Play It Again, SimMR!

Abhishek Verma
University of Illinois
at Urbana-Champaign, IL, US.
verma7@illinois.edu

Ludmila Cherkasova
Hewlett-Packard Labs
Palo Alto, CA, US.
lucy.cherkasova@hp.com

Roy H. Campbell
University of Illinois
at Urbana-Champaign, IL, US.
rhc@illinois.edu

***Abstract*—A typical MapReduce cluster is shared among different users and multiple applications. A challenging problem in such shared environments is the ability to efficiently control resource allocations among the running and submitted jobs for achieving users' performance goals. To ease the task of evaluating and comparing different provisioning and scheduling approaches in MapReduce environments, we have designed and implemented a simulation environment *SimMR* [1] which is comprised of three inter-related components: *i)* Trace Generator that creates a replayable MapReduce workload; *ii)* Simulator Engine that accurately emulates the job master functionality in Hadoop; and *iii)* a pluggable scheduling policy that dictates the scheduler decisions on job ordering and the amount of resources allocated to different jobs over time. We validate the accuracy of SimMR environment by, first, executing a set of realistic MapReduce applications in a 66-node Hadoop cluster and then by replaying the collected job execution traces in SimMR. Our simulator accurately reproduces the original job processing: the completion times of the simulated jobs are within 5% of the original ones. SimMR can process over one million events per second. This allows users to simulate complex workloads in a few seconds instead of multi-hour executions in the real testbed. Finally, by using SimMR we analyze and compare the performance of two novel deadline-driven schedulers over a diverse set of real and synthetic workloads.**

***Keywords*-MapReduce; Simulator; Traces; Schedulers.**

I. Introduction

The MapReduce model is becoming popular for performing advanced analytics over unstructured information and for enabling efficient "Big Data" processing. The list of companies exploiting Hadoop and new data processing opportunities is growing every day[2]. Cheap data storage and the scalable operational model underlying Hadoop allows users to apply advanced data mining and machine learning algorithms to extract information and discover novel data insights in non-traditional, game-changing ways. Typically, a MapReduce cluster is used for hosting different datasets, and its compute capacity is shared among multiple applications.

One of the most challenging tasks in such shared environments is the ability to tailor and efficiently control resource allocations among different jobs for achieving their performance goals. Often, the jobs are partitioned in different classes of service (e.g., platinum, silver, and bronze at Facebook [1]) and then processed by different Hadoop clusters with specially created management and resource allocation strategies. This is mainly done in order to guarantee performance isolation and have a predictable completion time for production jobs. However, when there is a need to expand the set of production jobs with new applications and additional data processing, first, one has to evaluate whether additional resources are required, and then how they should be allocated for meeting performance goals of the jobs in the extended set. To assist system administrators in performance evaluation and simplify MapReduce cluster management, new fast and accurate tools are needed.

In the past couple of years, job scheduling and workload management issues in MapReduce environments have received much attention. Currently, there are at least three different schedulers broadly used for job processing: the default FIFO scheduler, the *Capacity* scheduler [2], and the *Hadoop Fair Scheduler* (HFS) [3]. Each scheduler's decisions are based on several factors like simplicity, throughput, fairness, data locality, capacity guarantee, etc. Moreover, there are several research prototypes, e.g., FLEX [4], Dynamic Priority (DP) scheduler [5], ARIA [6], that aim to enhance the existing schedulers by exploiting new principles and performance models for supporting additional features.

Designing, prototyping, and evaluating new resource allocation and job scheduling algorithms in large-scale distributed systems such as Hadoop is a challenging, labor-intensive, and time-consuming task. Experiments performed in a real MapReduce testbed can take hours (to days) to obtain any preliminary results. Such evaluation is often limited to a set of specific applications (or benchmarks) available for experimentation. These experiments cannot be performed in production clusters of interest. Our goal is to design an accurate and fast simulation environment for evaluating workload management and resource optimization decisions in MapReduce environments. It will assist Hadoop cluster administrators in their daily tasks, helping them avoid error-prone, guess-based decisions.

In this work, we present a new MapReduce simulator, called *SimMR* (pronounced as *simmer*), that can replay execution traces of real workloads collected in Hadoop clusters (as well as synthetic traces based on statistical properties of workloads) for evaluating different resource allocation and scheduling ideas in MapReduce environments.

SimMR consists of the following three components:

1) **Trace Generator –** a module that generates a replayable workload trace by processing the job tracker logs or using a synthetic workload description.
2) **Simulator Engine –** a discrete event simulator that accurately emulates the Hadoop job master decisions for map/reduce slot allocations across multiple jobs.

[1]This work was partially completed during A. Verma's internship at HP Labs. R. Campbell and A. Verma are supported in part by NSF CCF grant #0964471.

[2]"Powered by" Hadoop, http://wiki.apache.org/hadoop/PoweredBy

3) **Scheduling policy –** a pluggable scheduling module that dictates the ordering of jobs and the amount of allocated resources to different jobs over time.

We validate the accuracy of SimMR by, first, executing a set of realistic MapReduce applications in a 66-node Hadoop cluster and then replaying the collected job execution traces in SimMR. The simulator accurately reproduces the original job processing with less than 2.7% average (6.6% maximum) error across the applications in the simulated set. We compare SimMR with the available open source, Apache's MapReduce simulator, called Mumak [7]. This simulator replays traces collected with a log processing tool, called Rumen [8]. In our evaluation study, we observe that Mumak's trace replay deviates significantly from the original job processing: Mumak's simulation exhibits 37% average (51.7% maximum) error while replaying the same traces.

The main difference between Mumak and SimMR is that Mumak omits modeling the shuffle/sort phase. For many applications this could lead to a significant error in completion time estimates and inaccurate workload modeling over time. We believe that the modeling framework proposed in SimMR for replaying the shuffle/sort and reduce phases could be adopted by Mumak to make it more accurate.

