The Journal of Supercomputing (2020) 76:10050–10089
https://doi.org/10.1007/s11227-020-03241-x

Tails in the cloud: a survey and taxonomy of straggler management within large-scale cloud data centres

Sukhpal Singh Gill[1] · Xue Ouyang[2] · Peter Garraghan[3]

Published online: 12 March 2020

Abstract

Cloud computing systems are splitting compute- and data-intensive jobs into smaller tasks to execute them in a parallel manner using clusters to improve execution time. However, such systems at increasing scale are exposed to stragglers, whereby abnormally slow running tasks executing within a job substantially affect job performance completion. Such stragglers are a direct threat towards attaining fast execution of data-intensive jobs within cloud computing. Researchers have proposed an assortment of different mechanisms, frameworks, and management techniques to detect and mitigate stragglers both proactively and reactively. In this paper, we present a comprehensive review of straggler management techniques within large-scale cloud data centres. We provide a detailed taxonomy of straggler causes, as well as proposed management and mitigation techniques based on straggler characteristics and properties. From this systematic review, we outline several outstanding challenges and potential directions of possible future work for straggler research.

Keywords Computing · Stragglers · Cloud computing · Straggler management · Distributed systems · Cloud data centres

✉ Sukhpal Singh Gill
s.s.gill@qmul.ac.uk

Xue Ouyang
ouyangxue08@nudt.edu.cn

Peter Garraghan
p.garraghan@lancaster.ac.uk

[1] School of Electronic Engineering and Computer Science, Queen Mary University of London, London, UK

[2] School of Electronic Sciences, National University of Defense Technology, Changsha, China

[3] School of Computing and Communications, Lancaster University, Lancaster, UK

1 Introduction and motivation

Nowadays, applications spanning various domains including social networks, e-commerce sites, and healthcare generate vast quantities of data. The growing velocity and volume of such data generation has subsequently required the substantial computing capacity in order to store and process such data effectively [1]. Such large-scale computing systems, encompassing data centre clusters, comprise hundreds and thousands of individual machines interconnected together that underpin application operation consumed by both businesses and consumers alike.

A combination of increasing application demand and technological innovations has resulted in greater system scale in the regions of tens of thousands of servers within an individual cluster [2]. However, such complexity has subsequently resulted in an increase in complexity within such systems, manifesting in the form of emergent phenomena whereby system operation exhibits behaviour unforeseen at design time. Such emergent phenomenon manifesting within large-scale cloud data centres has been observed to negatively impact application performance. One such phenomenon, known as the *long-tail problem*, is characterized by a minor subset of task stragglers that operate unusually slower in comparison with normal task behaviour within a job. Task stragglers occur within any highly parallelized system and become even more apparent for jobs containing many tasks executing across a large number of machines.

Frameworks such as MapReduce, Spark, and Dryad [1, 3, 4] process vast quantities of data via parallelizing jobs into a smaller subset of tasks and thus make such applications susceptible to stragglers. For example, within MapReduce, a job can only complete once all tasks have completed their execution. However, the occurrence of stragglers results in an atypically long task execution duration, thus degrading the performance of the entire job. The challenge in effectively addressing stragglers is that their root-cause is not well understood [5] and can be resultant due to various reasons spanning daemon processes, data skew, failures, resource contention, and energy management tools [6, 7], manifesting within the application, operating systems (OS), or physical hardware. This can subsequently lead to subsequent applications that depend on job outputs to also fail pending on its completion [8, 9].

This has resulted in a growing body of straggler research pertaining to analysing their underlying causes [9, 10], straggler forecasting [11, 12], and straggler mitigation techniques [13–16] including speculative execution [17], replication, load balancing, and scheduling [18]. Each of these works predominantly focuses on a certain subset phenomenon within a particular context of system operation of application framework. Thus, straggler research has reached sufficient level of maturity whereby it is worthwhile to appraise the landscape of research within the field, identify cross-cutting challenges within areas, and evaluate future challenges on the horizon for future generation computing systems.

1.1 Motivation

The core motivation behind this methodical survey is to conduct a systematic review of straggler research within large-scale cloud data centres. This systematic

review encompasses clearly defining and analysing the impact of stragglers, a taxonomy of various straggler management techniques for forecasting and mitigations, as well as identify future directions within the field.

1.2 Article organization

The rest of the article is structured as follows: Sect. 2 presents the background information for straggler definition as well as straggler management within large-scale systems. Section 3 presents the taxonomy of straggler causes. Section 4 explores the existing literature for straggler management techniques. Section 5 presents the comparison of straggler management techniques based on the taxonomy of straggler causes and outlines the observation, trend analysis, and future research directions. Finally, Sect. 6 summarizes the article.

2 Background

2.1 Straggler definition and impact

Applications execute within large-scale computing systems such as data centres and clusters by submitting jobs via a resource manager (YARN, Mesos, Borg, etc.). In this context, a job is composed of multiple smaller tasks (defined as the smallest unit of computation observable by the resource manager) [19]. Such jobs and subsequent tasks are scheduled onto different machines in a parallelized manner to accelerate job completion and are often divided into phases creating a direct acyclic graph (DAG) [20]. Application frameworks (such as MapReduce) attempt to sub-divide jobs so that tasks will approximately complete within the same timeframe for each phase [21]. This is achieved by providing a subset of data (known as shards) to each task, and allocating the appropriate resources to tasks (CPU, memory, etc.). This is calculated via the resource requirement module of the resource manager [22].

However, even with such measures in place, within large-scale cloud data centres a subset of tasks within a job will manifest as stragglers [23, 24]. In this context, a straggler is defined as task which execute abnormally slow in comparison with the average task duration within a job [2]. The phrase ‘abnormally slow’ is typically identified as any task with a task completion time 50% greater than the (average) task completion time for a job phase [25, 26]. Slowly executing tasks (stragglers) affect the performance and completion time of the entire job [14], increasing resource utilization and performance degradation of applications at increased scale [27, 28], thus reducing system availability and incurring additional operational costs [29]. It has been identified from analysis of production systems at scale [28] that approximately 4–6% of task stragglers negatively affect over 50% of the overall jobs within the greater system.

2.2 Straggler management

Due to the impact of long-tail problem within distributed computing systems, there have been concentrated efforts in order to effectively mitigate their effects. This has been tackled by the research community via the creation of various straggler management techniques. In this context, straggler management comprises all mechanisms that have been created in order to mitigate the effects and impact of straggler manifestation. Figure 1 shows the depiction of straggler tasks and non-straggler tasks.

Such straggler management techniques can be predominantly considered into two main classes: detection and mitigation [30, 31]. *Detection* focuses on approaches to identify straggler manifestation a priori or post-priori job execution within the cloud data centre, such as offline analytics and online monitoring mechanisms [32, 33] and an example of straggler detection is NearestFit [1]. *Mitigation* approaches focus on avoiding [34] or tolerating (detected) straggler manifestation during job execution such as scheduling, load balancing, and replication [26, 35, 36]. The examples of straggler mitigation are Dolly [13], GRASS [14], LATE [16], and Wrangler [15].

2.3 Related surveys and our contributions

To present day, to the best of our knowledge, only two works have conducted a survey pertaining to straggler research. Umesh and Jitendar [37] discussed an overview of straggler handling algorithms for MapReduce framework, while Ashwin et al. [38] reviewed several straggler handling techniques. While these reviews cover specific cases of stragglers related to specific frameworks and installations, they do not necessarily provide a comprehensive survey of the straggler causes and straggler management techniques which exist within the research community. Furthermore, these works do not discuss in detail the precise root-causes and analysis of straggler behaviour, which underpin the design of straggler management techniques. Therefore, this paper attempts to provide a systematic review and taxonomy of straggler causes and map them directly to straggler management techniques along with trend analysis.

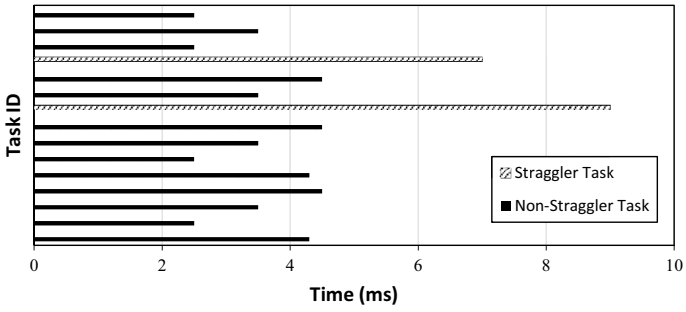

Fig. 1 Depiction of straggler tasks and non-straggler tasks

3 Taxonomy of straggler causes

As mentioned in Sect. 1, the challenge within this research area is the myriad of potential causes of straggler manifestation. According to our comprehensive appraisal of the literature, we have identified eight key causes for straggler occurrence that manifest within large-scale cloud data centres. Figure 2 shows the taxonomy of straggler causes.

1. *Data abstraction* Stragglers can occur due to information obfuscation at different levels of the system. The literature [39–42] has identified that information can be hidden at two different levels: (i) OS level and (ii) application level. During the execution of resources, the master node (controller) hides information from workers (cluster nodes) at *OS level*. (ii) At *application level*, the information regarding platform services and infrastructure services is kept hidden from the software services.
2. *CPU utilization* It has been identified that there is a strong correlation between high system CPU utilization and straggler occurrence [7, 12, 43, 44]. The reason for this occurrence is resource contention. This is further compounded due to Head-of-Line blocking (HOL blocking), task interference during execution, busy locks, queue issues, hazard rates of task execution and launching additional *speculative* replicas, which requires *additional time* for execution.
3. *Scheduling* It has been identified that scheduling and resource allocation decisions also influence straggler manifestation [45–48]. For *job scheduling*, stragglers can occur due to a large number of enqueued jobs within a (machine, master scheduler) that are pending for available resources to be revoked (i.e. only a portion of tasks within a job are able to successfully acquire their necessary resources to commence execution). Furthermore, straggler may occur due to the poor admission control mechanisms, which is used to submit the jobs for execution [49]. The poor admission control mechanism launches multiple tasks together, resulting in resource exhaustion causing slowdown. Lastly, dynamicity of QoS requirements at runtime results in an inability to effectively manage the resources which leads to further the straggler occurrence. In terms of *resource scheduling*, stragglers can occur in following situations [49–52]: (1) when resources are allocated to the jobs in an inefficient manner without available resource optimization, leading to ineffective scheduling of resources for job execution and (2) sometimes resources are still in active stage even they are not utilized for execution of jobs, which consumes more energy and affects the performance of other resources because some resources need more power to run continuously.
4. *Inaccessible local disk* Stragglers may occur when a machine hard disk is not accessible to residing tasks. Such inaccessibility is predominantly caused by [9, 53–59]: (i) increasing backup tasks and (ii) failing to store output. Stragglers can occur, when it is difficult to find the required task due to the large backlog of the tasks waiting for execution. Sometimes, an error can occur while storing the output on the disk, causing a problem when some tasks want to access those data during execution.

