# Straggler Root-Cause and Impact Analysis for Massive-scale Virtualized Cloud Datacenters

Peter Garraghan, Xue Ouyang, Renyu Yang, David McKee, and Jie Xu, *Member, IEEE*

**Abstract**—Increased complexity and scale of virtualized distributed systems has resulted in the manifestation of emergent phenomena substantially affecting overall system performance. This phenomena is known as "Long Tail", whereby a small proportion of task stragglers significantly impede job completion time. While work focuses on straggler detection and mitigation, there is limited work that empirically studies straggler root-cause and quantifies its impact upon system operation. Such analysis is critical to ascertain in-depth knowledge of straggler occurrence for focusing developmental and research efforts towards solving the Long Tail challenge. This paper provides an empirical analysis of straggler root-cause within virtualized Cloud datacenters; we analyze two large-scale production systems to quantify the frequency and impact stragglers impose, and propose a method for conducting root-cause analysis. Results demonstrate approximately 5 percent of task stragglers impact 50 percent of total jobs for batch processes, and 53 percent of stragglers occur due to high server resource utilization. We leverage these findings to propose a method for extreme straggler detection through a combination of offline execution patterns modeling and online analytic agents to monitor tasks at runtime. Experiments show the approach is capable of detecting stragglers less than 11 percent into their execution lifecycle with 95 percent accuracy for short duration jobs.

**Index Terms**—Straggler, distributed systems, root-cause analysis, datacenter, cloud

## 1 Introduction

Modern day computing services are provisioned globally through the use of Cloud datacenters. These Internet-based virtual computing environments are distributed systems composed by hundreds and thousands of interconnected nodes, and are critical for fulfilling consumer Quality of Service (QoS) demands and business objectives. Cloud datacenters heavily exploit virtualization to form compute clusters capable of effectively deploying parallelizable frameworks such as MapReduce [1], Dryad [2], and Spark [3]–all of which require vast amounts of compute power and storage capacity to operate at scale. This has subsequently driven enormous consumer uptake for Cloud-based applications resulting in explosive data growth. This has driven the formation of Cloud datacenters composed by thousands of nodes and millions of virtualized Cloud-based services, leading to increased system scale and complexity amongst interacting components. Subsequently, manifestation of previously unseen emergent system behavior has arisen within these distributed systems, and represents a significant threat towards providing effective virtualized service performance.

This behavior is defined as the Long Tail problem, which occurs when a *job*–composed of multiple smaller *tasks* executing in parallel–incur significant delay. This delay is resultant of a subset of tasks known as *stragglers* executing abnormally slower in comparison to typical task execution [4]. It has been demonstrated that stragglers impose a substantive challenge towards rapid and predictable service execution for parallelizable applications [5], and is further aggravated by increased occurrence at growing system scale and complexity [6]. Addressing such behavior is particularly important when considering organizations such as VMWare and Amazon have spent substantial effort optimizing their virtualization technologies to operate effectively within massive-scale systems.

There have been concentrated efforts by academia and industry towards mitigating the effect of stragglers upon virtualized service operation. These approaches primarily use speculative execution based methods that create replicas of detected stragglers which leverage redundant computation [7], [8], network congestion [9], and data locality [10] to reduce overall job completion time. While such works have been demonstrated to reduce the impact of stragglers upon service operation, their effectiveness is dependent on realistic assumptions pertaining to system behavior. An example of one such assumption is that all stragglers are accurately detected within the system. This is challenging in practice due to (i) diverse task computation patterns within the system [11], (ii) straggler detection occurring late within the job execution lifecycle [12], and (iii) different underlying root causes for stragglers [6], [13].

While root-cause analysis is a cross-cutting challenge across all straggler research, there is a lack of in-depth analysis for Cloud datacenters which quantifies the frequency and impact of stragglers and its underpinning root-cause within the system. Such work is key for effective fault-

- *P. Garraghan is with the School of Computing and Communications, Lancaster University, Lancaster LA1 4WA, United Kingdom. E-mail: p.garraghan@lancaster.ac.uk.*
- *X. Ouyang, D. McKee and J. Xu are with the School of Computing, University of Leeds, Leeds LS2 9JT. E-mail: {scxo, scdwm, j.xu}@leeds.ac.uk.*
- *R. Yang is with the School of Computing, Beihang University, Beijing 100191, China. E-mail: yangry@act.buaa.edu.cn.*

*Manuscript received 1 Dec. 2015; revised 11 July 2016; accepted 4 Sept. 2016. Date of publication 20 Sept. 2016; date of current version 6 Feb. 2019.*
*For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.*
*Digital Object Identifier no. 10.1109/TSC.2016.2611578*

diagnosis for Cloud-based services, and is urgently needed for researchers to ascertain an intrinsic understanding of stragglers within real systems – transitioning away from developing detection and mitigation strategies built upon imprecise knowledge of occurrence. In order to achieve this objective, such analysis must come from large-scale production Cloud datacenters that heavily exploit virtualization in order to discover scientific understanding of straggler behavior and construct assumptions derived from realistic operational scenarios.

An effective application of performing this in-depth analysis is enhancing straggler detection; it has been identified that analyzing historical data of task execution can be leveraged as an effective means to model task computation patterns [14], enable effective speculative task execution [15], and avoid scheduling tasks onto faulty nodes [10]. However, approaches were not designed specifically for detecting straggler occurrence; from studying existing straggler mitigation mechanisms and historical analysis of task execution patterns, there is an opportunity to leverage both online and offline analytics in order to detect stragglers as soon as possible into a tasks' lifespan.

This work presents an in-depth root-cause and impact analysis of stragglers in large-scale virtualized Cloud datacenters, providing key insight for fault-diagnosis towards reliable Cloud-based service. Our approach statistically analyzes production systems to empirically ascertain straggler occurrence, quantify its impact on application execution, and determine its underlying cause. We exploit these findings to propose a novel data-centric approach for straggler detection combining offline and online analytics. The three core contributions are as follows:

- *Empirical analysis of straggler occurrence and impact.* We analyze two real-world large-scale production Cloud datacenters comprising thousands of nodes, and study the probability of straggler occurrence and quantify their impact on service performance and system overhead. This provides empirical evidence of straggler behavior in modern distributed systems, and exemplifies to the larger research community the challenge towards designing reliable Cloud-based services at scale.
- *Method of straggler root-cause analysis.* We detail a method for determining a straggler's root-cause from vast quantities of heterogeneous semi-structured trace data, and provide a study on root-cause from a 12,500+ node system, representing the first root-cause analysis of stragglers in production. We discuss the practical limitations and challenges in conducting root-cause analysis, and opportunities to improve current methods.
- *An approach for extreme straggler detection leveraging both offline analysis and online agent-based monitoring.* The offline analytics comprises linear and non-linear regression to model task execution patterns from historical data, and is used to inform online monitoring of task execution at runtime. The approach is evaluated through conducting experiments and simulation, and can be used to enhance straggler mitigation approaches.

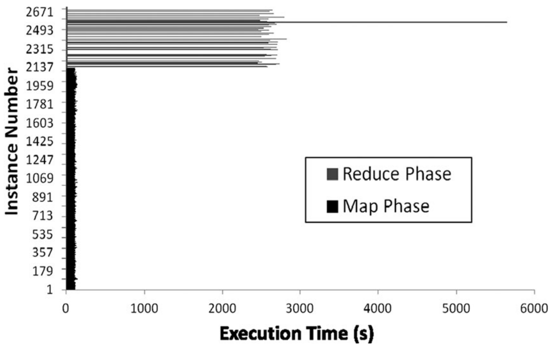


Fig. 1. Straggler occurrence within a job for a production cloud system.

Section 2 presents the research background; Section 3 discusses related work; Section 4 presents an empirical analysis of straggler occurrence and impact; Section 5 details the root-cause straggler analysis; Section 6 presents the approach for straggler detection; Section 7 presents the experiment setup and evaluation of the proposed approach; Section 8 discusses conclusions and future work.

## 2 Background

### 2.1 Stragglers

Frameworks such as MapReduce, Dryad, Hadoop and Spark decompose jobs into tasks which are executed across numerous nodes in order to achieve improved performance gains through parallelization. While these frameworks have seen extensive uptake in recent years, they all face identical cross-cutting challenges towards effective task execution at scale. Specifically, it has been established that it is problematic to achieve predictable execution within Cloud datacenter environments due to volatile network conditions, resource interference, node heterogeneity, and scheduling practices [9]. Such a challenge has resulted in virtualized Cloud services requiring longer periods of time to complete execution. This is undesirable for both consumers and providers alike; for consumers, it results in reduced service performance, and potential QoS violations with respect to time (i.e., real-time applications). For providers, services that require additional time for completion results in decreased system availability waiting for all compute resources assigned to a job to be released.

Furthermore, with the increased uptake of service creation and usage in Cloud datacenters, such behavior has been demonstrated to become increasingly frequent and important to mitigate [6]. This is especially true for Long Tail phenomena, manifesting within these frameworks in large-scale computing infrastructure. Long Tail phenomena results in ineffective job execution due to abnormally slow task execution defined as stragglers. An example of straggler occurrence is shown in Fig. 1 recorded from a job executing within a production Cloud datacenter, and demonstrates how a single task straggler can impede total job completion substantially. Stragglers significantly impede job completion, as it is unable to finish until all respective tasks within the job have successfully completed.

Long Tail phenomena will further aggravate performance degradation within virtualized computing environments. This is due to the non-negligible I/O overheads (as typical Virtual Machines use shared-storage rather than local disk storage). When considering virtualization benefits such as cost reduction, simplified management and operations, virtualized services such as Hadoop/Spark will continue to increase in scale, and thus stragglers will become an increasing concern for virtualized infrastructure.

