Efficient Task Replication for Fast Response Times in Parallel Computation

Da Wang
EECS Dept., MIT
Cambridge, MA, 02139, USA
dawang@mit.edu

Gauri Joshi
EECS Dept., MIT
Cambridge, MA, 02139, USA
gauri@mit.edu

Gregory Wornell
EECS Dept., MIT
Cambridge, MA, 02139, USA
gww@mit.edu

ABSTRACT

Large-scale distributed computing systems divide a job into many independent tasks and run them in parallel on different machines. A challenge in such parallel computing is that the time taken by a machine to execute a task is inherently variable, and thus the slowest machine becomes the bottleneck in the completion of the job. One way to combat the variability in machine response is to replicate tasks on multiple machines and waiting for the machine that finishes first. While task replication reduces response time, it generally increases resource usage. In this work, we propose a theoretical framework to analyze the trade-off between response time and resource usage. Given an execution time distribution for machines, our analysis gives insights on when and how replication helps. We also propose efficient scheduling algorithms for large-scale distributed computing systems.

1. INTRODUCTION

One of the typical scenarios in cloud computing is large scale computation in a data centers with a large number of computers, which is pioneered by companies like Google with the support from distributed computing frameworks such as MapReduce [5]. An important category of large scale computation in data center is called "embarrassingly parallel" computation, where the computation can be easily separated into a number of parallel tasks, that are executed on separate machines.

For an embarrassingly parallel job, the completion time is determined by the slowest computing node, as one needs to wait for all parallel tasks to finish. However, machine response time in data centers are inherently variable due to factors such as co-hosting, virtualization, network congestion, etc. As the computing scale increases, waiting for the slowest machine to finish its assigned task delays the job completion significantly. For example, [4, Table 1] shows that while the 99%-percentile finishing time for each task is 10ms, the 99%-percentile finishing time for the slowest task in a large computation job could take up to 140ms.

SIGMETRICS'14, June 16–20, 2014, Austin, Texas, USA.
ACM 978-1-4503-2789-3/14/06.
http://dx.doi.org/10.1145/2591971.2592042 .

One of the techniques used by system designers to combat variability in machine response time is *task replication*, i.e., sending the same task to multiple machines and taking the result from whichever finishes first. The idea of replicating tasks was recognized by system designers for parallel commutating [3, 6], and first adopted in cloud computing via the "backup tasks" in MapReduce [5]. A line of system work [1, 2, 10, 12] further develop this idea to handle various performance variability issues in data centers.

While this approach of replicating tasks reduces task completion time, it may cause additional resource usage in terms of machine running time. In this paper, we introduce a stylized yet realistic system model to analyze this trade-off between completion time and resource usage. Our analysis reveals when and how task replication works, and provides insights into the design of scheduling algorithms in practical distributed computing systems.

To the best of our knowledge, this is the first theoretical analysis of task replication in distributed computing. A few theoretical works [7–9] have investigated the use of replication or redundancy to reduce latency, but in the context of content download from distributed storage.

2. SYSTEM MODEL

We consider the problem of executing a set of n *embarrassingly parallel tasks* in a data center. Due to randomness in execution time, replicating a task at multiple machines can significantly improve its completion time.

A *scheduling policy* π assigns the starting times of each task at different machines,

$$\pi \triangleq \left[(t_{i,j}), i \in \{1, 2, \ldots, n\}, t_{i,j} \in \mathbb{R}_+, j \in \mathbb{Z}^+\right],$$

where i is the task of interest, and $t_{i,j}$ is the start time for the j-th replica of task i.

We consider that the scheduler receives an instantaneous notification when any machine finishes its assigned task. When the earliest replica of task i finishes, the scheduler terminates all the replicas of task i. We assume that the scheduler cannot preferentially terminate some replicas while keeping others running. Further, we consider only static policies, where all the starting times $t_{i,j}$ are chosen at time 0 and not changed afterwards. Although there is loss of generality in static launching, it may be of interest in practice because it allows more time for resource provisioning.

Let $X_{i,j}$ be the running time of the j-th replica of task i. We assume $X_{i,j}$ are i.i.d. discrete random variables, with

probability mass function (PMF),

$$\text{or, } P_X(\alpha_i) = \Pr(X = \alpha_i) = p_i, \quad 1 \le i \le l, \tag{1}$$

where $p_i \in [0,1]$ and $\sum_{i=1}^{l} p_i = 1$.

We can use log files or traces to estimate the execution time distribution, for example, by finding a histogram of the execution times with certain bin size (e.g., 10 seconds).

2.1 Performance metrics

We evaluate the performance of a scheduling policy π by the following two performance metrics:

- completion time $T(\pi)$: the time when at least one replica of each of the tasks finishes;

$$T(\pi) \triangleq \max_i \min_j (t_{i,j} + X_{i,j}) \tag{2}$$

- machine time $C(\pi)$: sum of the running times for all machines, normalized by the number of tasks n;

$$C(\pi) \triangleq \frac{1}{n} \sum_{i=1}^{n} \sum_j |T_i(\pi) - t_{i,j}|^+, \tag{3}$$

where $|x|^+ = \max\{0, x\}$.