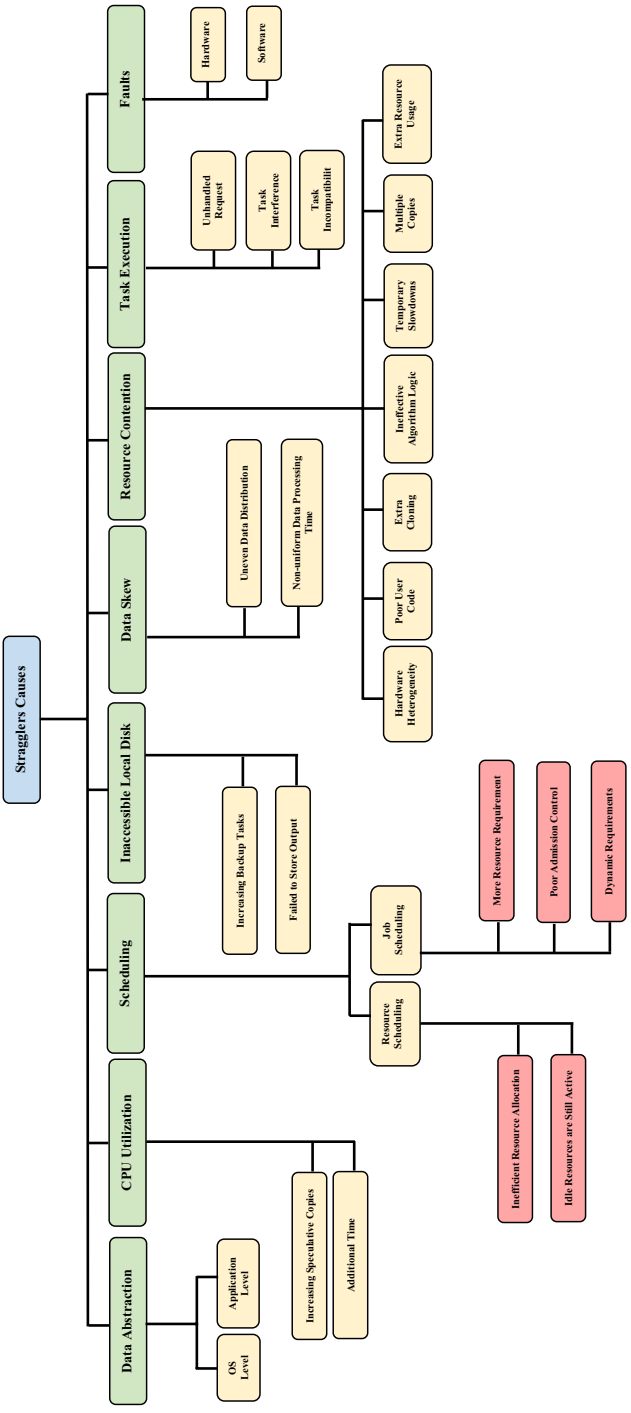

Fig. 2 Taxonomy of straggler causes

5. *Data skew* Straggles can occur due to the data skew, caused by the different data sizes and time variation in accessing required data [56, 57, 60, 61]. With several tasks operating on a split version of a very large shared dataset, an uneven distribution of the data amongst these tasks potentially results in some tasks to progress slowly in comparison with tasks within the same phase (and subsequently delays the future sub-phases and the entire job). Data non-uniformity can also impact data access and processing time data, directly affecting the timing delays between tasks, further increasing the probability of straggler occurrence. Moreover, data locality for job execution results in lower latencies, while distant data will take longer to be accessed, incurring additional delays in task completion, again, manifesting as a straggler.
6. *Resource contention* Resource contention occurs when the same resource is shared by multiple tasks [9, 13, 14, 17, 53–55, 58, 59, 62–69]. Resource contention occurs due to conflict over task access and oversubscription to a resources within multi-tenant machines which can be exuberated within different scenarios including: (1) hardware heterogeneity, (2) poor user code, (3) extra cloning, (4) ineffective algorithm logic, (5) temporary slowdowns, (6) additional task clones requiring more resources and (7) resource usage being higher than accepted threshold value. *Hardware heterogeneity* is the main reason of resource contention, which occurs due to a mismatch between hardware specification and specified application constraints (e.g. budget, deadline, etc.) leading to task performance degradation. The *source code* of scheduling algorithm also affects the performance of the scaling system due to its coding style in terms of space and time complexity. Sometimes, poorly written source code schedules resources inefficiently, which can increase resource consumption and unavailability of required resources to specific jobs [35]. The *cloning* of tasks is creating a similar of copy to task to run parallel on another resource for fast execution.

   The *cloning* of tasks needs more resources (increases resource usage), which can also put tasks of other jobs on hold and when the tasks are waiting for other resources, then stragglers can occur. An *ineffective logic* in the resource scheduling algorithm can also lead to an inefficient allocation of resources and increase resource usage, which leads to resource contention for future tasks. Temporary slowdown can occur due to inefficient allocation of resources, which needs to be corrected; otherwise, it will cause straggler occurrence during execution of resources.
7. *Task execution* The successful execution of a task is important to avoid straggler occurrence during execution of jobs [10, 28, 70–74]. During job execution, stragglers can occur due to unhandled requests or ineffective task interference and task incompatibility management. When a processing request is unhandled or not fully handled, tasks expecting the results of this request will have to wait until the full request output is ready, manifesting in straggling tasks. This occurs due to data dependency and task dependency. If the tasks are not oblivious to the heterogeneity of the underlying resources of the platform, their incompatibility (non-synchronization) due to different types of workloads or requirements can manifest in slower execution and ultimately straggler occurrence.

8. *Faults* Faults within software and hardware resulting in to crash-stop and late-timing failure can cause straggler occurrence in large-scale systems [17, 18, 63, 64, 75, 76]. The main reasons for software-induced faults can be: development, logic or overflow errors as well as misconfigurations. In terms of hardware, the main fault occurrence reasons are: physical damage, device failures, daemon processes, or power-related issues such as effective energy management. Ironically, fault tolerance and recovery mechanisms can themselves result in straggler manifestation (for example, checkpointing introduces burst in disk access and increases resource contention, resulting in a higher system hazard rate).

3.1 Relationship between straggler causes

Based on different types of causes of stragglers in large-scale systems, we have identified the correlation among them, as described in Table 1. As identified in [28], stragglers are not resultant of a singular cause, but can potentially be correlated. For example, data abstraction can occur due to tasks in a queue waiting for execution. Resource contention is the main reason of stragglers due to the sharing of resources among different applications, which are running on different nodes, which further affects the CPU utilization by overloading the resources. Straggler occurs during scheduling of jobs as well as resources, and the reasons for straggler occurrence during resource scheduling can be heterogenous resources, poor user code or logic error, and too many copies of straggler tasks that are running simultaneously. The reasons for inaccessible local disk can be large copies of backup tasks and failing to store required output, which happens due to task interference and its incompatibility with other tasks. The other reason can be that requirements are changing dynamically. Data skew happens due to straggler occurrence at application level due to data hiding or failing to write data. The other reason can be inefficient allocation of resources for processing of data, which can increase running time of resource. The resource contention occurs at OS level, when master node hides the information from workers. Further, the overutilization of CPU causes the resource contention due to increasing speculative copies as well as when the performance of node degrades. Moreover, poor admission control can also affect the resource utilization and creates resource contention when the value of required resources is increased than the available resources. Further, resource contention affects the task execution due to unavailability of shared resources. Fault occurrences during job execution can happen due to resource failure and resource misconfiguration [77].

4 Straggler management techniques: current status

Straggler management techniques can be categorized into two broad categories: straggler detection and straggler mitigation [78]. Each category can be further subdivided into specific areas as shown in Fig. 3.

Table 1 Correlation among straggler causes

Stragglers causes	Dependent
Data abstraction	
1. OS level	Resource contention (1), resource contention (6), inaccessible local disk, task execution
2. Application level	
CPU utilization	
1. Increasing speculative copies	Resource contention, faults (1), scheduling (2b), scheduling (1c)
2. Additional time	
Scheduling	
1. Job scheduling	Resource contention (1), resource contention (2), resource contention (3), resource contention (4), faults
a. Number of jobs more than available resources	
b. Poor admission control mechanism	
c. Dynamic requirements	
2. Resource scheduling	
d. Inefficient resource allocation	
e. Idle resources are still active	
Inaccessible local disk	
1. Increasing backup tasks	Task execution (1), task execution (2), task execution (3), scheduling (1c)
2. Failing to store output	
Data skew	
1. "Uneven data distribution among tasks"	Inaccessible local disk (2), data abstraction (2), scheduling (1b)
2. "Non-uniform data processing time"	
Resource contention	
1. Hardware heterogeneity	Data abstraction (1), CPU utilization, inaccessible local disk (1), data skew (1), task execution, scheduling (2a)
2. Poor user code	
3. Extra cloning	
4. Ineffective algorithm logic	
5. Temporary slowdowns	
6. More number of copies of same task needs more resources	
7. Resource usage is more than threshold value	
Task execution	
1. Unhandled request	Resource contention (1), resource contention (2), scheduling (1a)
2. Task interference	
3. Task incompatibility	
Faults	
1. Hardware	Resource contention (6), resource contention (7), task execution (1), task execution (2)
2. Software	

4.1 Straggler detection techniques

Straggler detection techniques are leveraged in order to identify straggler occurrence during job execution.

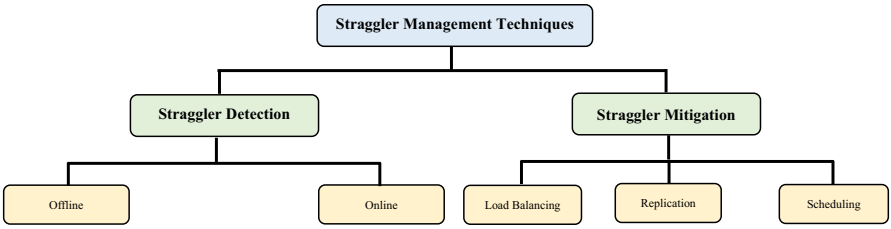

Fig. 3 Taxonomy of straggler management techniques

4.1.1 Offline straggler detection

Offline straggler detection technique attempts to identify straggler manifestation in order to enhance speculative execution via leveraging offline analytics (i.e. analysing and modelling task execution and progress patterns derived from empirical data a priori execution).

