	n/a	0.108	0.208	Table D.8, SN2
D_4	Task Run Time	Weibull	0.531	0.000	82.0	Table D.11, Yahoo!

Table 5.7: Distributions as input for Algorithm 5.1.

5.4.2 Procedure using the Family of Complex Models

We propose to generate synthetic workloads, from the family of Complex Models, using the procedure shown in pseudo-code in Algorithm 5.2. This procedure takes two kinds of inputs: a duration δ in seconds, and probability distributions D_{1-19}. Duration δ specifies the length of the period in seconds, during which job submissions are generated. Distributions D_{1-19} are used to sample values for various job- and task-characteristics. We show input distributions, the model parameters, in Table 5.8. Unfortunately, most of the MapReduce properties could be modeled for only a single workload trace. For the model parameters we have selected the best fitting distribution, regardless to which workload it belongs. Essentially we present the model parameters for a hypothetical system which borrows properties from all studied systems. Each of the properties corresponds to a real system.

Algorithm 5.1 Algorithm for generating synthetic MapReduce workloads from the simple model.
Note: In this algorithm the notation $\bowtie(X)$ means "generate a random value from probability distribution X".

Input:

▷ δ, the duration in seconds during which to generate Jobs.

▷ D_{1-4}, the distributions specified in Table 5.7.

Output:

▷ $\{\mathbf{a}, \mathbf{y}, \mathbf{r}\}$, a synthetic MapReduce workload.

```
1:  loop
2:    i ←⋈ (D_1) {interarrival time}
3:    if max(a) + i ≤ δ then
4:      a_{|a|+1} ← max(a) + i {submit time for a new Job}
5:    else
6:      exit loop
7:  for j = 1 to |a| do
8:    n_j ←⋈ (D_2) {number of tasks}
9:    nr_j ←⋈ (D_3) {map-/reducetasks ratio}
10:   for t = 1 to n_j do
11:     y_{j,t} ←⋈ (nr_j) {task type (map/reduce)}
12:     r_{j,t} ←⋈ (D_4) {task run time}
```

The procedure generates a set of 7 vectors and 5 two-dimensional vectors which are specified below, each containing information about a characteristic, such as the job arrival time. The j-th element of each vector stores information about the j-th job in the workload. For the task-specific characteristics, the t-th sub element of the vector stores information about the t-th task of the j-th job. We discuss in the following each output vector of this procedure in turn, starting with the job-specific output vectors.

Job Arrival Times (vector $\mathbf{a}$) The procedure generates vector $\mathbf{a}$ with the arrival times for all jobs in the workload. The contents of this vector are generated by repeatedly sampling job inter arrival times from distribution D_1 and calculating the arrival times from the inter arrival times. Job submissions are generated for a period of length δ, i.e., $max(\mathbf{a}) - min(\mathbf{a}) \leq \delta$.

Executable Identifier (vector $\mathbf{x}$) The procedure generates an integer executable identifier $\mathbf{x}_j$, for each job j by sampling from distribution D_2 and rounding to an integer value. Although $\mathbf{x}$ is not used in our model, in the future we intend to extend the use of distributions D_{3-19} such that the correlation between them and the executable is explicitly modeled.

Tasks Count (vector $\mathbf{n}$) The procedure generates the total number of tasks $\mathbf{n}_j$ for each job j by sampling from distribution D_3.

Map/Reduce tasks Ratio (vector $\mathbf{nr}$) The procedure generates a map/reduce tasks ratio $\mathbf{nr}_j$ for each job j by sampling from distribution D_4.

Forced Quit Time (vector $\mathbf{q}$) Although MapReduce runtime systems are fault-tolerant (they support task restarts), jobs may still fail to complete (see Section 5.3.2). We generate a vector $\mathbf{q}$ with a forced quit time for each job by sampling from distribution D_5. This forced quit time is the number of seconds elapsed since the submission of the job and until the user forced quits the job. For jobs with $\mathbf{q}_j \leq 0$, the job is not forced quitted.

Two three dimensional vectors are used internally, these vectors contain job-specific values used to generate task-specific characteristics. Values for these two vectors are sampled per job, for both map tasks and reduce tasks.

Job-specific Distributions (vector $\mathbf{d}_{1-5}$) Job-specific distributions are used to generate values for the properties of all the tasks of a job: task type, CPU demand, disk i+o, disk i/o ratio, memory demand, and the task exit state. The properties of the individual tasks of a job are generated from the job-specific probability distributions $\mathbf{d}_{1-5}$. For each task property we have chosen a fixed type of distribution (see Section 5.3.2), with its parameters sampled from the probability distributions D_{6-15}.

Task-specific Parameters (vector $\mathbf{p}_{1-2}$) The procedure generate task-specific parameters $\mathbf{p}_{1-2}$ by sampling from distributions D_{16-19}. These task-specific parameters are used to calculate the task disk IO (see line 20 in Algorithm 5.2).

The procedure generates seven two-dimensional task-specific output vectors.

Task Type (vector $\mathbf{y}$) The procedure generates for task t of job j a task type $\mathbf{y}_{j,t}$ by sampling from distribution $\mathbf{nr}_j$.

CPU Demand (vector $\mathbf{c}$) The procedure generates for task t with type y of job j the CPU $\mathbf{c}_{j,t}$ by sampling from distribution $\mathbf{d}_{1,j,y}$.

Runtime (vector $\mathbf{r}$) The procedure generates for task t with type y of job j the runtime $\mathbf{r}_{j,t}$ by sampling from the task-specific distribution $\mathbf{d}_{2,j,y}$.

Disk I+O (vector $\mathbf{h}$) The procedure generates for task t with type y of job j the sum of the task's disk in- and output $\mathbf{h}_{j,t}$ as function of $\mathbf{r}_{j,t}$ and the two parameters $\mathbf{p}_{1,j,y}$ and $\mathbf{p}_{2,j,y}$. As can be seen in Tables 5.6a and 5.6b, there is a strong correlation between the amount of disk I+O and the task runtime. In the proposed function for calculating $\mathbf{h}$ we exploit this correlation.

Disk I/O Ratio (vector $\mathbf{hr}$) The procedure generates for task t with type y of job j the disk I/O ratio $\mathbf{hr}_{j,t}$ by sampling from the task-specific distribution $\mathbf{d}_{3,j,y}$.

Memory (vector $\mathbf{m}$) The procedure generates for task t with type y of job j the memory usage $\mathbf{m}_{j,t}$ by sampling from the task-specific distribution $\mathbf{d}_{4,j,y}$.

Task Exit State (vector $\mathbf{e}$) The procedure generates for task t with type y of job j the exit state $\mathbf{e}_{j,t}$ by sampling from the task-specific distribution $\mathbf{d}_{7,j,y}$.

			Parameters				
Input	**What**	**Function**	**Shape**	**Location**	**Scale**	**Remarks**	**D**
D_1	**Job Inter-Arrival Times**	Log-Normal	1.22	-0.430	6.90	Table D.1, SN1	0.05
D_2	**Executable**	Gamma	1.58	33.8	6460	Table D.6, Google	0.26
D_3	**Number of Tasks**	Log-Normal	2.54	0.941	93.2	Table D.7, Yahoo!	0.13
D_4	**Map/Reduce Ratio**	Normal	n/a	0.108	0.208	Table D.8, SN2	0.35
D_5	**Forced-quit Time**	Normal	n/a	2350	34800	Table D.9, Google	0.45
D_6	**Map Task CPU Demand**	Normal				Table D.18, Google	
	loc	Weibull	1.00	1.72	0.380		0.29
	scale	Normal	n/a	0.098	0.227		0.48
D_7	**Map Task Run Time**	Normal				Table D.17, Yahoo!	
	loc	Log-Normal	1.25	-0.024	15.8		0.09
	scale	Weibull	0.503	0.00	6.69		0.14
D_8	**Map Task I/O Ratio**	Gamma				Table D.19, Yahoo!	
	shape	Log-Normal	3.09	0.093	38.0		0.11
	loc	Gamma	0.108	0.00	324		0.34
	scale	Weibull	0.266	0.00	0.061		0.07
D_9	**Map Task Memory**	no fits					
D_{10}	**Map Task Exit State**	no fits					
D_{11}	**Reduce Task CPU Demand**	no fits, Use D_6					
D_{12}	**Reduce Task Run Time**	Gamma				Table D.17, Yahoo!	
	shape	Pareto	0.303	-0.828	0.863		0.12
	loc	Weibull	0.338	0.00	44.6		0.22
	scale	Pareto	0.748	-0.718	0.718		0.06
D_{13}	**Reduce Task I/O Ratio**	no fits, Use D_8					
D_{14}	**Reduce Task Memory**	no fits					
D_{15}	**Reduce Task Exit State**	no fits					
D_{16}	**Map Task Disk Param.** α	Log-Normal	6.10	0.00	720000	Table D.4, Yahoo!	0.20
D_{17}	**Map Task Disk Param.** β	Normal	n/a	200000	7040000	Table D.5, Yahoo!	0.29
D_{18}	**Reduce Task Disk Param.** α	Use D_{16}					
D_{19}	**Reduce Task Disk Param.** β	Use D_{17}					

Table 5.8: Distributions as input for Algorithm 5.2.

5.5 Concluding Remarks

In this chapter we have shown the advantages models and synthetic workloads. We have proposed models for MapReduce workloads, as well as procedures to generate realistic MapReduce workloads based on these models. Unfortunately, the proposed indirect modeling approach does not work well. We find that not only is it difficult to find a distribution that fits a task property well for a significant amount of the total tasks, it also turns out to be difficult to fit the parameters of such a distribution using meta-distributions.

Algorithm 5.2 Algorithm for generating synthetic MapReduce workloads.
Note: In this algorithm the notation $\bowtie(X)$ means "generate a random value from probability distribution X".

Input:

▷ δ, the duration in seconds during which to generate Jobs.

▷ D_{1-19}, the distributions specified in Table 5.8.

Output:

▷ $\{\mathbf{a}, \mathbf{x}, \mathbf{q}, \mathbf{y}, \mathbf{c}, \mathbf{h}, \mathbf{hr}, \mathbf{m}, \mathbf{r}, \mathbf{e}\}$, a synthetic MapReduce workload.

1: **loop**
2: $i \leftarrow\!\bowtie(D_1)$ {interarrival time}
3: **if** $\max(\mathbf{a}) + i \leq \delta$ **then**
4: $\mathbf{a}_{|\mathbf{a}|+1} \leftarrow \max(\mathbf{a}) + i$ {submit time for a new Job}
5: **else**
6: **exit loop**
7: **for** $j = 1$ **to** $|a|$ **do**
8: $\mathbf{x}_j \leftarrow\!\bowtie(D_2)$ **{executable id}**
9: $\mathbf{n}_j \leftarrow\!\bowtie(D_3)$ **{number of tasks}**
10: $\mathbf{nr}_j \leftarrow\!\bowtie(D_4)$ **{map-/reducetasks ratio}**
11: $\mathbf{q}_j \leftarrow\!\bowtie(D_5)$ **{forced-quit time}**
12: $\mathbf{d}_{1-5,j,\mathsf{MAP}} \leftarrow\!\bowtie(D_{6-10})$ **{task probability distributions}**
13: $\mathbf{d}_{1-5,j,\mathsf{RED}} \leftarrow\!\bowtie(D_{11-15})$ **{task probability distributions}**
14: $\mathbf{p}_{1-2,j,\mathsf{MAP}} \leftarrow\!\bowtie(D_{16-17})$ **{task probability distributions}**
15: $\mathbf{p}_{1-2,j,\mathsf{RED}} \leftarrow\!\bowtie(D_{18-19})$ **{task probability distributions}**
16: **for** $t = 1$ **to** $\mathbf{n}_j$ **do**
17: $\mathbf{y}_{j,t} \leftarrow\!\bowtie(\mathbf{nr}_j)$ **{task type (map/reduce)}**
18: $\mathbf{c}_{j,t} \leftarrow\!\bowtie(\mathbf{d}_{1,j,\mathbf{y}_{j,t}})$ **{CPU}**
19: $\mathbf{r}_{j,t} \leftarrow\!\bowtie(\mathbf{d}_{2,j,\mathbf{y}_{j,t}})$ **{run time}**
20: $\mathbf{h}_{j,t} \leftarrow \mathbf{r}_{j,t} \times \mathbf{p}_{1,j,\mathbf{y}_{j,t}} + \mathbf{p}_{2,j,\mathbf{y}_{j,t}}$ **{disk io}**
21: $\mathbf{hr}_{j,t} \leftarrow\!\bowtie(\mathbf{d}_{3,j,\mathbf{y}_{j,t}})$ **{disk i/o ratio}**
22: $\mathbf{m}_{j,t} \leftarrow\!\bowtie(\mathbf{d}_{4,j,\mathbf{y}_{j,t}})$ **{memory}**
23: $\mathbf{e}_{j,t} \leftarrow\!\bowtie(\mathbf{d}_{5,j,\mathbf{y}_{j,t}})$ **{exit state (succes/fail)}**

Chapter 6

Building Better Systems

The key element in building better systems is being able to evaluate how well they perform. Therefore, one of the main goals of our work is to evaluate and compare existing, enhanced, and new MapReduce systems. In this chapter we present a method to evaluate systems using synthetic workloads and simulations. We also perform an experiment to validate our MapReduce system assessment approach. With the work presented in this chapter we fulfill technical objective T4 and provide an answer to research question Q4: "Which MapReduce scheduler performs best in scheduling a certain workload?"

In Section 6.1 we survey the available MapReduce simulators and select one of them. In Section 6.2 we present our experimental setup. In Section 6.3 we present the results of our experiment. We conclude in Section 6.4.

6.1 Assessing MapReduce Systems in Simulation

We have chosen to simulate the execution of the workloads instead of actually executing them on a real MapReduce cluster. Executing the workloads on a real MapReduce cluster is undesirable for a number of reasons. We identify four such reasons. First, running tests on a production cluster will interfere with production workloads, so testing would require one or more separate test clusters. Second, using a simulator, cluster configurations of arbitrary size can be evaluated. Third, simulations might run faster than real time, and many simulations can run in parallel, thus potentially saving huge amounts of time. Finally, specific simulators might also allow you to investigate circumstances which are difficult to reproduce in a real cluster, like for example hardware failures.

Using a simulator also has its flaws; none of the existing simulators captures all aspects of a real MapReduce environment. However, we believe that the advantages of using a simulator outweigh the drawbacks.

6.1.1 Overview of MapReduce Simulators

We have chosen to use an already existing MapReduce simulator. We present the list of simulators we have considered in Table 6.1. Each of these MapReduce simulators was developed with a specific use in mind, which leads to the distinct features shown in the table.

Simulator	Open	Last Release	Build On	Language	Scheduler	Distinct Feature
MRPerf [6, 40]	+	2009-7-14	ns 2.33	C++	Custom (in TCL)	Low-level Network
Cardona et al. [11]	–	?	GridSim	Java	Custom (in Java)	HDFS
MrSim [12]	+	2010-10-27	GridSim	Java	Custom (in Java)	Low-level I/O
Mumak [13]	+	2011-06-12	Hadoop	Java	Hadoop (in Java)	Hadoop Scheduler

Table 6.1: Comparison of MapReduce simulators.

MRPerf by [6, 40] is built on an old version of the ns[1] network simulator, which requires libraries that are no longer shipped with modern Linux distributions, so it is difficult to get running.

Cardona et al.'s [11] simulator is not made available, so it can not be used by other researchers. The simulator was developed to evaluate schedulers for the distributed file system.

MrSim was developed, according to [12], because none the other simulators mentioned in Table 6.1 were able to deliver accurate results. Although, MrSim was shown to be superior in a constrained environment, we have decided not to use it for one main reason: MrSim has no support for pluggable schedulers.

