Three Pieces of the MapReduce Workload Management Puzzle*

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

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

Vijay S. Kumar
Hewlett-Packard Labs, Palo Alto
vijay.s.kumar@hp.com

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

1 Problem Definition

MapReduce paradigm has become the compelling choice for performing advanced analytics over unstructured information and enabling efficient "Big Data" processing. There is an increasing number of MapReduce applications, e.g., personalized advertising, sentiment analysis, spam detection, real-time event log analysis, etc., that require completion time guarantees and are deadline-driven. In an enterprise setting, users share Hadoop clusters and benefit from processing a diverse variety of applications over the same or different datasets. The existing Hadoop schedulers (Hadoop Fair Scheduler, Capacity Scheduler) do not support completion time guarantees. Given a MapReduce workload consisting of diverse jobs with deadlines, how do we schedule them in order to meet these service level objectives (SLO)?

2 Three Pieces of the Puzzle

Typical production jobs are run periodically on new data. We take advantage of this observation, and for a job that is routinely executed on a new dataset, we automatically build its job profile that reflects critical performance characteristics of the underlying application during all the execution phases: map, shuffle, sort, and reduce phases. Our profiling technique [1] does not require any modifications or instrumentation of either the application or the underlying Hadoop execution engine. All this information can be obtained from the counters at the job master during the job's execution or parsed from the logs.

Using the knowledge about the job profiles, we design a set of MapReduce performance models with complementary functionality: *i)* for a given job, we can estimate the job completion time as a function of allocated resources, and *ii)* for a given job with a specified soft deadline (job's SLO), we can estimate the amount of map and reduce slots required for completing the job within the deadline.

In this work, we introduce and analyze a set of complementary mechanisms that enhance workload management decisions for processing MapReduce jobs with deadlines. The three pieces of the MapReduce workload management puzzle are as follows:

1) *An ordering policy for the jobs in the processing queue.* For example, even if no profiling information is available about the arriving MapReduce jobs, one can utilize the job deadlines for ordering. The job ordering based on the EDF policy (*Earliest Deadline First*) was successfully used in real-time processing. The EDF job ordering might be used with a default resource allocation policy in Hadoop, where the maximum number of available map (or reduce) slots is allocated to each job at the head of the queue. The possible drawback of this scheme is that in many cases, it is impossible to preempt/reassign the already allocated resources to a newly arrived job with an earlier deadline without killing the running tasks.

2) *A mechanism for allocating a tailored number of map and reduce slots* to each job for supporting the job completion goals. If the job profiling information is available, then our resource allocation policy can be much more precise and intelligent: for each job with a specified deadline, we can estimate and allocate the minimum quota of map and reduce slots required for completing the job within the deadline. The interesting feature of this mechanism is that as the time progresses and the job deadline gets closer, the introduced mechanism can recompute the amount of resources needed by each job to meet its deadline.

3) *A mechanism for allocating and deallocating (if necessary) spare resources* in the system among the active jobs. Assume that a cluster has spare resources, i.e., unallocated map and reduce slots left after each job was assigned its minimum resource quota for meeting a given deadline. It would be beneficial to design a mechanism that allocates these spare resources among the running jobs to improve the Hadoop cluster utilization and its performance. The main challenge in designing such a mechanism is accurate decision making on how the slots in the cluster should be re-allocated or de-allocated to the newly-arrived job with an earlier deadline. The naïve, straightforward approach could de-allocate the spare resources by canceling their running tasks, and then by re-allocating these slots to the new job. However, it may lead to undesirable churn in resource allocation and unproductive usage of cluster resources.

We introduce a novel mechanism that enables a scheduler to accurately predict whether the cluster

*R. Campbell and A. Verma are supported in part by NSF CCF grants #0964471, IIS #0841765 and Air Force Research grant FA8750-11-2-0084.