Coppa and Finocchi [1] identified three different challenges such as straggling tasks, load unbalancing and data skewness, which affects the performance of computing systems. To overcome these challenges, authors proposed a profile-guided progress indicator called NearestFit to gather the required combination of closest neighbour regression using statistical curve fitting approach. NearestFit is mainly suitable for long running applications and helps to identify the above discussed challenges to increase the efficiency of computing systems. Authors implemented the NodeIterator triangle counting algorithm using homogeneous clusters in Hadoop to test the capability of NearestFit dynamically in terms run time and progress.

Ouyang et al. [70] proposed a technique for modelling and ranking node-level stragglers (MRNLS) in CDCs based on analysing the execution trace log data of parallel jobs. This was conducted by a graph-based algorithm that is used to partition the server nodes into small nodes to execute more jobs in parallel. The proposed techniques improve the performance of computing systems by reducing task stragglers occurrence. Cong et al. [72] proposed a machine learning-based straggler detection (MLSD) technique using unsupervised clustering method. The proposed technique effectively manages the resources while executing the jobs and diagnosing the stragglers at runtime. Wei et al. [10] proposed straggler detection approach (SDA) for data-intensive computing in cloud environment to detect stragglers at early stage to preserve the efficiency of the CDC. Further, statistical method for outlier detection called Turkey is developed to detect straggler at run time because it starts the speculative execution earlier than the standard deviation method.

4.1.2 Online straggler detection

Online straggler detection technique detects the straggler to improve speculative execution using online monitoring tools.

Farshid [79] analysed that map phase of MapReduce (MR) framework takes longer with the increase in the number of servers, which further affects negatively the execution time of MapReduce job. Moreover, authors designed an analytical model to identify the impact of stragglers on efficiency of computing system using map phase in terms of application, system, and hardware parameters. Experimental results show that model reduces the execution time during execution of MapReduce applications. Zaharia et al. [80] proposed a resilient distributed datasets (RDDs), a distributed memory abstraction, which enables developers to provide a fault-tolerant module while performing in-memory computations on a huge number of clusters. RDD uses coarse-grained transformations to offer controlled form of shared memory to perform different memory-intensive computations in an iterative manner. Further, Spark is used to implement RDDs in a controlled environment to evaluate its performance.

Wang et al. [17] proposed heuristic algorithm (HA) to search for the best replication to reduce latency in computing systems. The proposed algorithm is used to implement the proposed algorithm, and experimental results demonstrate that this is capable of reducing latency and its impact on cost of execution of workloads. Jeffrey and Sanjay [54] explored data processing on large clusters (DPRCs) to perform different aspects such as (1) providing fault tolerance by distributing computations, (2) optimizing network bandwidth by decreasing the quantity of data transferred throughout the network, and (3) decreasing impact of slow machines and improving fault tolerance. In DPRC [54], speculative copy of task is executed by MapReduce on another node for increasing job completion time and reducing response time. It is challenging to select the task for which to execute speculation because it is not trivial to identify the machine or node, which is running slower than average. To implement DPRC effectively, stragglers are recognized at the earliest possible stage calculated by progress scores.

Garraghan et al. [28] explored the root-cause of stragglers (RCS) and provided a method to analyse the root-cause analysis in a massive scale virtualized CDCs to solve the long-tail challenge effectively. Authors used online analytic agents and offline execution patterns modelling for straggler detection while monitoring tasks dynamically. Heecheol et al. [81] proposed secure distributed computing (SDC) approach using recovery threshold value to efficiently deal with the impact of straggling [82], which uses polynomial codes on sub-tasks allocated to nodes.

4.2 Straggler mitigation techniques

Straggler mitigation technique comprises all mechanisms and approaches to tolerate or avoid the impact of straggler manifestation. Such techniques can be further sub-divided into three sub-categories [68, 83–85]: load balancing based, replication based, and scheduling based.

4.2.1 Load balancing-based straggler mitigation

Load balancing-based straggler mitigation technique manages the load during mitigation of stragglers.

Ouyang et al. [2] proposed a method to reduce late-timing failure (LTF) and analyse the root-cause of stragglers in cloud data centres (CDC) such as server failures or task concurrency and resource contention. Further, this study identified the high temporal resource contention as a main root-cause of stragglers. Further, the output of experiments demonstrates that this technique maintains the efficiency of the computing systems while tolerating the system failures effectively. Yanfei et al. [86] proposed a user transparent task slot management approach called *FlexSlot*, which identifies the stragglers automatically and resizes their slots to improve the speed of execution of task. The approach also balances the usage of resources by automatically changing the number of available slots of nodes to improve its utilization. Moreover, FlexSlot uses adaptive speculative execution approach to improve mitigation of skew data.

Neda et al. [71] proposed log-assisted straggler-aware (LASA) I/O scheduler for high-end computing to mitigate the impact of storage server stragglers. Further, a scheduling algorithm is proposed to make effective decisions to manage stragglers at runtime. The output of experiments demonstrate that LASA is performing better in load balancing while mitigating the storage server stragglers dynamically. Eman et al. [62] proposed a parallel model for straggler mitigation in distributed spatial simulation called priority asynchronous parallel (PAP) to exploit data dependencies of parallel processes to be computed and synchronized based on data priority to the other workers. Moreover, load balancing and partitioning method are proposed to balance the workloads among different nodes and help to improve the performance speedup by a large extent. Haozhao et al. [87] proposed heterogeneity-aware gradient coding (HGC) scheme to execute the jobs in heterogenous environment and efficiently tolerate the stragglers without degrading the effectiveness of the cloud services [34]. The output of experiments demonstrates that HGC scheme outperforms in computation time.

4.2.2 Replication-based straggler mitigation

Replication-based straggler mitigation technique replicates the adequate number of tasks during mitigation of stragglers.

Mehmet et al. [8] analysed the trade-off between latency and cost (TLC) using simple replication or erasure coding for straggler mitigation in executing jobs with many tasks. Experimental results show that delaying redundancy is not effective in reducing cost. Further, Mehmet et al. [55] developed a straggler mitigation (SM) technique using delayed relaunch of tasks, which helps to reduce cost and latency effectively. Wang et al. [9] proposed an idea of an efficient task replication technique (TRT) for straggler management to improve the response time in parallel computations. This technique is implemented in [88] and demonstrates empirically that replicating all operations can result in significant mean and tail latency reduction in real-world systems including domain name system (DNS) queries, database servers, and packet forwarding within networks.

Tien-Dat [11, 64] proposed energy-efficient straggler mitigation (EESM) technique for effective management of big-data applications in the cloud computing environment to optimize the energy consumption during straggler occurrence.

Firstly, authors characterize the effect of straggler mitigation on energy efficiency. Secondly, a straggler detection framework is developed, and they identified that only 12% of the detected tasks are real stragglers [64]. The usage of huge number of speculative copies is the main reason for unnecessary energy consumption. Thirdly, a reservation-based straggler handling approach is proposed to optimize the energy efficiency by allocating the required resources at runtime effectively.

Wang et al. [89] analysed the trade-off between latency and cost to find out the best replication technique for straggler management based on following parameters: (1) when to perform replication for straggling tasks, (2) number of replicas to be launched, and (3) is it necessary to destroy the original copy or not. Further, a straggler management approach (SMA) is proposed to calculate the value of latency-based empirical distribution of execution time of task. The output of experiments demonstrates that this work gives better for two performance parameters such as cost and latency. Lei et al. [90] proposed a straggler management technique called Combination Re-Execution Scheduling Technology (CREST) for fast speculation of straggler tasks in MapReduce framework, which further reduces the response time of MapReduce jobs. The re-execution of set of tasks on set of computing nodes in CREST improves the speed of task execution.

Radheshyam et al. [91] proposed a job-aware scheduling (JAS) technique to optimize the running time of different jobs by maintaining the harmony among them, which are executing on the same cluster. JAS technique is implemented using for MapReduce framework. Further, proposed algorithm selects the most compatible task with executing task to reduce more execution time. Moreover, a heuristic-based load balancing technique is developed to avoid the underloading and overloading of resources. Matei et al. [16] explored the MapReduce framework for straggler management and improved its performance in heterogenous environment. Further, a resource scheduling algorithm, longest approximate time to end *(LATE)* is proposed to improve the robustness in regard to heterogeneity and improves response time of tasks. LATE scheduling algorithm [80] estimates the longest approximate time and select the task with the longest approximate time as straggler tasks and execute its speculative copy on another fast node to speed up the job completion time. SAMR scheduling technique [18] computes the completion of tasks at runtime and discovers the straggler task based on execution time. Historic information of node is used to detect more reliable node in SAMR and weights of reduce and map stages are updated after completion of every task.

Farhat et al. [63] proposed a straggler management technique for modelling and optimization (SMMO) of straggling mappers to show the stochastic behaviour of mapper nodes and its negative effect on completion time of MapReduce jobs. Authors identified task inter-arrival time of jobs to map the required nodes of heterogenous CDC in an optimized way. The experimental results demonstrate that the proposed technique reduced the execution time of jobs at runtime. Behrouzi-Far et al. [92] proposed an efficient straggler replication framework in large-scale parallel computing to analyse the performance of the system in terms of latency–cost trade-off. Further, it identifies the best replication technique based on different criteria such as: (i) number of replicas required, (ii) time to replicate straggling tasks,

and (iii) determine whether to kill the original task. Finally, performance evaluation is described that latency and cost are reduced in Google Cluster Trace as compared to MapReduce.

4.2.3 Scheduling-based straggler mitigation

Scheduling-based straggler mitigation technique schedules the resource for jobs during mitigation of stragglers.

Ananthanarayanan et al. [13] explored the straggler mitigation techniques and identified the impact of reasons of stragglers in latency-sensitive jobs. Further, authors designed workloads with small number of jobs and performed cloning of small jobs. It has been identified that the cloning of small jobs uses less resources but improves the reliability of computing services. Moreover, a system named *Dolly* is developed to generate multiple clones of jobs and execute jobs within their specified budget. Experimental results demonstrate that Dolly sped up jobs by 46% by using only 5% extra resources.