Mumak is a MapReduce simulator included with Hadoop. It is fairly easy to get running, makes use of the native Hadoop schedulers, and is input- and output-compatible with Gridmix – a tool that is able to run synthetic workloads on a real cluster.

6.1.2 Mumak, with the help of Rumen

Mumak [13] works together with Rumen [10] to simulate the execution of MapReduce jobs on a single cluster. These tools were first introduced to the public in two separate bug reports in July 2009, and committed into the Hadoop SVN in September 2009. They are included in the unstable Apache Hadoop versions 0.21 (August 2010) and 0.22.

[1] `http://isi.edu/nsnam/ns/`

Mumak

Mumak is a MapReduce simulator. It simulates the JobTracker (the component in Hadoop that accepts job submissions and schedules the execution of their tasks) and all TaskTrackers (the Hadoop component that executes tasks on worker nodes), and it submits jobs to the JobTracker. Mumak has three policies to submit jobs to the JobTracker:

1. The REPLAY policy replays the input workload, adhering to the submit times in this workload.

2. The SEQUENTIAL policy does not adhere to the submit times, but submits the next job as soon as the previous job has finished.

3. The STRESS policy does not adhere to the submit times, but keeps submitting the next jobs until a specific load level is reached.

The configuration format for Mumak is almost the same as for Hadoop itself; there are just some additional properties that can be configured, like for example the job submission policy. When starting Mumak, paths to a cluster topology file and a workload description file are specified as arguments. The cluster topology file is an XML-encoded tree of nodes; the current version requires all the leaves of this tree to be at the same level. The workload file is a JSON-encoded file, describing all the jobs and their tasks.

Rumen

Rumen is a tool that extracts job models and the cluster topology from Hadoop log files, and produces a JSON-encoded workload file and an XML-encoded topology file for Mumak. Using these input files, Mumak is able to perform the simulation. Rumen is also used as a library by Mumak, to read the workload input file.

At first glance Rumen seems to have an overlap with our work in Chapter 5. However, when inspecting Rumens inner workings, it becomes clear that Rumen is only able to generate a *replay* of the original log files. This replay contains the same jobs as the original Hadoop log files. Rumen only produces a very limited distribution function for run times, it calculates discrete CDF's, which Mumak uses to sample run times at run time (via calls to Rumen). Rumen does not look for correlations; it does however apply a configurable constant slowdown to tasks that are not allocated to the preferred task-executing resource specified in the workload description.

In our work, on the other hand, we model many properties with probability distribution functions, and we also exploit correlations between properties. We extract parameters for this model from original log files, and we use our model and the extracted parameters, to generates realistic synthetic workloads of arbitrary size.

In the remainder of this work, we use the term Mumak, collectively, for both Rumen and Mumak.

6.1.3 Mumak Selected!

We have chosen to use Mumak as simulator for our experiments. The main reasons for this choice are that:

- Mumak is bundled with Hadoop versions 0.21 and 0.22.
- Mumak uses unaltered versions of the Hadoop Schedulers, so all schedulers bundled with Hadoop can be used. Also, if a researcher develops a scheduler for Mumak, it can be used unaltered in a real Hadoop environment.
- Mumak is, input- and output-compatible with Gridmix3 [14], a tool that is able to run synthetic workloads on a real Hadoop cluster, instead of simulating the execution.

Mumak also has some disadvantages, of which the most notable are:

- Mumak lets all reduce tasks run until the last map task has finished. This leads to longer task run times in the simulation, than expected in real execution.
- Mumak does not take into account the bandwidth and latencies of the HDFS, network, and disk I/O.
- Mumak does not simulate the partition and shuffle phases (see Section 1.1).
- Mumak is not able to simulate failures in the cluster.

6.2 Experimental Setup

We perform an experiment to validate our MapReduce system evaluation approach. In this experiment we compare Hadoop's default FIFO scheduler with the Fair scheduler by [2]. In our experiment we run simulations for these 2 schedulers, for 8 different load levels, and for 4 different workload lengths; each of these combinations is repeated 6 times, this adds up to a total of 384 simulations. Because this amount of work is too demanding for a single computer, we have run our simulations on the DAS-3 and DAS-4 super-computers.

We have created an simulation toolbox that automates the individual parts of the evaluation process when using super-computers:

- Generation of all the synthetic workloads, and the distribution of the generated workloads to the cluster sites of the super-computers.
- Generation of all simulation jobs to run on the super-computers, a job consists of the following steps:
  - Configure the simulator to simulate the specified MapReduce cluster and scheduler configurations.

  - Let the simulator simulate the execution of the specified MapReduce workload.
  - Create an archive containing the results of the simulation.
  - Clean up all temporary files.

- A simple meta-scheduler submits the generated simulation jobs to cluster sites of the super-computers where a slot is available. The cluster scheduler will then execute the simulation job on one of the nodes of the super-computer.
- Gathering of the simulation result archives from the super-computers, and the analysis of these results.
- Plotting graphs in which the results of the various simulations are shown next to each other, for easy comparison.

Although, in this process we use Mumak as the simulator, other users can easily use any other simulator. There are only two components that need to be adapted: First, the workload definitions and cluster configurations need to be converted in a format that the simulator accepts. Second, the traces outputted by the simulator need to be converted into the CWA data format.

6.2.1 Simulated Workloads

In our work we do not use Rumen to generate an input workload file for Mumak, but instead we generate this file ourselves. We then let Mumak execute this workload using the REPLAY submission policy.

We have chosen to generate synthetic workloads based on our MapReduce workload models. We want to accommodate the evaluation of systems that vary significantly in size, and we want to evaluate systems under various levels of load. For this we need workloads that impose a specified load on a system of specified size, while containing jobs that are statistically the same across the workloads. Obtaining workloads that meet these constraints from traces of real MapReduce clusters is infeasible. As discussed in Chapter 5, the best way to obtain these workloads is to generate synthetic workloads. We generate workloads based on our simple model (see Section 5.3.1) and not based on one of the complex models (see Sections 5.3.2-5.3.4) because the modeling and generating using the complex models is much more time consuming, we leave this for furture work.

We generate synthetic workloads for 8 load levels (1%, 30%, 40%, 50%, 60%, 70%, 80%, and 90%), and 4 different durations (1, 6, 24, and 96 hours) based on our procedure in Section 5.4.2. With a load level we mean the average task slot occupation, which we denote as a percentage. We calculate the load level as $\lambda = \frac{100}{n \cdot s \cdot \delta} \cdot \sum_i \mathbf{t}_i$, where λ is the load level percentage, $\mathbf{t}$ is a vector of all task run times, n is the number of nodes in the system, s is the number of task slots per node, and, finally, δ is the duration for which we generate a workload.

To be able to simulate the workloads on Mumak we have added two additional rules to the workload generator:

1. Each job has at least one map task, as reduce-only jobs always fail in Mumak.

2. The run time of each task is capped at seven days. This prevents the generation of extreme long running tasks (in the case we are sampling from a long tailed distribution). This is a reasonable maximum run time, as it is a more relaxed constraint than in some existing production clusters, where *jobs* running longer than seven days are automatically killed.

In the workloads we generate jobs of two different application types: jobs of application type 0 and jobs of application type 1. The jobs of application type 1 have shorter inter-arrival times, less tasks, and shorter task run times, than jobs of application type 0.

The model parameters we used to generated these workloads are shown in Table 6.2. These model parameters have been selected before all the modeling was done, so therefore they deviate from the parameters shown in Table 5.7, still we have chosen fairly good values. The most important of these parameters, the (task) run time is on average 60 seconds for application 0 and on average 15 seconds for application 1, which is in the same range as in Table 5.7 where the all-over average task run time is about 40 seconds, and we have chosen the same distribution. In this table the Map/Reduce ratio shows the ratio of the maps, while table Table 5.7 shows the reduce ratio, here we also have chosen the same distribution and the same range of on average about 10% reduce tasks. For the number of tasks, we have selected the Weibull distribution while for the Yahoo traces, the log-normal distribution is a slightly better fit, as can be seen in Table D.7. The number of tasks per job are on average 154, 67, and 980 for respectively the SN1, Google, and Yahoo workload traces. So the average number of tasks in our chosen parameters of about 50 for application 0 and 10 for application 1 are on the low side. The inter-arrival time is adapted by the workload generator to obtain the requested load-level, and the wait time is indirectly a part of the inter-arrival time.

App.	Inter-arrival Time			Wait Time			Num. Tasks			Map/Reduce Ratio			Run Time		
0	Weibull	50	0.5	Weibull	2	0.5	Weibull	100	0.5	Normal	0.85	0.05	Weibull	120	0.5
1	Weibull	20	0.5	Weibull	1	0.5	Weibull	20	0.5	Normal	0.9	0.05	Weibull	30	0.5

Table 6.2: Model parameters.

In Figure 6.1 we show the run time CDFs of the 6-hour workloads generated by our workload generator based on these model parameters. We observe that there is no significant difference in the CDFs of the workloads generated for different load levels. In Figure 6.2 we show the same CDFs, but now with the horizontal axis truncated at 200 seconds, here we observe that the median is somewhere between $120 \times 0.5 = 60$ and $30 \times 0.5 = 15$ (the expected means for the run times of the two applications based on the distribution). Based on these two observations

we conclude that the generated workload is valid, that is, it matches the parameter values selected for this experiment.

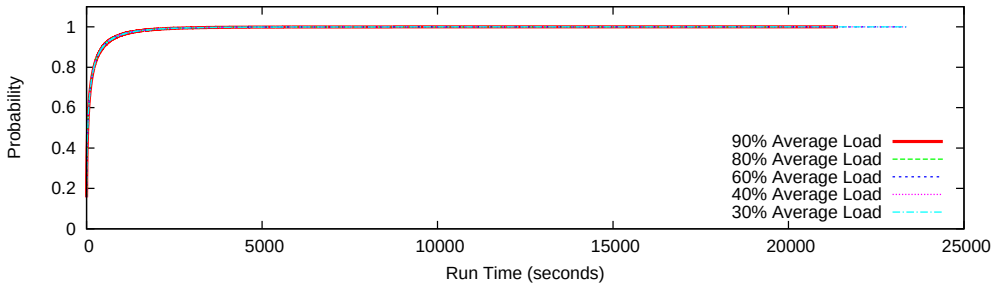


Figure 6.1: Task run time CDFs for various load levels of the generated six hour long workloads.

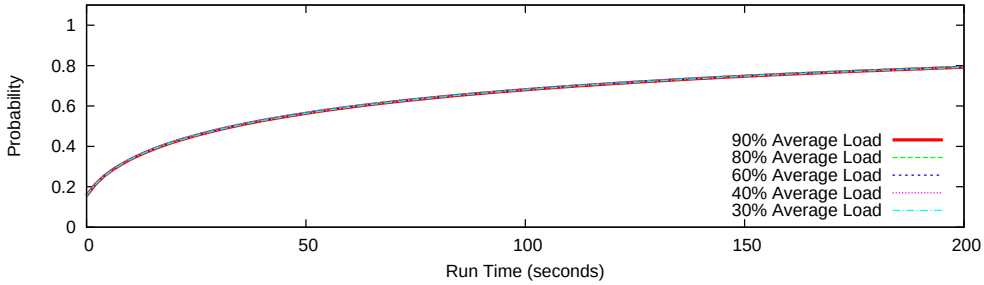


Figure 6.2: Task run time CDFs, capped at 200 seconds, for various load levels of the generated six hour long workloads.

6.2.2 Topology of the Simulated Cluster

In our experiment we used the example "19-job" topology that is included in the Mumak source code. This cluster topology consists of 1545 (TaskTracker) nodes distributed over 41 racks. We have configured Mumak to have four map task slots and four reduce task slots per simulated node.

6.2.3 Configuration of the Simulated Scheduler

In this experiment we compare two MapReduce schedulers, Hadoop's default (FIFO) scheduler and the Fair scheduler. We have used Hadoop's default scheduler without any modification.

The Fair scheduler work with *pools* that each get allocated *fair share* of the system. In our workloads we have two types of applications, a less frequent and heavier application 0, and a more frequent and lighter application 1 (see Section

6.2.1). For the Fair scheduler we have defined, next to the default pool (which will only be used by applications of type 0), an additional pool for applications of type 1. This pool has been configured with triple the weight of the default pool, and on top of that a guaranteed capacity of 500 map slots and 100 reduce slots.

6.2.4 Evaluation Metrics

After we have simulated the execution of all the workloads on all the systems under test, we analyze the results. When these analyses are finished, we need a way to decide what system performed best, so, we need an answer to research question Q4: "Which MapReduce scheduler performs best in scheduling a certain workload?"

We cannot give a generic answer to this question, as the this depends on the specific needs of the user. Nonetheless, we propose two metrics that could be used to find the best system. The two proposed metrics are *job response time* and *cost*. For both these metrics it holds that: the lower, the better.

Job response time is the wall clock time elapsed since a job has been submitted up until the job has finished. For production jobs the job response time should be low enough to make the job deadline, for interactive jobs the job response time should be as low as possible.

Cost is the number of node-hours used to execute the entire workload. An entire node hour is counted, if during an hour a distinct node has been executing at least one task. This is like the pricing scheme of Amazon EC2, and can thus be used to approximate the cost of "running the workload in the cloud."

6.3 Experimental Results

The actual simulations of the 6-hour workloads took from $5\frac{1}{2}$ minutes for the 1% load level workload, up to $5\frac{1}{2}$ hours for the 90% load level workload. Now we have performed the simulations, we make a decision on which of the two schedulers performed best. We first show the operational profile, then we show the analysis of the two evaluation metrics we selected; the job response times and the cost.

6.3.1 Simulator Validation Through Operational Profile

We show the number of running jobs and tasks over time in respectively Figures 6.3 and 6.4. The figures show the count of tasks and jobs that ran during 10-minute intervals, so in the likely case that short jobs ran in such an interval, the count can be larger than the total number of task slots.

These figures show that the simulator actually executed the workloads, and that the entire workloads finished in a reasonable time. Apart from this validation of the simulator there are already three things, concerning the evaluation of the two systems, that we observe in these two figures.

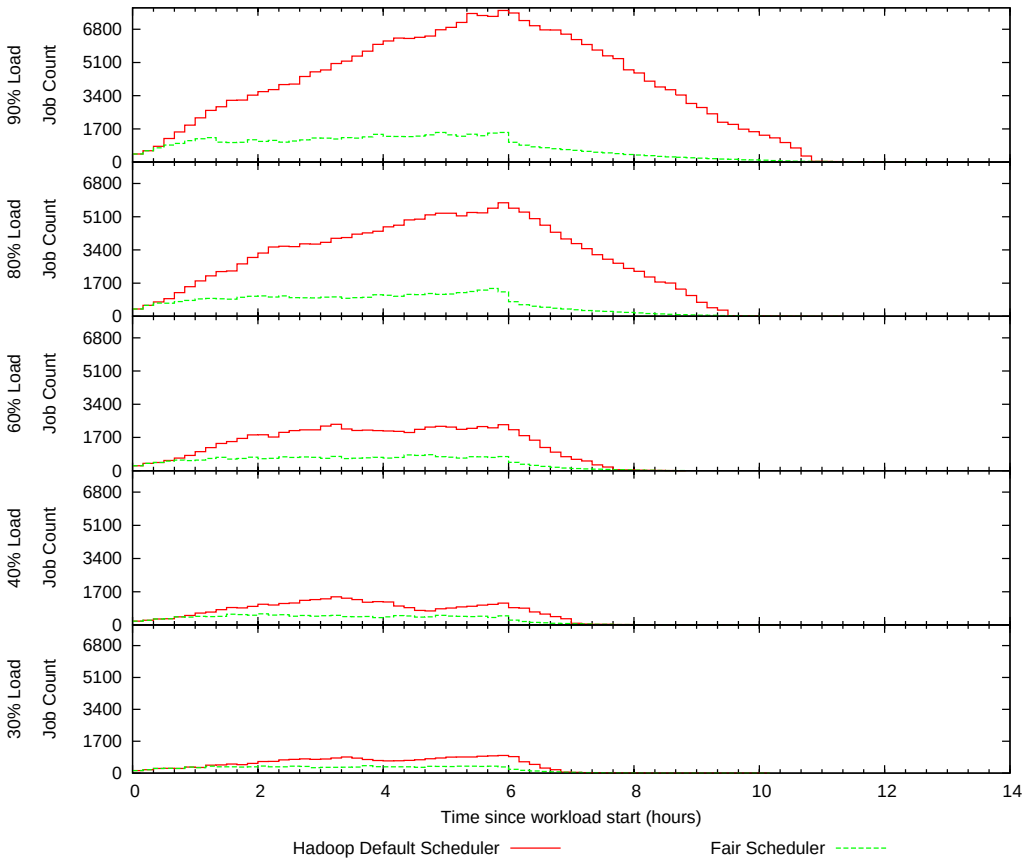


Figure 6.3: Number of running jobs for both scheduling policies under various load levels.