To assess the simulator speed, we collected traces from 1148 jobs run on our 66 node cluster during the last 6 months. The results show that SimMR replays these jobs in 1.5 seconds, while Mumak's execution takes 680 seconds to replay the same set of jobs. Thus, SimMR is two orders of magnitude faster than Mumak.

Finally, we present a case study with SimMR that is used for a fast (but accurate) performance analysis and comparison of two different deadline-driven Hadoop schedulers over a diverse set of workloads.

This paper is organized as follows. Section II provides a brief background on the MapReduce framework and motivates the problem. Section III describes the design choices and overall architecture of SimMR. We evaluate SimMR in Section IV. Section V provides a case study with SimMR by comparing two different deadline schedulers. Section VI describes the related work. Finally, we summarize the results and outline future work in Section VII.

II. Background and Motivation

MapReduce jobs are distributed and executed across multiple machines. The map stage is partitioned into *map tasks* and the reduce stage is partitioned into *reduce tasks*.

Each map task processes a logical split of the input data that generally resides on a distributed file system. The map task applies the user-defined map function on each record and buffers the resulting output. This intermediate data is hash-partitioned for the different reduce tasks and written to the local hard disk of the worker executing the map task.

The reduce stage consists of three phases: *shuffle*, *sort* and *reduce* phase. In the *shuffle phase*, the reduce tasks fetch the intermediate data files from map tasks, thus following the "pull" model. In the *sort phase*, the intermediate files from all the map tasks are sorted. An external merge sort is used in case the intermediate data does not fit in memory. After all the intermediate data is shuffled, a final pass is made to merge all these sorted files. Thus, the shuffle and sort phases are interleaved. We call this combined phase as just a shuffle phase. Finally, in the *reduce phase*, the sorted intermediate data is passed to the user-defined reduce function. The output from the reduce function is generally written back to the distributed file system.

Job scheduling in Hadoop is performed by the job master node called the *JobTracker*, which orchestrates a number of worker nodes called *TaskTrackers* in the cluster. Each TaskTracker has a fixed number of *map* and *reduce slots*, which can run tasks. The number of map and reduce slots is statically configured (typically to one or two per core). The TaskTrackers periodically send heartbeats to the JobTracker reporting the number of free slots and the progress of the tasks that they are currently running. Based on the availability of free slots and the rules of the scheduling policy, the JobTracker assigns map and reduce tasks to slots.

The amount of allocated resources may have a significant performance impact on the job completion time. Let us consider *WordCount*, a popular MapReduce application that counts the word frequencies in a text corpus. The map task tokenizes each line into words, while the reduce task counts the occurrence of each word. In our example, this job has 200 map tasks and 256 reduce tasks. Let us demonstrate and explain differences between the job executions as a function of the amount of resources allocated to the job. For this purpose, we have modified the default FIFO scheduler in Hadoop such that it allocates a requested number of map/reduce slots for a job execution (instead of maximum). First, in the 64 worker-node cluster configured with 2 map and 2 reduce slots per node, we run WordCount with 128 map and 128 reduce slots (the testbed's remaining details can be found in Section IV). Figure 1 shows the progress

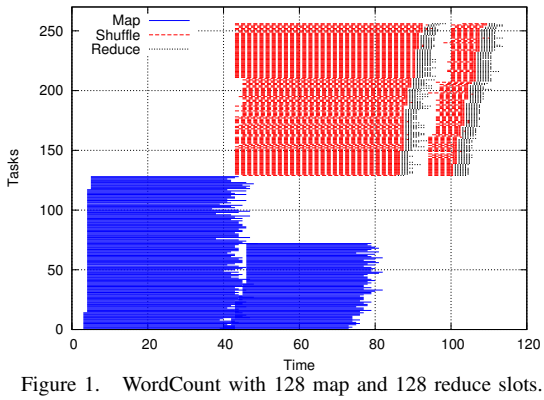


Figure 1. WordCount with 128 map and 128 reduce slots.

of the map and reduce tasks over time (on the x-axis) vs the 128 map slots and 128 reduce slots (on the y-axis). Since there are 200 map tasks, while only 128 map slots

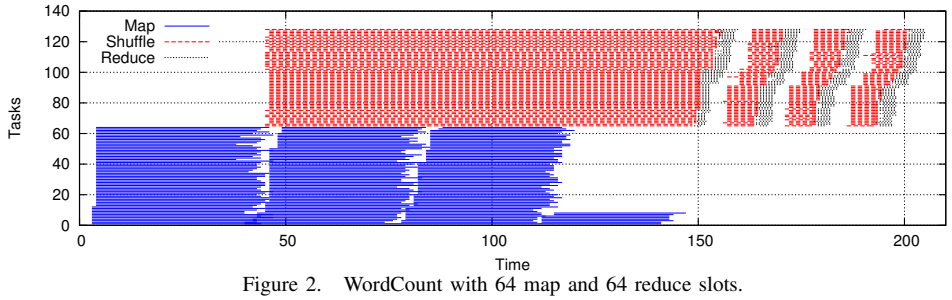


Figure 2. WordCount with 64 map and 64 reduce slots.