Ananthanarayanan et al. [14] proposed greedy speculative scheduling and resource-aware speculative scheduling (GRASS) technique, which uses speculation to mitigate the impact of stragglers in approximation jobs. GRASS uses extra resources for speculation and improves accuracy for deadline-bound jobs by 47% and speeds up error-bound jobs by 38%. Aaron et al. [53] addressed the straggler problem for iterative convergent parallel (ICP) machine learning technique to identify the behaviour (in terms of delay) of the system during execution of jobs by injecting the stragglers. Amazon EC2 and Microsoft Azure [93] are used to evaluate the performance of system in terms of execution time.

Ouyang et al. [25] proposed a straggler management technique (SMT) to find the task stragglers by calculating threshold value at runtime. Further, this technique considers important key parameters such as resource utilization, task execution, and job QoS timing constraints to manage straggler tasks effectively. Neeraja et al. [15] proposed straggler management technique called *Wrangler* to proactively avoid the conditions, which cause stragglers. Wrangler [13] uses interpretable linear modelling approach to reduce the resource wastage by eradicating the requirement for replicating tasks. It uses fewer resources to complete the job in a faster way and avoids the straggler proactively by predicting in advance. A cluster resource utilization-based statistical learning technique is used for confidence measure to offer reliable task scheduling by predicting errors in advance. The output of experiments shows that Wrangler produces improvements in terms of job completion time and resource utilization as compared to speculative execution.

Quan et al. [18] proposed a self-adaptive MapReduce (SAMR) scheduling technique for straggler management, which estimates task progress automatically and adapts to the changing conditions of environment dynamically. SAMR uses MapReduce mechanism to divide jobs into tasks and execute on different available nodes. SAMR does not create backup tasks for regular tasks. SAMR reduces the execution time of MapReduce jobs while executing tasks in heterogenous environment. Enhanced SAMR (ESAMR) [27] uses the k-means clustering algorithm to categorize the historic data of each node into k-clusters and identifies the straggler task more

accurately. Furthermore, ESAMR uses weights of reduce and map stages to find the *Time to End* on different nodes, which can easily identify the more reliable node.

Ananthanarayanan et al. [27] studied and explored the straggler management in resource-aware techniques and identified the main causes of stragglers such as varying bandwidth, network congestion, workload imbalance, and contention of resources (network, memory, and processor). Furthermore, *Mantri* [27] is used to monitor task execution and take a proactive action to sustain the efficiency of the CDC in the case of resource contention or hardware/software failure [94–96]. It uses Bing traces to evaluate the performance, and it improves job completion time to a large extent.

Ouyang et al. [97] proposed a straggler management mechanism (SMM) to improve the execution efficiency of Internet-ware applications by dynamically calculating the straggler threshold, considering important parameters such as optimal system resource utilization, task execution progress, and job QoS timing constraints. Further, YARN architecture is used to implement dynamic straggler threshold to test the performance of the proposed mechanism and experimental results give the better outcomes in terms of response time. Yan et al. [98] developed large-scale multimedia semantic concept (LMSC) model to improve the scalability of the computing systems with heterogenous environment. Robust subspace bagging algorithm is used to improve learning process, and further, a task scheduling algorithm is proposed to improve the scalability by executing heterogenous tasks. Proposed model is tested on MapReduce framework, and experimental results demonstrate its superiority.

Figure 4 presents the evolution (2008–2019) of different types of straggler management techniques along with their focus of study and QoS. Table 2 shows the comparison of different types of straggler management techniques based on different parameters.

5 Comparison of straggler management techniques based on taxonomy

Table 3 shows the comparison of straggler management techniques based on taxonomy of straggler causes from Fig. 1 and Table 2.

5.1 Analysis of experimental results: practical use-case

The existing straggler management techniques have been categorized into two categories, i.e. straggler detection and mitigation techniques. Table 4 shows the analysis of experimental results of straggler detection and mitigation techniques in the context of different performance parameters. Future researchers can use Table 4 to validate their research work based on the values of various performance parameters identified from the existing literature. The literature reported that there are four types of data abstraction levels (OS, application, server, and VM), where straggler can occur.

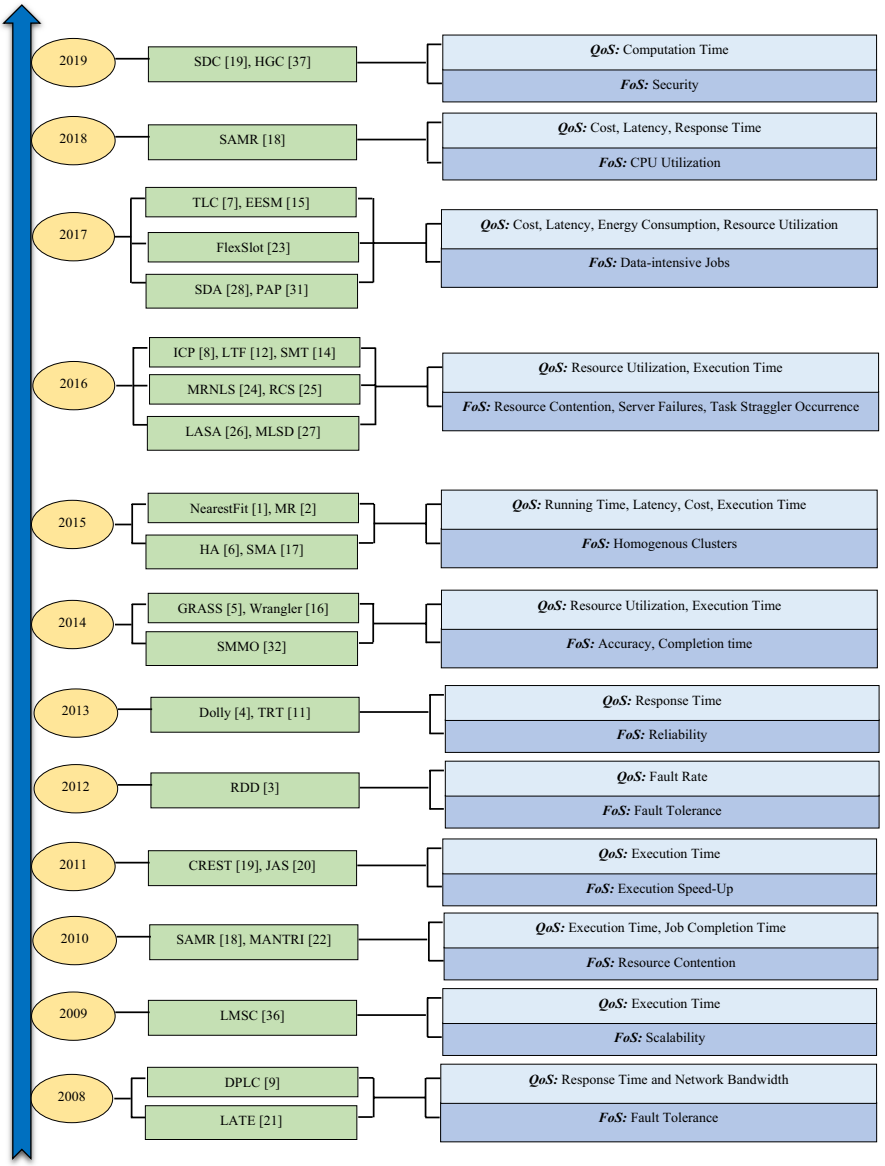

Fig. 4 Evolution of straggler management techniques

5.2 Trend analysis

Our systematic review has identified different types of result outcomes for different categories of straggler management techniques developed from year 2008 to year 2019. The scheduling-based straggler mitigation technique appears prominent across the years except year 2012. After the scheduling-based techniques, researchers

Table 2 Comparison of straggler management techniques

Work	Straggler type	Cause	Environment	Type	Dataset	Delay	Merits	Open challenges
[1]	Slow tasks, non-local task, data skew	Wall-clock times of slow running tasks far from prediction from linear progress assumption	Hadoop	Pro	50 GB archive Wikipedia articles	Low	Higher accuracy in profile optimization	If a job is too short, unable to collect enough profiling data to obtain accurate prediction
[79]	Delay time is larger	Shuffle phase overlaps with the Map phase of MapReduce	Hadoop	Rea	Word Count Data Set (3 GB)	High	Optimized execution time	Inaccurate estimation of time to execution
[80]	Memory utilization	Lack of data sharing abstractions	Spark	Pro	1 TB Wikipedia logs (2 years)	Low	Efficient data recovery	Memory requirements grow as the dataset size increases
[13]	Resource contention	Extra clones cause contention for intermediate data and shared resources	Hadoop	Pro	Facebook and Bing traces	High	46% job speedup, 5% extra CPU	1. Homogenous job sizes required for optimization 2. Extensive analysis needed to determine straggler chance
[14]	CPU utilization	Spawning speculative copy leads to task using two (or more) resources simultaneously	Spark and Hadoop	Pro & Rea	Facebook and Bing traces	Med	Improves deadline job accuracy 47%; error jobs 38%	Weigh the impact of speculating a running task with scheduling a new task of any job

Table 2 (continued)

Work	Straggler type	Cause	Environment	Type	Dataset	Delay	Merits	Open challenges
[17]	Larger response time	Resource contention due to resource sharing (deadlock)	Hadoop	Rea	Google Trace Data	High	"Reduced latency and cost of computing resources"	Develop online strategy to learn execution time distribution to launch replica, instead of using historical traces"
[8]	Delaying redundancy	Delaying redundancy increases latency	NA	Rea	Google Trace Data	High	Analysed trade-off between cost and latency	Degree of redundancy can be reduced without affecting latency
[53]	Transient slowdown of worker thread	Temporary slowdowns (due to resource contention with a background activity) often occurs on non-blacklisted machines	Amazon EC2 and Microsoft Azure	Pro	Netflix dataset (480 k-by-18 k sparse matrix with 100 m elements)	High	Reduced execution time	Resource overloading
[54]	Inaccessible local disk	Completed map tasks are re-executed on failure because output is stored on local disk of failed machine, thus inaccessible	Hadoop	Rea	Google Zeitgeist data	Low	Fault tolerance, locality, optimization load balancing	Redundant execution can be used to reduce the impact of slow machines, but consumes more energy

Table 2 (continued)