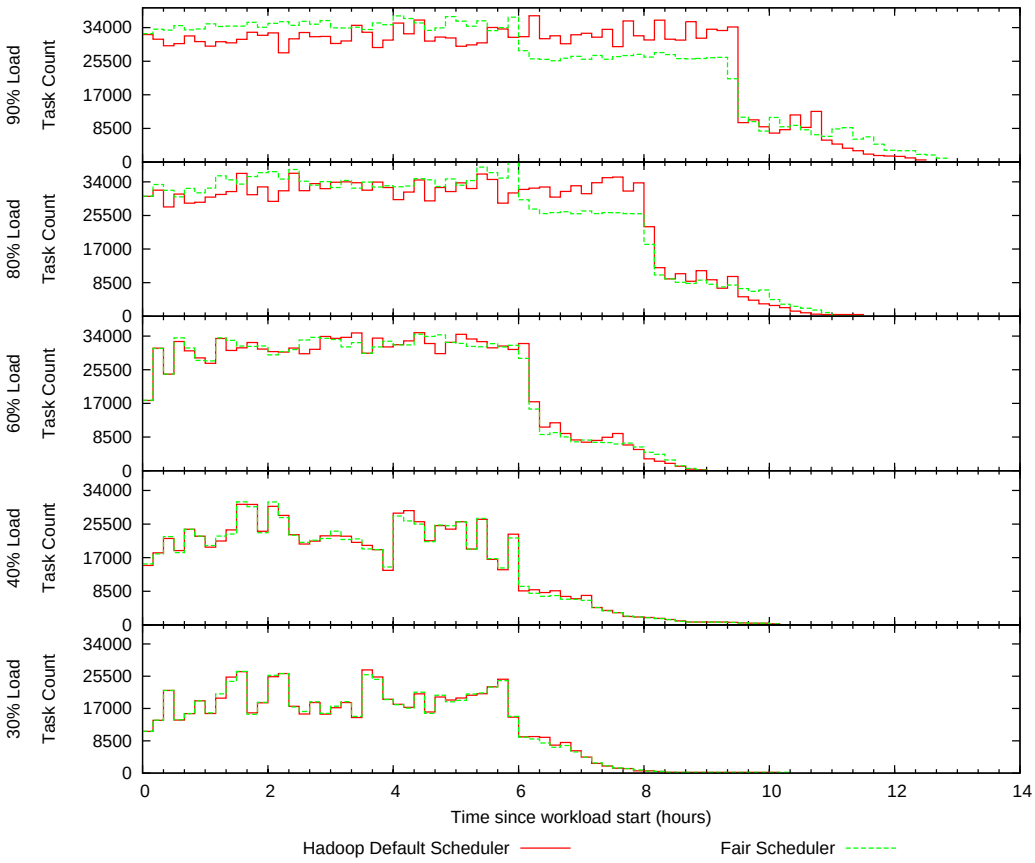


Figure 6.4: Number of running tasks for both scheduling policies under various load levels.

The first thing we observe is the large difference in the number of concurrent running jobs between the two schedulers, which increases with the load. From this difference we can already conclude that the Fair scheduler, which keeps the number of concurrent running jobs significantly lower than Hadoop's default scheduler, will have lower job run times than the default scheduler. This difference is not visible for the number of running tasks over time, but a large difference is not likely to happen, as the number of task slots is finite.

The second thing we observe is, that there is no significant difference in the time needed to execute the entire workload. So, on the whole, the utilization of the cluster does not really differ between the two schedulers, the only difference will be observed in the individual jobs.

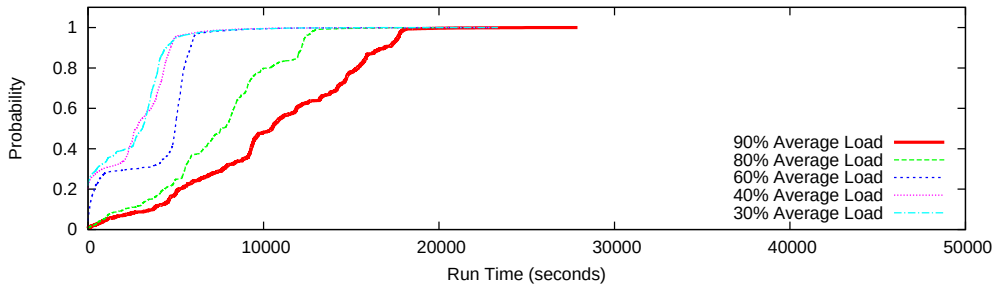
The third observation is that although we observe a drop in jobs and tasks after the six hour period, for which we generated the workload, it still takes some time to finish all jobs. This additional tail is caused by three things. For one, jobs that were submitted just before the end of the six hour period, can not finish earlier than the moment which is sum of their submit time and their longest running task, this will be the main cause for the lower load levels. Also, it takes some time to "ramp up the system", while the desired load level is calculated over the entire duration, including the ramp up time, during which there are not yet enough tasks in the system to occupy all task slots. The third cause will be that the scheduler may not be able to use the task slots optimal, this will be the main cause for the higher load levels.

6.3.2 Analysis of Job Response Times

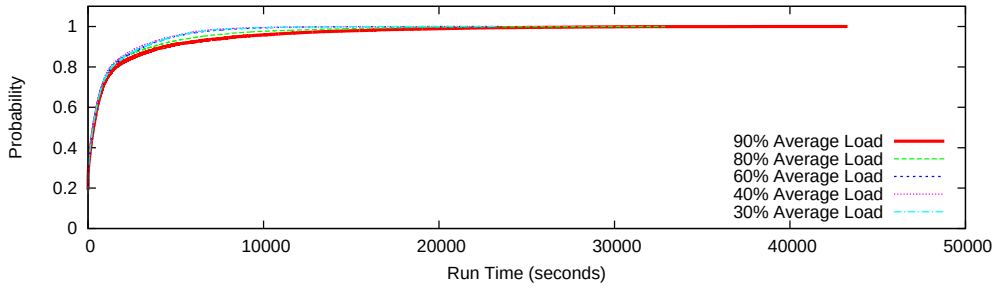
We now evaluate the run times of the jobs for the two scheduling policies under five different load levels. For this evaluation we compare the cumulative distribution functions of the job run times for Hadoop's default scheduler in Figure 6.5a, with those for the Fair scheduler in Figure 6.5b.

These graphs show a slow and erratic increase of the CDF value from 0.0 to 0.9 for the Hadoop's default scheduler, while the same increase for the Fair scheduler happens fast and steady. From this observation we conclude that, for the majority of the jobs, the job response times are significantly shorter when scheduled by the Fair scheduler. However, we also observe a difference in the length of the curves. For the default scheduler the curves stop before 30,000 seconds, while for the Fair scheduler the curves continue to well over 40,000 seconds. This indicates that a relatively small number of jobs has a significantly longer job run times, when scheduled by the Fair scheduler.

This behavior is expected from the Fair scheduler. Looking back at both the workload generation in Section 6.2.1 and the configuration of the Fair scheduler in Section 6.2.3, we see that we have configured the Fair scheduler to use separate pools for the heavier and the lighter, and that we have given the pool for the lighter jobs triple the weight of the other pool. Although the Fair scheduler lets tasks from both pools run in parallel, this configuration lets the scheduler allocate a larger



(a) Hadoop's Default Scheduler



(b) Fair Scheduler

Figure 6.5: Job run time CDFs for various load levels.

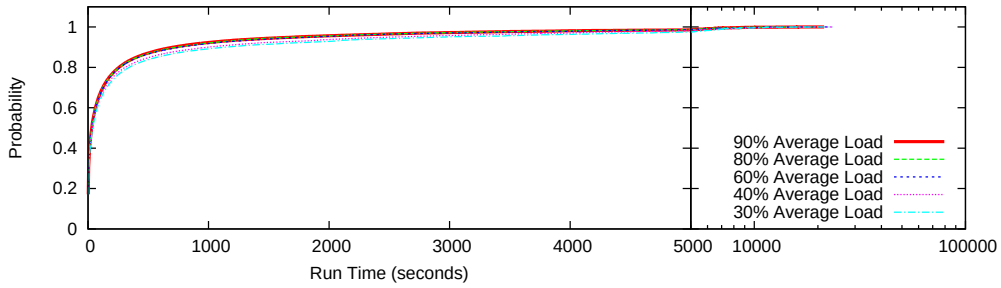
share of the cluster for running the tasks of lighter jobs. As effect of this, the run times of lighter jobs are significantly shorter, at the expense of an increase in the run times of the heavier jobs.

We show the CDFs for the run time of the tasks in Figure 6.6. To make the differences between the curves better visible, we "zoom in" at the first 5,000 seconds – larger values on the horizontal axis are shown in log scale. The maximum values on the horizontal axis are about 23,500 seconds for the Hadoop's default scheduler and 41,400 seconds for the Fair scheduler.

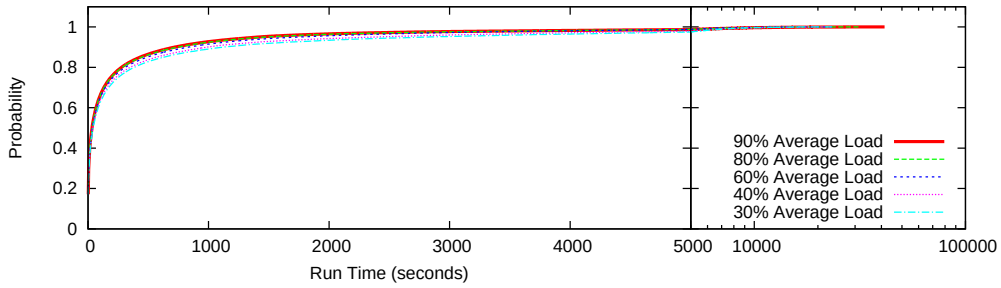
In Figure 6.6, on the first 5,000 seconds there is no visible difference in task run times between the two schedulers. However, we find a significant difference in the lengths of the tails. The absence of a visible difference in the first 5,000 seconds is due to both schedulers allocating map tasks to each run on a task slot, and not further interfering with the execution.

The run times of the reduce tasks, unlike map tasks, depend on when other (map) tasks finish; the last reduce task can only finish after the last map task has finished. Mumak lets every reduce task run until the very last map task has finished, we can use this fact to explain the large difference in tail length we observed: In long running jobs (which occur most for the Fair scheduler) the last map task finishes late, this increases the run times of all reduce tasks in these jobs.

We also observe a difference in the distribution of response times per load level: the higher the load level, the shorter the majority of the tasks require to complete.



(a) Hadoop's Default Scheduler



(b) Fair Scheduler

Figure 6.6: Task run time CDFs, horizontal axis values over 5,000 in log scale.

We hypothesize that this difference is also caused by Mumak's way of letting reduce tasks run longer than specified. With decreasing load, the reduce tasks will take significantly less time to complete. As an unexpected consequence, their shorter run time may lead to a lower value for the 90th-percentile of the distribution (they become shorter than some of the map tasks); we could incorrectly conclude that "lower loads lead to higher [90th-percentile] run times."

We have plotted the median run times as a function of the load level in Figure 6.7, and we have plotted box plots of run times as function of the load level in Figure 6.8 (the box plots show, from the top down: the maximum value, the third quartile, the median, the first quartile, and the minimum value). For the tasks, we have plotted next to the results of the two schedulers also the input workload for comparison.

In Figure 6.7a we clearly see that the almost linear increase of the median job run time with the increase of the load level for Hadoop's default scheduler, while for the same loads, the median job run time for the Fair scheduler remains around the same low value. In Figure 6.8a we observe that although the maximum job run times for the Fair scheduler are always larger than for Hadoop's default scheduler, the majority of jobs finish faster than jobs scheduled by the default scheduler. This leaves us to conclude that the Fair scheduler has far better scaling behavior than Hadoop's default scheduler.

The median task run times in Figure 6.7b seem to confirm our hypothesis, that the influence of the longer running reduce tasks interfere more with the normal tasks for the lower load levels. There is also something strange we observe in this graph, for some load levels the median task run times of Hadoop's default scheduler are lower than the median task run time is lower than in the generated workload. A possible cause can be that in the same way that reduce run times can become larger than specified, they can also become lower than specified, if for example a reduce task is started just before the last map task finishes.

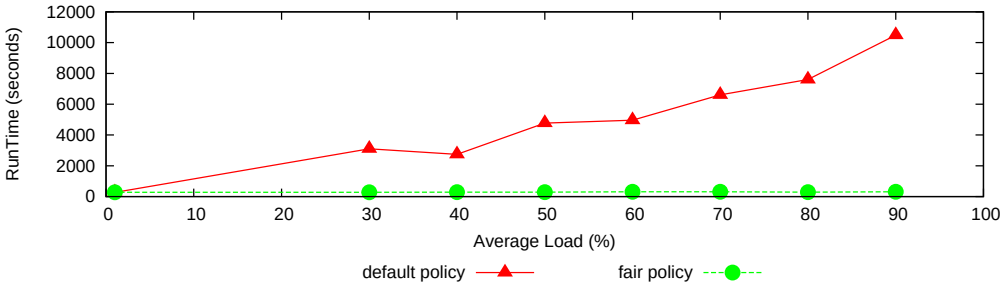
6.3.3 Analysis of Cost

We have calculated the cost, as described in Section 6.2.4, of running the 6-hour workloads on a 1545-node cluster. We have both plotted the cost over time in Figure 6.9 and we show the total cost for running the entire workloads in Table 6.3.

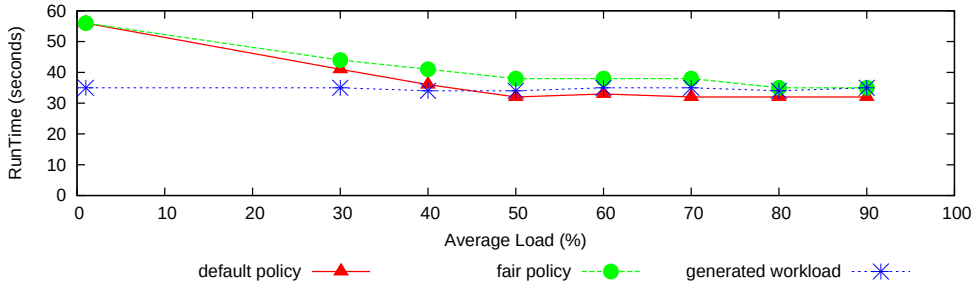
	Load Level							
Scheduler	**1%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**
Hadoop's Default Scheduler	7385	**12496**	14091	**14037**	**13416**	**15903**	16716	**18372**
Fair Scheduler	**3514**	12553	**13840**	14076	13548	16827	**16674**	19774
Fraction of Fair/Default	0.475	1.004	0.982	1.002	1.009	1.058	0.997	1.076

Table 6.3: Total cost in node-hours for running the entire 6-hour workloads.

