IJETST- Vol.||04||Issue||09||Pages 5891-5897||September||ISSN 2348-9480 2017

Open access Journal International Journal of Emerging Trends in Science and Technology
IC Value: 76.89 (Index Copernicus) Impact Factor: 4.219 DOI: https://dx.doi.org/10.18535/ijetst/v4i9.06

Big Data Cluster Processing Through Optimized Speculative Execution

Authors
D. Sasi Redkha[1], T.N. Ranganadham[2], Dr. M. Rudra Kumar[3]
[1]M.Tech., Dept of CSE,Annamacharya Institute Of Technology & Sciences, Rajampet, Kadapa.
[2]Assistant Professor, Dept of CSE,Annamacharya Institute Of Technology & Sciences, Rajampet, Kadapa.
[3]Professor, Dept of CSE, Annamacharya Institute Of Technology & Sciences, Rajampet, Kadapa.

Abstract A big parallel processing job can be delayed substantially as long as one of its many tasks is being assigned to an unreliable or congested machine. To tackle this so-called straggler problem, most parallel processing frameworks such as MapReduce have adopted various strategies under which the system may speculatively launch additional copies of the same task if its progress is abnormally slow when extra idling resource is available. In this paper, we focus on the design of speculative execution schemes for parallel processing clusters from an optimization perspective under different loading conditions. For the lightly loaded case, we analyze and propose one cloning scheme, namely, the Smart Cloning Algorithm (SCA) which is based on maximizing the overall system utility. We also derive the workload threshold under which SCA should be used for speculative execution. For the heavily loaded case, we propose the Enhanced Speculative Execution (ESE) algorithm which is an extension of the Microsoft Mantri scheme to mitigate stragglers. Our simulation results show SCA reduces the total job flowtime, i.e., the job delay/ response time by nearly 6% comparing to the speculative execution strategy of Microsoft Mantri. In addition, we show that the ESE Algorithm outperforms the Mantri baseline scheme by 71% in terms of the job flowtime while consuming the same amount of computation resource.
Keywords:Job scheduling, speculative execution, cloning, straggler detection, optimization.

1. Introduction

EMPIRICAL performance studies of large-scale computing clusters have indicated that the completion time of a job [7] is often significantly and unnecessarily prolonged by one or a few so-called “stragglers” or straggling tasks, i.e., tasks which are unfortunately assigned to either a failing or overloaded server within a cluster of hundreds of thousands of commodity servers. To mitigate stragglers, recent big data frameworks such as the MapReduce system or its variants have adopted various preventive or reactive speculation strategies under which the system launches extra (backup) copies of a task on alternative machines in a judicious manner. In particular, there exist two main classes of speculative execution strategies, namely, the Cloning approach [5] and the Straggler-Detection-based one [6], [7], [13], [16], [20], [32], [35], [41]. Under the Cloning approach, extra copies of a task are scheduled in parallel with the initial task as long as the computation cost of the task is expected to be low and the system resource is available. For the Straggler-Detectionbased approach, the progress of each task is monitored by the system and backup copies are launched only when a straggler is detected.

As one may expect, the cloning-based strategy is only suitable for a lightly loaded cluster as it launches the clones in a greedy, indiscriminately fashion. On the other hand, the straggler-detection based strategy is applicable to both the lightly-loaded and heavily-loaded regimes but at the expense of extra system instrumentation and performance overhead as discussed in [10]. The situation is particularly challenging when the progress of a large number of tasks have to be tracked. However, previous works do not compare the performance between these two different speculation approaches. Furthermore, most of the existing speculative execution schemes are based on simple heuristics and do not consider the optimization based on specific performance objectives. With the aforementioned observations in mind, in this paper, we take a more systematic, optimization-based approach for the design and analysis of speculative execution schemes. Our objective is to optimize two performance metrics which are the total job delay/ response time (which is also referred as job flowtime) and the computation cost by defining a utility function. The optimizations are conducted by coordinating speculating with job scheduling, which is an opportunity to gain significant performance improvement compared to speculation-only policies. We also characterize the differences between the Cloning approach and the Straggler-Detection based speculative execution scheme through both theoretical analysis and extensive simulations.