Work	Straggler type	Cause	Environment	Type	Dataset	Delay	Merits	Open challenges
[55]	Ineffective CPU utilization	Relaunching tasks before minimum task completion time causes work loss and latency	Hadoop	Pro	Google Trace Data	High	Reduced redundancy, cost, latency	Need of empirical evidence to accurately model execution time for resource load
[9, 88]	Larger response time	Latency in locating small disk file, time needed to load file from disk	Apache Cassandra	Pro	Facebook and Google Trace	High	Reduced latency using redundancy	Ignoring data locality for launching speculative map tasks leads to performance degradation
[2]	Resource contention	Unbalanced workload aggregation and poor user code	Hadoop	Pro	NA	Low	Reduced late-timing failures	Request handling inefficiency is due to overloaded file request
[25]	CPU and memory utilization	Contention of shared resources, node disk failures, and imbalanced task workloads	SEED	Rea	Microsoft Bing traces	High	Improved job completion time	To design a cost function for further optimization of proposed method
[11, 64]	Energy consumption, resource contention	Large resource cost with many speculated tasks, increased energy usage	Hadoop	Pro	MapReduce Application (Wordcount, Cloud Burst)	Med	Improved energy efficiency	Speculative copies can be launched adaptively to improve further the output accuracy

Table 2 (continued)

Work	Straggler type	Cause	Environment	Type	Dataset	Delay	Merits	Open challenges
[15]	Resource contention	Local conditions exceed fixed thresholds defined over set of resource usage statistics	Hadoop	Pro	Facebook 2009, Cloudera Hadoop trace	High	Improved resource utilization, job completion time	Reduce time spent for capturing training data per node in a cluster
[89]	Long task delay	Additional time needed for original copy to finish more likely to be shorter than new copy execution time	Hadoop	Pro	Google Cluster Trace	High	Latency and cost are improved	Technique can be tested on heterogeneous servers, identify task dependencies (some tasks need to complete prior to others)
[18]	Task replication	Larger number of task replication resulting in high overhead	Hadoop	Pro	MapReduce Application (Word-count and Sort)	Low	Reduced execution time	Ignoring of data locality for launching speculative map tasks may lead to severe performance degradation
[90]	High speculative map task execution	Ignoring data locality for launching speculative map task may lead to performance degradation	Hadoop	Rea	Specific gene segment from 50 GB DNA	Low	Reduced running time of a speculative map task	Non-uniform data processing time; performance interference from co-running jobs

Table 2 (continued)

Work	Straggler type	Cause	Environment	Type	Dataset	Delay	Merits	Open challenges
[91]	No synchronization between tasks	Task incompatibility due to their different requirements	Hadoop	Rea	Terasort, grep, webcrawling, WordCount, video conversion	Low	Reduced execution time	Scale-up/down cluster by switching on/off virtual machines or nodes based on resource usage to reduce energy
[16]	Performance degradation in heterogeneous environment	Many speculative tasks may be launched leading to resource exhaustion	Hadoop and EC2	Pro	Facebook and Yahoo traces	Low	Improved task response time	Workload imbalance between workers
[27]	Varying bandwidth, network congestion, workload imbalance and resource contention (network, memory, CPU)	Disk-intensive tasks scheduled to a node with slow disk	Hadoop	Pro	Bing traces	Med	Improved job completion time	Longer communication delays
[86]	High resource contention (CPU utilization)	Uneven data distribution among tasks, non-uniform data processing time; performance interference from co-running jobs	Hadoop	Pro	Wikipedia traces	Med	Data skew mitigation, reduced job completion time	Adaptive speculation execution approach can be developed for resource management in YARN

Table 2 (continued)

Work	Straggler type	Cause	Environment	Type	Dataset	Delay	Merits	Open challenges
[70]	Task-level stragglers during job execution	Lack of correlation between straggler tasks and available slots at system runtime	Hadoop	Rea	Google Trace	Low	Reduced task straggler occurrence	Performance-aware algorithm can improve straggler mitigation, node-level identification
[28]	Resource contention	Hardware heterogeneity	SEED	Pro	Google Trace	Low	Monitored and detected straggler tasks at runtime	Root-cause analysis via machine learning to cross-correlate heterogeneous system traces for intelligent failure detection
[71]	Storage server stragglers	Workload imbalance between workers	Simulator	Rea	Synthetic workloads generated based on real-world traces	Low	Mitigate storage server stragglers dynamically	Develop prototype of integrating proposed I/O scheduling into existing parallel file systems
[72]	Workload imbalance	Datasets on stragglers are expected to demonstrate imbalanced label distribution	Spark	Rea	Google Trace	Low	Runtime straggler detection	Fine-grained straggler identification by per slowest node and cluster
[10]	High job completion time	Late speculative execution on straggler tasks does not reduce job completion time	Simulator	Rea	NA	Low	Runtime straggler diagnosis	Slow task execution (e.g. non-local task and data skew)

Table 2 (continued)

Work	Straggler type	Cause	Environment	Type	Dataset	Delay	Merits	Open challenges
[81]	Data abstraction obfuscation	Master node wants to hide input data from workers	NA	Rea	NA	Med	Reduced straggling effects on sub-tasks	Communication delays and computation delays
[97]	High CPU utilization	Imbalanced workload or uneven input data Sizes	SEED	Pro	Sort, Wordcount, Hive query	Med	Improved response time	Design a cost function beyond CPU and memory utilization, including disk volume and network speed
[62]	Data dependencies of parallel process	Workload imbalance between workers in cloud data centres	SMARTS	Pro	OpenStreetMap dataset	High	Improved performance speedup	High resource overhead associated with increased backups
[63]	Resource contention	Hardware heterogeneity	Hadoop	Rea	Bing, Facebook traces	High	Reduced job execution time	Hardware/software faults identified and removed from task scheduling
[98]	Task-level stragglers during execution of jobs	Hardware heterogeneity	Hadoop	Pro	Google Trace	Med	Improved Scalability	Computation delay is larger
[87]	Resource contention	Hardware heterogeneity	Simulator	Rea	NA	High	Reduced job completion time	Overloading of resources

Pro, proactive; Rea, reactive; high means the value is more than 80%, medium (Med) means the value between 50 and 80%, and low means the value less than 50%

Table 3 Comparison of straggler management techniques based on taxonomy of straggler causes

Work	Data abstraction	CPU util.	Scheduling		Inaccessible local disk	Data skew	Resource contention	Task execution	Faults
			Resource schedule	Job schedule					
[1]	OS level	NA	Inefficient resource allocation	NA	Increased backup tasks	Uneven data distribution	Extra cloning	Task interference	NA
[79]	NA	Additional time	NA	More resource requirement	NA	NA	Hardware heterogeneity	Task interference	Software
[80]	Application level	NA	Idle resources	High resource requirement	Failed to store output	Uneven data distribution	Extra cloning	Task incompatibility	NA
[13]	OS level	NA		NA	NA		Extra cloning	NA	Hardware
[14]	NA	Increased speculative copies	Idle resources still active	Poor admission control	NA	Uneven data distribution	Hardware heterogeneity	Unhandled request	Software
[17]	Application level	NA	Idle resources still active	Poor admission control	Failed to store output	Non-uniform data processing time	Ineffective algorithm logic	Unhandled request	NA
[8]	NA	NA	NA	NA	NA	Non-uniform data processing time	Temporary slowdowns	Task incompatibility	Hardware
[53]	OS level	NA	NA	NA	NA	NA	Temporary slowdowns	NA	Hardware
[54]	NA	NA	Inefficient resource allocation	Dynamic requirements	Failed to store output	Uneven data distribution	Ineffective algorithm logic	Task interference	NA
[55]	NA	Additional time	NA	NA	NA	NA	Extra cloning	NA	Hardware
[9, 88]	Application level	Additional time	NA	NA	Increasing backup tasks	NA	Hardware heterogeneity	NA	Hardware
[2]	Application level	NA	NA	NA	NA	NA	Poor user code	NA	NA

Table 3 (continued)

Work	Data abstraction	CPU util.	Scheduling		Inaccessible local disk	Data skew	Resource contention	Task execution	Faults
			Resource schedule	Job schedule					
[25]	NA	Increasing speculative copies	Idle resources still active	Dynamic requirements	Increasing backup tasks	NA	Ineffective algorithm logic	Unhandled request	Hardware
[11, 64]	OS Level	NA	NA	Dynamic requirements	NA	Uneven data distribution	Multiple copies	NA	NA
[15]	NA	NA	NA	NA	NA	NA	Extra resource usage	NA	Software
[89]	NA	Additional time	NA	NA	NA	NA		NA	Software
[18]	Application level	NA	Idle resources are still active	More resource requirement	Increasing backup tasks	NA	Extra cloning	Unhandled request	
[90]	OS level	Increasing speculative copies	NA	NA	NA	Non-uniform data processing time	Extra cloning	NA	Hardware and software
[91]	OS level	NA	NA	NA	NA	NA	Multiple copies	Task incompatibility	NA
[16]	OS level	Increasing speculative copies	NA	Poor admission control	Failed to store output	NA	Multiple copies	NA	NA
[27]	Application level	NA	NA	NA	NA	NA	Temporary slowdowns	Task incompatibility	Hardware
[86]	NA	NA	Inefficient resource allocation	Dynamic requirements	Increasing backup tasks	uneven data distribution	NA		Software
[70]	NA	Additional time	NA	NA	NA	NA	NA	Task interference	Hardware

Table 3 (continued)

Work	Data abstraction	CPU util.	Scheduling		Inaccessible local disk	Data skew	Resource contention	Task execution	Faults
			Resource schedule	Job schedule					
[28]	NA	NA	NA	NA	NA	NA	Hardware heterogeneity	NA	Hardware
[71]	NA	NA	Idle resources are still active	More resource requirement	Increasing backup tasks	NA	NA	Task interference	Software
[72]	Application and OS level	Increasing speculative copies	Idle resources are still active	Poor admission control	Failed to store output	Non-uniform data processing time	Extra cloning	Unhandled request	NA
[10]	OS Level	Increasing speculative copies	NA	Dynamic requirements	NA	NA	Hardware heterogeneity	NA	Hardware
[81]	OS level	NA	NA	NA	NA	NA	Poor user code	Task incompatibility	NA
[97]	Application level	Increasing speculative copies	Inefficient resource allocation	More resource requirement	NA	Uneven data distribution	Ineffective algorithm logic	NA	Software
[62]	NA	NA	Inefficient resource allocation	Poor admission control	Failed to store output	Non-uniform data processing time	Temporary slowdowns	Task interference	NA
[63]	Application level	Additional time	NA	More resource requirement	Increasing backup tasks	Non-uniform data processing time	Hardware heterogeneity	Task incompatibility	Hardware
[98]	OS Level	Additional time	NA	Poor admission control	Failed to store output	NA	Hardware heterogeneity	NA	Hardware

Table 3 (continued)