Stragglers stem from a number of root-causes, including hardware heterogeneity [4], resource contention [6], background network traffic [9], I/O discord [10], and OS and application-level related sources [16]. There has been considerable effort towards studying stragglers caused by data skew categorized as either Map or Reduce skew, and can be further subdivided into partitioning skew, record size skew and computation skew [17], [18], [19]. How the distribution of an input dataset causes data skew (and subsequently introduce stragglers into the system) is detailed in [20].

As the size of computing infrastructure and submitted jobs continues to expand, the impact of stragglers increases dramatically. Stragglers substantially extend job execution time, thus impacting QoS and consumer Service Level Agreement (SLA) [21]. Even rare performance abnormalities can affect a significant portion of all requests in large-scale distributed systems [6], [22]. As a result, analyzing stragglers is critical in order to speed up job completion and enhance operational efficiency of Cloud datacenters.

### 2.2 Straggler Mitigation and Detection

There are two approaches to mitigate stragglers; avoidance and tolerance. Akin to the nature of faults defined within the context of dependability [28], eliminating all sources of stragglers in large-scale computing systems is impractical due to system scale and complexity [29], [30], as well as the increased use of multi-tenancy to collocate tasks within the same physical servers through virtualization.

As a result, it is typical to instead tolerate task stragglers for mitigation through means of speculative execution. Initially proposed in [1], this technique observes the execution progress of tasks using a percentage score (values ranging between 0 to 1 representing start and completion, respectively), and will launch speculative copies (or backup copies) for task progress 20 percent less than average. This approach operates under the assumption that the speculative copy will execute faster and complete prior to the original task straggler, and is currently deployed within many production clusters from Google, Facebook, Bing, Alibaba and Yahoo. Although straggler mitigation approaches have been demonstrated to enhance job execution performance, their effectiveness is underpinned by the assumption of accurate straggler detection.

Current straggler detection approaches can be classified as either online or offline analytics and both face challenges. The use of online analytics for detection can occur too late within the task execution lifecycle. As a result, even after applying speculative copies, stragglers still execute 8x slower compared to average task duration within a job, increasing its duration by 47 percent [8]. On the other hand, offline analytics are predominantly applied for straggler avoidance, an approach that becomes less feasible for systems at increased scale (and are more heavily impacted in terms of straggler behavior due to numerous underlying causes). As a result, there is a clear opportunity to combine both online and offline analytic techniques together to improve the effectiveness of straggler detection in an attempt to preserve the temporal guarantees in a Cloud system.

## 3 Related Work

### 3.1 Straggler Analytics

Jeffrey et al. [6] study a real Google service to quantify the impact stragglers impose on system performance, and demonstrate through statistical analysis that the slowest 5 percent of completed requests are responsible for half of the total 99th percentile latency. The work discusses the positive correlation between straggler probability and cluster size, concluding that the probability of longer latency increases within larger systems.

Ananthanarayanan et al. [9] analyze trace data from Microsoft Bing's production cluster. Their analysis shows that 80 percent of stragglers have a uniform probability of delay between 150-250 percent compared to the median task duration, with 10 percent exhibiting a delay 1,000 percent greater than median task duration. This work also studies the characteristics of speculative copies within the cluster, and discover that their dispersion from average execution duration is minimal, however 3 percent of stragglers require 10 times longer to successfully complete.

Garraghan et al. [22] study two production Cloud datacenters to study the frequency of straggler occurrence within Google and an anonymous large-scale e-commerce Cloud provider. Through analysis of task execution patterns extracted from system trace logs, they discover that a small proportion of tasks negatively impact the execution for approximately half the jobs within the entire system. Their work also observes that the distribution of straggler occurrence per server is weakly skewed, and affects 20% and 100% of nodes for each Cloud datacenter, respectively.

While these works study the characteristics of stragglers within real systems and quantify their impact on overall system performance–their primary objective is the proposal of detection or mitigation approaches. This results in analytics limited to observations pertaining to straggler occurrence, and does not study the precise root-cause of stragglers in detail. Work such as [9], [22] only briefly discuss the need to differentiate stragglers by dataskew, resource contention and faulty nodes, yet provide no analysis to support this objective.

### 3.2 Online Straggler Detection

There have been numerous straggler mitigation methods proposed that are dependent on online monitoring and speculative execution.

Zaharia et al. [4] propose LATE, a method of speculative execution which emphasizes improved effectiveness within a heterogeneous cluster. This work proposes a Progress Rate matrix to calculate the estimated completion time in addition to the absolute Progress Score for straggler detection. Furthermore, this work also defines concepts such as a slow node threshold to ensure speculative copies are launched on powerful nodes, slow task threshold to avoid needless speculation for fast tasks, and speculative cap to limit the number of speculative tasks running simultaneously within a system. When combined together, LATE demonstrates improvement to default Hadoop job response times by a factor of two, and is presently the dominant method of straggler detection for distributed systems.

Ananthanarayanan et al. [9] introduce MANTRI; the concept of preferential replication and resource constraint-aware placement of speculative copies in LATE. Specifically, the approach only replicates the output of tasks which are either likely to be lost or require substantial re-computation calculated through a cost-benefit analysis. Furthermore, the approach also launches speculative copies based off the present network congestion characteristics of the system. Experiments within Bing's production cluster demonstrate that Mantri can improve job completion times by 32 percent in comparison to LATE.

Despite improvements to straggler mitigation using online detection, experiments conducted using production data [12] shows that as many as 90 percent of launched speculative copies are unneeded. This is due to speculative copies launched too late within the task lifecycle, resulting in the original task completing prior to the replica (and is subsequently killed by the system). This behavior results in many speculative copies producing resource overhead with no improvement to job execution time. As a result, a critical requirement for online mitigation is the ability to identify stragglers as quickly and accurately as possible.

### 3.3 Offline Straggler Detection

There are several approaches that leverage historical data to improve speculative execution effectiveness through offline analytics.

Chen et al. [14] propose SAMR; a self-adaptive scheduling algorithm. They use historical data to adjust temporal weightings for each execution stage for calculating task progress, with results demonstrating up to a 25 percent decrease in job completion time in comparison to Hadoop default scheduler and a 14 percent decrease compared to LATE. Lin et al. [23] further augments SAMR within a multi-tenant system, and show that their method only generates a 10 percent relative mean square error for task completion prediction for reduce tasks and 30 percent for map tasks.

Ananthanarayanan et al. [9] propose a smart speculative strategy that leverages historical data to select the most suitable node candidates for launching the replica copies using a cost-benefit model. Their results demonstrate that MCP can run jobs up to 39 percent faster and improve the cluster throughput by up to 44 percent compared to Hadoop default.

Yadwadkar et al. [15] use a statistical learning technique based on cluster resource utilization counters to select the fewest resources needed for efficient speculation and significantly improved the resource consumption by up to 55 percent while still achieving an improvement to job completion time by 61 percent compared to default speculative execution.

Furthermore, there are methods that use historical data to proactively avoid scenarios that cause stragglers: Yadwadkar et al. [24] also propose a method for proactive straggler avoidance that performs a regression tree algorithm using the node-level statistics and avoid assign tasks onto nodes that tend to cause stragglers.

All of above works show that machine learning and offline data analytics techniques support straggler detection preciseness. However, it is observable that offline analytics are predominantly applied for calculating estimated task execution times within the system instead of predicting stragglers, which becomes less feasible when approaching systems at increasing scale. Furthermore, both online and offline detection is underpinned by a deep understanding of straggler behaviour; presently there is limited work that specifically analyses how stragglers affect system performance, and the underlying root-cause which leads to straggler manifestation.

TABLE 1
Studied Cloud Datacenter Characteristics

| System | Google [20] | Cloud Datacenter B |
|---|---|---|
| **Cluster size** | 12,532 | 2,841 |
| **Time period** | 29 days | 14 days |
| **Application Types** | Batch, MapReduce, Latency sensitive, streaming, etc. | Direct Acyclic Graph (DAG) - Multiple MapReduce phases |

## 4 Straggler Impact Analysis

It is necessary to first fully understand stragglers within the context of real system operation. This enables researchers to study frequency and impact that task stragglers impose on Cloud datacenters, as well as focus research and developmental effort for enhancing straggler detection.

To achieve this, we have empirically studied stragglers within two large-scale production Cloud datacenters; the Google cluster [25] and Cloud Datacenter B–a large-scale e-commerce provider (for commercial reasons we are unable to disclose the provider's identity). Each of these systems use OS-level virtualization (such as LXC), and vary dramatically in terms of cluster size, server heterogeneity, business objectives and application types as summarized in Table 1.

As operational trace data produced from these systems are semi-structured and voluminous - composed of multiple files detailing information concerning task resource usage, event logs and server utilization–it is necessary to filter the trace data within each system in order to identify different job types. Specifically, we are particularly interested in studying straggler manifestation within batch jobs (i.e., DAG, MapReduce, MPI); a common type of application typically deployed within Cloud datacenters.

The approach for filtering batch jobs within each cluster follows the same method, varying only by bespoke extraction from heterogeneous trace data structure. Information pertaining to job ownership of tasks is identified through use of recorded job IDs attached to all submitted tasks. Once the grouping of tasks to specific jobs has been established, their execution time is calculated through recorded start and completion events within the trace. Furthermore, we also consider the resource characteristics of tasks when identifying batch jobs to avoid serial task execution (i.e., all tasks within a job have the same requested resources and are submitted fraction of timestamps apart from each other). Once the execution duration for all tasks has been determined, we calculate the difference between an individual task's execution duration and the average duration of all tasks within a job. In the case for the Google cluster, there exists multiple applications types described in [26]. As a result, we filter all jobs with priority 4 (identified as batch processing). Through using this filtering criteria it is possible to identify 3,043 jobs comprised of 252,290 tasks within Google and 875 jobs comprised of 1,223,879 tasks for Cloud Datacenter B.