II Literature Survey

Job scheduling in a MapReduce-like cluster

In a big data processing cluster like MapReduce and its variants or derivatives, different applications/ jobs need to share and compete for resources in the cluster. Thus, job scheduling plays a very important role. Throughout the whole paper, we only consider the centralized scheduling paradigm under which a global scheduler of the cluster manages all jobs where each job may consist of many small tasks. The scheduler allocates resources across jobs and also

handles straggling tasks. Widely deployed schedulers to-date include the fair scheduler [4] and the capacity scheduler [3]. However, the main goal of these schedulers is to provide fair and efficient resource sharing among different organizations. As such, other key performance metrics such as the job response time have not received adequate considerations under their designs. To enhance system performance, the design of job schedulers for MapReduce-like systems has been an active research area lately [11], [12], [23], [25], [36], [40], [42]. In particular, several works focus on deriving performance bounds for minimizing the total job completion time [11], [12], [40]. Tan et al. design the Coupling scheduler [36], which mitigates the starvation problem caused by reduce tasks in large jobs. It is well known in scheduling literature that the SRPT (Shortest Remaining Processing Time) scheduler is optimal for the overall flowtime on a single machine where there is one task per job. As such, some works extend the SRPT scheduler to minimize the total job flowtime under different settings [23], [25], [40], [42]. However, all of these studies assume accurate knowledge of task durations and hence do not support speculative copies to be scheduled dynamically.

Speculative Execution Policies

Several speculative execution strategies have been proposed for MapReduce-like systems. The initial Google MapReduce system only begins to launch backup tasks when a job is close to completion. It has been shown that speculative execution can decrease the job service time by nearly 44% [16]. This scheme is easy to implement but it would unnecessarily launch backup copies for tasks of normal progress. The speculative execution strategies in the initial versions of Hadoop [2] and Microsoft Dryad [20] closely follow that of the Google MapReduce system. However, Zaharia et al. present a new strategy called LATE (Longest Approximate Time to End) in [41] for the Hadoop-0.21 implementation. It monitors the progress rate of each task and estimates their remaining time to completion. Tasks with progress rate below certain threshold are chosen as backup candidates and the one with the longest remaining time is given the highest priority. The system also imposes a limit on the maximum number of backup tasks in the cluster. In contrast, Microsoft Mantri [7] proposes a new speculative execution strategy for Dryad in which the system estimates the remaining time to finish (i.e., trem), for each task and predicts the required service time of a relaunched copy of the task (i.e., tnew). Once a server becomes available, the Mantri system makes a decision on whether to launch a backup task based on the statistics of trem and tnew. Mantri would schedule a duplicate if the total computation cost is expected to decrease while it does not explore the tradeoffs between the job completion time (flowtime) and the computation cost. To accurately and promptly identify stragglers, Chen et al. propose a Smart Speculative Execution strategy in [13] and Sun et al. present an Enhanced Self-Adaptive MapReduce Scheduling Algorithm in [35]. The main ideas of [13] include: i) use the exponentially weighted moving average to predict the process speed and compute the remaining time of a task and ii) determine which task to backup based on the load of a cluster using a cost-benefit model. The limitation is that those works only focus on the optimization of task level rather than job level performance. Ananthanarayanan et al. proposes to mitigate the straggler problem by cloning every small job and avoid the extra delay caused by the straggler monitoring/ detection process [5]. When most of the jobs in the system are small, the cloned copies only consume a small amount of additional resources. As an extension from [5], Ananthanarayanan further presents GRASS [6], which carefully adopts the Detection-based approach to trim stragglers for approximation jobs. GRASS also provides a unified solution for normal jobs. Recently, Ren et al. propose Hopper [32], a speculation aware scheduler, which coordinates job scheduling with speculative execution. In Hopper, the scheduler allocates computing slots based on the virtual job size, which is larger than the actual size, and can immediately schedule a speculative copy once a straggler is detected. For most of the speculative execution schemes presented above, the speculation algorithms are designed independently of job scheduling. Hopper and the recently proposed SRPTMS+C [39] are the only exceptions. However, Hopper still has several downsides that can degrade the cluster performance. Firstly, Hopper is non-work-conserving: it is possible for its scheduler to keep a computing slot idle as a reservation for a future straggler while other jobs/ tasks already queue up for computation resource 1. Secondly, the job size is computed/ estimated based on only the number of tasks instead of taking the product with the task service time (i.e. the time between the task is launched and the task is finished). In practice, the task service times have shown to be varying widely even among tasks of the same job. (e.g., a Map task vs. a Reduce task). As a comparison, in our work, we incorporate the task service time when estimating the job size. Moreover, SRPTMS+C is limited to investigate the cloning approach only whereas the work in this paper combines job scheduling with speculative execution and judiciously applies proactive cloning or reactive speculation under different operating regimes. Another body of work related to this paper investigate a study on the effectiveness of scheduling redundant copies from a queuing perspective. In particular, Vulimiri et al. characterize when a global redundancy policy improves latency performance of the whole system [38]. However, this work does not consider killing the unfinished copies of the same task. Chen et al. adopts the approach of redundant requests in storage codes and theoretically analyzes its optimality when the service time of each request is exponentially distributed [14]. Based on these works, Qiu et al. adopt the MAP model to represent task arrivals and study the distribution of task-response time when redundancy is applied [27]–[30]. Moreover, Kristen et al. present in [17] an exact analysis of systems with redundancy when the service time for the redundant class follows an exponential distribution. One fundamental limitation of [14], [17], [27]–[30] is that they do not theoretically characterize the efficiency of redundancy when the task service time follows a more general distribution. Besides exponential distribution, [21] and [34] also analyze how different redundancy strategy can influence the latency and the computation cost when the job service time follows a heavy-everywhere or light everywhere distribution. However, their derived results do not hold when the service time follows other hevay-tailed distributions (e.g., the Pareto Distribution) and thus cannot be applied to our work.