Work	Data abstraction	CPU util.	Scheduling		Inaccessible local disk	Data skew	Resource contention	Task execution	Faults
			Resource schedule	Job schedule					
[87]	Application level	NA	Inefficient resource allocation	Dynamic requirements	NA	NA	Hardware heterogeneity	NA	NA

NA not applicable

Table 4 Analysis of experimental results of straggler detection and mitigation techniques

Study	Type	Data abstraction level	Context								
			Number of nodes/workers	Energy consumption	CPU utilization	Disk utilization	Data transfer	Memory utilization	Deadline	Execution time	Number of tasks
[1]	Straggler detection techniques	Application	8–32	NA	>61%	NA	NA	NA	NA	NA	NA
[79]		Server	40–200	NA	NA	NA	NA	>71%	NA	0.5–1000 s	NA
[80]		Application	25–100	NA	NA	NA	NA	NA	NA	0–184 s	NA
[17]		OS	NA	90 kWh	>82%	NA	NA	NA	NA		400–488
[54]		Server	NA	NA	NA	NA	0–20 k MB/s	NA	NA	0–634 s	3351
[70]		Server	132	NA	NA	>40%	NA	NA	NA	50–300 s	NA
[10]		Application	NA	NA	>80%	NA	NA	NA	NA	10–14 m	252,950
[28]		Application	NA	NA	>80%	NA	NA	90%	NA	10–14 m	1,233,879
[72]		Server	12,000	NA	NA	NA	NA	NA	NA		NA
[13]	Straggler mitigation techniques	VM	48	NA	NA	>60%	NA	NA	NA	0–35 s	1–5
[14]		VM	NA	NA	NA	NA	NA	NA	2–20%	30–50%	
[8]		Application	NA	NA	NA	NA	NA	NA	NA	50–110 s	NA
[53]		Server	90	NA	NA	NA	NA	NA	NA	0–100 s	
[55]		Application	NA	NA	NA	NA	NA	NA	NA	50–110 s	NA
[2]		Application	60	NA	>80%	>80%	NA	NA	NA		NA
[9, 88]		Servers	1–10	NA	NA	NA	NA	NA	NA	0–12 s	NA
[25]		Application	20–80	NA	27–135%	NA	NA	27–135%	NA	100–1100	NA
[11, 64]		Application	NA	14–18 10^6 J	NA	NA	NA	NA	NA	12–52 s	1,759,434
[15]		Application	50–100	NA	NA	NA	NA	NA	NA	0–70 s	NA
[89]		Application	NA	NA	NA	NA	2.87 MB/s	NA	NA	NA	NA
[18]		Application	1–8	NA	NA	NA	NA	NA	NA	210–330 s	NA
[90]		VM	100	NA	NA	NA	NA	NA	NA	NA	1–8
[91]		Application	12	NA	NA	NA	NA	NA	NA	NA	NA
[16]		VM	871	NA	NA	NA	52.1 MB/s	NA	NA	10–70 s	NA

Table 4 (continued)

Study	Type	Data abstraction level	Context								
			Number of nodes/workers	Energy consumption	CPU utilization	Disk utilization	Data transfer	Memory utilization	Deadline	Execution time	Number of tasks
[27]		Server	NA	NA	20–80%	NA	NA	NA	NA	0–300 s	NA
[86]		VM	32	60 kWh	NA	NA	NA	NA	NA	0–300 s	NA
[71]		Application	300	NA	NA	NA	NA	NA	NA	20–200 s	NA
[97]		VM	40	NA	NA	NA	11 MB/s	NA	NA	5–85 s	500–10,000
[62]		Application	80	NA	NA	NA	NA	NA	NA	15–80 s	NA
[63]		Server	NA	NA	NA	> 55%	NA	NA	NA	NA	NA
[98]		OS	16	NA	NA	NA	NA	NA	NA	NA	NA
[87]		Application	8–48	NA	> 40%	NA	NA	NA	NA	0–15 s	NA

Study	Type	Data abstraction level	Context								
			Number of jobs	Data size	Latency	Failure prediction accuracy	Number of failures	Average error	Slowdown/ delay	Interval arrival rate	Running time	Response time
[1]	Straggler detection techniques	Application	50–150	NA	NA	NA	NA	2.5–5%	5–15%	NA	0–30 m	NA
[79]		Server	NA	1–3 GB	6.5–9 s	NA	NA	NA	0.5–2 s	10–30 J/s	NA	NA
[80]		Application	NA	100 GB–1 TB	NA	NA	1–119	NA	NA	NA	NA	1.7–6.6 s
[17]		OS	1017	NA	4–8 s	NA	NA		NA	NA	0–3 k s	NA
[54]		Server	29,423	3228 TB	NA	NA	NA	NA	NA	NA	0–1000 s	NA
[70]		Server	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
[10]		Application	3043	NA	NA	NA	NA	NA	NA	NA	29 days	NA
[28]		Application	875	NA	1200–1500 ms	NA	NA	NA	6%	NA	14 days	30 m
[72]		Server	417	NA	NA	62.92%	NA	NA	NA	NA	NA	NA
[13]	Straggler mitigation techniques	VM	1–500	NA	NA	NA	NA	NA	NA	NA	1–6 h	NA

Table 4 (continued)

Study	Type	Data abstraction level	Context									
			Number of jobs	Data size	Latency	Failure prediction accuracy	Number of failures	Average error	Slowdown/ delay	Interval arrival rate	Running time	Response time
[14]		VM	50–500	NA		20–52%	NA	5–30%	NA	NA	NA	NA
[8]		Application	NA	NA	0–120 s	NA	NA	1.5–3.5%	NA	NA	NA	NA
[53]		Server	NA	NA	NA	NA	NA	NA	NA	NA	5–32 s	NA
[55]		Application	NA	NA	60–1600 s	70%	NA	1.5–3.5%	NA	NA	NA	NA
[2]		Application	NA	NA		NA	6–22	NA	8 s	NA	NA	NA
[9, 88]		Servers	700	NA	0–1 s	NA	NA	NA	NA	0–12 req/s	NA	1–1000 s
[25]		Application	NA	24.6 GB	NA	NA	NA	2–3%	4–6 s	NA	NA	680–1080 ms
[11, 64]		Application	1735	20 GB	NA	40–60%	NA	NA	NA	NA	7–14 10^3 s	NA
[15]		Application	22,974	NA	NA	20–85%	NA	NA	0.22–21.84%	NA	9 days	NA
[89]		Application	NA	NA	1750–5200 ms	NA	NA	NA	NA	NA	0–1600 s	18 s
[18]		Application	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
[90]		VM	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
[91]		Application	0–35	64 MB	NA	NA	50–91	NA	11%	NA	0–125 s	NA
[16]		VM	NA	30 GB	NA	NA	NA	NA	NA	NA	0–2.5 s	NA
[27]		Server	NA	NA	NA	NA	NA	NA	NA	20–80 req/s	NA	NA
[86]		VM	NA	150 GB	NA	NA	NA	7–9%	NA	NA	NA	NA
[71]		Application	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
[97]		VM	NA	NA	NA	NA	NA	NA	NA	NA	NA	130–213 s
[62]		Application	NA	NA	NA	NA	82–98	NA	NA	NA	NA	NA
[63]		Server	NA	NA	NA	NA	NA	3–6%	4–9 s	0–25 req/s	NA	NA
[98]		OS	NA	NA	NA	60–70%	NA	NA	NA	NA	NA	NA
[87]		Application	NA	NA	NA	NA	NA	<2%	4 s	NA	NA	50 s

NA not available; s seconds; ms milliseconds; GB giga bytes; kWh kilo Watt hour; MB/s mega bytes per second; req/s number of requests per second; m minutes

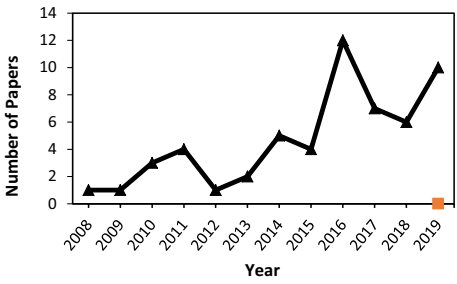

Fig. 5 Publications of straggler management techniques

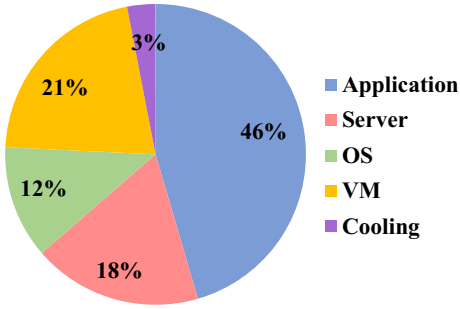

Fig. 6 Straggler-type breakdown in the literature

focused on replication-based straggler mitigation, during the year 2013 to 2019. The offline, online, and load-balancing straggler management techniques are less focused on from year 2008 to year 2019 requiring research to improve the straggler management in large-scale systems. Researchers focused on scheduling and replication-based straggler management in years 2018 and 2019. Figure 5 shows the year-wise publications of straggler management techniques, and it has been clearly depicted that research from year 2008 to 2016 was highly progressive in this area, declining after 2017 and 2018 while progressing in 2019.

The literature reports the research related to straggler management is mostly published in journals (31%), followed by conferences (28%), transactions (21%), and book chapters (10%). The rest of the research is published in symposiums, workshops, white papers, and PhD thesis. Figure 6 shows the research conducted related to straggler management at different levels such as Application, Server, OS, VM, and cooling. Figure 6 clearly shows that most of the research work has been done at the application level (46%) and followed by VM level (21%). Only 3% of research work has been done at cooling level.

The literature reports 44% of research work considered between 0 and 100 nodes for performance evaluation, and only 7% research work considered 1000 + nodes. There are four different types of studies identified from the literature: real testbed based (63%), systematic reviews (7%), conceptual models (10%), and simulation based (20%). Most of the technical research papers (63%) consider real testbeds for performance evaluation. There are only two reviews [37, 38], which have been done in this area. Table 5 shows the different research work related to different performance parameters identified from Table 4.