Figs. 2, 3 shows the difference between an individual task's execution duration and the mean and median

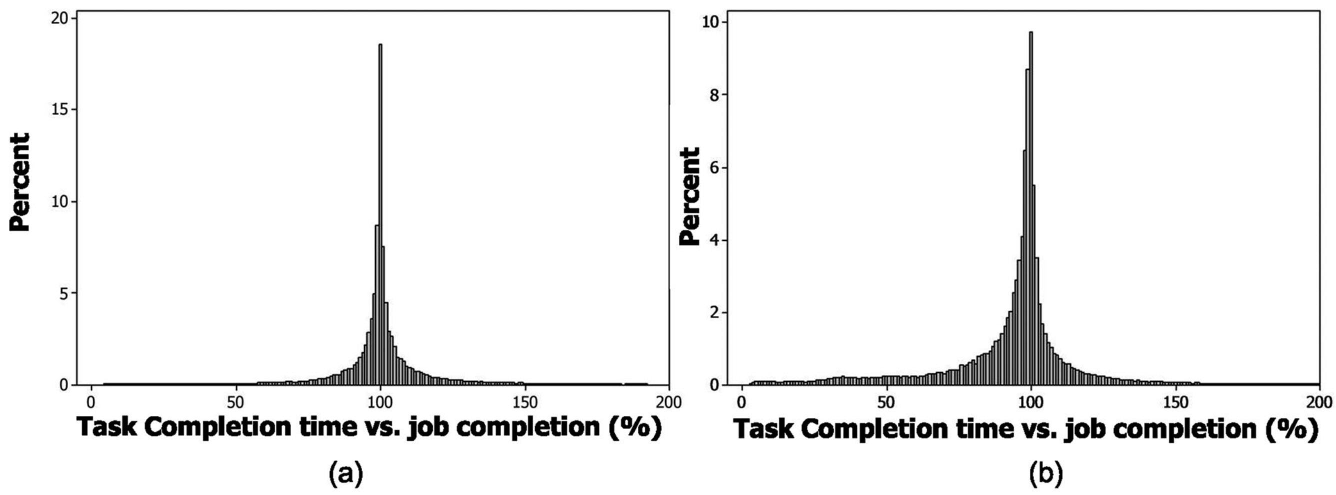


Fig. 2. Google datacenter task—job completion difference percent (a) median, (b) mean.

execution of all tasks within the same job. It is observable that the majority of tasks exhibit similar proportions for completion situated around 100 percent (i.e., an individual task execution duration is equal to the average job execution duration for all other tasks) for both studied systems. In accordance with [4], [8], [18], [19] task stragglers are defined as tasks whose execution is $\geq$ 150 percent the average execution of all tasks within the same job.

We observe that calculating this difference using different central tendency measurements of mean and median results in substantially different patterns for straggler detection. This is particularly noticeable within Cloud datacenter B shown in Fig. 3, exhibiting different dispersion patterns for task execution. This is resultant of extremely fast or slow tasks affecting the central tendency and dispersion for task completion within a job when using the mean. As a result, while existing literature use the mean task execution duration for defining stragglers, there are additional advantages when studying the median task duration instead. Most notably that median job execution duration is less affected by extreme execution times caused by task stragglers; this is especially true when considering jobs composed of thousands and tens of thousands of tasks. This results in 6.54 and 3.48 percent of tasks to be identified as stragglers within the two respective Cloud datacenters.

While it is intuitive to assume that such a small proportion of task stragglers would have limited impact towards the performance of all jobs, findings demonstrate that between 37.79 and 49.49 percent of all jobs are negatively affected. This result is due to a job's inability to complete until its respective tasks (including stragglers) have all completed execution, with distribution of job execution delay shown in Figs 4a and 4b for Google and Cloud Datacenter B, respectively. Such results resonates with theorized impact of stragglers in large-scale systems discussed in [6], and corroborates findings in [9] demonstrating a small proportion of jobs being delayed by up to 1,000 percent. This negative affect can be directly quantified in terms of system overhead and application performance as shown in Table 2. It is observable that task stragglers cause job completion to be delayed on average between 12 and 865 seconds. The reason for this large disparity is primarily driven by the applications executing within the different Cloud systems. Cloud datacenter B is composed of shorter lived DAG jobs, while Google cluster is comprised of longer running batch jobs. While it can be argued for the latter that extended job execution lends itself to less focus on timing requirements, this directly translates into increased system overhead and reduced system availability, reflected by 2.49 percent additional compute hours required with the entire system (and

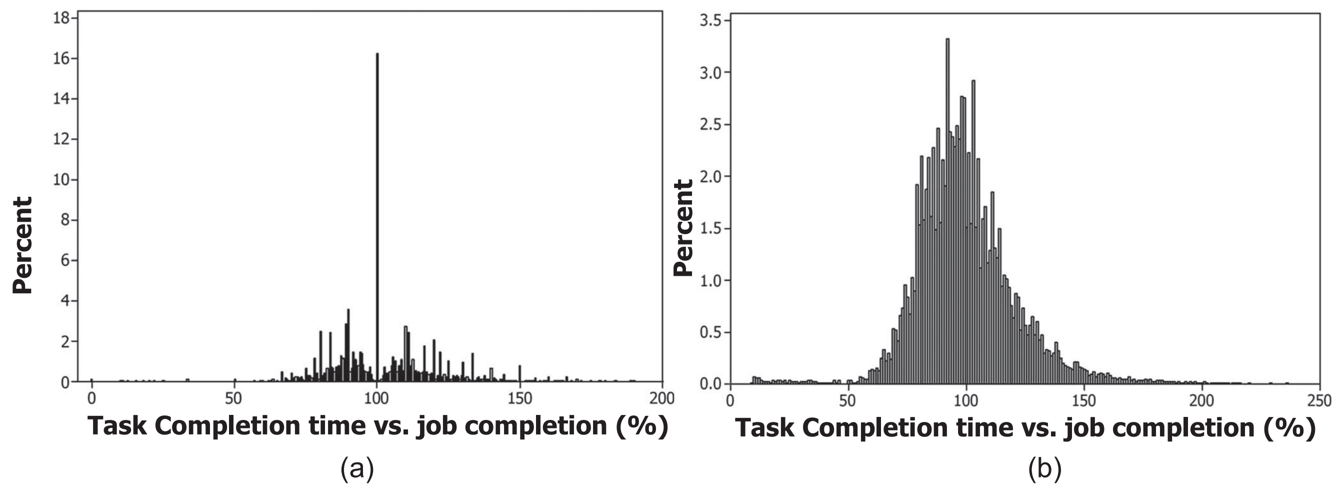


Fig. 3. Cloud Datacenter B task—job completion difference percent (a) median, (b) mean.

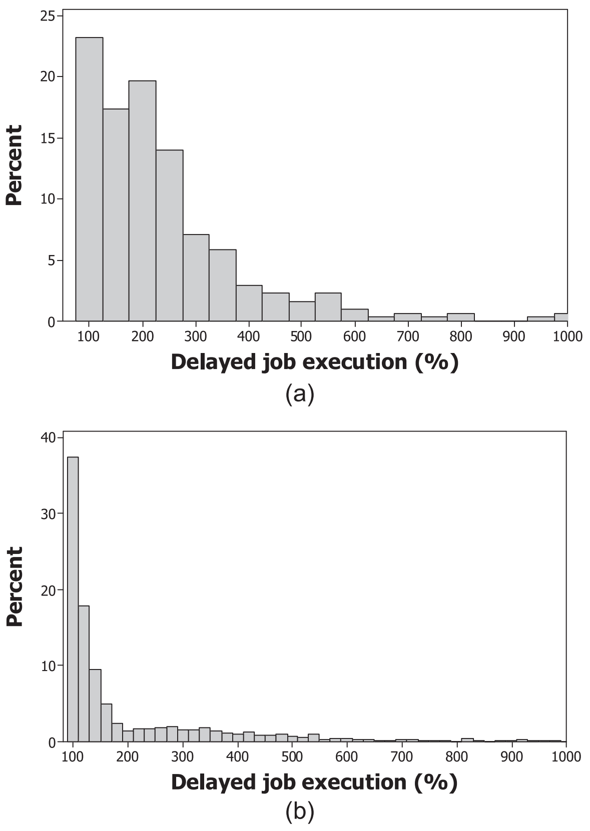


Fig. 4. Job execution delay distribution (a) Google datacenter, (b) cloud datacenter B.