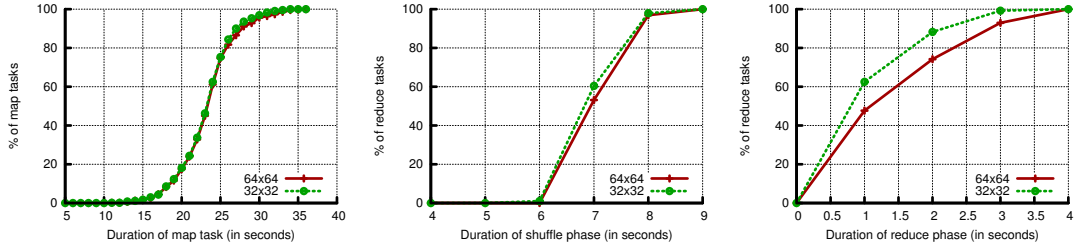


Figure 3. Comparison of CDFs of task durations for WordCount under two different resource allocations.

are allocated to the job, the map stage proceeds in multiple rounds of slot assignment, i.e., with 2 map *waves* clearly observed in Figure 1. Similarly, the reduce stage proceeds in 2 waves as well. We split each reduce task into its constituent shuffle and reduce phases. The shuffle phase begins only after the first map task has completed. The shuffle phase (of any reduce wave) can be completed when the entire map stage is complete and all the intermediate data generated by the map tasks has been shuffled to the reduce tasks and has been sorted. As seen in the figure, a part of the first shuffle phase overlaps with the map stage. Only when the shuffle phase completes, the reduce phase is performed.

Next, we run WordCount with lesser resources: 64 map slots and 64 reduce slots. As shown in Figure 2, both map and reduce stages proceed in 4 waves. The shuffle phase of the first reduce wave is significantly different from the shuffle phase in the next reduce waves (see Fig. 1, 2). This happens because the "first" shuffle phase overlaps with the entire map stage, and hence it depends on the number of map waves and their durations. Therefore we consider two sets of measurements: *first shuffle*, i.e., the shuffle phase of the first reduce wave, and *typical shuffle*, i.e., the shuffle phase of the other waves. Since we are looking for measurements that are invariant to the amount of allocated resources to the job, we characterize the first shuffle in a special way: the measurements represent only non-overlapping portions of the first shuffle with the map stage.

In our earlier work [6], we used a compact job profile comprised of performance *invariants* that characterize the job execution during map, shuffle, and reduce phases via *average* and *maximum* task durations. We demonstrated that these simple metrics are very stable (within 10-15%) across different job executions. These earlier results are encouraging for generating synthetic distributions driven by these parameters. The question we need to answer is whether the distributions of task durations (in addition to their averages and maxima) are also similar for different job executions? Figure 3 shows the distributions of map, reduce, and shuffle task durations for two job executions with different resources. Indeed, the duration distributions of these two different executions are very similar. To perform a more representative study, we compared the distributions (map, shuffle, and reduce task durations) of five different executions of six applications (these applications are described in more detail in Section IV). The Kullback-Leibler divergence [9], [10] is often used as a standard measure to compare the difference (or distance) between two probability distributions P and Q. The KL divergence of a distribution function Q from the distribution function P is defined as:

$$D(P||Q) = \sum_i P(i) \cdot log \frac{P(i)}{Q(i)}$$

We use a symmetric version of KL divergence defined as

$$D'(P||Q) = \frac{1}{2}(D(P||Q) + (D(Q||P)$$

Table I shows the summary of KL divergence for 10 pairwise comparison of different executions of the same application. We show the minimum, average and 95-th percentile of collected KL values.

Application	**Map** (in seconds)			**Shuffle** (in seconds)			**Reduce** (in seconds)		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
WordCount	0.05	0.07	0.09	0.13	0.34	0.72	0.00	0.04	0.07
WikiTrends	0.09	0.10	0.13	0.71	2.44	3.94	0.11	0.25	0.35
Twitter	0.04	0.06	0.08	0.28	0.84	1.39	0.02	0.07	0.12
Sort	0.02	0.09	0.20	0.35	1.69	3.42	0.11	0.22	0.33
TFIDF	0.00	0.02	0.05	0.14	1.39	4.40	0.07	0.27	0.73
Bayes	0.07	0.12	0.15	0.19	0.57	1.14	0.02	0.04	0.07

Table I
KULLBACK-LEIBLER VALUES FOR DIFFERENT APPLICATIONS.

Indeed, these values are very small. On the other hand, when we compared the runs of different applications, the (min, avg, max) KL values for map task were (7.34, 11.56, 13.25), for shuffle task were (11.31, 13.05, 13.49) and

reduce task were (9.11, 12.66, 13.30). These values are much higher than the KL values for executions of the same application. Thus, the phase duration distributions are very similar for the same application and different for different applications. Therefore any one of the executions (as a job representative) can be used for a future job replay in the simulation environment when exploring different resource allocation and scheduling policies.

III. SimMR Design

Our goal is to build a simulator which is capable of replaying the scheduling decisions over a large workload (several months of job logs) in a few minutes on a single machine. We focus on simulating the job master decisions and the task/slot allocations across multiple concurrent jobs. This would aid in understanding the efficacy of our scheduling and resource allocation algorithms. It is a non-goal to simulate details of the TaskTracker nodes (their hard disks or network packet transfers) as done by MRPerf [11]. Instead, we use job profiles (with task durations) to represent the latencies during different phases of MapReduce processing in the cluster.

Figure 4 shows the overall design of SimMR. The job traces can be generated using two methods. Firstly, they can be obtained from actual jobs executed on the real cluster using the **MRProfiler**. Alternatively, the trace can be synthetically generated using **Synthetic TraceGen** by observing the statistical properties of the workloads. These collected traces are stored persistently in the **Trace Database**. Using the job trace and a **Scheduling policy** as input, the **Simulator Engine** replays the trace by enforcing the scheduling and resource allocation decisions and generates the output log. Various scheduling policies, such as **FIFO**, **MinEDF** and **MaxEDF** that are considered in these paper, can be enforced by SimMR.