III. Proposed Work

In this paper we propose the design of speculative execution schemes for parallel processing clusters from an optimization perspective under different loading conditions. For the lightly loaded case, we analyze and propose one cloning scheme, namely, the Smart Cloning Algorithm (SCA) which is based on maximizing the overall system utility. We also derive the workload threshold under which SCA should be used for speculative execution. For the heavily loaded case, we propose the Enhanced Speculative Execution (ESE) algorithm which is an extension of the Microsoft Mantri scheme to mitigate stragglers. Our simulation results show SCA reduces the total job flowtime, i.e., the job delay/ response time by nearly 6% comparing to the speculative

execution strategy of Microsoft Mantri. In addition, we show that the ESE Algorithm outperforms the Mantri baseline scheme by 71% in terms of the job flowtime while consuming the same amount of computation resource.

Ⅳ Methodology

Smart Cloning Algorithm (SCA):-

The SCA algorithm consists of two separate parts. At the beginning of each time slot, we first schedule the remaining tasks of unfinished jobs and then check whether the computation resource is available. If it is available we will determine number of clones for each task. Otherwise, we will clone each task exactly once and sort the set of unscheduled jobs, according to the increasing order of the workload.

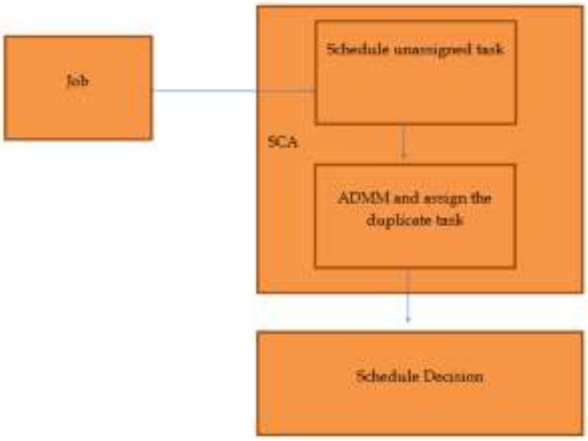

Fig: heavily loaded cluster

Design details of ESE:-

Our ESE Algorithm includes three scheduling levels. At the beginning of time slot l, the scheduler estimates the remaining time of each running task and puts the tasks whose remaining time satisfies the constraint of P3 in the backup candidate set.

The scheduler then schedules the remaining tasks of the jobs which have already been scheduled but have not left the cluster yet this are set of unfinished jobs at time slot l and the jobs are sorted based on remaining workloads. Upon scheduling, the jobs which have smaller remaining workload are given the higher priorities. The number of available machines, i.e., N(l) is updated after the aforementioned scheduling and the scheduler proceeds to allocate machines to these unscheduled jobs. To be specific, denote all the jobs that have not been scheduled yet where the jobs are sorted based on their non-decreasing order of workloads. The scheduler launches one copy for each task if there are available machines.