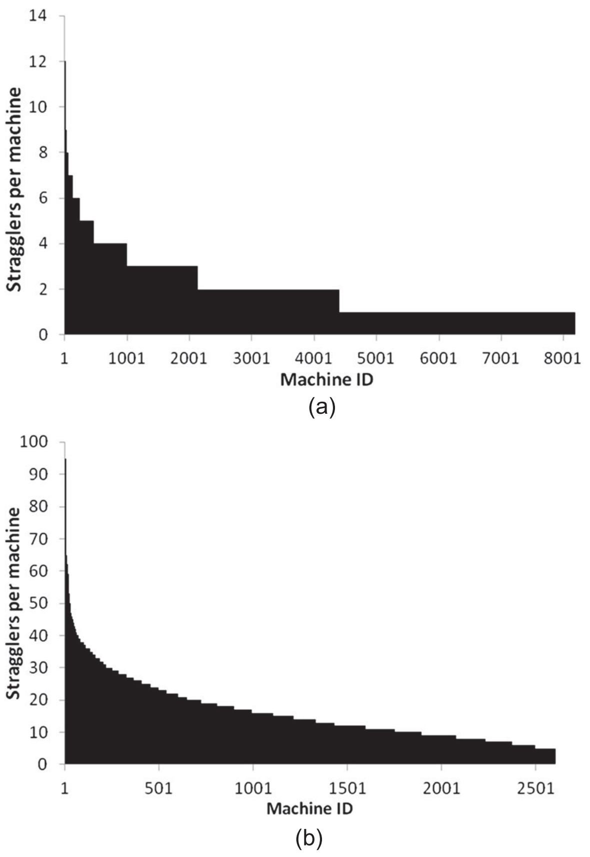


Fig. 5. Comparison of filtered stragglers from (a) Google datacenter (b) cloud datacenter B.

effectively doubles to 5% when applying speculative execution). We also studied the manifestation of task stragglers within servers, with Fig. 5a and 5b depicting the distribution of stragglers per server, and observe that 65.07 and 99.78 percent of servers experience stragglers with a weak right-skewed distribution within Google and Cloud datacenter B, respectively.

This analysis of stragglers within production Cloud datacenters has discovered a non-intuitive finding of particular interest. Specifically, while stragglers occur in 3-7 percent of total tasks submitted, they impact a greater proportion of jobs by a factor of 10. By empirically demonstrating this surprising affect stragglers impose on large-scale systems, researchers and industry will be able to convey the scale and importance addressing straggler behavior to the wider community. The next step is to investigate the underlying causes which produce these identified stragglers.

TABLE 2
Straggler Occurrence and Impact in Production Systems

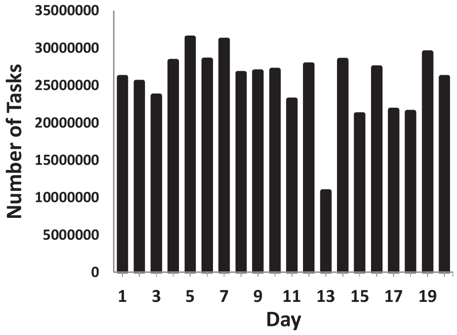
| | Google Datacenter | | Cloud Datacenter B | |
|---|---|---|---|---|
| | Mean | Median | Mean | Median |
| **Total tasks** | 252,950 | | 1,233,879 | |
| **Task stragglers** | 11,210 | 16,543 | 33,322 | 42,925 |
| **Task stragglers (%)** | 4.43 | 6.54 | 2.70 | 3.48 |
| **Total jobs** | 3,043 | | 875 | |
| **Job stragglers** | 1081 | 1150 | 512 | 433 |
| **Job stragglers (%)** | 35.52 | 37.79 | 58.51 | 49.49 |
| **Median straggler duration (s)** | 865 | | 12 | |

## 5 Straggler Root-cause Analysis

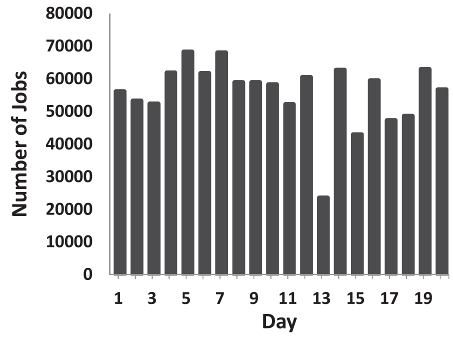
This section details a method currently applied within industry for conducting straggler root-cause analysis—detailing straggler filtration, analysis limitations in live systems, and presents an analysis of straggler root-cause stemming from numerous causes. Here we focus on the operational practices conducted within Cloud Datacenter B; due to obfuscation of low-level system logs within the Google trace, it is not possible to derive deep insight into their methods for straggler root-cause analysis.

The method is composed by two components; *correction* and *diagnosis*. Correction entails a reactive approach of direct intervention by technical staff to perform fault correction upon straggler detection. This is performed by periodic execution of a health checker processes using Tsar [27] and Nagios [28] to monitor system metrics at a specific time interval, and alerts potential atypical system behavior to technical staff (i.e., abnormally high CPU utilization, extended task execution). Correction allows for technical staff to identify and manually correct potential problems within the system for reducing QoS violations and catastrophic failure prevention (such as system outages).

While correction allows for rapid fault correction to reduce straggler impact towards system QoS, it is advantageous to understand the precise operational scenarios and causes that result in straggler occurrence. This is important in order to focus technical and developmental efforts towards reducing future straggler occurrence within the system. As a result, diagnosis involves offline analysis of



(a)



(b)

Fig. 6. Workload statistics for cloud datacenter B (a) tasks, (b) jobs.

system historical data to conduct in-depth investigation of precise causes for stragglers.

A challenge when performing diagnosis is the large quantity of stragglers detected daily within system; Fig. 6 illustrates the proportion of jobs and tasks submitted daily within the greater Cloud datacenter B cluster, comprising over 12,000 servers over 20 days of operation. If assuming 3.48 percent of the 25,600,000 tasks submitted daily are detected as stragglers (890,880), based on findings in Section 4, the ability to perform fault correction and diagnosis becomes infeasible due to the sheer number of occurrences. Therefore, it is necessary to further filter and characterize straggler behavior, thereby focusing on root-cause analysis for stragglers towards a specific design objective. A particularly important objective for production systems is mitigating the impact of extreme stragglers (i.e., tasks whose execution time far exceeds typical behavior) due to their noticeable impact to user perception of application performance.

To achieve this, we propose a new criterion for straggler detection termed Degree of Straggler (*DoS-index*)–a system metric comprising task execution time and input size for an individual task *Ti* for *n* tasks in a job as shown as follow:

$$DoS\text{-}Index = \left(\frac{Dur(T_i)}{Inp(T_i)}\right) \Bigg/ \left(\frac{\left(\frac{\sum_{j=1}^{n} Dur(T_j)}{n}\right)}{\left(\frac{\sum_{j=1}^{n} Inp(T_j)}{n}\right)}\right) \tag{1}$$

where *Dur(Ti)* is the current execution duration of *Ti*, and *Inp(Ti)* is the data volume that *Ti* is required to process. Based on this definition, it is possible to control the strictness for straggler detection. The DoS-index indicates a relative speed of data processing (i.e., the time consumed when processing one unit of input data) for an individual task contrasted against all other tasks within the same job. A higher *DoS-index* value indicates a task with extended execution time and/or low input size in comparison to tasks within the same job. *DoS-index* is configured by default in Cloud datacenter B as $\geq$ 2.5.

Although conducting data analytics for daily straggler reporting can be automatically deployed and generated using Big Data techniques, there are still numerous limitations towards automatic diagnosis–requiring fine-grained analysis for investigating precise straggler root-cause. Straggler diagnosis requires manual intervention from technical staff, and is conducted on a case-by-case basis. This is due to the requirement for technical staff to study heterogeneous semi-structured system logs from multiple sub-systems including kernel processes, error logs, application logs (it is worth noting that the same system log may be heterogeneous from each other).

This results in an inability to produce a single unified query for straggler diagnosis, and data analysis is viewed as one tool to support technical staff when conducting root-cause analysis. Furthermore, data queries themselves require considerable system resources for computation and data mining, and will impose overhead affecting the production system operation. Such overhead threatens the system's ability to provision acceptable levels of QoS, and is produced within numerous aspects including server and application heterogeneity (semi-structured logs), network condition, I/O performance and remote data access (read and write), and current system utilization. As a result of this challenge combined with the design philosophy for Cloud datacenter B to focus developmental efforts on the extreme stragglers behavior within the system, the DoS-index is also configured to $\geq$ 10 for conducting diagnosis.

Table 3 presents statistics for stragglers within a 20 day period under different detection methods. It is observable that using different straggler criteria such as >150 percent progress and *DoS-index* strictness criteria results in filtering the number of stragglers down from 890,880 to 7,319 (365 a day on average) as depicted in Fig. 7. It is hypothesized in [15] that high resource usage of a server plays a key factor for straggler occurrence. As a result from using system profiling tools in [27], [28], we monitor and collect system information from servers which execute all tasks with *DoS-Index* $\geq$ 10. Information collected includes server CPU utilization $\geq$ 80 percent, Disk usage $\geq$ 80 percent, and slow Read-Write request handling (i.e., latency from file system > 400ms).

We observe that approximately 59 and 42 percent of stragglers with *DoS-Index* $\geq$ 10 occur under the presence of high server CPU and disk overloading, respectively. This result indicates that high server resource utilization is a common cause for straggler occurrence. It is also observed that 34.3 percent of stragglers experience slow request handling. Although it is possible for CPU utilization and disk utilization to be correlated, we are unable to find significance in correlation (indicated by a Pearson Correlation Coefficient of 0.072). This is likely result of diversity in workload characteristics within Cloud datacenter B (i.e., CPU, memory, disk, and network intensive tasks) imposing different server characteristics, and requires further study of straggler categorized by workload behavior and characteristics.

