Optimizing Speculative Execution of Deadline-Sensitive Jobs in Cloud

Maotong Xu, Sultan Alamro, Tian Lan, and Suresh Subramaniam
Department of Electrical and Computer Engineering
The George Washington University
{htfy8927, alamro, tlan, suresh}@gwu.edu

ABSTRACT

In this paper, we bring various speculative scheduling strategies together under a unifying optimization framework, which defines a new metric, Probability of Completion before Deadlines (PoCD), to measure the probability that MapReduce jobs meet their desired deadlines. We propose an optimization problem to jointly optimize PoCD and execution cost in different strategies. Three strategies are prototyped on Hadoop MapReduce and evaluated against two baseline strategies using experiments. A 78% net utility increase with up to 94% PoCD and 12% cost improvement is achieved.

KEYWORDS

MapReduce; Straggler; Speculative Strategy; PoCD

1 INTRODUCTION

Hadoop, built on the MapReduce programming model, has been widely employed by giant companies such as Facebook, Google, and Yahoo! for processing big data. It splits large amount of data into blocks and distributes them across machines to process the data in parallel. However, such parallel data processing framework is susceptible to heavy tails in response time, and job execution times could be adversely impacted by a few slow tasks, called stragglers. These stragglers are inevitable in the cloud environment due to resource contentions and hardware/software errors, and they could result in high latency and impact the overall performance of deadline-sensitive cloud applications.

Both proactive and reactive techniques are proposed to mitigate stragglers. Dolly [1] is a proactive cloning approach. It launches multiple attempts for each task, and the task completes when the earliest attempt finishes. LATE [2] presents a scheduling algorithm to check if the task is a straggler, and launch speculative attempts for each straggler. However, no existing work provides any guarantee to meet application deadlines.

Meeting desired deadlines is crucial since cloud applications are becoming increasingly deadline-sensitive. In this paper, we present an optimization framework for three straggler mitigation strategies– Clone, Speculative-Restart, and Speculative-Resume. For each MapReduce job, the optimization finds the optimal number r of speculative/clone attempts for each strategy to exploit the PoCD and cost tradeoff. The optimization framework unifies all three strategies and maximizes a *net utility* that is defined as a utility function of r. The utility function ($U(r)$) consists of PoCD ($R(r)$), i.e., the probability of meeting job deadlines, and total execution cost, which is measured by the total expected (virtual) machine time ($E(T)$). More specifically,

$$U(r) = \log(R(r) - R_{\min}) - \theta \cdot C \cdot E(T), \tag{1}$$

where $R_{\min}$ is a minimum required PoCD, and T is the execution time of a job. θ is the tradeoff between the PoCD and cost, and C is the usage-based VM price per unit time. To the best of our knowledge, this is the first work to provide a systematic study of various scheduling strategies and to offer an analytical framework for joint optimization of the probability of meeting deadlines and the associated execution cost.

We evaluate Clone, Speculative-Restart, and Speculative-Resume strategies and compare them against two baseline strategies including default Hadoop without speculation (Hadoop-NS), and default Hadoop with speculation (Hadoop-S), in our cloud testbed consisting of 80 nodes. Using two classic benchmarks, WordCount and Sort, we show that Speculative-Resume outperforms the baseline algorithms by an average of 78% in net utility improvement, which results from a significant PoCD increase (up to 94%) and/or cost reduction (up to 12%), while the improvement is even higher for more stringent application deadlines.

2 BACKGROUND AND SYSTEM MODEL

Suppose M jobs are submitted to a datacenter, where job i is associated with a deadline D_i and consists of N_i tasks for $i = 1, 2, \ldots, M$. Job i meets the deadline if all N_i tasks are processed before D_i. A task whose execution time exceeds D_i is considered as a straggler. To mitigate stragglers, we launch r_i extra attempts of each task along with an original attempt, and a task is completed as long as one of the attempts is successfully executed. We denote the (random) execution time of attempt k of job i's task j as $T_{i,j,k}$. Thus, we define job i's completion time T_i and task completion time $T_{i,j}$ by:

$$T_i = \max_{j=1,\ldots,N_i} T_{i,j}, \text{ where } T_{i,j} = \min_{k=1,\ldots,r_i+1} T_{i,j,k}, \ \forall j. \tag{2}$$

[3–5] model the execution times of tasks by using the Pareto distribution. Following these papers, we assume the execution time $T_{i,j,k}$ of each attempt follows a Pareto distribution with parameters $t_{\min}$, and β, where $t_{\min}$ is the minimum execution time and β is the exponent, and the execution times of different attempts are independent. We use progress score, i.e., the percentage of workload processed at a given time t, to determine if extra attempts are needed. More specifically, the estimated execution time equals the sum of the amount of time to launch the task and the amount of

SIGMETRICS '17, June 05-09, 2017, Urbana- Champaign, IL, USA

DOI: http://dx.doi.org/10.1145/3078505.3078541

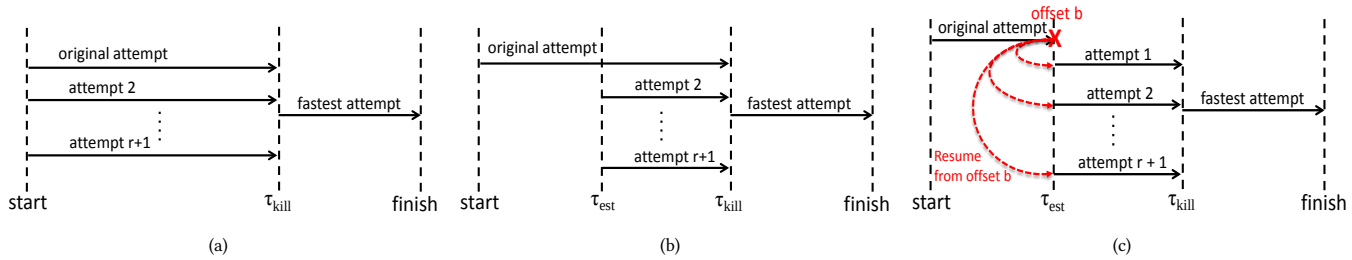

Figure 1: (a) Clone Strategy, (b) Speculative-Restart Strategy, (c) Speculative-Resume Strategy.

time used for processing data divided by the progress score. In the following, we describe the details of the three strategies.

