Optimizing resources to mitigate stragglers through virtualization in run time

Ajay Kumar Bansal · Manmohan Sharma · Ashu Gupta, Students

Abstract Modern computing systems are generally enormous in scale, consisting of hundreds to thousands of heterogeneous machine nodes, to meet rising demands for Cloud services. MapReduce and other parallel computing frameworks are frequently used on such cluster architecture to offer consumers dependable and timely services. However, Cloud workloads’ complex features, such as multi-dimensional resource requirements and dynamically changing system settings, such as dynamic node performance, are posing new difficulties for providers in terms of both customer experience and system efficiency. The straggler problem occurs when a small subset of parallelized jobs takes an excessively long time to execute in contrast to their siblings, resulting in a delayed job response and the possibility of late-timing failure. Speculative execution is the state-of-the-art method to straggler mitigation. Speculative execution has been used in numerous real-world systems with a variety of implementation improvements, but the results of this thesis’ research demonstrate that it is typically wasteful. The failure rate of speculative execution might be as high as 71 percent, according to different data center production trace logs. Straggler mitigation is a difficult task in and of itself: 1) stragglers may have varying degrees of severity in parallel job execution; 2) whether a task should be considered a straggler is highly subjective, depending on various application and system conditions; 3) the efficiency of speculative execution would be improved if dynamic node quality could be adequately modeled and predicted; 4) Other sorts of stragglers, such as those generated by data skews, are beyond speculative execution’s capabilities.

1 Introduction

The datacenters generally associated with thousands and/or tens of thousands of machine nodes and other homogenous hardware such as Graphics Processing Units (GPUs), Field Programmable Gate Array (FPGA)s, and other heterogeneous hardware, modern-day IT has grown at a significant rate: with annual global IP traffic overtaking the zettabyte (1,000 exabytes) threshold in 2016 and IP traffic per capita reaching 22GB by 2019, datacenters typically. A great number of large-scale distributed linked systems capable of offering computation as a service have been created to tackle the problems posed by such extraordinary growth. Cloud computing, in which parallel and/or distributed computing methods are developed to the solving of computationally intensive applications over networks of computers, has developed as a strong paradigm to support this large-scale computing infrastructure. This notion has seen a lot of traction in recent years. However, these models confront substantial performance problems, particularly when cluster sizes rise rapidly. When one distributed subtask runs unreasonably slowly, all sister tasks in the same job must wait for that straggler to finish, for example, under the present parallel computing paradigm.

For users, a Quality of Service (QoS) violation may arise owing to job completion delays, resulting in lower customer satisfaction; for system managers, committed resources squandered while waiting for stragglers result in poor utilization and financial loss. As a re-
result, a substantial in-depth study is required to define and quantify straggler behavior, particularly in large-scale and complex systems where this issue has already resulted in significant performance loss. It is advantageous for data-driven analytics research to be inspired and refined by results from real-world systems. Trace log data from production clusters are utilized to make valuable observations on the straggler effect, straggler causes, and other straggler-related problems like current mitigation efficacy, and so on. For this study, three production datasets were provided, one of which is a Google Cloud datacenter trace log. Since November 2011, this dataset has been open to the public and is available in [5]. The normalized CPU and memory use data are gathered every 5 minutes in the second edition of this trace, which spans 30 days. The trace provides information on 25 million tasks organized into 650,000 jobs, as well as the resource usage of 12,000+ servers in operation. [6] has further details on the data structure, monitoring, and normalization procedure for this data.

1.1 Straggler and Its Impact

By submitting jobs through resource management, applications may run on large-scale computing systems like data centers and clusters (YARN, Mesos, Borg, etc). A job is made up of several smaller tasks (defined as the lowest unit of computation visible by the resource management) in this context [7]. To speed up work completion, such jobs and subsequent tasks are typically separated into stages and scheduled on various computers in a parallelized way, resulting in a Direct Acyclic Graph (DAG) [8]. Application frameworks (such as MapReduce) try to subdivide jobs so that each step takes about the same amount of time to finish [9]. This is accomplished by giving each job a subset of data (known as shards) and devoting the necessary resources to it (CPU, memory, etc). This is determined using the resource manager’s resource needs module [10]. Even with such safeguards in place, a subset of activities inside a job would show as stragglers in large-scale cloud datacenters [11,12]. A straggler is defined in this context as a task that executes unusually slowly in relation to the typical task length inside a job [13]. The term “abnormally slow” refers to any work that takes 50 percent longer to complete than the (average) task completion time for a job phase [14,15]. Slowly running tasks (stragglers) have an impact on the overall job’s performance and completion time [16], increasing resource use and application performance deterioration as the size grows [17,18], decreasing system availability and incurring additional operational expenses [12]. According to a study of large-scale production systems [18], around 4-6 percent of task stragglers have a detrimental impact on more than half of the overall jobs in the system.