TABLE 3
Cloud Datacenter B Straggler Detection with DoS-Index

| Day | *DoS-index*≥2.5 | *DoS-index*≥10 | System Utilization at Detection | | |
|---|---|---|---|---|---|
| | | | CPU Util≥80% | DiskUtil≥80% | Slow Req Handling |
| **1** | 9,937 | 136 | 46 | 61 | 29 |
| **2** | 7,232 | 151 | 114 | 14 | 23 |
| **3** | 7,540 | 280 | 161 | 84 | 35 |
| **4** | 7,277 | 213 | 147 | 23 | 43 |
| **5** | 12,373 | 376 | 158 | 149 | 69 |
| **6** | 8,402 | 384 | 129 | 184 | 71 |
| **7** | 10,450 | 562 | 352 | 128 | 82 |
| **8** | 8,494 | 552 | 348 | 129 | 75 |
| **9** | 9,109 | 313 | 121 | 94 | 98 |
| **10** | 10,834 | 426 | 116 | 77 | 233 |
| **11** | 8,486 | 382 | 100 | 150 | 132 |
| **12** | 8,773 | 586 | 179 | 239 | 168 |
| **13** | 2,728 | 534 | 126 | 247 | 161 |
| **14** | 9,414 | 283 | 117 | 41 | 125 |
| **15** | 8,472 | 448 | 104 | 259 | 85 |
| **16** | 12,194 | 335 | 131 | 66 | 138 |
| **17** | 9,700 | 395 | 236 | 92 | 67 |
| **18** | 10,941 | 368 | 163 | 63 | 142 |
| **19** | 12,552 | 313 | 172 | 83 | 58 |
| **20** | 11,526 | 282 | 154 | 101 | 27 |
| **Total** | **186,434** | **7,319** | **3,174(58.6%)** | **2,284(42.1%)** | **1,861(34.3%)** |

We conducted an in-depth root-cause analysis for stragglers with DoS-Index $\geq$ 10 to ascertain a deeper insight to straggler occurrence due to numerous underlying causes. This was performed from exploratory analysis of numerous system logs within the cluster, including application errors, kernel processes, resource managers, and system monitoring tools. Table 4 shows the categorization of the dominant factors that cause stragglers to occur, and their corresponding frequency.

It is observable that high CPU utilization is the most dominant type of cause, responsible for 30 percent of all straggler occurrences, and is caused by two reasons; unbalanced workload aggregation and poor user code. Unbalanced aggregation is caused by inefficient scheduling causing excessive workload co-allocation within a server. Poor user code is inefficiently designed executable logic (i.e., orphan processes, looping conditions) complied and executed by the user. Both of these reasons result in CPU bursting within a very short time period; this results to inefficient time-slice sharing within the server kernel, resulting in slowdown in CPU, memory and disk access.

Another straggler cause is resultant of faults within the server, specifically late timing failures and transient disk faults which result in slow disk I/O and file operations; task co-located within the same machine with the same resource characteristic (i.e., IO intensive) generate resource interference. We discovered that it is possible for tasks to read and write to the same disk block simultaneously, resulting in large amount of disk resource competition requiring conflict resolving.

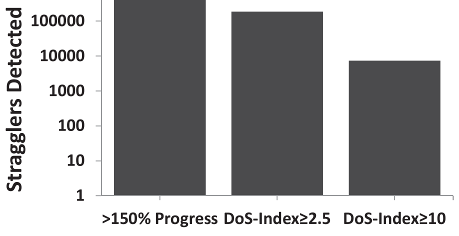


Fig. 7. Number of stragglers detected with approaches.

Another important reason we observe is the request handling inefficiency due to overloaded and surging file requests. Specifically, for a typical batch job such as MapReduce, there are a large number of read and write operational requests to the distributed file system (such as HDFS, GFS, etc.). Once the surging request number surpasses the handling capability of the file system master, it will become a bottleneck (even when the master has multiple replicas) and therefore many requests will be queued to await allocation. In fact, based on our analysis, we observe that in some cases the unreasonable configuration of Map or Reduce number or block size might lead to unexpected request increase thereby increasing the load of file system master with slow request handling.

Furthermore, we found that the network condition is also a variable that will affect reliable task execution, due to all remote copies operations after shuffle phase in Map Reduce being sent through the network. From our analysis by using "tsar retran" [27], 14 percent of stragglers were caused due to network package loss. Higher package re-transmission results in not only extended job end-to-end time-span, but aggravates the network congestion as well. Finally, other common factors include time-out faults and data skew, comprising 10 percent of straggler root-cause.

The proportions of affected tasks from each identified straggler root-cause share identified proportions similar to Table 4 (specifically for high CPU, Disk and slow request handling). Therefore, we are fully convinced from our practical experience that these results could be used as inspirable instructions to handle with different stragglers and can cover comprehensively multiple scenarios and fault-injection practices to simulate straggler behavior.

TABLE 4
Classification for Straggler Root-Cause

| Type | Category | Specified Description | Occurrence frequency |
|---|---|---|---|
| **1** | High CPU utilization | Low time-slice sharing and process scheduling due to certain bad user-defined worker logic, unbalanced workload aggregation etc. | 30% |
| **2** | High disk utilization | Local disk read and write conflicts, unbalanced tasks aggregation, disk faults etc. | 23% |
| **3** | Unhandled operational access request | Distributed file system request surging(usually read request) and overpass the capability of request handling. | 23% |
| **4** | Network package loss | Network traffic package loss, resulting in repeating intermediate file and data transmission. | 14% |
| **5** | Hardware faults | Server timing-out, hang etc. | 7% |
| **6** | Data skew | Uneven file block input resulting in data skew. | 3% |

## 6 System Architecture

In this section we propose a method and implementation for a straggler detection system for large-scale virtualized distributed systems which aims to mitigate the effects of extreme stragglers (i.e., task execution that is abnormally long). While our approach is applicable to numerous types of distributed systems such as Grids, Cyber-physical systems and the Internet of Things, this work focuses on Cloud computing datacenters; modern large-scale systems with explicit (SLAs, QoS, availbility) and implicit (energy-efficiency, user experience) requirements for provisioning high performance service to users.

Fig. 8 depicts the high-level system architecture for our task straggler detection system, and is divided into two primary components: *offline analytics* and *online analytics*. The offline analytics component analyses historical data detailing previous job execution to characterize and model task execution patterns in order to calculate a threshold parameter which determines a boundary which distinguishes between straggler and non-straggler behavior for an individual task at a given time interval. The online analytics then monitors and compares the current task execution progress at runtime against the historical patterns using agents within each server for straggler detection.

### 6.1 Offline Analytics

The offline analytics component is integrated into the straggler detection engine, and is responsible for analyzing and modeling task patterns and straggler manifestation. Specifically, this module is responsible for modeling task execution patterns and supports the decision making for straggler detection within the offline analytics components. The module is composed of three sub-components:

*Job profiler*. Responsible for profiling and modeling different types of job and task execution patterns. Such a components is important as tasks exhibit heterogeneous task execution lengths and resource consumption quantities across the system as detailed in [29]. The method for profiling job execution patterns is independent on the task characteristics executing within the system, and can be performed using several techniques including clusterization and modeling task progress execution [11]. Fig. 9 shows an approach for modeling task progress execution patterns for 500 Reduce tasks within a 50 node cluster. It is observable

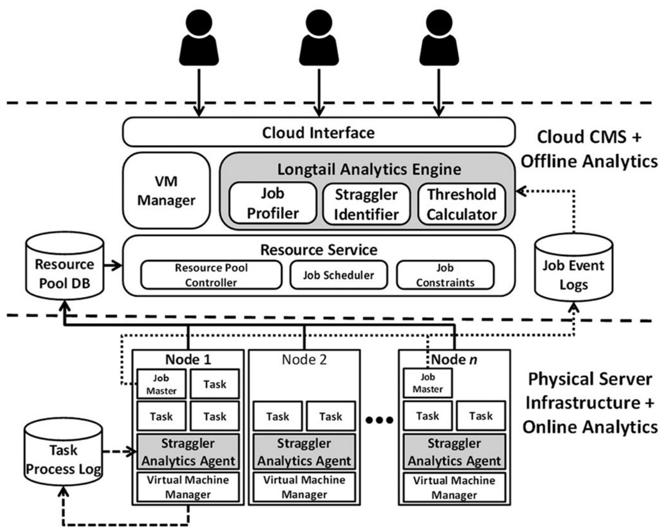


Fig. 8. Cloud datacenter model with integrated long tail analytics engine and agent based analytics.

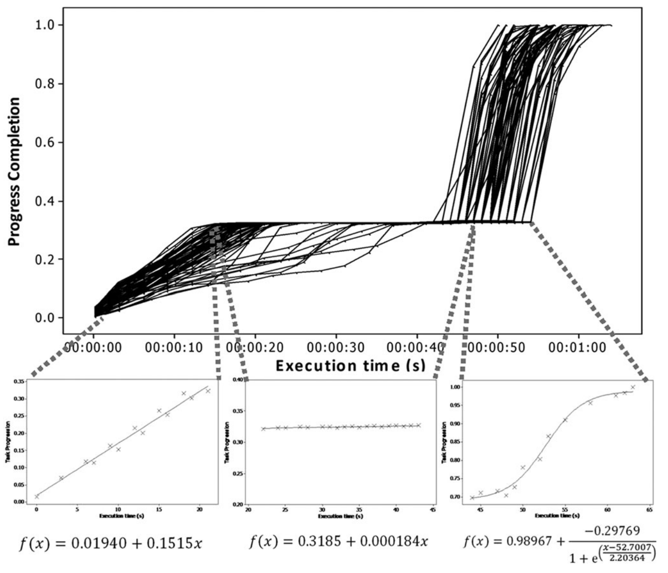


Fig. 9. Example of converting empirical task progression into regression models for task progress execution.