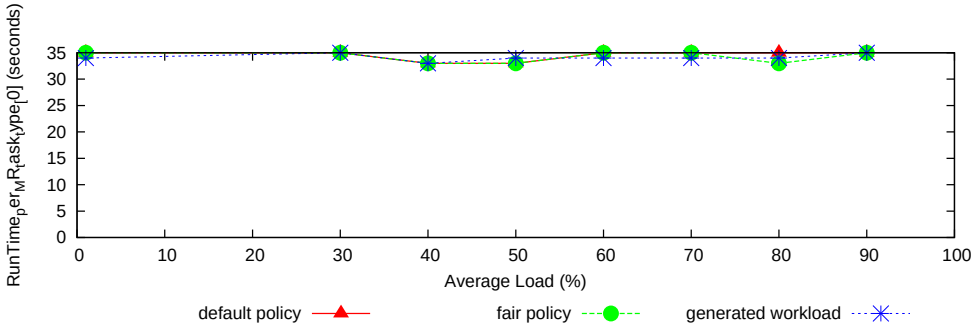
At first glance we observe in Figure 6.9 significant differences in cost during



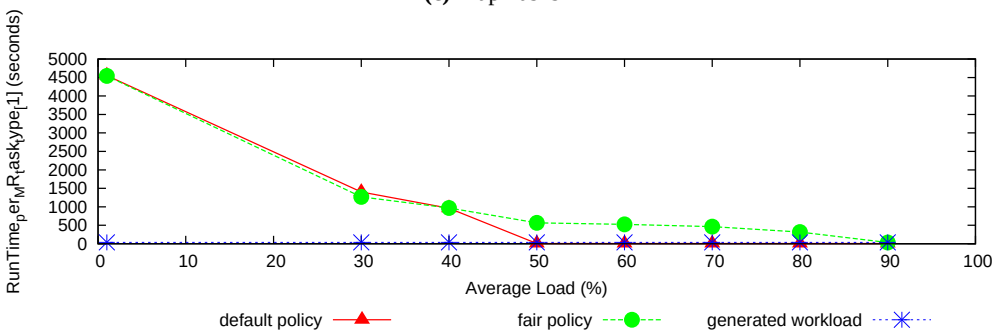
(a) Jobs



(b) Tasks

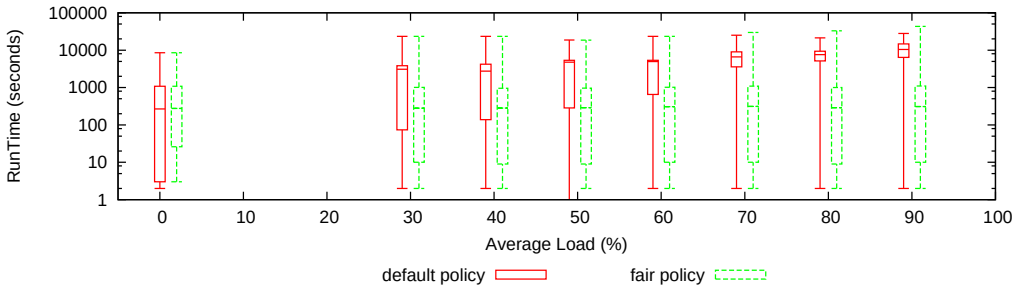


(c) Map Tasks

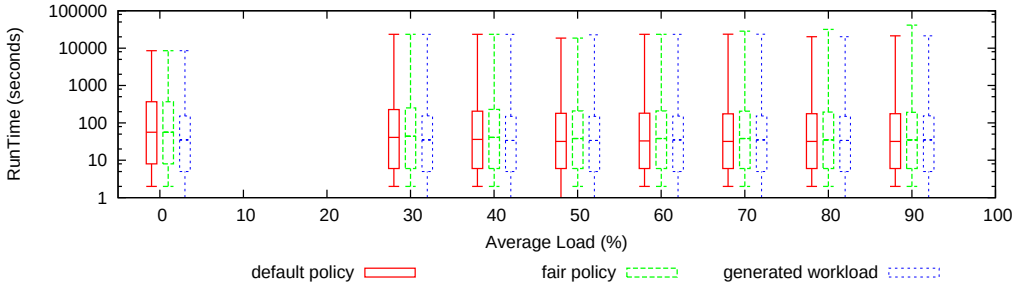


(d) Reduce Tasks

Figure 6.7: Medians of the run times for different scheduling policies over various load levels.



(a) Jobs



(b) Tasks

Figure 6.8: Box plots of the run times for different scheduling policies over various load levels.

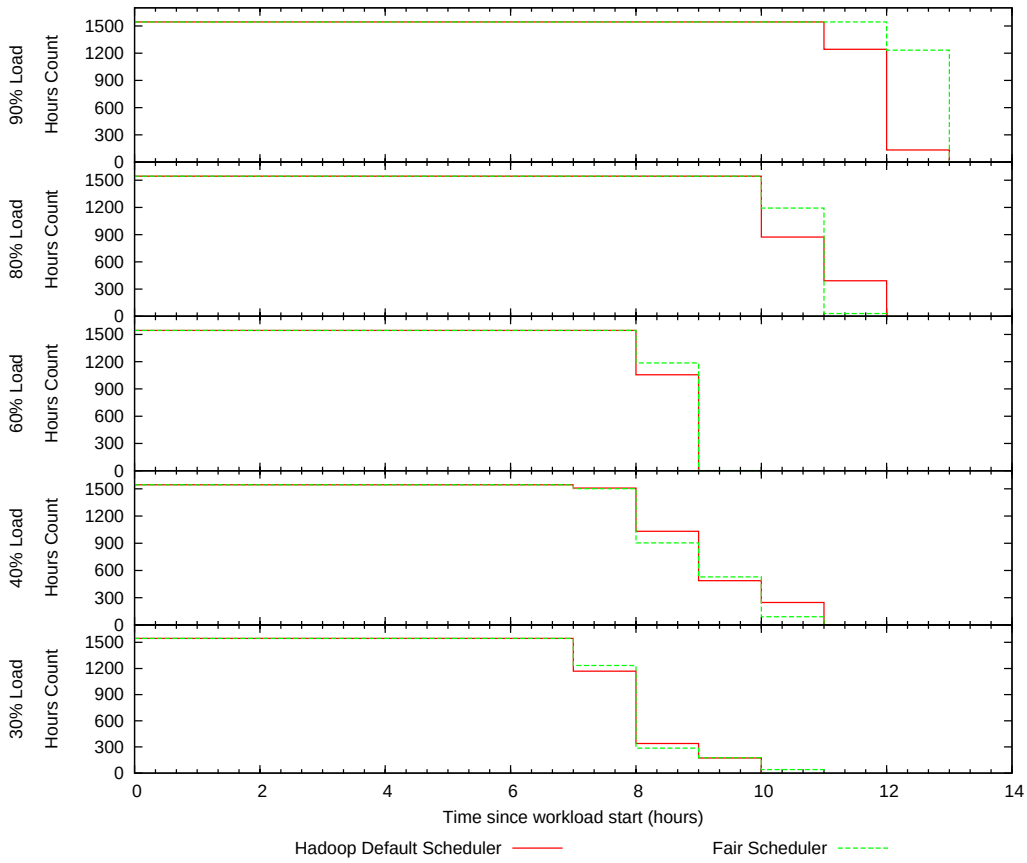


Figure 6.9: Cost in nodes per hours for both scheduling policies under various load levels.

the tail of the workload execution, however the scheduler that is the cheapest in executing the workload, seems to be fairly arbitrary. Table 6.3 makes it even more clear that the differences in cost are very small, and that there is no clear winning scheduler. Workloads with high load levels use all nodes most of the time and only show differences in the tail of the execution. The differences in the tail of the workload execution however, do not have a large impact on the total cost.

None of the used schedulers optimizes on cost in node-hours. A system that can scale down well enough to allow a scheduler to optimize in this way is non-trivial to develop, as currently the distributed file system in Hadoop can not easily scale down. An implementation of a MapReduce system with an elastic distributed file system would not only be beneficial in a cloud computing setting, it would for example also enable power conservation techniques in MapReduce clusters. It would be possible to investigate the behavior of the cost for different cluster configurations, a smaller cluster might take a longer time to execute the workload, but at a lower total cost.

6.4 Concluding Remarks

In this chapter we have shown an approach of how MapReduce systems can be evaluated with the help of out Cloud Workload Archive Toolbox. We have shown that using a simulator can be beneficial in comparison to running experiments on a real cluster, and we have shown that the use of generated synthetic workloads can be beneficial compared to using real workloads.

We have verified our approach by performing an experiment, from this experiment we are able to conclude that our approach actually allows us to evaluate MapReduce systems. The experiment also shows a huge differences between the two schedulers. The Fair scheduler does, compared to Hadoop's default scheduler, an excellent job in providing low response times for the lighter jobs. None of the two schedulers optimize on cost in node-hours, and this results in no significant difference in their performance in cost.

In this chapter we have also identified challenges for future work, first the cost as a function of the MapReduce cluster configuration can be investigated, an elastic distributed file-system for MapReduce can be developed, and a MapReduce scheduler that optimizes on cost can be developed. We have only used workloads generated from the Simple Model (see Section 5.3.1), we leave the use of workloads generated from the Complex Models (see Section 5.3.2) for future work.

Chapter 7

Conclusion

The goal of this research was to be able to evaluate MapReduce systems. To this end we have analyzed real-world MapReduce workloads. We have presented models for MapReduce workloads and procedures to generate synthetic workloads based on these models. And finally, we have shown that our tools can be used to evaluate MapReduce systems, by performing an experiment in which we compare the performance of two schedulers.

In this chapter we present our conclusion, reflect on the work, and propose possible directions for future work.

7.1 Overview

In Chapter 1 we introduced this thesis, we introduced MapReduce, and we presented our research questions and technical objectives. In Chapter 2 we surveyed the current state of literature related to this thesis. In Chapters 3-6 we presented our toolbox for MapReduce workload analysis and modeling, the analyses of real-world workload traces, models for MapReduce workloads, and how to use our work to evaluate MapReduce schedulers. These four chapters covered in fact the research questions, technical objectives, and experimental results, which we discuss in the following subsections.

7.1.1 The Research Question

The main research question for this thesis was: *"Is the MapReduce scheduler X better than MapReduce scheduler Y?"* This main research question lead to four sub-questions:

Q1 ***What are the characteristics of MapReduce workloads?***

In Chapter 4 we have analyzed MapReduce workload traces. These traces show:

- close to zero job wait times for the majority of the jobs

- run times of less than about 1.5 minute for the majority the tasks

When comparing map tasks with reduce tasks, we find that:

- map tasks run generally shorter than reduce tasks
- map tasks read more data from disk than they write, reduce tasks write more data to disk than they read
- there are in a job generally more map tasks than reduce tasks

Q2 ***How can we model MapReduce workloads?***

In Chapter 5 we presented four statistical models for MapReduce workloads. One simple model and three complex models. The simple model was used in our experiments and uses a direct-modeling approach for both the job and task characteristics. The complex models make a distinction between map and reduce tasks, exploit the correlation between run time and disk I/O, and use indirect-modeling for task characteristics.

Q3 ***How can we generate realistic synthetic MapReduce workloads?***

In Chapter 5 we presented two procedures to generate realistic synthetic MapReduce workloads: one procedure to generate workloads based on the simple model, and one procedure to generate workloads based on the family of complex models. Both procedures work by sampling from distributions given by the model parameters. A workload generator based on the simple model is included in the toolbox.

Q4 ***Which MapReduce scheduler performs best in scheduling a certain workload?***

We answer a more concrete version of this question in Chapter 6 by evaluating the Hadoop's default scheduler and the Fair Scheduler in an experiment. The answer to this question needs a metric to decide which is best; which metric actually to use differs per the requirements of the user. We have proposed two metrics: first, the job run-times, and second, the cost of running the workload in node-hours.

7.1.2 The Technical Objectives

We have developed the MapReduce Analysis Toolbox to fulfill the technical objectives T1-T3 – to automate the analyzing, modeling, and generation of MapReduce workloads. We have developed the toolbox for simulation using super-computers to fulfill technical objective T4 – automate the simulating of MapReduce workloads. We describe these toolkits in Chapter 3. In Appendix A we show to obtain and use the MapReduce Analysis Toolbox.

7.1.3 Experimental Results

In our experiment we have evaluated two MapReduce schedulers: Hadoop's default scheduler and the Fair Scheduler. We have successfully used Mumak to simulate for both schedulers individually synthetic workloads, generated from our model. We have used the DAS-4 super-computer to run the simulations.

We evaluate the schedulers based on the job run times and the total cost for running the workload in node-hours. The experiment shows that the Fair Scheduler yields much lower job run-times for the far majority of the jobs compared to Hadoop's default scheduler, as can be seen in Figure 6.5. When looking at the cost in node-hours, we can not observe a significant difference in the performance of the two schedulers with regard to this metric. This is not surprising, as none of the two schedulers tries to optimize this value.

7.2 Reflection

During this research we have made many choices, some of which might in hindsight not been the best choices. In this section we reflect on two of these choices.

7.2.1 Selection of Mumak

We have chosen Mumak as MapReduce simulator, and in the end it turned out that it did not respect the reduce task run times.

The use of the native Hadoop JobTracker and the input compatibility with Gridmix3 are two features that make Mumak really attractive as a simulator for our work. But, from the results of this experiment we have learned that Mumak does not respect the reduce task run times specified in the input workload. Instead, Mumak lets reduce tasks run until the last map task finishes. Given Mumak's disrespect for its input workload, we should have selected another simulator.

7.2.2 The Need for a Complex Model

Although, we have not used one of the complex models in our experiment, because of the amount of time needed to model and generate workloads using the complex model, the results of the experiment would not have been very different because of the current limitations of Mumak. We do, however, think that the complex model is a better choice, as it more-closely models real workloads:

1. The complex model allows differences in the behavior of map and reduce tasks, by modeling them separately.

2. The complex model allows similar behaving tasks within a single job, while still allowing significant differences in the behavior of tasks of different jobs, by using an indirectly-modeling approach.

7.3 Recommendations for Further Research

Throughout in this work we have mentioned possible directions for future work:

1. Use complex models.

   In this thesis we have used the simple model for the experiments, we would recommend future experiments with the use of the complex model.

2. Add AD goodness of fit test to the python model fitter.

   In the Matlab code we tested fits using both the KS test and the AD test, for reasons explained in Chapter 5. In the python version we have not yet implemented the AD test, although it is designed with multiple goodness of fitness tests in mind.

3. Publication of MapReduce workload traces in a Cloud Workloads Archive.

   Publication of the workload traces requires permission from the trace owners. We have not attempted to obtain these permissions, so we can not publish the traces. Public available traces will allow a larger set of researchers to study MapReduce systems.

4. Identification of MapReduce jobs in the Google traces.

   We deem it likely that the Google traces contain MapReduce jobs. It may be possible to identify the MapReduce jobs in these traces by observing the job characteristics.

5. Adaption of the CWA Data Format.

   The CWA Data Format should capture more aspects of MapReduce, such as partition and shuffle phases.

6. Exploitation in a model of possible correlations with the executable identifier.

   From our workload analyses in Chapter 4 it becomes clear that MapReduce jobs may have application/executable-specific behavior, for example a “select query job” will read a lot of data while a “append job” will only write little data. A future MapReduce model could exploit this kind of correlations.

7. Development of a scheduler that optimizes on cost in an elastic MapReduce cluster.

   We observe the demand for MapReduce clusters in the cloud. A MapReduce cluster of a fixed size will be under-provisioned at one moment and over-provisioned at another moment. A scheduler that scales an elastic MapReduce cluster up and down, while meeting deadlines and minimizing cost, would make MapReduce in the cloud more affordable.