1.2 Aim and objective of study

The straggler problem occurs when one or more parallel jobs perform considerably slower than their sibling sub-tasks, even though they are expected to take the same amount of time. As a result, the entire job execution time is impacted since the parallel job must wait until the final task is completed. Straggler tasks are those that are excessively sluggish. The first goal of this study is to enhance parallel task execution performance by reducing execution time by mitigating straggler behaviors, and the second goal is to conserve resources that would otherwise be spent on mitigating straggler behaviors by enhancing speculation efficiency. These goals necessitate a thorough examination of the straggler syndrome in today’s Cloud computing environments, a smart straggler identification method that always selects the most urgent and appropriate stragglers based on various operational environments and system behaviors, and an intelligent mitigation mechanism that predicts straggler occurrence and avoids assigning tasks to poorly performing. Due to the much longer latency for distant reads compared to local reads, they have a substantial influence on job performance, while others are less evident (for example, some OS-level background daemons such as garbage collection can temporarily affect machine performance). When stragglers are found in a system, it’s important to figure out what’s causing the performance deterioration so that suitable steps may be made to reduce the impact of the stragglers activities. The following are the specific aims of this study:

- Analyzing straggler-related statistics within Cloud computing systems.
- Identifying the most appropriate stragglers for mitigation.
- Avoiding straggler occurrence through modeling and predicting machine node performance.
- Developing a dedicated algorithm to deal with situations when speculative execution is not appropriate.

2 Evaluation of computing System

It is important to describe the evolution of current computing models in order to better comprehend how the Cloud computing system originates and evolves, as well as why the MapReduce framework has been popular in recent years.
2.1 Tiered Software System Architecture

Most software systems’ conceptual architecture is divided into three levels, as shown in Figure 2.1 (a): the display layer, the application logic layer, and the resource layer. These layers are also known as the business layer, network layer, or communication layer, as described in [18]–[20], but they all provide the same functions. The presentation layer is in charge of displaying data, interacting with, and connecting with components outside of the system, such as human users or other systems. A presentation layer can take several forms, such as a graphical user interface or a module that formats data into a certain syntax. The application logic layer is in charge of data acquisition and is the element that actually conducts the actions that the user has requested through the presentation layer. Regardless of the data source, the resource management layer is in charge of managing data. Data in databases, file systems, and other data repositories are all included.

2.2 Service Computing

The idea of constructing application elements into a network of services that can be generated randomly to create flexible, dynamic business processes and agile applications that span organizations and computing platforms are promoted by service cloud computing, which promotes the idea of assembling application components into a network of services that can be generated randomly to create flexible, dynamic business operations and agile applications that span organizations and computing platforms [21]–[22]. A software executed business function that is encased with a clearly specified interface [23] is how the term “service” is defined.

Owing to middleware decentralization, this design has a benefit over the N-tier architecture in that middleware complexity may be decreased, and updates are less likely to influence current services due to loose coupling. The provider makes their services available to the system, and the register keeps track of them in the form of service descriptions. The registry also allows users to find new services via the Universal Description Discovery and Integration (UDDI) framework [24], and in certain circumstances, it may help consumers with things like service selection, workload balance, and scheduling [25]. The maturity of service computing has allowed a long-sought notion to resurface: systems that provide services to users as computing utilities. The concept of a computing utility was first proposed in a speech given by John McCarthy to commemorate MIT’s centennial in 1961 [26], in which it was envisioned that networks would be highly developed, mature enough to make “computer utilities” a reality, and would work in a similar way to electrical and telephone utilities [27]. Within the last several decades, a number of distributed systems have emerged as a result of advancements in research and technology, such as peer-to-peer computing [28] and grid computing [29]. Each of them has its own set of features that enable it to achieve a specific customer goal in the form of a computing utility. Cloud computing is the most typical example among them.