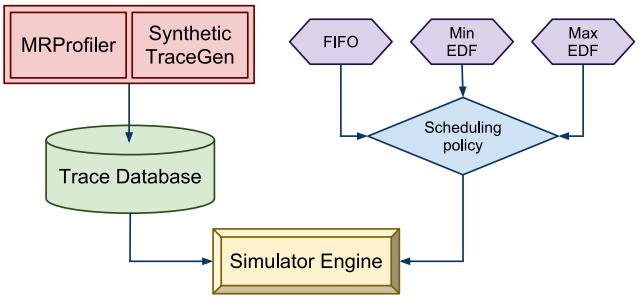


Figure 4. SimMR Design

A. Trace Generation

We can generate job traces using two methods: MRProfiler and Synthetic TraceGen. MRProfiler extracts the job performance metrics by processing the counters and logs stored at the JobTracker at the end of each job. The job tracker logs reflect the MapReduce jobs' processing in the Hadoop cluster. They faithfully record the detailed information about the map and reduce tasks' processing. The logs also have useful information about the shuffle/sort stage of each job. Alternatively, we can model the distributions of the durations based on the statistical properties of the workloads and generate synthetic traces using Synthetic TraceGen. This can help evaluate hypothetical workloads and consider what-if scenarios. We store job traces persistently in a Trace database (for efficient lookup and storage) using a *job template*. The job template summarizes the job's essential performance characteristics during its execution in the cluster. We extract the following metrics for each job J:

- $(\mathbf{N_M^J}, \mathbf{N_R^J})$ - the number of map tasks and reduce tasks respectively that constitute job J;
- **MapDurations (M^J):** the array consisting of N_m^J durations of map tasks.
- **FirstShuffleDurations (Sh_1^J):** the array representing durations of non-overlapping part of first shuffle tasks.
- **TypicalShuffleDurations (Sh_{typ}^J):** the array representing the durations of the typical shuffle tasks.
- **ReduceDurations (R^J):** the array consisting of N_R^J durations of reduce tasks.

B. Simulator Engine

Simulator Engine is the main component of SimMR which replays the given job trace. It manages all the discrete events in simulated time and performs the appropriate action on each event. It maintains data structures similar to the Hadoop job master such as a queue of submitted jobs $jobQ$. The slot allocation algorithm makes a new decision when a map or reduce task completes. Since our goal is to be fast and accurate, we simulate the jobs at the task level and do not simulate details of the TaskTrackers.

The simulator engine reads the job trace from the trace database. It communicates with the scheduler policies using a very narrow interface consisting of the following functions:

1) chooseNextMapTask(jobQ),
2) chooseNextReduceTask(jobQ)

These two functions ask the scheduling policy to return the jobId of the job whose map (or reduce) task should be executed next.

The simulator maintains a priority queue Q for *seven event types*: job arrivals and departures, map and reduce task arrivals and departures, and an event signaling the completion of the map stage. Each event is a triplet $(eventTime, eventType, jobId)$ where $eventTime$ is the time at which the event will occur in the simulation; $eventType$ is one of the seven event types; and $jobId$ is the job index of the job with which the event is associated.

The simulator engine fires events and runs the corresponding event handlers. It tracks the number of completed map and reduce tasks and the number of free slots. It allocates the map slots to tasks as dictated by the scheduling policy. When $minMapPercentCompleted$ percentage of map tasks are completed (it is the parameter set by the user), it starts

scheduling reduce tasks. We could have started the reduce tasks directly after the map stage is complete. However, the shuffle phase of the reduce task occupies a reduce slot and has to be modeled as such. Hence, we schedule a filler reduce task of infinite duration and update its duration to the first shuffle duration when all the map tasks are complete. This enables accurate modeling of the shuffle phase.

C. Scheduling policies

Different scheduling and resource allocation policies can be used with SimMR for their evaluation, e.g.:

- **FIFO:** This policy finds the earliest arriving job that needs a map (or reduce) task to be executed next.
- **MaxEDF:** Similar to FIFO, this policy finds the job with the earliest deadline which has an unscheduled map (or reduce) task.
- **MinEDF:** This policy calculates the minimum number of map and reduce slots that need to be allocated for the job to be completed within the user specified deadline, when the job arrives into the system as described later in Section V. It also keeps track of the number of running and scheduled map and reduce tasks so that they are always less than the "wanted" number of slots.

IV. SimMR Evaluation

In this section, we evaluate the accuracy and performance of SimMR, and compare it against the open source, Apache's MapReduce simulator, called Mumak [7].

A. Mumak and Rumen

Rumen [8] is a data extraction and analysis tool built for MapReduce environments. Rumen (similar to our MRProfiler) can process job history logs to generate trace files describing the task durations, the number of bytes and records read and written, etc. The trace files generated by Rumen can be replayed by the MapReduce simulator Mumak [7]. Rumen collects more than 40 properties for each map/reduce task and all the job counters. On the other hand, our MRProfiler is selective and stores only the task durations. However, MRProfiler is easily extendable if we find that additional job metrics are needed for the simulation.

An overarching design goal for Mumak is that it aims to execute the exact same MapReduce schedulers "as-is" without any changes. SimMR, on the other hand, does not have this objective and interfaces with the scheduling policy using a very narrow interface. Mumak does not simulate the running of the actual map/reduce tasks. Similar to SimMR, Mumak uses a special AllMapsFinished event generated by the SimulatedJobTracker to trigger the start of the reduce-phase. However, Mumak models the total runtime of the reduce task as the summation of the time taken for completion of all maps and the time taken for an individual task to complete the reduce phase (without the shuffle). Thus, Mumak does not model the shuffle phase accurately.

B. Experimental Testbed

We perform our experiments on 66 HP DL145 GL3 machines. Each machine has four AMD 2.39MHz cores, 8 GB RAM and two 160GB hard disks. The machines are set up in two racks and interconnected with gigabit Ethernet. We used Hadoop 0.20.2 with two machines for JobTracker and NameNode, and remaining 64 machines as worker nodes. Each slave is configured with a single map and reduce slot. The default blocksize of the file system is set to 64MB and the replication level is set to 3. We disabled speculation as it did not lead to any significant improvements.