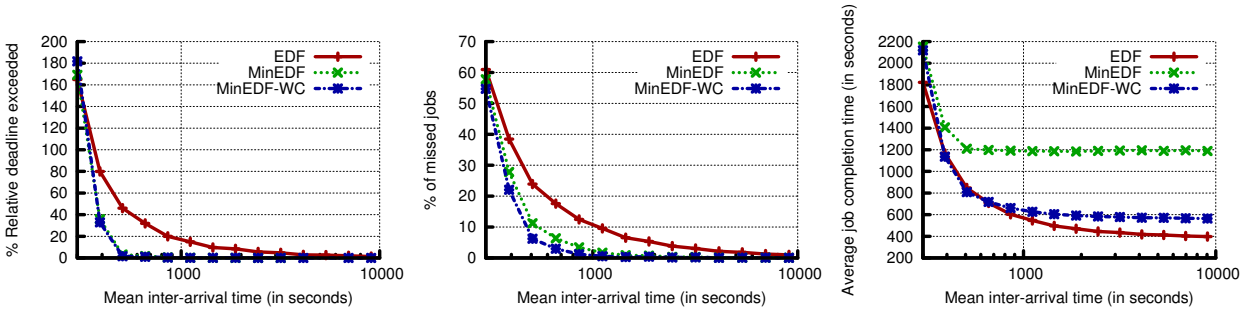

Figure 1: Comparing different metrics for 1000 job workload, averaged over 100 runs. Deadline = $[2 \cdot T_J, 4 \cdot T_J]$.

will have a sufficient amount of released resources over time for the new job to be completed within its deadline. The mechanism exploits the job profile information for making the prediction. It uses a novel modeling technique to avoid canceling the currently running tasks if possible. The mechanism deallocates the spare slots (i.e., cancels the execution of extra tasks above the minimum resource quota for each job) only when the amount of released resources over time does not guarantee a timely completion of the newly arrived job.

3 Work-in-Progress

We analyze the functionality and performance benefits of each mechanism via an extensive set of simulations over diverse workload sets. We compare performance of the three schedulers: *EDF*, *MinEDF*, and *MinEDF-WC*. Note, that the *EDF* scheduler uses only one out of three mechanisms: it applies EDF job ordering with a default resource allocation policy. The *MinEDF* scheduler uses two out of three introduced mechanisms: in addition to EDF job ordering it utilizes the mechanism for allocating a tailored amount (minimum quota) of map and reduce slots to each job for meeting its deadline. Finally, *MinEDF-WC* represents a scheduler that integrates all the three mechanisms for workload management of jobs with deadlines. The analysis presents a set of performance metrics that reflect the quality of job scheduling and slot allocation decisions provided by these different mechanisms. The solution that integrates all the three mechanisms is a clear winner in providing the most efficient support for serving MapReduce jobs with deadlines.

We analyze these mechanisms and their performance with our simulation environment SimMR [2]. We use a mix of six different applications: WordCount, Sort, Bayesian classifier, Term Frequency - Inverse Document Frequency (TF-IDF), Twitter Asymmetry and WikiTrends with three different dataset sizes. We run these 18 jobs applications with three different datasets in our 66-nodes Hadoop testbed, and 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.

Figure 1 shows the simulation results for processing 1000 jobs (consisting of multiple permutations of the 18 jobs described above) under the *EDF*, *MinEDF*, and *MinEDF-WC* schedulers and the deadlines being generated uniformly in the interval $[2 \cdot T_J, 4 \cdot T_J]$. The results are averaged over 100 runs. These simulation results reflect our scheduler's performance over a broader range of loads in the cluster. At very high loads, the cluster is under significant overload, most of the jobs miss their deadline and the performance of the three schedulers becomes very similar. At medium loads (inter-arrival times $\geq$ 300s), we see significantly better results under *MinEDF* and *MinEDF-WC* schedulers as compared to *EDF* with respect to two major metrics: lower relative deadline exceeded and very low percentages of jobs with missed deadlines. The *MinEDF-WC* scheduler is able to support the smallest fraction of jobs missing their deadlines: it accurately tailors required resources per job and intelligently allocates and de-allocates spare resources in the cluster to significantly improve the quality of resource allocation decisions and to maximize the number of jobs meeting their deadlines.

The impact of this mechanism can be also observed in Figure 1 that reflects the most improved average completion time under the work-conserving nature of *MinEDF-WC* scheduler. Since *MinEDF* does not utilize spare resources and only allocates the minimum resource quota required for job completion within the deadline, the average job completion time does not improve under *MinEDF* even as the load decreases.

We observe similar simulation results with a workload trace from Facebook. This leads us to believe in the generality of presented conclusions. We validated these results through experiments on our 66-node Hadoop cluster. We observe that metrics from testbed measurements and metrics from simulations are within 10% of each other.

References

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

[2] ——, "Play It Again, SimMR!" in *Proc. of Intl. IEEE Cluster 2011*. Austin, TX, USA, 2011.