8. Development of an elastic distributed file-system, to enable an elastic MapReduce cluster.

   We assume that the in-elasticity of the current version of the Hadoop Distributed File-System would be the main obstacle for creating an elastic MapReduce cluster. An elastic distributed file-system would also be beneficial for using MapReduce in grid environments.

Bibliography

[1] J. Dean, S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, *Symposium on Operating System Design and Implementation (OSDI)*, 137–150, 2004.

[2] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, I. Stoica, “Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling”, *Proceedings of the 5th European conference on Computer systems*, EuroSys ’10, ACM, New York, NY, USA, ISBN 978-1-60558-577-2, 265–278, 2010.

[3] S. Kavulya, J. Tan, R. Gandhi, P. Narasimhan, “An Analysis of Traces from a Production MapReduce Cluster”, *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID ’10, IEEE Computer Society, Washington, DC, USA, ISBN 978-0-7695-4039-9, 94–103, 2010.

[4] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, I. Stoica, “Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center”, *In Proceedings of the 8th Usenix Symposium on Networked System Design and Implementation*, 2011.

[5] A. Ganapathi, Y. Chen, A. Fox, R. Katz, D. Patterson, “Statistics-driven workload modeling for the Cloud” (2010) 87–92.

[6] G. Wang, A. R. Butt, P. Pandey, K. Gupta, “Using realistic simulation for performance analysis of mapreduce setups”, *Proceedings of the 1st ACM workshop on Large-Scale system and application performance*, LSAP ’09, ACM, New York, NY, USA, ISBN 978-1-60558-592-5, 19–26, 2009.

[7] K. Kim, K. Jeon, H. Han, S.-g. Kim, H. Jung, H. Y. Yeom, “MRBench: A Benchmark for MapReduce Framework” (2008) 11–18ISSN 1521-9097.

[8] Y. Chen, A. Ganapathi, R. H. Katz, “To compress or not to compress - compute vs. IO tradeoffs for mapreduce energy efficiency”, *Proceedings of the first ACM SIGCOMM workshop on Green networking*, Green Networking ’10, ACM, New York, NY, USA, ISBN 978-1-4503-0196-1, 23–28, 2010.

[9] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, I. Stoica, "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types" .

[10] "Rumen: A tool to extract Job Characterization Data from Job Tracker Logs", URL https://issues.apache.org/jira/browse/MAPREDUCE-751, 2009.

[11] K. Cardona, J. Secretan, M. Georgiopoulos, G. Anagnostopoulos, "A Grid Based System for Data Mining UsingMapReduce" .

[12] S. Hammoud, M. Li, Y. Liu, N. K. Alham, Z. Liu, "MRSim: A discrete event based MapReduce simulator" 6 (2010) 2993–2997.

[13] "Mumak: Map-Reduce Simulator", URL https://issues.apache.org/jira/browse/MAPREDUCE-728, 2009.

[14] D. Tankel, "Gridmix3 — Emulating Production Workload for Apache Hadoop", URL http://developer.yahoo.com/blogs/hadoop/posts/2010/04/gridmix3_emulating 2010.

[15] A. C. Murthy, "The Hadoop Map-Reduce Capacity Scheduler", URL http://developer.yahoo.com/blogs/hadoop/posts/2011/02/capacity-scheduler 2011.

[16] T. Sandholm, K. Lai, "Dynamic Proportional Share Scheduling in Hadoop Job Scheduling Strategies for Parallel Processing", vol. 6253 of *Lecture Notes in Computer Science*, chap. 7, Springer Berlin / Heidelberg, Berlin, Heidelberg, ISBN 978-3-642-16504-7, 110–131, 2010.

[17] M. Zaharia, A. Konwinski, A. D. Joseph, Y. Katz, I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments" .

[18] J. Polo, D. Carrera, Y. Becerra, M. Steinder, I. Whalley, "Performance-driven task co-scheduling for MapReduce environments", 373–380, 2010.

[19] A. Iosup, A Framework for the Study of Grid Inter-Operation Mechanisms, Ph.D. thesis, 2008.

[20] W. Leland, T. J. Ott, "Load-balancing heuristics and process behavior", *SIGMETRICS Perform. Eval. Rev.* 14 (1986) 54–69, ISSN 0163-5999.

[21] M. Calzarossa, G. Serazzi, "Construction and use of multiclass workload models", *Perform. Eval.* 19 (1994) 341–352, ISSN 0166-5316.

[22] M. H. Balter, A. B. Downey, “Exploiting process lifetime distributions for dynamic load balancing”, *ACM Trans. Comput. Syst.* 15 (3) (1997) 253–285, ISSN 0734-2071.

[23] D. G. Feitelson, “Packing Schemes for Gang Scheduling”, *IPPS ’96: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, London, UK, ISBN 3540618643, 89–110, 1996.

[24] D. G. Feitelson, B. Nitzberg, “Job Characteristics of a Production Parallel Scientivic Workload on the NASA Ames iPSC/860”, *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, London, UK, ISBN 3-540-60153-8, 337–360, 1995.

[25] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, J. Riordan, “Modeling of Workload in MPPs”, *In Job Scheduling Strategies for Parallel Processing*, 95–116, 1997.

[26] U. Lublin, D. G. Feitelson, “The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs”, *Journal of Parallel and Distributed Computing* 63.

[27] H. Li, D. Groep, L. Wolters, “Workload Characteristics of a Multi-cluster Supercomputer”, *D. G. Feitelson, L. Rudolph, U. Schwiegelshohn (Eds.), Job Scheduling Strategies for Parallel Processing*, vol. 3277 of *Lecture Notes in Computer Science*, chap. 10, Springer Berlin / Heidelberg, Berlin, Heidelberg, ISBN 978-3-540-25330-3, 33–53, 2005.

[28] E. Medernach, “Workload Analysis of a Cluster in a Grid Environment”, *D. Feitelson, E. Frachtenberg, L. Rudolph, U. Schwiegelshohn (Eds.), Job Scheduling Strategies for Parallel Processing*, vol. 3834 of *Lecture Notes in Computer Science*, chap. 2, Springer Berlin / Heidelberg, Berlin, Heidelberg, ISBN 978-3-540-31024-2, 36–61, 2005.

[29] B. Song, C. Ernemann, R. Yahyapour, “User group-based workload analysis and modelling” 2 (2005) 953–961 Vol. 2.

[30] H. Li, M. Muskulus, “Analysis and modeling of job arrivals in a production grid”, *SIGMETRICS Perform. Eval. Rev.* 34 (2007) 59–70, ISSN 0163-5999.

[31] A. Iosup, O. Sonmez, S. Anoep, D. Epema, “The performance of bags-of-tasks in large-scale distributed systems”, *Proceedings of the 17th international symposium on High performance distributed computing*, HPDC ’08, ACM, New York, NY, USA, ISBN 978-1-59593-997-5, 97–108, 2008.

[32] D. G. Feitelson, Workload Modeling for Computer Systems Performance Evaluation, 0.34 edn., 2011.

[33] D. G. Feitelson, “Parallel Workloads Archive”, URL `http://www.cs.huji.ac.il/labs/parallel/workload/`, 2005.

[34] D. Kondo, B. Javadi, A. Iosup, D. Epema, “The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems”, *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, IEEE, ISBN 978-1-4244-6987-1, 398–407, 2010.

[35] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, D. H. J. Epema, “The Grid Workloads Archive”, *Future Generation Computer Systems* 24 (7) (2008) 672–686, ISSN 0167739X.

[36] B. Zhang, A. Iosup, D. Epema, “The Peer-to-Peer Trace Archive: Design and Comparative Trace Analysis”, Tech. Rep. PDS-2010-003, Delft University of Technology, 2010.

[37] A. Iosup, R. Griffith, A. Konwinski, M. Zaharia, A. Ghodsi, I. Stoica, “Data Fromat for the Cloud Workloads Archive”, *DRAFT, v.3* .

[38] A. Iosup, H. Li, C. Dumitrescu, L. Wolters, D. H. J. Epema, “The Grid Workloads Archive Format” .

[39] D. Talby, D. G. Feitelson, J. P. Jones, “The Standard Workload Format” .

[40] G. Wang, A. R. Butt, P. Pandey, K. Gupta, “A simulation approach to evaluating design decisions in MapReduce setups”, *2009 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems*, IEEE, ISBN 978-1-4244-4927-9, ISSN 1526-7539, 1–11, 2009.

Appendix A

Result Availability

The technical result of our work is the Cloud Workloads Archive Toolbox. We have made this software available as an open-source project. It would have been nice if we could also make the workload traces available as open-access data. We leave that task as future work, as it requires obtaining permission from the trace owners.

A.1 Obtaining the Cloud Workloads Archive Toolbox

We have published the source code of the CWA Toolbox as a git repository on Atlassian Bitbucket. You can either download the source code from the code repository website, or clone the entire git repository.

Code Repository Website

`https://bitbucket.org/tader/cwa-toolbox/`

`Clone the Git Repository`

```
$ git clone https://bitbucket.org/tader/cwa-toolbox.git
```

A.2 Dependencies

The CWA Toolbox requires the following software to be available. (The version numbers indicate the versions used to develop the toolbox, other versions might also work.)

- **Python** (v2.7)
  Almost all components of the CWA Toolbox are written in Python.
- **Gnuplot** (v4.4)
  Gnuplot is used to plot the graphs from the analysis data.

- **Matlab** (v7.11.0 (R2010b))
  Matlab is used for modeling of the workload trace.

- **SciPy** (v0.10.1)
  SciPy is used for modeling of the workload trace.

- **NumPy** (v1.6.2)
  NumPy is required for SciPy.

A.3 Installation

There is no real installation needed as such, you just need to copy the `cwa` script into a directory which is in your `$PATH`, e.g., in `/usr/local/bin` or `~/bin`. Then change the path inside this script so that it correctly indicates the location where you placed the toolbox's `src` directory.

A.4 Creating a CWA "Project"

To work on a new CWA "project", we need to create a directory structure with the original input traces and a configuration file. With this in place, we can use the CWA Toolbox.

A.4.1 Directory Structure

The CWA Toolbox expects the following layout for a CWA workload trace project. This layout is defined in the default configuration and everything can be overridden in the local configuration file if needed, except for the path to the local configuration file itself of course.

- `trace-name/` ..... Root path for the CWA project
  - `cwa/` ..... Toolbox Work Directory
    - `config` ..... CWA configuration file (see example)
    - `traces/` ..... Traces in CWA format (after `cwa import`)
    - `data/` ..... Analysis results (after `cwa analyze`)
    - `plots/` ..... Gnuplot plots (after `cwa plot`)
    - `model/` ..... Modeling results (after `cwa model`)
  - `raw/` ..... Original trace data

A.4.2 Configuration File

An example configuration file is shown in Figure A.1. For all possible settings and their default values, please see the file `cwa-toolbox/src/default_config`. The root path of the CWA project is available as variable in the configuration files, it can be used by writing "`%(path)s`", as shown in the example configuration file.

The setting "`module = cwa.import.hadoop`" could have been omitted as this is already set as default value in the default configuration file. The simplest possible configuration file is an empty file.

```
1 [trace]
2 title   = Hadoop Workload
3
4 [import]
5 input   = %(path)s/raw/logs/history/done
6 module  = cwa.import.hadoop
```

Figure A.1: Example configuration file for a CWA project.

A.4.3 Example Usage

If you have created such a directory structure and configuration file, you are ready to use the CWA Toolbox. First place the original trace data in the raw directory, and create a suitable configuration file. The default process would be entering the following commands:

1. `cwa import`
   Imports the raw trace data into the CWA format.
2. `cwa analyze`
   Analyzes the traces in CWA format.
3. `cwa plot`
   Plots the results of the analyses.
4. `cwa model`
   Fit the model parameters of the trace.
5. `cwa model report`
   Plot the results of the model fitting.

A.5 General Usage

Type `cwa -h` to find out all the available options and commands, the output of this command is shown in Figure A.2. For even more usage information, see the source code.

```
$ cwa -h
usage: cwa [OPTIONS]... <command>

Cloud Workloads Archive Toolbox

OPTIONS:
    -h, --help     print command help
         --doc     print command modules pydoc
    -v             increase verbosity level

COMMANDS:
    analyze
    dump
    generate
    import
    model
    plot
    test
    trace
```

Figure A.2: Help information for the `cwa` command.

A.6 Contributing

You are invited to contribute to the CWA Toolbox. I look forward to receiving your "pull requests," which you can send me either by using the pull request feature of Bitbucket, or just by sending me an email at `thomas@de-ruiter.cx`.

Appendix B

Data Format for the Cloud Workloads Archive

The Data Format for the Cloud Workloads Archive is defined by Iosup et al. [37]. The data format specifies two levels of fields for job and task information. This data can in principle be stored in any database or file. An overview of all fields in the draft version 3 of this data format is given in the following table.

In addition to the data format specification, we present the following rules of thumb for reading/writing the trace data in a tab separated file format:

- The workload trace consists of four separate files containing job data (*.cwj), task data (*.cwt), detailed job information (*.cwjd), and detailed task information (*.cwtd).
- Records must be separated by a newline character, field values must be separated by a tab character.
- Empty lines and lines starting with the number symbol "#" must be ignored.
- The first not ignored line of each file should contain tab separated list of field names as column headers.
- The fields in the records and headers should be ordered by the field identifier.
- Field values must not contain tabs and should not be quoted.
- The values (ignoring case) "none", "null", and the empty string, must be handled as unknown data.
- Negative values in fields where this is obviously not logical, like the number of CPUs, should be handled as unknown data.
- The files may optionally be compressed using gzip compression, in that case the file names must be extended with the "`.gz`" extension.

ID	Name	Type	CWJ	CWT	CWJD	CWTD
1	JobID	Int	+	+	+	+
2	TaskID	Int	–	+	–	+
3	SubmitTime	Float	+	+	+	+
4	WaitTime	Float	+	+	+	+
5	RunTime	Float	+	+	+	+
6	CPUs	Float	+	+	+	+
7	TotalWallClockTime	Int	+	+	+	+
8	Memory	Float	+	+	+	+
9	Network	Float	+	+	+	+
10	Disk	Float	+	+	+	+
11	Status	Int[1]	+	+	+	+
12	UserID	String	+	–	+	–
13	GroupID	String	+	–	+	–
14	ExecutableID	String	+	+	+	+
15	QueueID	Int	+	+	+	+
16	PartitionID	Int	+	+	+	+
17	JobProperties	Int[2]	+	+	–	–
18	StructuralChanges	String[3]	+	+	+	+
19	StructuralChangeParams	String[4]	+	+	+	+
20	DiskIORatio	Float	+	+	+	+
21	MR_total_launched	Int	–	–	+	–
22	MR_total_from_job	Int	–	–	+	–
23	MR_total_failed	Int	–	–	+	–
24	MR_total_killed	Int	–	–	+	–
25	MR_total_splits	Int	–	–	+	–
26	MR_total_hdfs_read	Int	–	–	+	+
27	MR_total_hdfs_written	Int	–	–	+	+
28	MR_total_local_read	Int	–	–	+	+
29	MR_total_local_written	Int	–	–	+	+
30	MR_total_spilled_records	Int	–	–	+	+
31	MR_map_launched	Int[5]	–	–	+	–
32	MR_map_total	Int[5]	–	–	+	–
33	MR_map_finished	Int[5]	–	–	+	–
34	MR_map_failed	Int[5]	–	–	+	–
35	MR_map_killed	Int[5]	–	–	+	–
36	MR_map_hdfs_read	Int[5]	–	–	+	–
37	MR_map_hdfs_written	Int[5]	–	–	+	–
38	MR_map_local_read	Int[5]	–	–	+	–
39	MR_map_local_written	Int[5]	–	–	+	–
40	MR_map_input	Int[5]	–	–	+	+
41	MR_map_output	Int[5]	–	–	+	+

ID	Name	Type	CWJ	CWT	CWJD	CWTD
42	MR_map_input_records	Int[5]	–	–	+	+
43	MR_map_output_records	Int[5]	–	–	+	+
44	MR_data_local	Int	–	–	+	+
45	MR_data_rack	Int	–	–	+	+
46	MR_combine_input_records	Int	–	–	+	+
47	MR_combine_output_records	Int	–	–	+	+
48	MR_reduce_input_records	Int	–	–	+	+
49	MR_reduce_input_groups	Int	–	–	+	+
50	MR_reduce_output_records	Int	–	–	+	+
51	MR_reduce_finished	Int	–	–	+	–
52	MR_task_attempt_host	String	–	–	–	+
53	MR_task_attempt_shuffle_finished	Float	–	–	–	+
54	MR_task_attempt_sort_finished	Float	–	–	–	+
55	MR_task_attempt_counters	Float	–	–	–	+
56	MR_task_attempt_id	String	–	–	–	+
57	MR_task_type	Int[6]	–	–	–	+
58	MR_total_map_time	Int	–	–	+	–
59	MR_total_reduce_time	Int	–	–	+	–
60	MR_spilled_records	Int	–	–	–	+

[1] Status codes:
- 0: Failed
- 1: Success
- 2: Continued partial execution
- 3: Last partial execution, success
- 4: Last partial execution, failed
- 5: Canceled job
- 6: Retry of an earlier failed job

[2] Job property codes:
- 0: Interactive
- 1: User-facing
- 2: Batch

[3] Comma-separated values: MR/Opaque, MR[/Detailed], MR/Changed

[4] Semicolon-separated values: PrevTasks=String(,...); PrevProfile=Int

[5] Type unspecified in specification [37], but Int is certainly the only sane option.