The machine time $C(\pi)$ can be viewed as the cost of using a cloud service, such as Amazon Web Service (AWS), which charges user per hour of machine usage. Intuitively, task replication reduces T, but may increase C.

We investigate the trade-off between $T(\pi)$ and $C(\pi)$ using the cost function

$$J_\lambda(\pi) = \lambda \mathbb{E}[T(\pi)] + (1-\lambda)\mathbb{E}[C(\pi)], \tag{4}$$

where $0 \le \lambda \le 1$ reflects the relative importance of reducing $\mathbb{E}[T(\pi)]$. Given λ, the *optimal* scheduling policy π^* is

$$\pi^* = \arg\min_\pi J_\lambda(\pi).$$

3. OUR CONTRIBUTIONS

Our analysis shows that:

1. While in general there is a trade-off between expected completion time $\mathbb{E}[T(\pi)]$ and expected machine time $\mathbb{E}[C(\pi)]$, there exist scenarios where task replication simultaneously reduces $\mathbb{E}[T(\pi)]$ and $\mathbb{E}[C(\pi)]$.
2. Given the execution time distribution P_X, the optimal scheduling policy that minimizes the cost function J_λ has starting times $t_{i,j}$ that are integer combinations of $\alpha_1, \alpha_2, \ldots \alpha_l$, the possible values of the execution time. Thus we reduce the search space for the optimal policy to a discrete and finite set of policies.
3. When the machine execution time follows a bimodal distribution, we find the optimal single-task scheduling policy for the special case of two machines. In the optimal policy we start the task at one machine at time 0. The other machine is started at time α_1, or not started at all, depending on λ in (4), and the system parameters.
4. We propose a heuristic algorithm to choose the scheduling policy for both single-task and multi-task cases. The algorithm adds new machines one-by-one by looking ahead at the next k possible starting times, and chooses from among them the starting time that minimizes the cost function J_λ.
5. When scheduling multiple tasks, it is useful to take the interaction of completion times among different tasks into account, i.e., scheduling each task independently can be strictly suboptimal.

The detailed analysis and results are in [11], the extended version of this paper. A future research direction is to develop a strategy that can learn the execution time distribution while simultaneously scheduling jobs. It would also be useful to consider how queueing of requests impacts system performance.

Acknowledgement

We thank Devavrat Shah for helpful discussions.

4. REFERENCES

[1] Ananthanarayanan, G., Ghodsi, A., Shenker, S., and Stoica, I. Effective straggler mitigation: Attack of the clones. NSDI'13, USENIX, pp. 185–198.

[2] Ananthanarayanan, G., Kandula, S., Greenberg, A., Stoica, I., Lu, Y., Saha, B., and Harris, E. Reining in the outliers in map-reduce clusters using mantri. OSDI'10, USENIX, pp. 1–16.

[3] Cirne, W., Brasileiro, F., Paranhos, D., Luís Fabrício W. Góes, and Voorsluys, W. On the efficacy, efficiency and emergent behavior of task replication in large distributed systems. *Parallel Computing 33*, 3 (2007), 213–234.

[4] Dean, J., and Barroso, L. A. The tail at scale. *Commun. ACM 56*, 2 (2013), 74–80.

[5] Dean, J., and Ghemawat, S. MapReduce: simplified data processing on large clusters. *Commun. ACM 51*, 1 (2008), 107–113.

[6] Ghare, G. D., and Leutenegger, S. T. Improving speedup and response times by replicating parallel programs on a SNOW. In *Job Scheduling Strategies for Parallel Processing*. Springer Berlin Heidelberg, Jan. 2005, pp. 264–287.

[7] Joshi, G., Liu, Y., and Soljanin, E. Coding for fast content download. In *Proc. Annu. Allerton Conf. on Commu. Control. & Comput.* (Oct. 2012), pp. 326–333.

[8] Joshi, G., Liu, Y., and Soljanin, E. On the Delay-Storage trade-off in content download from coded distributed storage systems. *arXiv:1305.3945 [cs, math]* (May 2013).

[9] Shah, N. B., Lee, K., and Ramachandran, K. When do redundant requests reduce latency? *arXiv:1311.2851 [cs]* (Nov. 2013).

[10] Vulimiri, A., Godfrey, P. B., Mittal, R., Sherry, J., Ratnasamy, S., and Shenker, S. Low latency via redundancy. *arXiv:1306.3707 [cs]* (June 2013).

[11] Wang, D., Joshi, G., and Wornell, G. Efficient job replication for fast response times in parallel computation. *arXiv:1404.1328 [cs.DC]* (Apr. 2014).

[12] Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R., and Stoica, I. Improving MapReduce performance in heterogeneous environments. OSDI'08, USENIX, pp. 29–42.