Table 5 Research work related to performance parameters

Performance parameters	Study
Number of physical nodes/workers Number of virtual nodes/workers	[1, 9, 10, 15, 16, 18, 25, 62, 70–72, 79, 80, 86, 88, 90, 91, 97]
Energy consumption	[11, 64]
CPU utilization	[2, 25, 27, 28]
Disk utilization	[2]
Data transfer	[16, 54, 89]
Memory utilization	[25]
Deadline	[14]
Execution time	[2, 8–11, 13–18, 25, 27, 28, 53–55, 62, 64, 70–72, 79–81, 86–89, 97, 98]
Number of tasks	[11, 13, 17, 28, 54, 64, 97]
Number of jobs	[11, 13–17, 53, 64, 86]
Data size	[64]
Latency	[8, 9, 15, 17, 55, 72, 79, 88]
Failure prediction accuracy	[14, 15]
Number of failures	[80]
Average error	[1, 8, 14, 55]
Response time	[9, 25, 64, 80, 97]
Slowdown/delay	[1, 15, 63, 79]
Interval arrival rate	[9, 27, 63, 79, 88]
Running time	[1, 9, 11, 13, 15–17, 28, 53, 54, 64, 89, 91]

5.3 Observations

From the trend analysis, it is observable that current related works focus on studying and mitigating specific straggler types, ranging from resource contention to data skew as shown in Table 2. This appears to be a necessity given the complexities and management strategies appropriate for each straggler type. The challenge is that it is possible for straggler manifestation to be correlated in terms of system phenomena, but also management techniques themselves (e.g. use of speculative copies to address data skew causes increased resource contention).

The important research challenges within the large-scale cloud data centres such as latency, scalability, energy consumption, and data processing are contributing to the rise in research in the field of straggler management, which can be solved by using artificial intelligence techniques. On the other hand, there is a need of real cloud infrastructure (at least 50 physical nodes) to test the performance of future straggler management techniques, but it would be very expensive to afford for academic institutes. To solve this problem, industries such as Facebook, Google, and Amazon should collaborate with academic institutes to provide required infrastructure to do real experiments.

This systematic review also identifies various research directions for perspective researcher scholars, who are working in the field of straggler management for

distributed systems and searching for new research challenges to improve the performance of cloud services. The straggler management is an evolving field of research for large-scale systems, and it is quite challenge ring to execute user workloads without occurrence of stragglers. To solve this problem, there is a need to recognize the reasons of long-tail problem or stragglers and their correlations, which can help to find out the dependency among stragglers. This study [1] developed straggler management technique for profile guide more accurately, but accurate predication is difficult to get if job is very small to gather required profiling data. An efficient data recovery is achieved in [80], but it has been identified that the memory requirements do not grow to intolerable levels as the size of dataset is increasing, which further causes the stragglers. The jobs are increasing with time, but there is need to analyse the impact of multiple jobs on probability of stragglers [13]. Existing techniques use historic data to estimate resource requirement [17]. However, there is a need to develop an online strategy to simultaneously learn the execution time distribution and launch replicas, instead of estimating time using historical traces. Further, the replication increases the reliability of execution of jobs, but it consumes more energy, which is a global challenge to address [8]. The scale-up/down infrastructure by switching on/off the virtual machines/nodes based on the resource usage of the cluster to save energy is required [91]. The dependency among tasks during task execution further effects causes the stragglers because some tasks need to complete in order to begin others [89]. Existing straggler management techniques are required to improve to attain to reduce straggler occurrence. By using this systematic review, causes of straggler can be identified easily. Therefore, an effective straggler management technique can be developed to execute the jobs without straggler occurrence while fulfilling the dynamic requirements of job, which helps to increase the efficiency of large-scale cloud data centres.

5.4 Future research directions

Although a substantial progress has been made in straggler management techniques for large-scale systems, there are still many pressing issues and challenges in this field that need to be addressed. Based on existing research, we have identified various open issues pending in this area.

5.4.1 Data processing

Data processing in straggler management is an important challenge [54]. It happens due to the skew in data that the computing system is able to process effectively. There are two types of problems which reduce the data processing capability of systems: (1) large variation of data size and (2) non-uniformity of data. These two reasons degrade the performance of large-scale computing systems. To improve the straggler management mechanism, there should be less variation as well as less non-uniformity of data. Tackling this challenge can further improve the processing speed of computing systems in terms of execution time and latency.

5.4.2 Heterogeneity

Hardware heterogeneity is the main reason for resource contention, which occurs due different types of resources (with different configurations, different providers, etc.) being used, and sometimes some resources are not compatible to execute jobs in a coordinative manner. There is a need for a single interface, which can provide a stable platform for interaction of different types of hardware in a collaborative manner.

5.4.3 Latency

The latency is another important challenge in straggler management of large-scale systems, which can affect the performance of computing systems. There are different types of reasons for latency: (1) non-uniformity of data, (2) resource contention, (3) poor user code, and (4) extra cloning. To improve the processing of computing systems, there is the need to make data uniform initially. Further, efficient resource scheduling algorithms are required, which can reduce resource contention at runtime and reduce the latency [99]. The extra cloning of tasks to speed up the execution can increase the latency because there is a requirement for more number of resources to process more number of copies. There is a need to develop an effective straggler management technique, which schedules resources and reduces latency at runtime.

5.4.4 Scalability

To improve the performance of computing systems, the systems must be more scalable to serve the jobs within their specific deadline without further delay at runtime [100]. The scalability of the computing system can increase the capacity of the system when the load increases, which can further reduce the problem of occurrence of stragglers.

5.4.5 Resource sharing

The sharing of resources among different jobs can improve resource utilization, but it leads to resource contention, which can degrade the performance of large-scale computing systems [101]. There is a need for an effective resource contention technique, which can identify the reasons of resource contention and provide the possible solutions to avoid additional resource over-allocation, ultimately contributing to straggler occurrence.

5.4.6 Energy management

The literature reports [99–102] that the straggler management techniques create several copies of the same task to mitigate the effects of stragglers. Copying a task reserves additional resources such as the disk, memory of CPU time, and increasing

use of particular resource. As the resource is more continuously used, its energy consumption rises. Depending on the type of the resource, its performance can degrade as its energy consumption increases above a certain threshold level.

6 Summary and conclusions

In this paper, we have provided a comprehensive literature review of current straggler research within Computer Science, an important problem which directly debilitates the performance of large-scale computing systems. We proposed a taxonomy of straggler causes as identified from different types of straggler management techniques. Moreover, various straggler management techniques have been reviewed and classified into two categories: straggler detection and straggler mitigation. The comparison of straggler detection and straggler mitigation has been presented in detail, and the taxonomy mapping-based comparison has been described, and various result outcomes related to straggler management have been presented. Observations of interest include the focused nature of straggler causes, and mitigation solutions may potentially interfere with each other due to correlated root-causes. Hence, there is a possibility of designing a multi-purpose straggler management technique which profiles and acts based on the type of identified straggler.

Acknowledgements This work is supported by the Engineering and Physical Sciences Research Council (EPSRC) (EP/P031617/1). We would like to thank Damian Borowiec and Shreshth Tuli for their valuable comments, useful suggestions, and discussion to improve the quality of the paper. We would like to thank the Editor-in-Chief and anonymous reviewers for their valuable suggestions to help and improve paper.

References

1. Coppa E, Finocchi I (2015) On data skewness, stragglers, and MapReduce progress indicators. In: Proceedings of the Sixth ACM Symposium on Cloud Computing. ACM, pp 139–152
2. Ouyang X, Garraghan P, Yang R, Townend P, Xu J (2016) 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
3. Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, Meng X et al (2016) Apache spark: a unified engine for big data processing. Commun ACM 59(11):56–65
4. Isard M, Budiu M, Yu Y, Birrell A, Fetterly D (2007) Dryad: distributed data-parallel programs from sequential building blocks. In: ACM SIGOPS Operating Systems Review, vol 41, no 3. ACM, pp 59–72
5. Gill SS, Chana I, Singh M, Buyya R (2019) RADAR: self-configuring and self-healing in resource management for enhancing quality of cloud services. Concurr Comput Pract Exp 31(1):e4834
6. Dean J, Barroso LA (2013) The tail at scale. Commun ACM 56(2):74–80
7. Shen H, Li C (2018) Zeno: a straggler diagnosis system for distributed computing using machine learning. In: International Conference on High Performance Computing. Springer, Cham, pp 144–162
8. Aktas MF, Peng P, Soljanin E (2017) Effective straggler mitigation: which clones should attack and when? ACM SIGMETRICS Perform Eval Rev 45(2):12–14
9. Wang D, Joshi G, Wornell G (2014) Efficient task replication for fast response times in parallel computation. ACM SIGMETRICS Perform Eval Rev 42(1):599–600

10. Dai W, Ibrahim I, Bassiouni M (2017) An improved straggler identification scheme for data-intensive computing on cloud platforms. In: 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud). IEEE, pp 211–216
11. Phan T-D (2017) Energy-efficient straggler mitigation for big data applications on the clouds. Ph.D. dissertation, ENS Rennes
12. Ozfatura E, Gündüz D, Ulukus S (2018) Speeding up distributed gradient descent by utilizing non-persistent stragglers. arXiv preprint arXiv:1808.02240
13. Ananthanarayanan G, Ghodsi A, Shenker S, Stoica I (2013) Effective straggler mitigation: attack of the clones. NSDI 13:185–198
14. Ananthanarayanan G, Hung MCC, 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
15. Yadwadkar NJ, Ananthanarayanan G, Katz R (2014) Wrangler: predictable and faster jobs using fewer resources. In: Proceedings of the ACM Symposium on Cloud Computing. ACM, pp 1–14
16. Zaharia M, Konwinski A, Joseph AD, Katz RH, Stoica I (2008) Improving MapReduce performance in heterogeneous environments. Osdi 8(4):7
17. Wang D, Joshi G, Wornell G (2015) Using straggler replication to reduce latency in large-scale parallel computing. ACM SIGMETRICS Perform Eval Rev 43(3):7–11
18. Chen Q, Zhang D, Guo M, Deng Q, Guo S (2010) Samr: a self-adaptive MapReduce scheduling algorithm in heterogeneous environment. In: 2010 IEEE 10th International Conference on Computer and Information Technology (CIT). IEEE, pp 2736–2743
19. Gill SS, Garraghan P, Stankovski V, Casale G, Thulasiram RK, Ghosh SK, Ramamohanarao K, Buyya R (2019) Holistic resource management for sustainable and reliable cloud computing: an innovative solution to global challenge. J Syst Softw 155:104–129
20. Lama P, Wang S, Zhou X, Cheng D (2018) Performance isolation of data-intensive scale-out applications in a multi-tenant cloud. In: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, pp 85–94
21. 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
22. Gill SS, Buyya R (2018) A taxonomy and future directions for sustainable cloud computing: 360 degree view. ACM Comput Surv (CSUR) 51(5):104
23. Mitsuzuka K, Koibuchi M, Amano H, Matsutani H (2018) Proxy responses by FPGA-based switch for MapReduce stragglers. IEICE Trans Inf Syst 101(9):2258–2268
24. Ouyang X, Wang C, Jie X (2019) Mitigating stragglers to avoid QoS violation for time-critical applications through dynamic server blacklisting. Future Gener Comput Syst 101:831–842
25. Ouyang X, Garraghan P, McKee D, Townend P, Xu J (2016) Straggler detection in parallel computing systems through dynamic threshold calculation. In 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA). IEEE, pp 414–421
26. Phan T-D, Pallez G, Ibrahim S, Raghavan P (2019) A new framework for evaluating straggler detection mechanisms in MapReduce. ACM Trans Model Perform Eval Comput Syst (TOMPECS) 4(3):14
27. Ananthanarayanan G, Kandula S, Greenberg AG, Stoica I, Yi L, Saha B, Harris E (2010) Reining in the outliers in map-reduce clusters using Mantri. Osdi 10(1):24
28. Garraghan P, Ouyang X, Yang R, McKee D, Xu J (2016) Straggler root-cause and impact analysis for massive-scale virtualized cloud datacenters. IEEE Trans Serv Comput
29. Gill SS, Tuli S, Xu M, Singh I, Singh KV, Lindsay D, Tuli S et al (2019) Transformative effects of IoT, blockchain and artificial intelligence on cloud computing: evolution, vision, trends and open challenges. Internet of Things 8:100118
30. Hamandawana P, Mativenga R, Kwon SJ, Chung TS (2019) EPPADS: an enhanced phase-based performance-aware dynamic scheduler for high job execution performance in large scale clusters. In: International Conference on Database Systems for Advanced Applications. Springer, Cham, pp 140–156
31. Ren X, Ananthanarayanan G, Wierman A, Yu M (2015) Hopper: decentralized speculation-aware cluster scheduling at scale. In: ACM SIGCOMM Computer Communication Review, vol 45, no 4. ACM, pp 379–392
32. Krishna LS, Natarajan LP (2019) Distributed inference with straggler mitigation. Ph.D. dissertation, Indian institute of technology Hyderabad