C. Workload Trace

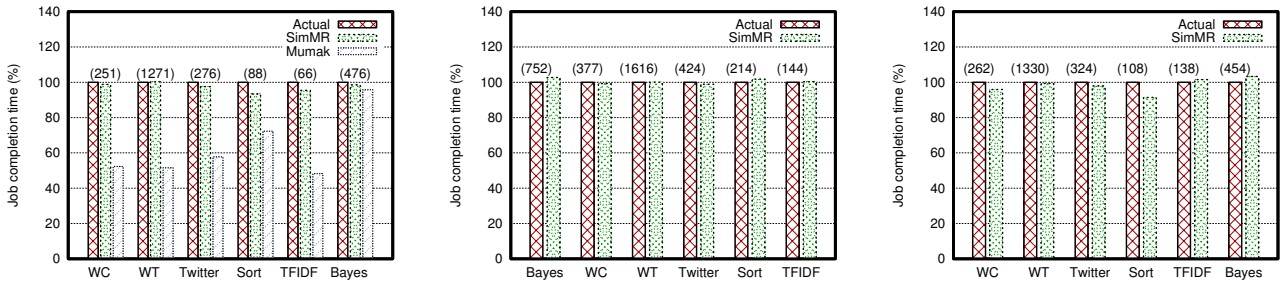
Our workload consists of a set of representative applications executed on three different datasets as follows:

1) **Word count:** This application computes the occurrence frequency of each word in 32GB, 40GB and 43GB Wikipedia article history dataset.
2) **Sort:** The Sort application sorts 16GB, 32GB and 64GB of random data generated using random text writer in GridMix2[3].
3) **Bayesian classification:** We use a step from the example of Bayesian classification trainer in Mahout[4]. The mapper extracts features from the input corpus and outputs the labels along with a normalized count of the labels. The reduce performs a simple addition of the counts and is also used as the combiner. The input dataset is the same Wikipedia article history dataset, except the chunks are split at page boundaries.
4) **TF-IDF:** The Term Frequency - Inverse Document Frequency application is often used in information retrieval and text mining. It is a statistical measure to evaluate how important a word is to a document. We used the TF-IDF example from Mahout and used the same Wikipedia articles history dataset.
5) **WikiTrends:** We use the data from Trending Topics (TT)[5]: Wikipedia article traffic logs that were collected (and compressed) every hour in April, May and June 2010. Our MapReduce application counts the number of times each article has been visited.
6) **Twitter:** This application uses the 12GB, 18GB and 25GB twitter dataset created by Kwak et. al. [12] containing an edgelist of twitter userids. Each edge (i, j) means that user i follows user j. The Twitter application counts the number of asymmetric links in the dataset, that is, $(i, j) \in E$, but $(j, i) \notin E$.

D. SimMR Accuracy

First, we evaluate the accuracy of SimMR and compare it against Mumak using the FIFO scheduler (this scheduler is

[3]http://hadoop.apache.org/mapreduce/docs/current/gridmix.html
[4]http://http://mahout.apache.org/
[5]http://trendingtopics.org



(a) FIFO (b) MinEDF (c) MaxEDF

Figure 5. Simulator accuracy across different scheduling policies. The numbers in the parentheses above the bars represent the actual job completion time in seconds.

available in both simulators). We collected a real workload trace consisting of three executions of the six applications. We replay this trace using SimMR and Mumak. Figure 5(a) shows a comparison of the duration of the simulated job with respect to the real job duration across different applications. We observe that SimMR faithfully replays the traces with less than 2.7% average (6.6% maximum) error across all the applications. On the other hand, Mumak underestimates the job completion time and has 37% average (51.7% maximum) error while replaying the same traces.

Additionally, we validate the accuracy of SimMR by simulating MinEDF and MaxEDF schedulers (discussed in more detail in the next Section V) and comparing the simulation results to the testbed runs respectively. We collected a real workload trace consisting of three executions of the six applications using both the schedulers. Figures 5(b) and 5(c) show a comparison of the duration of the simulated job with respect to the real job duration across different applications. We observe that SimMR replays the traces with less than 3.7% average (8.6% maximum) error across all the applications for MaxEDF and less than 1.1% average (2.7% maximum) error for MinEDF.

In summary, for considered diverse applications and different schedulers, SimMR replays traces with high fidelity.

E. SimMR Performance

We collected traces from 1148 jobs run on our 66 node cluster during 6 months of November 2010 to April 2011. We created a single trace file (without inactivity periods) and replayed it using SimMR and Mumak. These jobs would take about a week (152 hours) if they were to be executed serially. Figure 6 shows the performance comparison of the two simulators. Note, that Y-axes are in log scale. SimMR replays these jobs in 1.5 seconds, while Mumak takes 680 seconds to replay the same set of jobs. Thus, SimMR is more than 450 times faster than Mumak in simulating these traces. On closer inspection, we observe that Mumak simulates the TaskTrackers and the heartbeats between them, which leads to greater number of simulated events and computation.

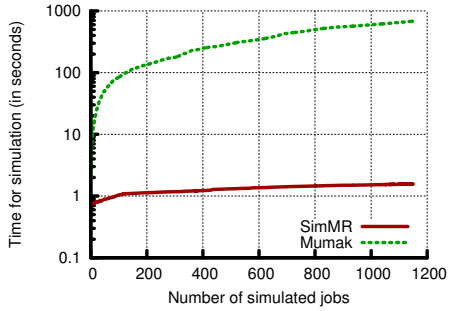


Figure 6. Performance comparison of simulators.