Clone Strategy. Under this strategy, $r + 1$ attempts of each task are launched at the beginning, including one original attempt and r extra attempts. At time τ_{kill}, the progress scores of all attempts are checked, and the attempt with the best progress score is left running, while the other r attempts are killed to save machine running time. Figure 1(a) illustrates the Clone strategy for a task.

Speculative-Restart (S-Restart) Strategy. Under this strategy, one attempt (original) of each task is launched at the beginning. The attempt completion time is estimated at time τ_{est}. If the estimated attempt completion time exceeds D, r extra attempts are launched to process data from the beginning. The progress scores of all attempts are checked at time τ_{kill}, and the attempt with the smallest estimated completion time is left running, while the other r attempts are killed. Figure 1(b) illustrates the Speculative-Restart strategy for a task whose estimated execution time exceeds D.

Speculative-Resume (S-Resume) Strategy. This strategy is similar to the Speculative-Restart strategy in its straggler detection. The difference is that at time τ_{est}, the straggler is killed and $r + 1$ attempts are launched for the straggling task. These attempts, however, do not reprocess the data that has already been processed by the original attempt; they process the data starting from the byte after the last byte processed by the original straggler task. The progress scores of all attempts are checked at time τ_{kill}, and the attempt with the smallest estimated completion time is left running while the other r attempts are killed. Figure 1(c) illustrates the Speculative-Resume strategy for a task whose estimated execution time exceeds D. The b denotes the byte offset from which extra attempts start processing.

3 EVALUATION

We compare Hadoop-NS, Hadoop-S, Clone, S-Restart, and S-Resume with respect to PoCD, cost, and net utility. In each of the three strategies, the optimal number, r_{opt}, of clone/speculative attempts is found by solving our proposed net utility optimization. We execute 100 MapReduce jobs, where each job consists of 10 tasks, on our testbed consisting of 80 nodes, where each node has 8 vCPUs and 2GB memory. The physical servers are connected to a GigE switch and the link bandwidth is 1Gbps. We evaluate the strategies by using the Map phases of two classic benchmarks, WordCount and Sort. WordCount is a CPU-bound application and Sort is an I/O-bound application. We download 1.2GB workload for WordCount from Wikipedia, and generate 1.2GB workload for Sort by using the RandomWriter application. We measure the PoCD by calculating the percentage of jobs that completed before their deadlines and the cost by the average job running time (i.e., VM time required), assuming a fixed price per unit VM time that is obtained from Amazon EC2 average spot price ($C = 0.009\$/hr$). In all experiments, we set $\theta \cdot C = 0.0001$ and solve the corresponding net utility optimization. The deadlines are set to 200 sec and 270 sec for Sort and WordCount, respectively. For our three strategies, τ_{est} and τ_{kill} equal 60 sec and 120 sec, respectively. For net utility, since we use the PoCD of Hadoop-NS as R_{min}, its utility is negative infinity.

Table 1: Comparison of different strategies.

	Metrics	H-NS	H-S	Clone	S-Restart	S-Resume
Sort	PoCD	0.50	0.78	0.97	0.96	0.94
	Cost(e-3, $)	2.15	2.39	3.63	3.15	2.75
	Utility	$-\infty$	-0.65	-0.47	-0.46	-0.45
WC	PoCD	0.46	0.64	0.81	0.79	0.85
	Cost(e-3, $)	3.56	4.10	5.67	4.64	3.60
	Utility	$-\infty$	-0.90	-0.32	-0.29	-0.20

Table 1 summarizes the corresponding PoCD and cost in the optimal solutions, and compares the performance of the five strategies in terms of the overall net utility. Results show that our three strategies outperform Hadoop-NS and Hadoop-S by up to 78% on net utility value. In particular, the three strategies can improve PoCD by up to 94% and 33% over Hadoop-NS and Hadoop-S, respectively, while S-Resume introduces little additional cost compared with Hadoop-NS and Hadoop-S. This significant improvement comes from not only launching multiple attempts for stragglers, but also maintaining only the fastest attempt at τ_{kill}, thereby introducing limited execution time overhead.

4 ACKNOWLEDGEMENT

This work was supported in part by NSF grant 1320226.

REFERENCES

[1] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013, pp. 185–198.

[2] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments." in *OSDI*, vol. 8, no. 4, 2008, p. 7.

[3] G. Ananthanarayanan, M. C.-C. Hung, X. Ren, I. Stoica, A. Wierman, and M. Yu, "Grass: trimming stragglers in approximation analytics," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 289–302.

[4] D. Wang, G. Joshi, and G. Wornell, "Using straggler replication to reduce latency in large-scale parallel computing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 3, pp. 7–11, 2015.

[5] X. Ren, G. Ananthanarayanan, A. Wierman, and M. Yu, "Hopper: Decentralized speculation-aware cluster scheduling at scale," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 379–392, 2015.