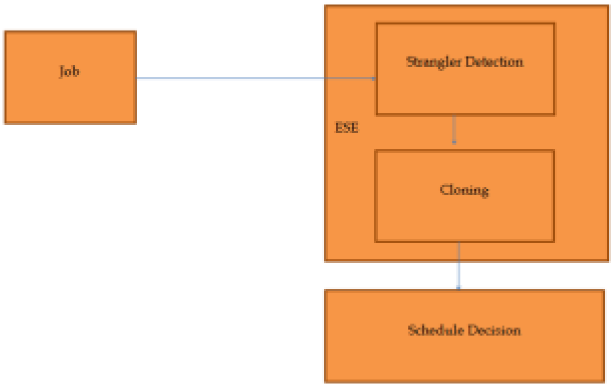

Fig: lightly loaded cluster

V. Conclusion

In our proposed work attempt to combine job scheduling and speculative execution for the design of redundancy algorithms in big data processing clusters. More importantly, we focus on two key performance metrics which are the average job flowtime and the overall system computation costs. By utilizing the distribution information of the task service time, we build an optimization framework to maximize the overall system utility. We then design two approximation algorithms to tackle this optimization problem, i.e., the SCA Algorithm and ESE Algorithm, corresponding to the cloning-based and detection-based approaches respectively. To differentiate the applicability of these two algorithms, we also categorize the cluster into the lightly loaded and heavily loaded cases and derive the cutoff threshold for these two operating regimes.

Future Work:

As future work, we will design speculative execution schemes for more complex jobs which can have additional task-dependency constraints. In addition, we plan to characterize the theoretical performance bounds of our proposed redundancy algorithms.

Reference

1. Residual lives, hazard rates, and long tails.
2. Apache. http://hadoop.apache.org, 2013.
3. Capacity Scheduler. http://hadoop.apache.org/ docs/r1.2.1/capacity scheduler.html, 2013.
4. Fair Scheduler. http://hadoop.apache.org/docs/r1.2.1/fair scheduler.html, 2013.
5. G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Effective straggler mitigation: Attack of the clones. In NSDI, April 2013.
6. G. Ananthanarayanan, M. C.-C. Hung, X. Ren, and I. Stoica. Grass: Trimming stragglers in approximation analytics. In NSDI, April 2014.
7. G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoic, Y. Lu, B. Saha, and E. Harris. Reining in the outliers in MapReduce clusters using mantri. In USENIX OSDI, Vancouver, Canada, October 2010.
8. D. P. Bertsekas. Nonlinear Programming: 2nd Edition. Athena Scientific, 1999.
9. [9] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends® in Machine Learning, 4:1–122, January 2011.
10. D. Breitgand, R. Cohen, A. Nahir, and D. Raz. On cost-aware monitoring for self-adaptive load sharing. IEEE JSAC, 28(1):70–83, January 2010.
11. H. Chang, M. Kodialam, R. R. Kompella, T. V. Lakshman, M. Lee, and S. Mukherjee. Scheduling in MapReduce-like systems for fast completion time. In Proceedings of IEEE Infocom, pages 3074–3082, March 2011.
12. F. Chen, M. Kodialam, and T. Lakshman. Joint scheduling of processing and shuffle phases in MapReduce systems. In Proceedings of IEEE Infocom, March 2012.
13. Q. Chen, C. Liu, and Z. Xiao. Improving MapReduce performance using smart speculative execution strategy. IEEE Transactions on Computers, 63(4), April 2014.
14. S. Chen, Y. Sun, U. C. Kozat, L. Huang, P. Sinha, G. Liang, X. Liu, and N. B. Shroff. When queueing meets coding: Optimal-latency data retrieving scheme in storage clouds. In Infocom, April 2014.
15. R. B. Cooper. Introduction to queuing theory. The Macmillan Company, New York, 1972.
16. J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In OSDI, pages 137–150, December 2004.
17. K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, and E. H. Aalto. Reducing latency via redundant requests: Exact analysis. pages 347–360. ACM SIGMETRICS, June 2015.
18. M. Harchol-Balter. Performance Modeling and Design of Computer Systems: Queueing Theory in Action. Cambridge University Press, 2013.
19. M. Hong and Z.-Q. Luo. On the linear convergence of the alternating direction

method of multipliers. In arXiv:1208.3922, August 2012.