V. Case Study: Comparing Two Schedulers

In this section, we introduce two different deadline-based schedulers and demonstrate how to use SimMR for comparing them. Originally, Hadoop employed a simple FIFO job scheduler for efficient batch processing. Later, the Capacity scheduler [2] and Hadoop Fair Scheduler [3] were introduced for enabling multiple concurrent job executions on the same Hadoop cluster. However, these schedulers do not aim at a tailored control of allocated resources to achieve the application performance goals, e.g., the job completion time within a given (soft) deadline.

A. Deadline-based Scheduling: MaxEDF and MinEDF

The *deadline-based* scheduler should answer two interrelated questions: which job should the slots be allocated and how many slots should be allocated to the job? In this work, we consider two policies: *MaxEDF* and *MinEDF*.

Both schedulers execute the *Earliest Deadline First* algorithm (EDF) for job ordering since this real-time scheduling is known to maximize the utility function of all the users. The difference comes in the amount of resources allocated to the job by these schedulers.

The *MaxEDF* scheduler aims to allocate the maximum available number of map (or reduce) slots for each job in the queue (i.e., apart from the EDF job ordering, the resource allocation per job is the same as under the FIFO policy). In such a way, the job might finish much earlier than the given deadline. Intuitively, such an approach optimizes pipelining of map and reduce stages of different jobs as follows. Once a map stage of a first job is completed and it proceeds to the reduce stage execution, the next job could start processing its map stage, etc. Typically, such a pipelined execution might result in the best makespan (completion time) for a given set of jobs. However, the possible drawback of the proposed schema might be that in many cases, it is

impossible to preempt the already allocated resources to the earlier job (without killing its currently running tasks) to provide resources for a newly arrived job with a more "urgent" deadline.

The *MinEDF* scheduler allocates the minimal amount of map and reduce slots that would be required for meeting a given job deadline. So, this approach aims to allocate the minimum sufficient resources to the job for completing within the deadline and leaves the remaining, spare resources to the next arriving job. This minimal amount of resources is computed with a specially designed performance model introduced in [6] and briefly described below.

The proposed MapReduce performance model evaluates lower and upper bounds on the job completion time. It is based on a general model for computing performance bounds on a makespan of a given set of n tasks that are processed by k servers (e.g., n map tasks are processed by k slots in MapReduce environment). Let $T_1, T_2, \ldots, T_n$ be the duration of n tasks in a given set. Let k be the number of slots that can each execute one task at a time. The assignment of tasks to slots is done using an online, *greedy* algorithm: assign each task to the slot with the earliest finishing time. Let avg and max be the *average* and *maximum* duration of the n tasks respectively. Then the makespan of a greedy task assignment is at least $(n \cdot avg)/k$ and at most $(n-1) \cdot avg/k + max$. These lower and upper bounds on the completion time can be easily computed if we know the average and maximum durations of the set of tasks and the number of allocated slots.

As motivated by the above model, in order to approximate the overall completion time of a MapReduce job, we need to estimate the *average* and *maximum* task durations during different execution phases of the job, i.e., map, shuffle/sort, and reduce phases. The MRProfiler (described in Section III) creates the detailed job template which characterizes the task durations during all the phases of the job execution. This data is used to compute average and maximum task durations in different phases, and then to compute lower and upper bounds for each execution phase of the job. By applying the bounds model, we can express the estimates for job completion time (lower bound T_J^{low} and upper bound T_J^{up}) as a function of map/reduce tasks (N_M^J, N_R^J) and the allocated map/reduce slots (S_M^J, S_R^J) using the following equation form:

$$T_J^{low} = A_J^{low} \cdot \frac{N_M^J}{S_M^J} + B_J^{low} \cdot \frac{N_R^J}{S_R^J} + C_J^{low} \qquad (1)$$

The equation for T_J^{up} can be written similarly (for details, see [6]). Typically, the average of lower and upper bounds is a good approximation of the job completion time.

Note, that once we have a technique for predicting the job completion time, it also can be used for solving the inverse problem: finding the appropriate number of map and reduce slots that could support a given job deadline. Equation 1 yields a hyperbola if S_M^J and S_R^J are the variables. All integral points on this hyperbola are possible allocations of map and reduce slots which result in meeting the same deadline. There is a point where the sum of the required map and reduce slots is minimized. We calculate this minima on the curve using Lagrange's multipliers [6], since we would like to conserve (minimize) the number of map and reduce slots required for the adequate resource allocation per job.

In such a way, the *MinEDF* scheduler allocates the minimal amount of map and reduce slots that would be needed for meeting a given job deadline.

In the simulations and the respective testbed evaluations, we will assess the quality of scheduling and resource allocation decisions by observing the following metric. Let the execution consist of a given set of n jobs $J_1, J_2, \ldots, J_n$ with corresponding deadlines $D_1, D_2, \ldots, D_n$. Let these jobs be completed at times $T_1, T_2, \ldots, T_n$, and let Θ be the set of all jobs whose deadline has been exceeded. Then we compute the following utility function: $\sum_{J \in \Theta} \frac{T_J - D_J}{D_J}$ This function denotes the sum of *relative deadlines exceeded*. The scheduling and resource allocation policy that minimizes this value is a better candidate for a deadline-based scheduler.

To compare performance of *MaxEDF* and *MinEDF*, we analyze these policies with our simulator SimMR and the following workloads:

1) A real testbed trace comprised of multiple job runs in our 66-node cluster, and
2) A synthetic trace generated with statistical distributions that characterize the Facebook workload.