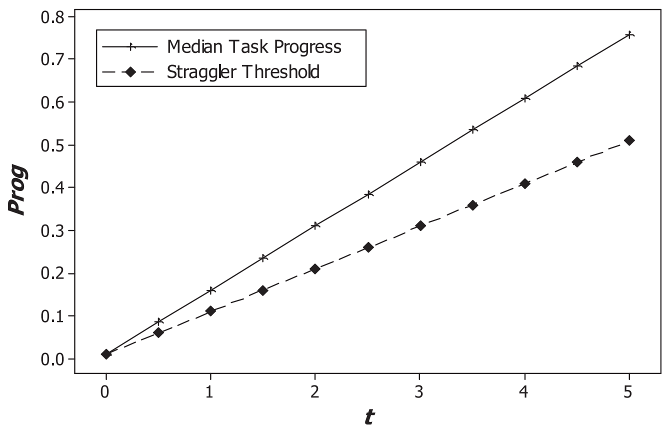


Fig. 10. Representation of median task progress ($T_{iProg}$) and straggler threshold ($T_{iS}$) at time $t$.

that it is possible to sub-divide the Reduce phase into multiple stages [1], [4], which can be successfully modeled through a combination of linear and non-linear regression analysis. Using this technique it is possible to profile task execution progress patterns over time for specific job types.

*Straggler Identifier*. Responsible for quantifying the type and impact of past stragglers within the distributed system. Work within [6], [9], [19] and findings in Section 5 have identified that there are numerous root cause for stragglers. Therefore, it is advantageous to analyze and identify the cause of stragglers which occur historically within a system in order to correct identified faults following the method in Section 5, and ascertain where developmental effort should be applied for maximum effectiveness.

*Threshold Calculator*. This components exploits the task execution patterns and regression models generated from the job profiler component to derive the (theoretical) minimum threshold for task progress at a certain time. Specifically, straggler threshold $S$ is defined as the minimum progress of task $Ti$ completed at time $t$ in relation to typical task progress *Prog* to avoid being identified as a straggler. *Diff* is calculated as the distance between *TiProg* and *TiS* at time $t$, and is used for determining violation of threshold value $S$, and is expressed as a percentage determined by the system administrator.

To give a hypothetical example, if a model expressing $Ti$ over period $t$ generated from the Job Profiler component is a linear function as shown in (2):

$$T_{iProg} = 0.01 + 0.15\mathrm{t} \tag{2}$$

and *Diff* is defined as task execution time 50 percent greater than median execution - a value commonly defined in the literature (i.e., speculated task straggler completion time of 180 minutes against typical task completion of 120 minutes), then straggler threshold is expressed as a function shown as follow:

$$T_{is} = 0.01 + 0.15 \cdot \frac{2}{3} t \tag{3}$$

As demonstrated in Fig. 10, in this example *TiS* will equal *TiProg* when $t$ is 50 percent greater (thus, a task is detected as a straggler when the time taken to reach a specific progress score at time $t$ is greater than 50 percent in comparison to typical task execution). The developed model generated from the offline component of the system is exploited by the online analytics at runtime for straggler detection.

### 6.2 Online Analytics

The online analytics component is comprised by the Straggler Analytics Agent which resides on each physical server as a lightweight process within the distributed system as shown in Fig. 10. The agent is responsible for monitoring and analyzing task execution progress and straggler detection at runtime. When a task is scheduled onto a server, each agent will periodically monitor task progress and extract key parameters from data traces generated by each task. Parameters of interest identified includes task timestamp, time of task instantiation, current task progress score as well as data blocks transferred and download rate (if applicable to the current Reduce phase).

The agent compares current task progress against the model produced by the offline analysis determined by the threshold calculator in Long Tail Analytics Engine. The agent then communicates with other agents in order to compare task progress against the median progress of all tasks within the same job at time $t$. The model derived from the offline analysis is of particular importance, as it safe guards against false positives due to (i) multiple stragglers within the same job, and (ii) boundary sensitivity at the beginning of job execution for low progress values. If $T_{iProg} < T_{iS}$, as well as 50 percent smaller than the median task progression at $t_i$ for its respective job, a task is identified as a straggler. Such an approach can encounter challenges in model sensitivity within the first time periods due to the short euclidian distance between progress scores at the start of task execution. As a result, it is necessary to combine both offline and online analytics together into a single approach for straggler detection.

## 7 Straggler Detection Evaluation

In this section we present the straggler detection evaluation to study its effectiveness in real systems. Furthermore, we demonstrate the advantages of combining online and offline analysis together in contrast to detection in isolation.

### 7.1 Experiment Setup

Experiments were conducted to evaluate the effectiveness of the proposed straggler detection method. The system was implemented within a 50 node cluster concurrently used by other users for research and University services, comprising 50 x quad-core Intel machines @ 3.40GHz CPU running CentOS. We deployed two types of applications. The first is Hive [30]–a database management system which interfaces and translates SQL like queries into MapReduce jobs. 40 jobs each comprising between 500-1000 Map tasks and 40-80 Reduce tasks were submitted to the cluster, with each job configured to perform various aspects of computation (i.e., multiplication, CEIL and FLOOR functions), JOIN clauses between data tables, and data attribute types. The data used to perform these functions consisted of operational trace logs from numerous Cloud, datacenter and HPC systems, totalling approximately 1 TB of system log data. The second application is the WordCount benchmark pre-installed within Hadoop, and processes a 548 MB file with each job creating 15 Map tasks per execution run.

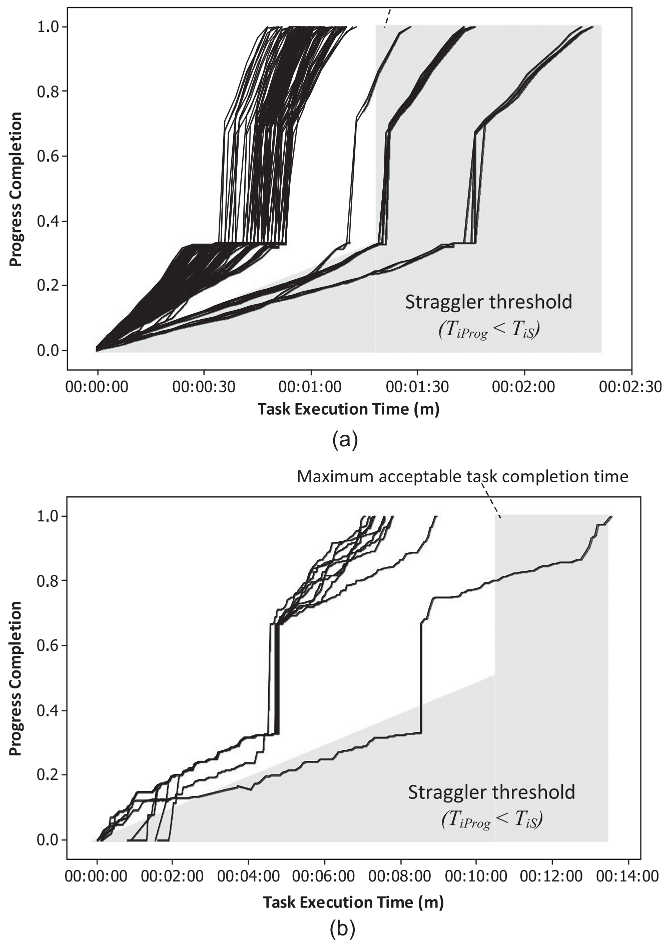


Fig. 11. Task progress with long tail identification analytics engine (a) HiveQL, (b) WordCount.

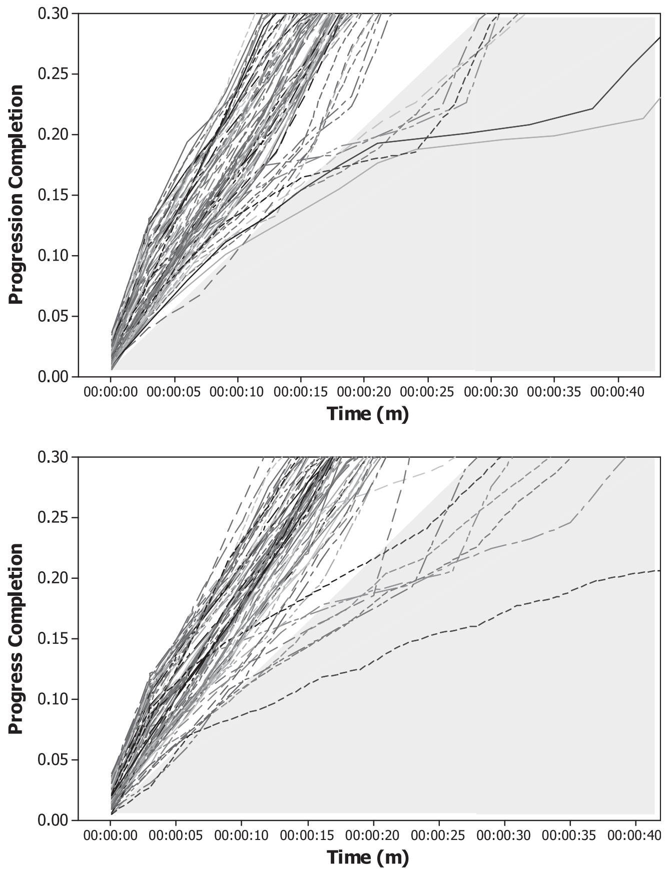


Fig. 12. Threshold boundary sensitivity.