[6] Task type codes:
- 0: Map
- 1: Reduce

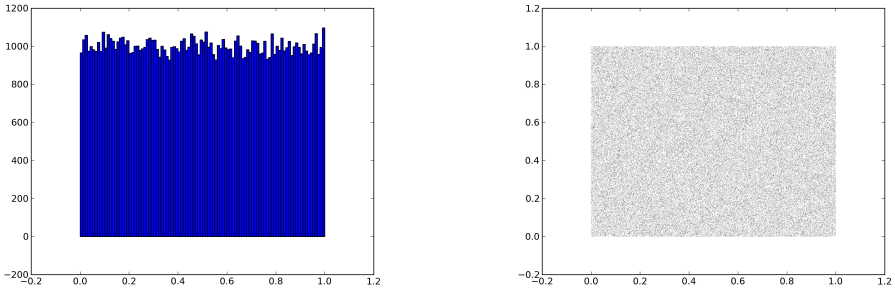
Appendix C

Validation of the Pseudo-Random Number Generator

We have sampled 50 MiB of data and attempted to compress it. The compression results in Table C.1 show only increased file sizes. We have also plotted a histogram of sampled values in Figure C.1a, and a scatter-plot of sampled points in Figure C.1b – as a visual validation of the random number generator. Inspection of these two figures does not reveal any significant deviation from an even distribution over the space. Based on the compression results and the visual validation, we conclude that this pseudo-random number generator is suitable to be used in our work.

Description	Size	Normalized
Sampled Random Bytes	52,428,800 B	1.0000
Compressed: `bzip2 -9`	52,662,999 B	1.0045
Compressed: `gz -9`	52,436,827 B	1.0002
Compressed: `zip`	52,436,956 B	1.0002

Table C.1: Compression of sampled bytes.



(a) Histogram. **(b)** Scatter-plot.

Figure C.1: Plots of 100,000 sampled values/points.

Appendix D

Modeling Results

D.1 Directly-Modeled Properties

		SN1	SN2	Google	Yahoo!
Normal	**shape**	0.000	0.000	0.000	0.000
	loc	14.0	14.2	3.76	55.3
	scale	239	37.4	6.92	1410
	KS	0.000	0.001	0.005	0.000
	D-stat.	0.426	0.298	0.261	0.426
Exponential	**shape**	0.000	0.000	0.000	0.000
	loc	-0.000	-0.000	-0.000	-0.000
	scale	15.1	14.2	3.76	55.3
	KS	**0.179**	**0.327**	0.000	0.002
	D-stat.	0.179	0.110	0.443	0.382
Weibull	**shape**	0.727	0.666	0.756	0.304
	loc	-0.000	-0.000	-0.000	-0.000
	scale	11.5	21.5	1.42	461
	KS	**0.344**	**0.129**	0.000	0.000
	D-stat.	0.059	0.165	0.452	0.435
Pareto	**shape**	1.00	1.00	1.00	1.00
	loc	0.000	0.000	0.000	0.000
	scale	0.000	0.000	0.000	0.000
	KS	0.000	0.000	0.000	0.000
	D-stat.	–	–	–	–
Log-Normal	**shape**	**1.22**	**1.25**	5.96	0.140
	loc	**-0.430**	**-0.471**	-0.000	-805
	scale	**6.90**	**7.47**	1.15	856
	KS	**0.419**	**0.382**	0.001	0.001
	D-stat.	**0.053**	**0.074**	0.439	0.309
Gamma	**shape**	0.000	0.000	0.132	0.000
	loc	13.5	13.7	1.24	28.8
	scale	121217	3030	19.0	75157
	KS	0.000	0.000	0.000	0.000
	D-stat.	0.727	0.632	0.575	0.705

Table D.1: Job inter-arrival time.

		SN1	SN2	Google	Yahoo!
Normal	**shape**	0.000	0.000	0.000	0.000
	loc	0.457	-1.00	8.18	6.53
	scale	2.62	0.000	567	26.3
	KS	0.000	0.000	0.000	0.000
	D-stat.	0.405	0.500	0.485	0.376
Exponential	**shape**	0.000	0.000	0.000	0.000
	loc	-0.000	0.000	-0.000	-0.000
	scale	0.457	0.000	8.18	6.53
	KS	0.000	0.000	0.000	0.000
	D-stat.	0.573	–	0.558	0.602
Weibull	**shape**	0.517	1.00	0.723	0.176
	loc	-0.000	0.000	-0.000	-0.000
	scale	0.073	0.000	4.12	34.7
	KS	0.000	0.000	0.004	0.000
	D-stat.	0.597	–	0.314	0.530
Pareto	**shape**	1.00	1.00	1.00	1.00
	loc	0.000	0.000	0.000	0.000
	scale	0.000	0.000	0.000	0.000
	KS	0.000	0.000	0.000	0.000
	D-stat.	–	–	–	–
Log-Normal	**shape**	9.61	1.00	0.245	9.27
	loc	-0.000	0.000	-18.0	-0.000
	scale	2.36	0.000	21.3	7.54
	KS	0.000	0.000	0.001	0.000
	D-stat.	0.603	–	0.400	0.487
Gamma	**shape**	0.000	–	0.000	0.032
	loc	0.428	–	5.25	1.85
	scale	242	–	109585	148
	KS	0.000	0.000	0.000	0.000
	D-stat.	0.597	–	0.948	0.769

Table D.2: Job wait time.

		SN1	SN2	Google	Yahoo!
Normal	**shape**	0.000	0.000	0.000	0.000
	loc	165	434	2856	513
	scale	654	3154	35709	1503
	KS	0.000	0.000	0.000	0.000
	D-stat.	0.398	0.443	0.465	0.363
Exponential	**shape**	0.000	0.000	0.000	0.000
	loc	-0.000	-0.000	-0.000	-0.000
	scale	165	434	2877	513
	KS	0.012	0.005	0.000	0.003
	D-stat.	0.343	0.344	0.619	0.376
Weibull	**shape**	0.593	0.583	0.467	0.567
	loc	-0.000	-0.000	-0.000	-0.000
	scale	83.0	173	454	240
	KS	**0.371**	**0.377**	**0.105**	**0.160**
	D-stat.	0.092	0.104	0.180	0.153
Pareto	**shape**	1.00	1.00	1.00	1.00
	loc	0.000	0.000	0.000	0.000
	scale	0.000	0.000	0.000	0.000
	KS	0.000	0.000	0.000	0.000
	D-stat.	–	–	–	–
Log-Normal	**shape**	**1.61**	**1.59**	**1.65**	**1.63**
	loc	**-0.206**	**-0.429**	**-0.424**	**-0.481**
	scale	**38.2**	**93.8**	**197**	**112**
	KS	**0.479**	**0.503**	**0.451**	**0.296**
	D-stat.	**0.047**	**0.032**	**0.070**	**0.105**
Gamma	**shape**	0.006	0.002	0.005	0.023
	loc	113	311	215	287
	scale	8325	80939	482742	10020
	KS	0.000	0.000	0.000	0.000
	D-stat.	0.762	0.803	0.552	0.734

Table D.3: Job run time.

		Yahoo!
Normal	**shape**	0.000
	loc	2e+08
	scale	6e+09
	KS	0.000
	D-stat.	0.441
Exponential	**shape**	0.000
	loc	-0.000
	scale	3e+08
	KS	0.000
	D-stat.	0.496
Weibull	**shape**	0.267
	loc	-0.000
	scale	4e+08
	KS	0.004
	D-stat.	0.309
Pareto	**shape**	1.00
	loc	0.000
	scale	0.000
	KS	0.000
	D-stat.	–
Log-Normal	**shape**	**6.10**
	loc	**-0.000**
	scale	**720312**
	KS	**0.093**
	D-stat.	**0.203**
Gamma	**shape**	0.000
	loc	2e+08
	scale	5e+11
	KS	0.000
	D-stat.	0.786

Table D.4: Job disk IO parameter α.

		Yahoo!
Normal	**shape**	0.000
	loc	199881
	scale	7036869
	KS	0.006
	D-stat.	0.291
Exponential	**shape**	0.000
	loc	-0.000
	scale	1704688
	KS	0.000
	D-stat.	0.691
Weibull	**shape**	0.203
	loc	-0.000
	scale	2701572
	KS	0.000
	D-stat.	0.645
Pareto	**shape**	1.00
	loc	0.000
	scale	0.000
	KS	0.000
	D-stat.	–
Log-Normal	**shape**	9.98
	loc	-0.000
	scale	3472
	KS	0.000
	D-stat.	0.655
Gamma	**shape**	0.482
	loc	-0.000
	scale	1e+07
	KS	0.000
	D-stat.	0.685

Table D.5: Job disk IO parameter β.

		SN1	SN2	Google	Yahoo!
Normal	**shape**	0.000	0.000	0.000	0.000
	loc	0.635	-1.00	10252	755
	scale	1.10	0.000	9616	986
	KS	0.003	0.000	0.022	0.014
	D-stat.	0.391	0.500	0.241	0.333
Exponential	**shape**	0.000	0.000	0.000	0.000
	loc	-0.000	0.000	636	-0.000
	scale	0.635	0.000	9616	755
	KS	0.000	0.000	0.027	0.000
	D-stat.	0.690	–	0.245	0.551
Weibull	**shape**	0.092	1.00	1.00	0.547
	loc	-0.000	0.000	636	-0.000
	scale	1.36	0.000	9616	1333
	KS	0.000	0.000	0.031	0.000
	D-stat.	0.667	–	0.238	0.540
Pareto	**shape**	1.00	1.00	1.93	1.00
	loc	0.000	0.000	-14898	0.000
	scale	0.000	0.000	14126	0.000
	KS	0.000	0.000	0.000	0.000
	D-stat.	–	–	0.385	–
Log-Normal	**shape**	7.64	1.00	1.00	1.00
	loc	-0.000	0.000	2916	2.78
	scale	0.008	0.000	4449	456
	KS	0.000	0.000	0.013	0.000
	D-stat.	0.672	–	0.332	0.566
Gamma	**shape**	0.258	–	**1.58**	0.451
	loc	-0.000	–	**33.8**	-0.000
	scale	1.61	–	**6460**	487
	KS	0.000	0.000	**0.076**	0.000
	D-stat.	0.702	–	**0.234**	0.520

Table D.6: Executable ID.

		SN2	Google	Yahoo!
Normal	**shape**	0.000	0.000	0.000
	loc	154	67.3	980
	scale	982	8486	3855
	KS	0.000	0.000	0.000
	D-stat.	0.385	0.468	0.390
Exponential	**shape**	0.000	0.000	0.000
	loc	-0.000	1.000	1.000
	scale	154	66.3	996
	KS	0.000	0.000	0.001
	D-stat.	0.697	0.764	0.352
Weibull	**shape**	0.223	0.942	0.432
	loc	-0.000	1.000	1.000
	scale	42.1	29.7	555
	KS	0.024	0.000	**0.218**
	D-stat.	0.310	0.755	0.136
Pareto	**shape**	1.00	0.817	0.227
	loc	0.000	1.000	-0.359
	scale	0.000	0.000	1.36
	KS	0.000	0.000	**0.083**
	D-stat.	–	0.697	0.205
Log-Normal	**shape**	8.07	0.186	**2.54**
	loc	-0.000	-4140	**0.941**
	scale	5.79	4313	**93.2**
	KS	0.007	0.000	**0.218**
	D-stat.	0.279	0.494	**0.124**
Gamma	**shape**	0.020	0.000	0.016
	loc	14.8	34.7	499
	scale	6955	2210235	30866
	KS	0.000	0.000	0.000
	D-stat.	0.783	0.906	0.723

Table D.7: Number of tasks.

		SN2	Google	Yahoo!
Normal	**shape**	0.000	0.000	0.000
	loc	0.108	0.000	0.002
	scale	0.208	0.000	0.040
	KS	0.002	0.000	0.000
	D-stat.	0.349	0.581	0.514
Exponential	**shape**	0.000	0.000	0.000
	loc	-0.000	0.000	-0.000
	scale	0.108	0.000	0.002
	KS	0.000	0.000	0.000
	D-stat.	0.598	–	0.998
Weibull	**shape**	0.377	1.00	0.501
	loc	-0.000	0.000	-0.000
	scale	0.129	0.000	0.000
	KS	0.000	0.000	0.000
	D-stat.	0.586	–	1.000
Pareto	**shape**	1.00	1.00	1.00
	loc	0.000	0.000	0.000
	scale	0.000	0.000	0.000
	KS	0.000	0.000	0.000
	D-stat.	–	–	–
Log-Normal	**shape**	13.9	1.00	0.252
	loc	-0.000	0.000	-0.000
	scale	0.006	0.000	0.000
	KS	0.000	0.000	0.000
	D-stat.	0.567	–	0.609
Gamma	**shape**	0.113	–	0.010
	loc	-0.000	–	-0.000
	scale	0.353	–	0.612
	KS	0.000	0.000	0.000
	D-stat.	0.581	–	0.493

Table D.8: Reduce-task ratio.

		SN1	SN2	Google	Yahoo!
Normal	**shape**	0.000	0.000	0.000	0.000
	loc	-1.00	-1.00	2353	-1.00
	scale	0.000	0.000	34819	0.000
	KS	0.000	0.000	0.000	0.000
	D-stat.	0.500	0.500	0.450	0.500
Exponential	**shape**	0.000	0.000	0.000	0.000
	loc	0.000	0.000	-0.000	0.000
	scale	0.000	0.000	2354	0.000
	KS	0.000	0.000	0.000	0.000
	D-stat.	–	–	0.700	–
Weibull	**shape**	1.00	1.00	0.698	1.00
	loc	0.000	0.000	-0.000	0.000
	scale	0.000	0.000	980	0.000
	KS	0.000	0.000	0.000	0.000
	D-stat.	–	–	0.592	–
Pareto	**shape**	1.00	1.00	1.00	1.00
	loc	0.000	0.000	0.000	0.000
	scale	0.000	0.000	0.000	0.000
	KS	0.000	0.000	0.000	0.000
	D-stat.	–	–	–	–
Log-Normal	**shape**	1.00	1.00	16.2	1.00
	loc	0.000	0.000	-0.000	0.000
	scale	0.000	0.000	733	0.000
	KS	0.000	0.000	0.000	0.000
	D-stat.	–	–	0.610	–
Gamma	**shape**	–	–	0.008	–
	loc	–	–	-0.000	–
	scale	–	–	597539	–
	KS	0.000	0.000	0.000	0.000
	D-stat.	–	–	0.607	–

Table D.9: Job forced-quit time.