2.3 Cloud Computing

The development of two main technologies, communication protocols and virtualization, has increased the practicality of Cloud computing [30]. Because computer systems may communicate across the globe through the Internet, the former allows for the establishment of potentially dispersed resource pools. The latter allows users to isolate computing resources from physical infrastructure, allowing them to add and release computing resources on demand via a virtual management system [19]. The construction of Virtual Machines (VMs), which are self-contained environments comprising encapsulated state and virtual computing resources, is a common use of virtualization technology. After Google introduced its new business model of delivering utility computing to customers, in which computer resources, development platforms, or apps are offered to users as a service, cloud computing has grown in popularity. Users and providers are the two parties involved in the supply of Cloud services. Providers are described as “entities that own and manage the underlying infrastructure to offer computer services” in the context of Cloud computing [30]. Users, who are described as “the actor responsible for originating and setting the volume of tasks to be computed” [30], are entities that demand computing capacity to fulfill business objectives. Even though the notion of cloud computing has been around for a long time, it still lacks a clear definition. Cloud computing is defined as “a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, data centers, storage, applications, and services) that can be rapidly provisioned and released with minimal administrative efficacy,” according to the National Institute of Standards and Technology (NIST) [31]. The following are the four most important characteristics that define Cloud computing as a distinct computing model [146]: (1) Pay-as-you-go: customers pay on a per-use basis for computing resources (such as storage and software). (2) Scalability: Cloud companies combine enormous volumes of data from thousands of servers.
into a single system. Users don’t have to invest anything upfront, so they may scale up or down as needed. (3) Virtualization: a crucial technique for creating a pool of resources with varying physical resources and dynamically assigning these resources and (4) Internet Centric: all public Cloud services are delivered over the Internet, allowing users to access their resources from a variety of devices. Cloud computing has a significant advantage over traditional computer system architectures because of these qualities. Typically, cloud computing systems are installed in data centers or large-scale clusters. Colocation of datacenters is popular to meet common environmental and physical security standards, as well as to simplify system maintenance [20]. The data center customers are given physical space in which to buy, install, and configure their own IT equipment, while the datacenter providers are in charge of physical security and operating ambient conditions.

2.4 Newly Develop Computing Methods

The Internet of Things (IoT) is one of the next developments (IoT). The phrase “internet of things” (IoT) was initially created in 1999 in the context of supply chain management, but it has subsequently been expanded to include a broader variety of applications such as healthcare, transportation, and so on. In the Internet of Things (IoT) concept, all of the items in our surroundings will be connected to a network in some way, with information and communication systems discreetly incorporated. Radio Frequency Identification (RFID), Wi-Fi, and sensor networks are all helping to bring this new design to life. Due to the unprecedented size of linked items, massive volumes of data will be generated, which must be stored, processed, and displayed in a seamless, efficient, and readily interpretable manner. Cloud solutions may enhance QoS for applications across many domains, and platforms like Amazon IoT illustrate the effectiveness of Cloud-centric IoT programming paradigms and resource orchestration methodologies. The pure Cloud-centric approach, however, still faces certain problems. Offloading large amounts of data to the cloud, for example, comes with related communication costs, latency difficulties, and privacy concerns, among other things. To address these issues, the edge computing paradigm has been developed, which asks for data processing at the network’s edge. Aside from edge computing, other types of computing models such as fog computing and osmotic computing, among others, are hotly debated in the community. The first focuses on mobile users, while the second aims for highly dispersed and federated environments across both edge and cloud infrastructures.

3 Conclusion

The system that efficiently tolerates speed variance and uncertainty about the number of stragglers in the system, was suggested and tested in this work. S2C2 distributes coded data to nodes and changes the computing work per node adaptively during runtime. As a result, the total execution time of many apps is considerably reduced. We exhibit a 39.3 percent decrease in execution time in the best scenario through our assessments utilizing machine learning and graph processing applications. We find that S2C2’s speed adaptive workload scheduling efficiently reduces overhead in coded computing frameworks and improves their effectiveness in real-world deployments.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., … & Zaharia, M. (2009). Above the clouds: A berkeley view of cloud computing (Vol. 17). Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.
2. Din, J. U., Asmat, H., & Guizani, M. (2019). A review of information-centric network-based internet of things: communication architectures, design issues, and research opportunities. Multimedia Tools and Applications, 78(21), 30241-30256.
3. Jouppi, N. (2016). Google supercharges machine learning tasks with TPU custom chip. Google Blog, May, 18(1).
4. Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., … & Yoon, D. H. (2017). In-datacenter performance analysis of a tensor processing unit. In Proceedings of the 44th annual international symposium on computer architecture (pp. 1-12).
5. Google Cluster Data V2. 2016. URL https://github.com/google/cluster-data. Cited on pages 4, 27, and 46.
6. Reiss, C., Wilkes, J., & Hellerstein, J. L. (2011). Google cluster-usage traces: format+ schema. Google Inc., White Paper, 1-14.
7. Gill, S. S., Garraghan, P., Stankowski, V., Casale, G., Thulasiram, R. K., Ghosh, S. K., … & Buyya, R. (2019). Holistic resource management for sustainable and reliable cloud computing: An innovative solution to global challenge. Journal of Systems and Software, 155, 104-129.
8. Lama, P., Wang, S., Zhou, X., & Cheng, D. (2018, May). Performance isolation of data-intensive scale-out applications in a multi-tenant cloud. In 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS) (pp. 85-94). IEEE.
9. Zhou, H., Li, Y., Yang, H., Jia, J., & Li, W. (2018). BigRoots: An effective approach for root-cause analysis of stragglers in big data system. IEEE Access, 6, 41966-41977.
10. Gill, S. S., & Buyya, R. (2018). A taxonomy and future directions for sustainable cloud computing: 360 degree view. ACM Computing Surveys (CSUR), 51(5), 1-33.
11. Maas, M. C. (2017). Hardware and Software Support for Managed-Language Workloads in Data Centers. University of California, Berkeley.
12. Maas, M. C. (2017). Hardware and Software Support for Managed-Language Workloads in Data Centers. University of California, Berkeley.
Optimizing resources to mitigate stragglers through virtualization in run time