In experiments we inject tasks which exhibit straggler behavior caused by data skew by invoking specific query types within Hive, and larger input size within WordCount. The probability of this occurrence per task is configured at 5 percent, reflecting values derived from the straggler impact analysis in Section 4. Within experiments we define straggler behaviour as task completion time 50 percent greater than median task execution within a job, in accordance to the definition within [4], [8], [18], [19]. Due to model sensitivity from initial experiments we configured for straggler detection to commence 5 seconds after the job commences. Online analytics agents were configured to monitor and compare current task progress against the models generated from Long Tail Analytics Engine at a time interval of three seconds (in line with Hadoop MapReduce log file reading). For evaluation we measure thresholds using the estimated finish time as opposed to the DOS-Index, as we assume full knowledge pertaining to extreme straggler task execution.

TABLE 5
Statistical Properties of Straggler Detection Experiments

| | HiveQL | WordCount |
|---|---|---|
| **Total tasks submitted** | 2500 | 350 |
| **Total stragglers submitted (%)** | 6.45 | 5.61 |
| **True positive rate (%)** | 95.71 | 78.31 |
| **False positive rate (%)** | 5.59 | 21.69 |
| **Straggler progress detection (%)** | 10.91 | 37.77 |
| **CPU usage of agent per node** | 0.20% | 0.21% |

## 7.2 Experiment Results

Figs. 11a and 11b depicts task progress execution over time and the generated threshold model for HiveQL and WordCount, respectively. It is observable within both job that there exists a substantial difference between normal and straggler task patterns, caused by the input size sent to a task being considerably larger than normal. Through statistical models generated from historical data combined with online analysis, it is possible within HiveQL to identify over 95 percent of stragglers caused by data skew which are detected 10.91 percent on average into a task progress at runtime detailed in Table 5. We observe false positive rate of 5.59 percent which is caused by task progress of straggler and non-straggler tasks exhibiting similar progress scores at the start of execution as shown in Fig. 11. In contrast, WordCount jobs are characterized as executing much longer compared to HiveQL (over 300 seconds), with 78.31 percent of stragglers detected 37.77 percent into their execution.

The predominate reason for the occurrence of false positives is a result of threshold sensitivity or interference from other users executing jobs on the same cluster. This is especially true for WordCount jobs, where it is possible for task execution to commence after the initial phase has commenced. Furthermore, it is possible for tasks to meet the conditions for detection, however quickly recover around the start of job execution as shown in Fig. 12. While this work focuses on detecting extreme tasks, this is a common

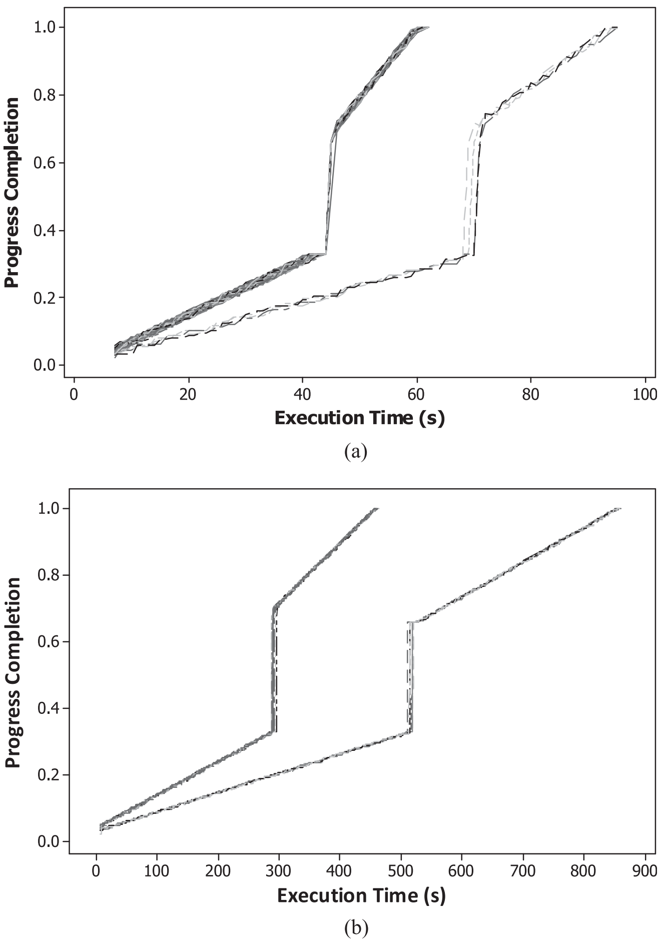


Fig. 13. Simulated normal and straggler task patterns for (a) HiveQL, (b) WordCount.

weakness shared across all detection systems for 'borderline stragglers' which require further investigation. This result demonstrates the need for refinement of any future detection techniques that are configured to handle potentially different sensitivity levels for straggler detection at different time frames into a task's execution. While this result indicates high accuracy of straggler detection for jobs with short duration and all of which start task execution at time zero, from our experiments we observe that there is further refinement required for longer running tasks and tasks which start later within the job lifecycle. Such refinement could be achieved through tuning *Diff* as well as data mining additional event parameters of interest from system logs (i.e., task process resource consumption, network usage, node location, etc.).

Furthermore, we observe that a minority of tasks are detected early within their execution, yet finish relatively close to the boundary of acceptable task completion (*TiS*). This behavior highlights potential issues for defining a fixed arbitrary value for stragglers (i.e., 50 percent greater than median task execution, *DoS-Index* $\geq$ 2.5), as tasks that complete just below the threshold will be detected as false positives, however still impede job execution completion. These results indicate the need for more intelligent metrics for straggler detection – transitioning away from a fixed temporal boundary as defined in [4], [8] towards an adaptive boundary that consider metrics such as task progression, system conditions and job QoS as detailed in [34]. We observe that the online analytics agent produce approximately 0.2 percent CPU usage for both job types, representing a fractional amounts of server usage, and observing no indication of causing an increment in straggler behavior caused by high CPU.

TABLE 6
Comparison of Straggler Detection Approaches

| Approach | Avg. Straggler detection (s) | Straggler detection % | True Positive % |
|---|---|---|---|
| | **HiveQL** | | |
| **Online** | 61 | 93.27 | 100.00 |
| **Offline** | 12.62 | 20.44 | 58.97 |
| **Combined** | 15.84 | 23.24 | 90.63 |
| | **WordCount** | | |
| **Online** | 108.92 | 23.58 | 100.00 |
| **Offline** | 98.3 | 21.37 | 98.76 |
| **Combined** | 219.11 | 47.63 | 100.00 |

## 7.3 Simulation

We also conducted an evaluation to study the advantages of combining offline and online analytics for straggler detection in contrast to each respective component in isolation. This was performed through simulation to study task execution within a larger-scale system. This was performed by simulating Cloud datacenter operation using SEED—an event-based simulator [36] that enables the creation of jobs (comprising multiple tasks) onto a set of machines for execution. In order to simulate task execution, we modeled progress patterns extracted from the HiveQL and WordCount workloads used in the experiments exploiting linear regression. At each simulation time-step the progress of task increments, and will perform straggler detection. We selected three approaches for detection - *online*, *offline* and *combined*. Online was based on the design in [4] defined as *"when a task's progress score is less than the average for its category (map or reduce) minus 0.2, and the task has run for at least one minute, it is marked as a straggler"*. Offline is defined as $T_{iProg} < T_{iS,}$ thus omitting the requirement for comparing against average job progress. Combined is a combination of offline and online, and implemented using the logic described in Section 6.2. 25 jobs of each workload type comprising 500 tasks (with an assigned a straggler probability of 5 percent) were submitted into 500 machines.

Fig. 13 and Table 6 presents the task execution patterns between normal and stragglers for HiveQL and WordCount. We observe in HiveQL that while the online approach is capable of successfully detecting 100 percent of stragglers, it occurs 93 percent into the execution lifecycle of the job. In contrast, the offline approach detects straggler within 20.44 percent, however incurs a false positive rate of 41.03 percent. This is due to major fluctuation of progress patterns proportional to current task progress at the start of the task lifecycle (i.e., progress score changes are magnified). The combined approach is capable of nullifying this fluctuation, resulting in straggler detection in 23.24 percent of the task lifecycle with 90.63 percent accuracy.

In contrast, online and offline detection appears to achieve greater detection effectiveness for workload that executes for extended periods of time–capable of

100 percent detection accuracy 21-24 percent within the lifecycle. The reason for this behavior is due to longer running tasks will reduce the sudden fluctuation of progress patterns each time-step. This results in the combined approach detecting stragglers just under 50 percent of task execution.

Theses result indicates that the combined approach is effective for jobs with smaller duration due to its ability to minimize false positives and rapid detection. In contrast for longer running tasks it is feasible to focus on online or offline straggler detection. This is reflected by various approaches proposed for speculative execution based on duration or job size [8], demonstrating that there presently does not exist a unified approach for straggler detection.

## 8 Conclusion

This paper presents an empirical analysis of two production large-scale virtualized Cloud datacenters to ascertain the impact and root-cause of stragglers; emergent phenomena found within distributed systems at scale. Findings were leveraged to guide the development of a detection system for extreme stragglers by combining offline analytics and agent based monitoring. The results present key empirical insight for stragglers in large-scale virtualized Cloud datacenters, and can be exploited by researchers for designing their system assumptions based on realistic operation scenarios. Our conclusions are summarized as follows:

*Stragglers Non-Intuitively Impact a Large Proportion of Job within Cloud Datacenters.* Our empirical analysis of two production Cloud datacenters demonstrates that 4–6 percent of total task stragglers affect 37–49 percent of total jobs, impeding their execution between 12-865 seconds. With the evolving trend of computing systems growing in complexity and scale, such findings demonstrate the threat that stragglers phenomena imposes towards guaranteeing virtualized service performance in next generation systems.