		SN2	Google	Yahoo!
Normal	**shape**	0.000	0.000	0.000
	loc	0.013	0.242	0.010
	scale	0.078	0.428	0.099
	KS	0.000	0.000	0.000
	D-stat.	0.450	0.498	0.529
Exponential	**shape**	0.000	0.000	0.000
	loc	-0.000	-0.000	-0.000
	scale	0.013	0.242	0.010
	KS	0.000	0.000	0.000
	D-stat.	0.851	0.757	0.987
Weibull	**shape**	0.071	0.390	0.683
	loc	-0.000	-0.000	-0.000
	scale	0.109	0.036	0.051
	KS	0.000	0.000	0.000
	D-stat.	0.823	0.762	0.991
Pareto	**shape**	1.00	1.00	1.00
	loc	0.000	0.000	0.000
	scale	0.000	0.000	0.000
	KS	0.000	0.000	0.000
	D-stat.	–	–	–
Log-Normal	**shape**	7.07	7.67	0.494
	loc	-0.000	-0.000	-0.000
	scale	0.001	0.000	0.000
	KS	0.000	0.000	0.000
	D-stat.	0.827	0.752	0.679
Gamma	**shape**	0.070	0.202	0.063
	loc	-0.000	-0.000	-0.000
	scale	0.455	0.388	0.605
	KS	0.000	0.000	0.000
	D-stat.	0.808	0.786	0.980

Table D.10: Job fail fraction.

		Google	Yahoo!
Normal	**shape**	0.000	0.000
	loc	2815	301
	scale	21252	236555
	KS	0.000	0.000
	D-stat.	0.446	0.492
Exponential	**shape**	0.000	0.000
	loc	0.000	0.000
	scale	2828	303
	KS	0.001	0.000
	D-stat.	0.38	0.49
Weibull	**shape**	0.525	**0.531**
	loc	0.000	**0.000**
	scale	1127	**82.0**
	KS	**0.302**	**0.314**
	D-stat.	0.110	**0.10**
Pareto	**shape**	1.00	1.00
	loc	0.000	0.000
	scale	0.000	0.000
	KS	0.000	0.000
	D-stat.	–	–
Log-Normal	**shape**	**1.77**	0.303
	loc	**-0.161**	-10315
	scale	**432**	11454
	KS	**0.39**	0.000
	D-stat.	**0.0890**	0.548
Gamma	**shape**	0.00331	0.000
	loc	1592	212
	scale	369374	618102864
	KS	0.000	0.000
	D-stat.	0.697	0.886

Table D.11: Directly-Modeled overall task run time.

D.2 Indirectly-Modeled Properties

D.2.1 Complex Model

	Google		Yahoo!	
	map	**reduce**	**map**	**reduce**
Normal	0.22 %	0.00 %	0.07 %	**0.28 %**
Exponential	0.19 %	0.00 %	0.07 %	0.00 %
Weibull	**0.22 %**	0.00 %	0.04 %	0.00 %
Pareto	0.03 %	0.00 %	0.00 %	0.00 %
Log-Normal	0.21 %	0.00 %	**0.10 %**	0.00 %
Gamma	0.05 %	0.00 %	0.01 %	0.00 %

Table D.12: Task inter-arrival time matches.

	Google		Yahoo!	
	map	**reduce**	**map**	**reduce**
Normal	6.49 %	0.00 %	**20.54 %**	11.12 %
Exponential	3.85 %	0.00 %	2.69 %	10.30 %
Weibull	3.70 %	0.00 %	3.84 %	3.58 %
Pareto	4.09 %	0.00 %	6.93 %	7.27 %
Log-Normal	**21.95 %**	0.00 %	8.03 %	13.25 %
Gamma	10.87 %	0.00 %	17.19 %	**31.74 %**

Table D.13: Task run time matches.

	Google		Yahoo!	
	map	**reduce**	**map**	**reduce**
Normal	**1.33 %**	0.00 %	**0.11 %**	**0.37 %**
Exponential	0.00 %	0.00 %	0.00 %	0.00 %
Weibull	0.00 %	0.00 %	0.00 %	0.00 %
Pareto	0.01 %	0.00 %	0.00 %	0.00 %
Log-Normal	0.00 %	0.00 %	0.00 %	0.00 %
Gamma	0.00 %	0.00 %	0.00 %	0.00 %

Table D.14: Task CPUs matches.

	Yahoo!	
	map	**reduce**
Normal	0.04 %	**0.00 %**
Exponential	0.05 %	0.00 %
Weibull	0.00 %	0.00 %
Pareto	0.01 %	0.00 %
Log-Normal	0.02 %	0.00 %
Gamma	**0.21 %**	0.00 %

Table D.15: Task disk I/O matches.

		Google						Yahoo!					
		map			**reduce**			**map**			**reduce**		
		weibull						**lognormal**			**normal**		
		shape	**loc**	**scale**				**shape**	**loc**	**scale**	**shape**	**loc**	**scale**
Normal	**shape**	**0.000**	**0.000**	**0.000**				0.000	0.000	0.000	**0.000**	0.000	0.000
	loc	**0.900**	**519**	**502**				1.53	0.660	1.79	**0.000**	0.388	0.503
	scale	**0.235**	**15528**	**9265**				1.51	5.25	14.3	**0.000**	13.9	20.1
	KS	0.000	0.000	0.000				0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	**0.366**	**0.486**	**0.480**				0.402	0.420	0.448	**0.581**	0.464	0.488
Exponential	**shape**	0.000	0.000	0.000				0.000	0.000	0.000	0.000	0.000	0.000
	loc	0.665	-0.000	-0.000				0.956	-0.000	0.000	0.000	-0.000	-0.000
	scale	0.235	519	502				0.572	0.667	1.79	0.000	0.387	0.503
	KS	0.000	0.000	0.000				0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	0.518	0.914	0.907				0.690	0.565	0.579	–	0.853	0.891
Weibull	**shape**	1.00	0.443	0.242				0.472	**0.450**	0.380	1.00	0.872	0.396
	loc	0.665	-0.000	-0.000				0.956	**-0.000**	0.000	0.000	0.000	-0.000
	scale	0.235	923	319				0.651	**0.833**	0.297	0.000	0.134	0.004
	KS	0.000	0.000	0.000				0.000	0.001	0.001	0.000	0.000	0.000
	D-stat.	0.514	0.911	0.809				0.520	**0.388**	0.379	–	0.875	0.885
Pareto	**shape**	4e+07	1.00	1.00				1.44	1.00	0.272	1.00	1.00	**1.00**
	loc	-3e+07	0.000	0.000				0.812	0.000	-0.005	0.000	0.000	**-0.000**
	scale	3e+07	0.000	0.000				0.144	0.000	0.005	0.000	0.000	**0.000**
	KS	0.000	0.000	0.000				0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	0.600	–	–				0.444	–	0.447	–	0.566	**0.453**
Log-Normal	**shape**	1.00	1.96	3.49				1.61	12.4	**5.01**	1.00	0.152	0.130
	loc	0.720	-0.000	-0.000				0.955	-0.000	**0.000**	0.000	-2.76	-9.51
	scale	0.109	0.000	0.000				0.103	0.245	**0.337**	0.000	2.88	9.74
	KS	0.000	0.000	0.000				0.000	0.000	0.001	0.000	0.000	0.000
	D-stat.	0.618	0.866	0.663				0.464	0.413	**0.330**	–	0.471	0.470
Gamma	**shape**	0.785	0.001	0.002				**0.142**	0.032	0.043	–	**0.001**	0.002
	loc	0.691	16.5	60.0				**0.959**	-0.000	0.000	–	**0.000**	0.000
	scale	0.265	479381	194255				**4.01**	35.6	106	–	**585**	798
	KS	0.000	0.000	0.000				0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	0.516	0.916	0.946				**0.209**	0.584	0.596	–	**0.116**	0.914

Table D.16: Task inter-arrival time.

		Google						Yahoo!					
		map			reduce			map			reduce		
		lognormal						normal			gamma		
		shape	loc	scale				shape	loc	scale	shape	loc	scale
Normal	**shape**	0.000	0.000	0.000				**0.000**	0.000	0.000	0.000	0.000	0.000
	loc	1.03	1873	175				**0.000**	51.7	13.8	1162321	71.1	12.8
	scale	0.457	24117	4338				**0.000**	353	162	1e+08	6245	106
	KS	0.000	0.000	0.000				0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	0.511	0.468	0.477				**0.581**	0.441	0.464	0.500	0.448	0.451
Exponential	**shape**	0.000	0.000	0.000				0.000	0.000	0.000	0.000	0.000	0.000
	loc	0.575	-0.000	-0.000				0.000	-0.000	-0.000	0.035	-0.000	0.000
	scale	0.457	1873	173				0.000	51.7	13.8	1162319	215	12.8
	KS	0.000	0.000	0.000				0.000	0.012	0.020	0.000	0.000	0.000
	D-stat.	0.582	0.594	0.850				–	0.344	0.318	0.975	0.453	0.481
Weibull	**shape**	1.00	0.325	0.482				1.00	0.665	**0.503**	0.168	**0.338**	0.469
	loc	0.575	-0.000	-0.000				0.000	0.000	**-0.000**	0.035	**-0.000**	0.000
	scale	0.457	7935	57.3				0.000	34.1	**6.69**	48229	**44.6**	3.23
	KS	0.000	0.000	0.000				0.000	**0.149**	**0.242**	0.000	**0.062**	**0.368**
	D-stat.	0.533	0.478	0.775				–	0.167	**0.139**	0.517	**0.223**	0.102
Pareto	**shape**	1.59	1.00	1.00				1.00	1.00	1.00	**0.303**	1.00	**0.748**
	loc	-0.321	0.000	0.000				0.000	-14.8	-0.000	**-0.828**	0.000	**-0.718**
	scale	0.435	0.000	0.000				0.000	12.8	0.000	**0.863**	0.000	**0.718**
	KS	0.000	0.000	0.000				0.000	0.030	0.000	**0.357**	0.000	**0.400**
	D-stat.	0.820	–	–				–	0.241	0.865	**0.122**	–	**0.063**
Log-Normal	**shape**	1.00	**1.56**	6.42				1.00	**1.25**	4.90	0.183	6.46	1.99
	loc	0.683	**-0.248**	-0.000				0.000	**-0.024**	-0.000	-1e+07	-0.000	-0.000
	scale	0.211	**173**	0.073				0.000	**15.8**	0.407	1e+07	22.4	1.20
	KS	0.000	**0.501**	0.000				0.000	**0.381**	0.002	0.000	0.017	**0.457**
	D-stat.	0.473	**0.062**	0.762				–	**0.086**	0.389	0.570	0.243	0.074
Gamma	**shape**	**0.011**	0.004	**0.003**				–	0.001	0.001	0.001	0.008	0.008
	loc	**0.983**	439	**-0.000**				–	41.2	9.83	0.035	104	3.56
	scale	**4.29**	405685	**129392**				–	11849	6600	8e+09	14159	1225
	KS	0.000	0.000	0.000				0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	**0.044**	0.758	**0.171**				–	0.849	0.742	0.985	0.770	0.728

Table D.17: Task run time.

		Google						Yahoo!					
		map			reduce			map			reduce		
		normal						normal			normal		
		shape	loc	scale				shape	loc	scale	shape	loc	scale
Normal	**shape**	**0.000**	0.000	**0.000**				**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
	loc	**0.000**	2.10	**0.098**				**0.000**	**1.00**	**0.000**	**0.000**	**1.00**	**0.000**
	scale	**0.000**	0.380	**0.227**				**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
	KS	0.000	0.000	0.000				0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	**0.581**	0.400	**0.475**				**0.581**	**0.500**	**0.441**	**0.581**	**0.500**	**0.441**
Exponential	**shape**	0.000	0.000	0.000				0.000	0.000	0.000	0.000	0.000	0.000
	loc	0.000	1.72	-0.000				0.000	1.00	0.000	0.000	1.00	0.000
	scale	0.000	0.380	0.098				0.000	0.000	0.000	0.000	0.000	0.000
	KS	0.000	0.000	0.000				0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	–	0.428	0.839				–	–	–	–	–	–
Weibull	**shape**	1.00	**1.00**	0.112				1.00	1.00	1.00	1.00	1.00	1.00
	loc	0.000	**1.72**	0.000				0.000	1.00	0.000	0.000	1.00	0.000
	scale	0.000	**0.380**	0.156				0.000	0.000	0.000	0.000	0.000	0.000
	KS	0.000	0.000	0.000				0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	–	**0.291**	0.840				–	–	–	–	–	–
Pareto	**shape**	1.00	1.29	1.00				1.00	2e+08	1.00	1.00	2e+08	1.00
	loc	0.000	0.075	0.000				0.000	-1177010	0.000	0.000	-1177010	0.000
	scale	0.000	0.925	0.000				0.000	1177011	0.000	0.000	1177011	0.000
	KS	0.000	0.000	0.000				0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	–	0.599	0.499				–	1.000	0.930	–	1.000	0.930
Log-Normal	**shape**	1.00	1.00	2.27				1.00	1.00	1.00	1.00	1.00	1.00
	loc	0.000	1.81	-0.000				0.000	1.00	0.000	0.000	1.00	0.000
	scale	0.000	0.176	0.000				0.000	0.000	0.000	0.000	0.000	0.000
	KS	0.000	0.000	0.000				0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	–	0.373	0.816				–	–	–	–	–	–
Gamma	**shape**	–	0.214	0.186				–	–	–	–	–	–
	loc	–	1.93	0.000				–	–	–	–	–	–
	scale	–	0.823	0.364				–	–	–	–	–	–
	KS	0.000	0.000	0.000				0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	–	0.529	0.833				–	–	–	–	–	–

Table D.18: Task CPUs.

		Yahoo!					
		map			reduce		
		gamma			normal		
		shape	loc	scale	shape	loc	scale
Normal	**shape**	0.000	0.000	0.000	**0.000**	**0.000**	**0.000**
	loc	1481	9.58	0.837	**0.000**	**0.000**	**0.000**
	scale	9520	71.3	4.66	**0.000**	**0.000**	**0.000**
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	0.430	0.452	0.421	**0.581**	**0.158**	**0.500**
Exponential	**shape**	0.000	0.000	0.000	0.000	0.000	0.000
	loc	0.094	-0.000	0.000	0.000	0.000	0.000
	scale	1481	13.1	0.837	0.000	0.000	0.000
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	0.536	0.616	0.588	–	1.00	1.00
Weibull	**shape**	0.379	0.522	**0.266**	1.00	1.00	1.00
	loc	0.094	-0.000	**0.000**	0.000	0.000	0.000
	scale	242	8.07	**0.061**	0.000	0.000	0.000
	KS	**0.385**	0.000	**0.498**	0.000	0.000	0.000
	D-stat.	0.149	0.390	**0.073**	–	0.867	1.00
Pareto	**shape**	0.176	1.00	0.140	1.00	1.00	1.00
	loc	-0.045	0.000	-0.000	0.000	0.000	0.000
	scale	0.139	0.000	0.000	0.000	0.000	0.000
	KS	**0.098**	0.000	0.048	0.000	0.000	0.000
	D-stat.	0.175	–	0.231	–	0.979	0.674
Log-Normal	**shape**	**3.09**	4.49	5.42	1.00	1.00	1.00
	loc	**0.093**	-0.000	0.000	0.000	0.000	0.000
	scale	**38.0**	0.027	0.006	0.000	0.000	0.000
	KS	**0.333**	0.000	**0.266**	0.000	0.000	0.000
	D-stat.	**0.107**	0.558	0.134	–	1.00	1.00
Gamma	**shape**	0.059	**0.108**	0.078	–	937	2.33
	loc	0.094	**-0.000**	0.000	–	0.000	0.000
	scale	60063	**324**	25.7	–	0.000	0.000
	KS	0.000	0.003	0.041	0.000	0.000	0.000
	D-stat.	0.477	**0.337**	0.230	–	0.447	0.709

Table D.19: Task disk I/O ratio.