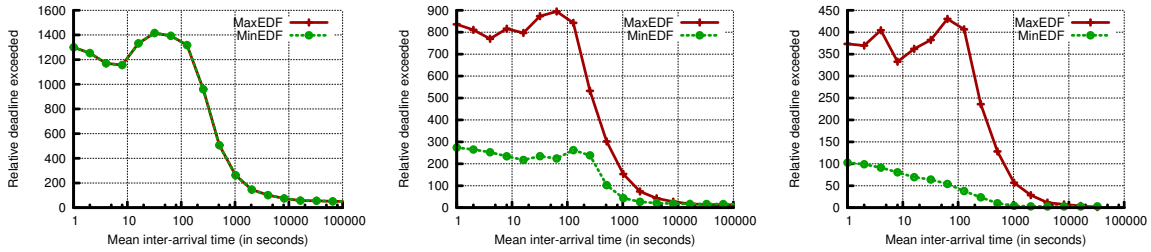
B. Replaying Real Traces with SimMR

For the real workload trace, we use a mix of the six realistic applications with different input dataset sizes as introduced and described in Section IV-C. We run these applications with three different datasets in our 66-nodes Hadoop testbed, and then by using MRProfiler, create the replayable job traces for SimMR. We generate an equally probable random permutation of arrival of these jobs and assume that the inter-arrival time of the jobs is exponential.

The job deadline (which is relative to the job completion time) is set to be uniformly distributed in the following interval $[T_J, df \cdot T_J]$, where T_J is the completion time of job J given all the cluster resources (i.e., maximum amount of map/reduce slots that job can utilize) and where $df \geq 1$ is a given deadline factor.

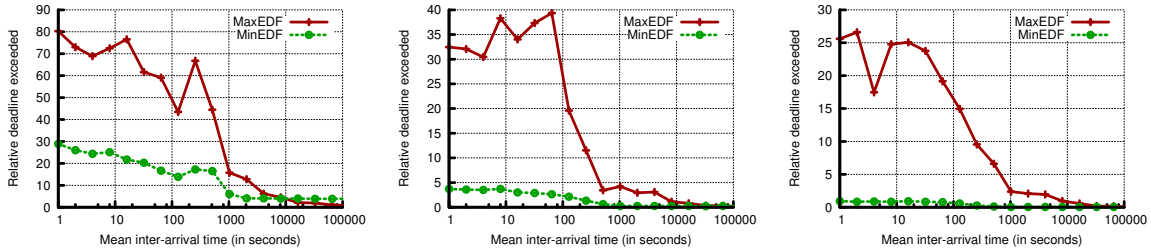
We run the simulation 400 times and report the average deadline exceeded metric while varying the mean of the exponential inter arrival times and the deadline factor as shown in Figure V-A.

When the deadline factor is set to 1, i.e., $df = 1$, then the performance of MinEDF and MaxEDF policies coincide as shown in Figure 7(a), because the maximum amount of map/reduce slots that job can utilize should be allocated under both policies. The relative deadline exceeded metric



(a) Deadline factor = 1 (b) Deadline factor = 1.5 (c) Deadline factor = 3

Figure 7. Simulating *MaxEDF* and *MinEDF* with **Real Testbed Workload**.



(a) Deadline factor = 1.1 (b) Deadline factor = 1.5 (c) Deadline factor = 2

Figure 8. Simulating *MaxEDF* and *MinEDF* with **Synthetic Facebook Workload**.

decreases as the arrival rate of jobs decreases and a larger fraction of them is capable of meeting the targeted deadline. There is a slight "bump" around the mean arrival time of 100s. On closer inspection we found that this is caused because the scheduler does not pre-empt tasks themselves. So, if a decision to allocate resources to a task has been made the slot is not available for allocation to the earlier deadline job which just arrived.

When the deadlines are relaxed by a factor of 1.5, MinEDF allocates the minimum required slots and shares the cluster resources among multiple jobs more efficiently. This leads to a smaller value of relative deadline exceeded metric as shown in Figure 7(b). The performance gap between MinEDF and MaxEDF policies increases when the deadline is further relaxed by a factor of 3 as shown in Figure 7(c).

In summary, for the realistic testbed workload and a variety of studied parameters, the MinEDF scheduler shows superior results compared to the MaxEDF policy.

C. Replaying Synthetically Generated Facebook Trace

In this section, we extend the performance comparison of MinEDF and MaxEDF policies by using SimMR and a set of synthetically generated traces. Zaharia et. al. [3] provide a detailed description about MapReduce jobs in production at Facebook in October 2009 (we use Figure 1 and Table 3 from [3]). We extract the CDF from the plot of map and reduce durations in Figure 1 of [3], and then we try to identify the statistical distributions which best fits the provided plot. We fit more than 60 distributions such as Weibull, LogNormal, Pearson, Exponential, Gamma, etc. using StatAssist[6]. Our analysis shows that the LogNormal distribution fits best the provided CDF of the Facebook task duration distribution. $LN(9.9511, 1.6764)$ fits the map task CDF with a Kolmogorov-Smirnov value of 0.1056, where $LN(x;\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}}\, e^{-\frac{(\ln x-\mu)^2}{2\sigma^2}}$ is the Log-normal distribution with mean μ and variance σ. For the reduce task duration, $LN(12.375, 1.6262)$ fits with a Kolmogorov-Smirnov value of 0.0451.

In our Synthetic TraceGen module, we use these respective LogNormal distributions to generate a synthetic workload that is similar to a reported Facebook workload. Figure 8 shows the SimMR's outcome of replaying the generated synthetic workloads with MinEDF and MaxEDF policies. The performance results are consistent with the outcome of testbed traces' simulations: the MinEDF scheduler significantly outperforms the MaxEDF policy.

[6]http://www.mathwave.com/help/easyfit/html/tools/assist.html

VI. Related Work

While MapReduce is a relatively new programming paradigm, there are a few on-going efforts on developing simulation tools for MapReduce environments.