13. Ouyang, X., Garraghan, P., Yang, R., Townend, P., & Xu, J. (2016, May). Reducing late-timing failure at scale: Straggler root-cause analysis in cloud datacenters. In Fast Abstracts in the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. DSN.

14. Ouyang, X., Garraghan, P., McKee, D., Townend, P., & Xu, J. (2016, March). Straggler detection in parallel computing systems through dynamic threshold calculation. In 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA) (pp. 414-421). IEEE.

15. Phan, T. D., Pallez, G., Ibrahim, S., & Raghavan, P. (2019). A new framework for evaluating straggler detection mechanisms in mapreduce. ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS), 4(3), 1-23.

16. Ananthanarayanan, G., Hung, M. C. C., Ren, X., Stoica, I., Wierman, A., & Yu, M. (2014). GRASS: Trimming stragglers in approximation analytics. In 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14) (pp. 289-302).

17. Ananthanarayanan, G., Hung, M. C. C., Ren, X., Stoica, I., Wierman, A., & Yu, M. (2014). GRASS: Trimming stragglers in approximation analytics. In 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14) (pp. 289-302).

18. Garraghan, P., Ouyang, X., Yang, R., McKee, D., & Xu, J. (2016). Straggler root-cause and impact analysis for massive-scale virtualized cloud datacenters. IEEE Transactions on Services Computing, 12(1), 91-104.

19. Chiu, D. K., Cheung, S. C., & Till, S. (2003, January). A three-layer architecture for e-contract enforcement in an e-service environment. In 36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the (pp. 10-pp). IEEE.

20. Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures. University of California, Irvine.

21. Broy, M., Renzel, K., & Schmidt, M. (1997). Software architectures and design patterns in business applications.

22. Papazoglou, M. P., Traverso, P., Dustdar, S., & Leymann, F. (2007). Service-oriented computing: State of the art and research challenges. Computer, 40(11), 38-45.

23. Papazoglou, M. P. (2003, December). Service-oriented computing: Concepts, characteristics and directions. In Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003. (pp. 3-12). IEEE.

24. OASIS, U. (2004). Version 3.0. 2 UDDI Spec Technical committee Draft.

25. Mezni, H., Chainbi, W., & Ghedira, K. (2011, December). An autonomic registry-based SOA model. In 2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA) (pp. 1-4). IEEE.

26. Garfinkel, S. (1999). Architects of the information society: 35 years of the Laboratory for Computer Science at MIT. MIT press.

27. Douglas, P. F. (1966). The challenge of the Computer Utility.

28. Schollmeier, R. (2001, August). A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In Proceedings First International Conference on Peer-to-Peer Computing (pp. 101-102). IEEE.

29. Foster, I., Kesselman, C., & Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. The International Journal of High Performance Computing Applications, 15(3), 200-222.

30. Buyya, R., Ranjan, R., & Calheiros, R. N. (2010, May). Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In International Conference on Algorithms and Architectures for Parallel Processing (pp. 13-31). Springer, Berlin, Heidelberg.

31. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., ... & Warfield, A. (2003). Xen and the art of virtualization. ACM SIGOPS operating systems review, 37(5), 164-177.

32. Mell, P., & Grance, T. (2011). The NIST definition of cloud computing.

33. Barroso, L. A., Clidaras, J., & Holzle, U. (2013). The datacenter as a computer: An introduction to the design of warehouse-scale machines. Synthesis lectures on computer architecture, 8(3), 1-154.