D.2.2 Relaxed Complex Model

	Yahoo!	
	map	**reduce**
Normal	**0.42 %**	**0.33 %**
Exponential	0.27 %	0.03 %
Weibull	0.19 %	0.03 %
Pareto	0.00 %	0.00 %
Log-Normal	0.14 %	0.03 %
Gamma	0.03 %	0.02 %

Table D.20: Task inter-arrival time matches.

	Yahoo!	
	map	**reduce**
Normal	**56.71 %**	**68.10 %**
Exponential	30.90 %	60.47 %
Weibull	39.21 %	62.79 %
Pareto	14.85 %	28.26 %
Log-Normal	18.83 %	45.06 %
Gamma	45.23 %	59.14 %

Table D.21: Task run time matches.

	Yahoo!	
	map	**reduce**
Normal	**0.13 %**	**0.40 %**
Exponential	0.00 %	0.00 %
Weibull	0.00 %	0.00 %
Pareto	0.00 %	0.00 %
Log-Normal	0.00 %	0.00 %
Gamma	0.00 %	0.00 %

Table D.22: Task CPUs matches.

	Yahoo!	
	map	**reduce**
Normal	0.04 %	**0.00 %**
Exponential	0.05 %	0.00 %
Weibull	0.00 %	0.00 %
Pareto	0.01 %	0.00 %
Log-Normal	0.02 %	0.00 %
Gamma	**0.21 %**	0.00 %

Table D.23: Task disk I/O ratio matches.

	Yahoo!	
	map	**reduce**
Normal	**0.00 %**	**0.00 %**
Exponential	**0.00 %**	**0.00 %**
Weibull	**0.00 %**	**0.00 %**
Pareto	**0.00 %**	**0.00 %**
Log-Normal	**0.00 %**	**0.00 %**
Gamma	**0.00 %**	**0.00 %**

Table D.24: Task memory matches.

		Yahoo!					
		map			reduce		
		normal			normal		
		shape	loc	scale	shape	loc	scale
Normal	**shape**	**0.000**	0.000	0.000	**0.000**	0.000	0.000
	loc	**0.000**	0.751	1.27	**0.000**	0.392	0.510
	scale	**0.000**	27.8	53.5	**0.000**	13.8	20.1
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	**0.581**	0.479	0.490	**0.581**	0.455	0.489
Exponential	**shape**	0.000	0.000	0.000	0.000	0.000	0.000
	loc	0.000	-0.000	-0.000	0.000	-0.000	-0.000
	scale	0.000	0.751	1.27	0.000	0.392	0.505
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	–	0.867	0.883	–	0.870	0.883
Weibull	**shape**	1.00	0.899	0.494	1.00	0.494	0.705
	loc	0.000	0.000	0.000	0.000	0.000	-0.000
	scale	0.000	0.477	0.392	0.000	0.016	0.647
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	–	0.859	0.883	–	0.852	0.880
Pareto	**shape**	1.00	1.00	**1.00**	1.00	1.00	**1.00**
	loc	0.000	0.000	**-0.000**	0.000	0.000	**-0.000**
	scale	0.000	0.000	**0.000**	0.000	0.000	**0.000**
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	–	0.540	**0.431**	–	0.716	**0.422**
Log-Normal	**shape**	1.00	0.119	0.120	1.00	0.120	0.119
	loc	0.000	-19.9	-38.3	0.000	-9.82	-14.3
	scale	0.000	20.1	38.7	0.000	9.98	14.5
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	–	0.400	0.472	–	0.396	0.461
Gamma	**shape**	–	**0.002**	0.002	–	**0.001**	0.001
	loc	–	**0.000**	0.000	–	**0.000**	0.000
	scale	–	**940**	1939	–	**585**	798
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	–	**0.113**	0.893	–	**0.135**	0.913

Table D.25: Task inter-arrival time.

		Yahoo!					
		map			reduce		
		normal			normal		
		shape	loc	scale	shape	loc	scale
Normal	**shape**	**0.000**	0.000	0.000	**0.000**	0.000	0.000
	loc	**0.000**	49.9	13.6	**0.000**	194	13.4
	scale	**0.000**	331	151	**0.000**	1111	102
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	**0.581**	0.440	0.459	**0.581**	0.431	0.446
Exponential	**shape**	0.000	0.000	0.000	0.000	0.000	0.000
	loc	0.000	-0.000	-0.000	0.000	1.000	-0.000
	scale	0.000	49.9	13.6	0.000	193	13.4
	KS	0.000	0.022	0.009	0.000	0.001	0.002
	D-stat.	–	0.326	0.313	–	0.420	0.412
Weibull	**shape**	1.00	0.654	**0.564**	1.00	0.591	0.553
	loc	0.000	0.000	**-0.000**	0.000	1.000	-0.000
	scale	0.000	30.0	**6.80**	0.000	92.7	11.6
	KS	0.000	**0.169**	**0.265**	0.000	**0.112**	0.041
	D-stat.	–	0.157	**0.129**	–	0.176	0.265
Pareto	**shape**	1.00	1.00	1.00	1.00	0.280	1.00
	loc	0.000	-14.0	-0.000	0.000	-0.309	-0.000
	scale	0.000	12.2	0.000	0.000	1.31	0.000
	KS	0.000	0.029	0.000	0.000	0.000	0.000
	D-stat.	–	0.242	0.869	–	0.373	0.887
Log-Normal	**shape**	1.00	**1.28**	4.46	1.00	**1.38**	**1.59**
	loc	0.000	**-0.019**	-0.000	0.000	**0.991**	**-0.126**
	scale	0.000	**15.0**	0.080	0.000	**44.0**	**3.08**
	KS	0.000	**0.377**	0.000	0.000	**0.299**	**0.165**
	D-stat.	–	**0.072**	0.534	–	**0.110**	**0.137**
Gamma	**shape**	–	0.001	0.001	–	0.005	0.001
	loc	–	40.5	10.1	–	113	10.2
	scale	–	11704	6549	–	15150	3163
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	–	0.830	0.714	–	0.795	0.841

Table D.26: Task run time.

		Yahoo!					
		map			reduce		
		normal			normal		
		shape	loc	scale	shape	loc	scale
Normal	**shape**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
	loc	**0.000**	**1.00**	**0.000**	**0.000**	**1.00**	**0.000**
	scale	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	**0.581**	**0.500**	**0.441**	**0.581**	**0.500**	**0.441**
Exponential	**shape**	0.000	0.000	0.000	0.000	0.000	0.000
	loc	0.000	1.00	0.000	0.000	1.00	0.000
	scale	0.000	0.000	0.000	0.000	0.000	0.000
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	–	–	–	–	–	–
Weibull	**shape**	1.00	1.00	1.00	1.00	1.00	1.00
	loc	0.000	1.00	0.000	0.000	1.00	0.000
	scale	0.000	0.000	0.000	0.000	0.000	0.000
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	–	–	–	–	–	–
Pareto	**shape**	1.00	2e+08	1.00	1.00	2e+08	1.00
	loc	0.000	-1177010	0.000	0.000	-1177010	0.000
	scale	0.000	1177011	0.000	0.000	1177011	0.000
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	–	1.000	0.930	–	1.000	0.930
Log-Normal	**shape**	1.00	1.00	1.00	1.00	1.00	1.00
	loc	0.000	1.00	0.000	0.000	1.00	0.000
	scale	0.000	0.000	0.000	0.000	0.000	0.000
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	–	–	–	–	–	–
Gamma	**shape**	–	–	–	–	–	–
	loc	–	–	–	–	–	–
	scale	–	–	–	–	–	–
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	–	–	–	–	–	–

Table D.27: Task CPUs.

		Yahoo!					
		map			reduce		
		weibull			normal		
		shape	loc	scale	shape	loc	scale
Normal	**shape**	**0.000**	0.000	0.000	**0.000**	**0.000**	**0.000**
	loc	**0.982**	12.3	1.29	**0.000**	**0.000**	**0.000**
	scale	**0.097**	84.4	9.11	**0.000**	**0.000**	**0.000**
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	**0.421**	0.458	0.439	**0.581**	**0.159**	**0.500**
Exponential	**shape**	0.000	0.000	0.000	0.000	0.000	0.000
	loc	0.884	-0.000	0.000	0.000	0.000	0.000
	scale	0.097	12.3	1.29	0.000	0.000	0.000
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	0.459	0.596	0.590	–	1.00	1.00
Weibull	**shape**	1.00	0.591	0.312	1.00	1.00	1.00
	loc	0.884	-0.000	0.000	0.000	0.000	0.000
	scale	0.097	12.6	0.306	0.000	0.000	0.000
	KS	0.000	0.000	**0.137**	0.000	0.000	0.000
	D-stat.	0.459	0.480	0.168	–	0.867	1.00
Pareto	**shape**	1.86	1.00	**0.483**	1.00	1.00	1.00
	loc	-0.451	0.000	**-0.014**	0.000	0.000	0.000
	scale	0.836	0.000	**0.014**	0.000	0.000	0.000
	KS	0.000	0.000	**0.145**	0.000	0.000	0.000
	D-stat.	0.574	–	**0.164**	–	0.979	0.674
Log-Normal	**shape**	1.00	**1.04**	3.04	1.00	1.00	1.00
	loc	0.907	**-0.088**	-0.000	0.000	0.000	0.000
	scale	0.045	**2.46**	0.044	0.000	0.000	0.000
	KS	0.000	**0.056**	**0.118**	0.000	0.000	0.000
	D-stat.	0.457	**0.197**	0.198	–	1.00	1.00
Gamma	**shape**	0.163	0.122	0.035	–	937	2.33
	loc	0.942	-0.000	0.000	–	0.000	0.000
	scale	0.241	64.1	74.3	–	0.000	0.000
	KS	0.000	0.000	0.000	0.000	0.000	0.000
	D-stat.	0.595	0.592	0.595	–	0.447	0.709

Table D.28: Task disk I/O ratio.

D.2.3 Safe Complex Model

		Google		Yahoo!	
		map	**reduce**	**map**	**reduce**
Normal	**shape**	0.000		0.000	0.000
	loc	26.9		0.187	0.356
	scale	2437		11.0	24.8
	KS	0.000		0.000	0.000
	D-stat.	0.494		0.460	0.483
Exponential	**shape**	0.000		0.000	0.000
	loc	-0.000		-0.000	-0.000
	scale	26.8		0.187	0.356
	KS	0.000		0.000	0.000
	D-stat.	0.940		0.888	0.921
Weibull	**shape**	0.256		0.956	0.624
	loc	-0.000		-0.000	-0.000
	scale	41.8		0.116	0.036
	KS	0.000		0.000	0.000
	D-stat.	0.949		0.892	0.912
Pareto	**shape**	1.00		1.00	1.00
	loc	0.000		0.000	0.000
	scale	0.000		0.000	0.000
	KS	0.000		0.000	0.000
	D-stat.	–		–	–
Log-Normal	**shape**	0.198		0.212	0.201
	loc	-927		-7.53	-14.8
	scale	966		7.90	15.5
	KS	0.000		0.000	0.000
	D-stat.	0.554		0.498	0.515
Gamma	**shape**	0.000		0.000	0.000
	loc	8.78		0.166	0.280
	scale	328728		5874	8149
	KS	0.000		0.000	0.000
	D-stat.	0.977		0.881	0.922

Table D.29: Task inter-arrival time.

		Google		Yahoo!	
		map	**reduce**	**map**	**reduce**
Normal	**shape**	0.000		0.000	0.000
	loc	2815		310	292
	scale	21252		264884	1396
	KS	0.000		0.000	0.000
	D-stat.	0.446		0.495	0.416
Exponential	**shape**	0.000		0.000	0.000
	loc	-0.000		-0.000	-0.000
	scale	2828		308	291
	KS	0.002		0.000	0.017
	D-stat.	0.360		0.550	0.292
Weibull	**shape**	0.525		**0.497**	**0.608**
	loc	-0.000		**-0.000**	**-0.000**
	scale	1127		**55.4**	**163**
	KS	**0.293**		**0.241**	**0.395**
	D-stat.	0.116		**0.113**	**0.087**
Pareto	**shape**	1.00		1.00	1.00
	loc	0.000		0.000	0.000
	scale	0.000		0.000	0.000
	KS	0.000		0.000	0.000
	D-stat.	–		–	–
Log-Normal	**shape**	**1.77**		0.231	1.56
	loc	**-0.161**		-162746	-0.056
	scale	**432**		174276	74.1
	KS	**0.411**		0.000	**0.343**
	D-stat.	**0.066**		0.590	0.089
Gamma	**shape**	0.003		0.000	0.007
	loc	1592		195	177
	scale	369374		6e+08	17009
	KS	0.000		0.000	0.000
	D-stat.	0.713		0.937	0.683

Table D.30: Task run time.

		Google		Yahoo!	
		map	**reduce**	**map**	**reduce**
Normal	**shape**	0.000		0.000	0.000
	loc	2.14		1.00	1.00
	scale	0.523		0.000	0.000
	KS	0.000		0.000	0.000
	D-stat.	0.535		0.500	0.500
Exponential	**shape**	0.000		0.000	0.000
	loc	1.61		1.00	1.00
	scale	0.523		0.000	0.000
	KS	0.000		0.000	0.000
	D-stat.	0.405		–	–
Weibull	**shape**	1.00		1.00	1.00
	loc	1.61		1.00	1.00
	scale	0.523		0.000	0.000
	KS	0.000		0.000	0.000
	D-stat.	0.410		–	–
Pareto	**shape**	1.28		7e+08	2e+08
	loc	0.075		-5148406	-1538968
	scale	0.925		5148407	1538969
	KS	0.000		0.000	0.000
	D-stat.	0.327		1.000	1.000
Log-Normal	**shape**	1.00		1.00	1.00
	loc	1.74		1.00	1.00
	scale	0.242		0.000	0.000
	KS	0.000		0.000	0.000
	D-stat.	0.401		–	–
Gamma	**shape**	0.407		–	–
	loc	1.80		–	–
	scale	0.820		–	–
	KS	0.000		0.000	0.000
	D-stat.	0.330		–	–

Table D.31: Task CPUs.

		Google		Yahoo!	
		map	**reduce**	**map**	**reduce**
Normal	**shape**			0.000	0.000
	loc			26792	0.000
	scale			1642977	0.000
	KS			0.000	0.000
	D-stat.			0.495	0.581
Exponential	**shape**			0.000	0.000
	loc			-0.000	0.000
	scale			26791	0.000
	KS			0.000	0.000
	D-stat.			0.943	–
Weibull	**shape**			0.272	1.00
	loc			-0.000	0.000
	scale			272	0.000
	KS			0.000	0.000
	D-stat.			0.519	–
Pareto	**shape**			1.00	1.00
	loc			0.000	0.000
	scale			0.000	0.000
	KS			0.000	0.000
	D-stat.			–	–
Log-Normal	**shape**			0.182	1.00
	loc			-816681	0.000
	scale			854997	0.000
	KS			0.000	0.000
	D-stat.			0.574	–
Gamma	**shape**			0.000	–
	loc			6588	–
	scale			1e+08	–
	KS			0.000	0.000
	D-stat.			0.988	–

Table D.32: Task disk I/O ratio.