The designers of *MRPerf* [11] aim to provide a fine-grained simulation of MapReduce setups throughout different phases. To model inter- and intra rack task communications over network as well as to accurately model the network behavior, *MRPerf* is based on the widely-used ns-2 network simulator [13]. The authors are interested in modeling different cluster topologies and in their impact on the MapReduce job performance. For map/reduce task modeling, *MRPerf* creates a number of simulated nodes, where each node might have several processors and a single disk (it is the *MRPerf* limitation). There are a few simplifying assumptions about the application behavior: that a job has simple map and reduce tasks with compute time requirements that are proportional to the data size but not the content (or type of processing) of the data.

In our work, we focus on simulating the job master decisions and the task/slot allocations across multiple jobs. We do not simulate details of the TaskTrackers (their hard disks or network packet transfers) as done by *MRPerf*. In spite of this, our approach accurately reflects the job processing because of our profiling technique to represent job latencies during different phases of MapReduce processing in the cluster. Our approach does not have many of *MRPerf*'s limitations. Moreover, it is very fast compared to *MRPerf* which deals with network-packet level simulations.

Another effort presents a simulator [14] that utilizes SimJava [15] and GridSim [16]. This tool is in very early stages of development. In the short paper, authors describe their goals to build a simulator for assessing a future application design (i.e., the applications that do not yet exist) rather than replaying traces of already existing applications. The authors are interested is evaluating the application scalability and parameter/configuration tuning.

Cardona et al. [17] discuss how to build a federated MapReduce environment on top of different Hadoop clusters. There are quite a few issues that need to be reconsidered in Hadoop while building such a system. One of the issues is that the original Hadoop assumes a homogeneous environment, and there are a few internal mechanisms that utilize this assumption. The authors discuss the modifications to Hadoop that are useful to support heterogeneity. To justify the set of proposed modifications the authors design a simulation environment based on GridSim [16].

The closest approach to our SimMR is the open source, Apache's MapReduce simulator, called Mumak [7]. This simulator replays traces collected with a log processing tool, called Rumen [8]. We discussed Mumak implementation details in Section IV. The main difference between Mumak and SimMR is that Mumak omits modeling the shuffle/sort phase. As we have shown it could lead to inaccurate results. We believe that the modeling approach undertaken in our simulator SimMR could be adopted by Mumak.

VII. Conclusion

To become enterprise-ready, the Hadoop open-source stack needs to be enhanced with new tools required in enterprise environments to support robust performance management. Due to lack of performance guarantees for job completion times when executed in shared environments (while many enterprise applications require such time guarantees), there is a need for new workload management strategies and supporting tools. In this work, we introduce a simulation environment SimMR that can assist system administrators in performance analysis and evaluation of new resource allocation and job scheduling algorithms in large-scale distributed systems such as Hadoop and other performance related tasks in MapReduce cluster management. The proposed SimMR simulator is accurate and fast: it can simulate a complex multi-hour workload in less than a second. It is aimed at helping Hadoop administrators in their daily tasks: SimMR can quickly replay production cluster workloads with different scenarios of interest, assess various *what-if* questions, and help avoiding error-prone decisions.

In our future work, we plan to design a trace-scaling technique where from the trace of a job execution on a small dataset, we could generate a trace that represents job processing of a larger dataset. We intend to analyze how SimMR can incorporate other useful job metrics and be integrated with complementary simulation tools, e.g., network simulators for modeling the shuffle phase.

References

[1] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu, "Data warehousing and analytics infrastructure at Facebook," in *Proc. of the 2010 Intl. Conference on Management of Data*. ACM, 2010.

[2] Apache, "Capacity Scheduler Guide," 2010. [Online]. Available: http://hadoop.apache.org/common/docs/r0.20.1/capacity_scheduler.html

[3] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *Proc. of EuroSys*. ACM, 2010, pp. 265–278.

[4] J. Wolf, D. Rajan, K. Hildrum, R. Khandekar, V. Kumar, S. Parekh, K.-L. Wu, and A. Balmin, "FLEX: A Slot Allocation Scheduling Optimizer for MapReduce Workloads," *Proc. of ACM/IFIP/USENIX Intl. Middleware Conference*, 2010.

[5] T. Sandholm and K. Lai, "Dynamic Proportional Share Scheduling in Hadoop," *LNCS: Proc. of the 15th Workshop on Job Scheduling Strategies for Parallel Processing*, 2010.

[6] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: Automatic Resource Inference and Allocation for MapReduce Environments," in *International Conference on Autonomic Computing (ICAC)*. ACM/IEEE, 2011.

[7] Apache, "Mumak: Map-Reduce Simulator," 2010. [Online]. Available: https://issues.apache.org/jira/browse/MAPREDUCE-728

[8] ——, "Rumen: a tool to extract job characterization data from job tracker logs," 2010. [Online]. Available: https://issues.apache.org/jira/browse/MAPREDUCE-751

[9] S. Kullback and R. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, 1951.

[10] S. Kullback, *Information theory and statistics*. Wiley, 1959.

[11] G. Wang, A. Butt, P. Pandey, and K. Gupta, "A simulation approach to evaluating design decisions in mapreduce setups," in *Intl. Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, 2009.

[12] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a social network or a news media?" in *Proc. of intl. conference on World Wide Web*. ACM, 2010, pp. 591–600.

[13] "Network Simulator (NS2)," 2005. [Online]. Available: http://www.isi.edu/nsnam/ns

[14] S. Hammoud, M. Li, Y. Liu, N. Alham, and Z. Liu, "MRSim: A Discrete Event Based MapReduce Simulator," in *Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on*, vol. 6, 2010.

[15] F. Howell and R. McNab, "SimJava: A Discrete Event Simulation Library for Java," *Simulation Series*, vol. 30, 1998.

[16] R. Buyya and M. Murshed, "Gridsim: A Toolkit for Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, 2002.

[17] K. Cardona, J. Secretan, M. Georgiopoulos, and G. Anagnostopoulos, "A grid based system for data mining using MapReduce," TR-2007-02, AMALTHEA, Tech. Rep., 2007.