Reducing Late-Timing Failure at Scale: Straggler Root-Cause Analysis in Cloud Datacenters

Xue Ouyang[1], Peter Garraghan[1], Renyu Yang[2], Paul Townend[1], Jie Xu[1]

School of Computing[1],
University of Leeds, Leeds, UK
{scxo, p.m.garraghan, p.m.townend, j.xu}@leeds.ac.uk

School of Computing[2],
Beihang University, Beijing, China
yangry@act.buaa.edu.cn

***Abstract* — Task stragglers hinder effective parallel job execution in Cloud datacenters, resulting in late-timing failures due to the violation of specified timing constraints. Straggler-tolerant methods such as speculative execution provide limited effectiveness due to (i) lack of precise straggler root-cause knowledge and (ii) straggler identification occurring too late within a job lifecycle. This paper proposes a method to ascertain underlying straggler root-causes by analyzing key parameters within large-scale distributed systems, and to determine the correlation between straggler occurrence and factors including resource contention, task concurrency, and server failures. Our preliminary study of a production Cloud datacenter indicates that the dominate straggler root-cause is resultant of high temporal resource contention. The result can assist in enhancing straggler prediction and mitigation for tolerating late-timing failures within large-scale distributed systems.**

I. INTRODUCTION

Modern Cloud datacenters are composed of thousands of servers to support increasing computing demand while fulfilling *Quality of Service (QoS)* requests. Within such large-scale systems, there exist a particular type of emergent phenomena that leads to increased likelihood of late-timing failures caused by task stragglers. In parallel computing frameworks such as MapReduce [1], a job is divided into multiple tasks, and *stragglers* are referred to those which exhibit abnormally long execution in comparison to sibling-tasks within the same job [2]. It has been shown that stragglers impose a substantive challenge towards rapid and predictable service execution, which is further aggravated within increasingly larger system scale [3].

Stragglers occur due to multiple causes, ranging from heterogeneous node capacities, network congestion, resource contention, to fault activation within tasks and servers [4]. There have been considerable efforts in academia and industry towards tolerating stragglers and their impact on efficient job execution. The dominant method is speculation that monitors slow task execution in order to create corresponding replicas under the assumption that it will complete prior to the straggler task. Such a method is widely adopted by production Cloud datacenters including Google, Yahoo, and Facebook.

There exist numerous speculation based techniques for straggler-tolerance, providing enhancement within different operational scenarios. For example, LATE [2] was designed for heterogamous clusters, Mantri [5] focuses on replication resource and opportunity cost, and Dolly [6] specifically tackle stragglers in small jobs. While these works demonstrate success for reducing job response time, their effectiveness is limited due to lack of precise knowledge pertaining to straggler root-causes. This is important when considering different straggler-tolerant techniques. For example, a specific approach such as SkewTune [7] will result in extensive replication overhead if the straggler is not caused by data skew. Furthermore, replication which occurs late within the task lifecycle will also result in increased overhead with minimal impact in reducing late-timing failures. While there are studies analyzing root causes for unsuccessful tasks within datacenters [8], there is an omission of work that analyzes straggler root-causes. Such analysis is critical to understanding the explicit relationship between straggler type, occurrence probability, and system operation as well as proposing effective straggler-tolerant techniques.

II. PROPOSED SOLUTION

A. Straggler-Tolerant System with Cause Analytics Engine

A straggler tolerant system is proposed to improve parallel job execution and to mitigate late-timing failures caused by stragglers. Figure 1. depicts how this system is integrated into existing large-scale cluster platforms such as YARN. The *scheduler* is responsible for assigning jobs into the Cloud datacenter comprising *M* servers. Each *Application Master* controls one job containing *N* tasks, with $task_j^i$ representing the i_{th} task belonging to *job j*. The *Straggler Cause Analytics Engine* communicates with both *Speculator* and *Node Manager,* collect data and conduct the analysis. By leveraging data mining techniques to correlate various system attributes with straggler occurrence, it will identify major causes and construct statistical models for straggler prediction.

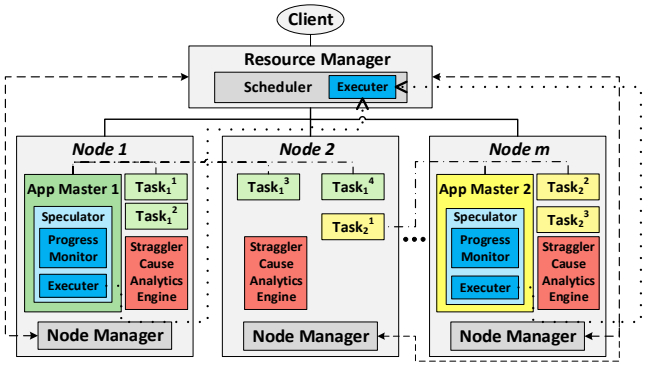

Figure 1. Straggler tolerant system integrated into YARN architecture

B. Straggler Root-Cause Analysis Method

There exist different criterions for identifying stragglers. Works such as LATE and Mantri determine stragglers based on the estimated finish time for each task as in Equation 1.

$$EFT_j^i = t_k + \frac{1 - PS(task_j^i)}{PS(task_j^i)}(t_k - t_0) \quad (1)$$

where EFT_j^i represents Estimated Finish Time for $task_j^i$, PS is the task Progress Score, t_0 is the start time of job j and t_k is the timestamp recording $PS(task_j^i)$.