*Straggler Root-Cause Stems from Numerous Faults—Predominately from High Server Resource Utilization.* Our analysis of a 12,000+ node production system indicates that stragglers are produced from numerous underlying faults including hardware faults, data skew, and network packet loss. Importantly, 53 percent of straggler root-cause is resultant of high CPU and disk server utilization stemming from unbalanced workload aggregation and inefficient user code.

*Research Into Automated Straggler Root-Cause Analysis is Urgently Required.* We discuss in detail the practical consideration and current limitations in straggler root-cause analysis. It is presently not possible for automated analysis due to the requirement of expertise in exploring and correlating heterogeneous sub-system trace data (kernel, application, server, etc.) combined with tacit knowledge of technical staff for identifying faults. This process is labor and resource intensive and outlines an open challenge within the straggler community to accelerate this process.

*Holistic Usage of Offline and Online Analytics is Capable of Detecting Extreme Straggler Behavior at Runtime.* Through combination of offline and online agent based analytics, we demonstrate through experiments that it is possible to identify 95 percent of task stragglers approximately 11 percent into a tasks execution for short lived jobs. The approach is capable of minimizing false positives for straggler detection caused by sudden fluctuation in task progress scores, which appears to be less of a concern for longer running jobs.

Future work includes integration of our approach into established straggler mitigation techniques including speculative execution to discover whether we can achieve substantial gains in job completion timeliness and system QoS. Furthermore, we intend to propose a means to accelerate the process for root-cause analysis through machine learning to cross-correlate heterogeneous system traces for more intelligent failure detection such as [35].

## Acknowledgments

The work is supported by National Key Research and Development Program (2016YFB1000503), China 863 program (2015AA01A202), China Natural Science Foundation (61421003), and the Beijing S&T Committee - Beijing Brain Inspired Computing Program in BCBD Innovation Center. Renyu Yang is the corresponding author.

## References

[1] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] M. Isard, M. Mihai, Y. Yu, A. Birrell, and D. Fetterly, “Dryad: Distributed data-parallel programs from sequential building blocks,” *ACM SIGOPS Operating Syst. Rev.*, vol. 41, no. 3, pp. 59–72, 2007.

[3] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. “Spark: Cluster computing with working sets,” *USENIX Conf. Hot Top. Cloud Comput.*, vol. 10, pp. 10–10, 2010.

[4] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, “Improving MapReduce performance in heterogeneous environments,” in *Proc. 8th USENIX Conf. Operating Syst. Des. Implementation*, 2008, p. 29–42.

[5] M. García-Valls, T. Cucinotta, and C. Lu, “Challenges in real-time virtualization and predictable cloud computing,” *J. Syst. Archit.*, no. 9, pp. 726–740, 2014.

[6] J. Dean and L. A. Barroso, “The tail at scale,” *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[7] S. Huang, T. Huang, S. Lyu, C. Shieh, and Y. Chou, “Improving speculative execution performance with coworker for cloud computing,” in *Proc. IEEE Int. Conf. Parallel Distrib. Syst.*, 2011, pp. 1004–1009.

[8] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, “Effective straggler mitigation: Attack of the clones,” in *Proc. 10th USENIX Conf. Netw. Syst. Des. Implementation*, 2013, vol. 13, pp. 185–198.

[9] G. Ananthanarayanan, et al., “Reining in the outliers in map-reduce clusters using Mantri,” in *Proc. 9th USENIX Symp. Operating Syst. Des. Implementation*, 2010, vol. 10, no. 1, Art. no. 24.

[10] Q. Chen, C. Liu, and Z. Xiao, “Improving mapreduce performance using smart speculative execution strategy,” *IEEE Trans. Comput.*, vol. 63, no. 4, pp. 954–967, Apr. 2014.

[11] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu. “An approach for characterizing workloads in google cloud to derive realistic resource utilization models,” in *Proc. IEEE Int. Symp. Serv.-Oriented Syst. Eng.*, 2013, pp. 49–60.

[12] E. Bortnikov, A. Frank, E. Hillel, and S. Rao, “Predicting execution bottlenecks in map-reduce clusters,” in *Proc. USENIX Conf. Hot Top. Cloud Comput.*, 2012, pp. 18–18.

[13] K. Wang, B. Tan, J. Shi, and B. Yang, “Automatic task slots assignment in hadoop mapreduce,” in *Proc. ACM Workshop Archit. Syst. Big Data*, 2011, pp. 24–29.

[14] Q. Chen, D. Zhang, M. Guo, Q. Deng, and S. Guo, “SAMR: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment,” in *Proc. IEEE Int. Conf. Comput. Inf. Technol.*, 2010, pp. 2736–2743.

[15] N. Yadwadkar, G. Ananthanarayanan, and R. Katz, “Wrangler: Predictable and faster jobs using fewer resources,” in *Proc. ACM Symp. Cloud Comput.*, 2014, pp. 1–14.

[16] J. Li, N. K. Sharma, D. Ports, and S. Gribble, “Tales of the tail: Hardware, OS, and application-level sources of tail latency,” in *Proc. ACM Symp. Cloud Comput.*, 2014, pp. 1–14.

[17] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "A study of skew in mapreduce applications," *Open Cirrus Summit*, 2011.

[18] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "Skewtune: Mitigating skew in mapreduce applications," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2012, pp. 25–36.

[19] J. Rosen and B. Zhao, "Fine-grained micro-tasks for MapReduce skew-handling," *White Paper*, Univ. Berkeley, 2012.

[20] J. Lin, "The curse of Zipf and limits to parallelization: A look at the stragglers problem in mapreduce," in *Proc. Workshop Large-Scale Distrib. Syst. Inf. Retrieval*, 2009, vol. 1, pp. 57–62.

[21] P. Patel, A. H. Ranabahu, and A. P. Sheth, "Service level agreement in cloud computing," in *Proc. Cloud Workshops OOPSLA*, 2009.

[22] P. Garraghan, X. Ouyang, P. Townend, and J. Xu. "Timely long tail identification through agent based monitoring and analytics," in *Proc. Int. Symp. Real-Time Distrib. Comput.*, 2015, pp. 19–26.

[23] C. Lin, W. Guo, and C. Lin, "Self-learning MapReduce scheduler in multi-job environment," in *Proc. IEEE Int. Conf. Cloud Comput. Big Data*, 2013, pp. 610–612.

[24] N. J. Yadwadkar and W. Choi, "Proactive straggler avoidance using machine learning," *White paper*, Univ. Berkeley, Berkeley, CA, USA, 2012.

[25] Google, "Google cluster data V2," (2011). [Online]. Available: http://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1

[26] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format+ schema," Google Inc., Tech. Rep., 2011.

[27] Tsar tools, (2012). [Online]. Available: https://github.com/alibaba/tsar/

[28] Nagios, (2009). [Online]. Available: https://www.nagios.org/

[29] I. S. Moreno, P. Garraghan, P. Townend, J. Xu, "Analysis, modeling and simulation of workload patterns in a large-scale utility cloud," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 208–221, Apr.-Jun. 2014.

[30] A. Thusoo, et al., "Hive: A warehousing solution over a map-reduce framework," in *Proc. VLDB Endowment*, 2009, pp. 1626–1629.

[31] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[32] Z. Zheng, T. Zhou, M. Lyu, and I. King, "FTCloud: A component ranking framework for fault-tolerant cloud applications," in *Proc. IEEE 21st Int. Symp. Softw. Rel. Eng.*, 2010, pp. 398–407.

[33] Y. Dai, B. Yang, J. Dongorra, and G. Zhang, "Cloud service reliability: Modelling and analysis," in *Proc. IEEE Pacific Rim Int. Symp. Dependable Comput.*, 2009, pp. 1–17.

[34] X. Ouyang, P. Garraghan, D. Mckee, P. Townend, and J. Xu, "Straggler detection in parallel computing systems through dynamic threshold calculation," in *Proc. IEEE Int. Conf. Adv. Inf. Netw. Appl.*, 2016, pp. 414–421.

[35] Y. Zhang, C. G. Guo, D. Li, R. Chu, H. Wu, and Y. Xiong, "CubicRing: Enabling one-hop failure detection and recovery for distributed in-memory storage systems," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, pp. 529–542, 2015.

[36] P. Garraghan, D. Mckee, X. Ouyang, D. Webster, and J. Xu, "SEED: A scalable approach for cyber-physical system simulation," *IEEE Trans. Serv. Comput.*, vol. 9, no. 2, pp. 118–123, Mar./Apr. 2016.

**Peter Garraghan** is a lecturer in the School of Computing & Communications, Lancaster University. He has industrial experience building large-scale systems and his research interests include distributed systems, cloud datacenters, dependability, data analytics and energy-efficient computing.

**Xue Ouyang** received the BEng degree in network engineering and the MEng degree in software engineering from National University of Defense Technology, China. She is working toward the PhD degree in the School of Computing, University of Leeds. Her research interest is improving service performance within distributed systems.

**Renyu Yang** received the BSc degree from Beihang University, in 2011 and was a visiting researcher with the University of Leeds, United Kingdom, in 2012. He is working toward the PhD degree at Beihang University, China. His research interests include resource management in large scale distributed system, system reliability and data management.

**David McKee** received the MEng degree in computer systems and software engineering from the University of York, United Kingdom. He is working toward the PhD degree in the School of Computing, University of Leeds, United Kingdom. His research interests include real world time service-orientation and large-scale distributed system simulation.

**Jie Xu** is a chair of computing with the University of Leeds and the director of the UK EPSRC WRG e-Science Centre. He has worked in the field of dependable distributed computing for more than 30 years, and is a Steering/Executive Committee member of IEEE SRDS, ISORC, HASE, SOSE, and co-founder of IEEE IC2E. He is a member of the IEEE.