To consider task execution time with input data size, we propose a new system metric for straggler detection termed *Degree of Straggler (DoS)* as shown in Equation 2.

$$DoS_j^i = \left(\frac{PS(task_j^i)}{Inp(task_j^i)}\right) \Big/ \left(\frac{\frac{1}{n}\sum_{i=1}^{n} PS(task_j^i)}{\frac{1}{n}\sum_{i=1}^{n} Inp(task_j^i)}\right) \quad (2)$$

where $Inp(task_j^i)$ is the data volume that $task_j^i$ is required to process. The *DoS-index* indicates a relative speed of data processing (i.e., the time consumed when processing one unit of input data). Using this criterion we can identify the straggler occurrence pattern, and correlate it with system attributes that are hypothesized to cause stragglers including hardware faults, high resource utilization (CPU, memory, disk), unhandled requests, network package loss and data skew, etc. Combining these runtime system features with dynamic task information, we can further design learning and prediction models such as Neural Networks (NN) to timely discriminate stragglers and the consequential late-timing failures, considering time-variant changes and instantaneous states.

C. Initial Results

The main reason leading to high resource usage includes unbalanced workload aggregation and poor user code. The former is caused by inefficient scheduling that leads to excessive workload co-allocation. The latter is caused by the inefficient design of execution logic (i.e. looping conditions). Such behavior has been identified in [9], analyzing how specific code can lead to high latency execution. For this specific cause, it is hypothesized that high resource usage of a server node (subcategorized as CPU, memory, and disk) is likely to result in high straggler occurrence due to contention.

Request handling inefficiency is mainly due to overloaded file requests, especially for MapReduce jobs with a large number of read and write requests to HDFS. Once the request number surpasses the handling capability of the master node, it will become the bottleneck, leading to a long request waiting queue. The unreasonable configuration of task number or block size can lead to request increase, thereby increasing the load of master node and causing slow request handling.

The exploration investigates how resource contention affects straggler occurrence. Figure 2. presents the initial analysis of straggler occurrence within a 20-day period from a large-scale production Cloud datacenter composed of 5000 servers and millions of tasks. We monitor and collect system information from servers which contain tasks with *DoS-Index* ≥ 10. System conditions including server CPU utilization ≥ 80%, disk usage ≥ 80%, and slow read-write request handling (i.e. latency from file system > 400ms) have been studied, and memory conditions will be included in future work.

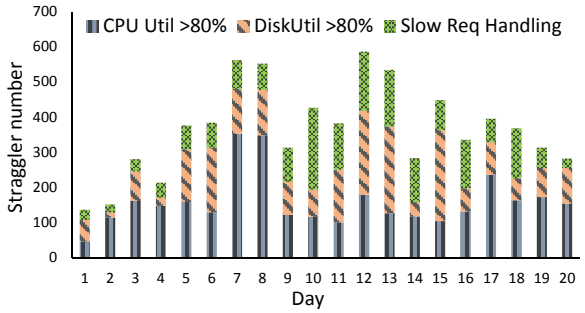

Figure 2. Straggler root-cause analysis in a production system

We observe that 59% and 42% of stragglers occur under the presence of high server CPU and disk overloading. Furthermore, it is also observable that 34.3% of stragglers occur due to slow request handling. This result indicates that high resource utilization is a dominant cause for straggler occurrence. Condition overlapping is not considered in this initial analysis (i.e. correlated and simultaneous root causes), explains why the sum of the three reasons exceeds 100%. The further refinement will be included in future work. Moreover, we also observe that additional factors such as network condition can result in stragglers as well due to all remote copies after shuffle phase in MapReduce job are sent through the network, and higher package re-transmission causes not only additional job end-to-end execution time, but also aggravated network congestion. Therefore, future work will explore such parameters in-depth.

III. Conclusion

In this paper we presented a straggler root-cause analysis method to reduce late-timing failures and study straggler occurrence. Our initial results provide new insights into straggler occurrence, however there is a requirement for more extensive analytics to determine precise relationships between temporal resource contention and stragglers. We plan to detail our method for system parameter collection as well as provide additional case studies from large-scale Cloud datacenters.

References

[1] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." In *Communications of the ACM 2008.*

[2] Zaharia, Matei, et al. "Improving MapReduce Performance in Heterogeneous Environments." *In USENIX OSDI 2008*.

[3] Dean, Jeffrey, and Luiz André Barroso. "The tail at scale." In *Communications of the ACM 2013.*

[4] Kumar, Umesh, and Jitendar Kumar. "A Comprehensive Review of Straggler Handling Algorithms for MapReduce Framework." In *IJGDC 2014.*

[5] Ananthanarayanan, Ganesh, et al. "Reining in the Outliers in Map-Reduce Clusters using Mantri." In *USENIX OSDI 2010.*

[6] Ananthanarayanan, Ganesh, et al. "Effective straggler mitigation: Attack of the clones." In *USENIX NSDI 2013.*

[7] Kwon, YongChul, et al. "Skewtune: mitigating skew in mapreduce applications." in *ACM SIGMOD 2012.*

[8] Rosa, Andrea, et al. "Understanding the Dark Side of Big Data Clusters: an Analysis beyond Failures." In *IEEE DSN 2015.*

[9] Li, Guanpeng, et al. "Fine-grained characterization of faults causing long latency crashes in programs." In *IEEE DSN 2015.*