33. Huang X, Li C, Luo Y (2019) Optimized speculative execution strategy for different workload levels in heterogeneous spark cluster. In: Proceedings of the 2019 4th International Conference on Big Data and Computing. ACM, pp 6–10
34. Tandon R, Lei Q, Dimakis AG, Karampatziakis N (2017) Gradient coding: avoiding stragglers in distributed learning. In: International Conference on Machine Learning, pp 3368–3376
35. Ouyang X, Wang C, Yang R, Yang G, Townend P, Xu J (2017) ML-NA: a machine learning based node performance analyzer utilizing straggler statistics. In: 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS). IEEE, pp 73–80
36. Panda B, Srinivasan D, Ke H, Gupta K, Khot V, Gunawi HS (2019) {IASO}: a fail-slow detection and mitigation framework for distributed storage services. In: 2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19), pp 47–62
37. Kumar U, Kumar J (2014) A comprehensive review of straggler handling algorithms for MapReduce framework. Int J Grid Distrib Comput 7(4):139–148
38. Bhandare A et al (2016) Review and analysis of straggler handling techniques. Int J Comput Sci Inf Technol 7(5):2270
39. Eppstein D, Goodrich MT (2007) Space-efficient straggler identification in round-trip data streams via Newton's identities and invertible bloom filters. In: Workshop on Algorithms and Data Structures. Springer, Berlin, pp 637–648
40. Ouyang X, Garraghan P, McKee D, Townend P, Xu J (2016) Straggler detection in parallel computing systems through dynamic threshold calculation. In: 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA). IEEE, pp 414–421
41. Singh S, Chana I (2016) Cloud resource provisioning: survey, status and future research directions. Knowl Inf Syst 49(3):1005–1069
42. Benavides Z, Gupta R, Zhang X (2016) Parallel execution profiles. In: Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing. ACM, pp 215–218
43. Eppstein D, Goodrich MT (2011) Straggler identification in round-trip data streams via Newton's identities and invertible Bloom filters. IEEE Trans Knowl Data Eng 23(2):297–306
44. Yu Z, Li M, Yang X, Zhao H, Li X (2015) Taming non-local stragglers using efficient prefetching in MapReduce. In: 2015 IEEE international conference on cluster computing. IEEE, pp 52–61
45. Singh S, Chana I (2016) QoS-aware autonomic resource management in cloud computing: a systematic review. ACM Comput Surv 48(3):46
46. Harlap A, Cui H, Dai W, Wei J, Ganger GR, Gibbons PB, Gibson GA, Xing EP (2016) Addressing the straggler problem for iterative convergent parallel ML. In: Proceedings of the seventh acm symposium on cloud computing (SoCC '16). Association for computing machinery, New York, NY, USA, pp 98–111. https://doi.org/10.1145/2987550.2987554
47. Ouyang X, Zhou H, Clement S, Townend P, Xu J (2017) Mitigate data skew caused stragglers through ImKP partition in MapReduce. In: 2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC). IEEE, pp 1–8
48. Martha VS, Zhao W, Xu X (2013) h-MapReduce: a framework for workload balancing in MapReduce. In: 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA). IEEE, pp 637–644
49. Huang SW, Huang TC, Lyu SR, Shieh CK, Chou YS (2011) Improving speculative execution performance with coworker for cloud computing. In: 2011 IEEE 17th International Conference on Parallel and Distributed Systems. IEEE, pp 1004–1009
50. Lin J (2009) The curse of zipf and limits to parallelization: a look at the stragglers problem in MapReduce. In: 7th Workshop on Large-Scale Distributed Systems for Information Retrieval, vol 1. ACM, Boston, pp 57–62
51. Zhou AC, Phan TD, Ibrahim S, He B (2018) Energy-efficient speculative execution using advanced reservation for heterogeneous clusters. In: Proceedings of the 47th International Conference on Parallel Processing. ACM, p 8
52. Wang Z, Gao L, Gu Y, Bao Y, Yu G (2017) FSP: towards flexible synchronous parallel framework for expectation-maximization based algorithms on cloud. In: Proceedings of the 2017 Symposium on Cloud Computing. ACM, pp 1–14
53. Harlap A, Cui H, Dai W, Wei J, Ganger GR, Gibbons PB, Gibson GA, Xing EP (2016) Addressing the straggler problem for iterative convergent parallel ML. In: Proceedings of the Seventh ACM Symposium on Cloud Computing. ACM, pp 98–111

54. Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51(1):107–113
55. Aktas MF, Peng P, Soljanin E (2018). Straggler mitigation by delayed relaunch of tasks. ACM SIGMETRICS Perform Eval Rev 45(3):224–231
56. Yu Q, Ali M, Avestimehr AS (2018) Straggler mitigation in distributed matrix multiplication: fundamental limits and optimal coding. In: 2018 IEEE International Symposium on Information Theory (ISIT). IEEE, pp 2022–2026
57. Baharav T, Lee K, Ocal O, Ramchandran K (2018) Straggler-proofing massive-scale distributed matrix multiplication with d-dimensional product codes. In: 2018 IEEE International Symposium on Information Theory (ISIT). IEEE, pp 1993–1997
58. Xu M, Alamro S, Lan T, Subramaniam S (2017) Optimizing speculative execution of deadline-sensitive jobs in cloud. ACM SIGMETRICS Perform Eval Rev 45(1):17–18
59. Haddadpour F, Yang Y, Chaudhari M, Cadambe VR, Grover P (2018) Straggler-resilient and communication-efficient distributed iterative linear solver. arXiv preprint arXiv:1806.06140
60. Zhao X, Kang K, Sun Y, Song Y, Xu M, Pan T (2013) Insight and reduction of MapReduce stragglers in heterogeneous environment. In: 2013 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, pp 1–8
61. Isaacs KE, Gamblin T, Bhatele A, Bremer PT, Schulz M, Hamann B (2014) Extracting logical structure and identifying stragglers in parallel execution traces. In: ACM SIGPLAN Notices, vol 49, no 8. ACM, pp 397–398
62. Bin Khunayn E, Karunasekera S, Xie H, Ramamohanarao K (2017) Exploiting data dependency to mitigate stragglers in distributed spatial simulation. In: Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, p 43
63. Farhat F, Tootaghaj DZ, Sivasubramaniam A, Kandemir MT, Das CR (2014) Modeling and optimization of straggling mappers. Technical report, Technical Report CSE-14-006, Pennsylvania State University
64. Phan TD, Ibrahim S, Zhou AC, Aupy G, Antoniu G (2017) Energy-driven straggler mitigation in MapReduce. In: European Conference on Parallel Processing. Springer, Cham, pp 385–398
65. Yang E, Kang DK, Youn CH (2019) BOA: batch orchestration algorithm for straggler mitigation of distributed DL training in heterogeneous GPU cluster. J Supercomput 76:1–21
66. Jiang J, Cui B, Zhang C, Yu L (2017) Heterogeneity-aware distributed parameter servers. In: Proceedings of the 2017 ACM International Conference on Management of Data. ACM, pp 463–478
67. Patgiri R, Das R. (2018) rTuner: a performance enhancement of MapReduce job. In: Proceedings of the 10th International Conference on Computer Modeling and Simulation. ACM, pp 176–183
68. Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I (2013) Discretized streams: Fault-tolerant streaming computation at scale. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM, pp 423–438
69. Yu Q, Maddah-Ali MA, Avestimehr AS (2020) Straggler mitigation in distributed matrix multiplication: fundamental limits and optimal coding. IEEE Trans Inf Theory 66(3):1920–1933
70. Ouyang X, Garraghan P, Wang C, Townend P, Xu J (2016) An approach for modeling and ranking node-level stragglers in cloud datacenters. In: 2016 IEEE International Conference on Services Computing (SCC). IEEE, pp 673–680
71. Tavakoli N, Dai D, Chen Y (2016) Log-assisted straggler-aware I/O scheduler for high-end computing. In: 2016 45th International Conference on Parallel Processing Workshops (ICPPW). IEEE, pp 181–189
72. Li C, Shen H, Huang T (2016) Learning to diagnose stragglers in distributed computing. In: 2016 9th Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers (MTAGS). IEEE, pp 1–6
73. Khunayn EB, Karunasekera S, Xie H, Ramamohanarao K (2017) Straggler mitigation for distributed behavioral simulation. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). IEEE, pp 2638–2641
74. Paik M (2010) Stragglers of the herd get eaten: security concerns for GSM mobile banking applications. In: Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications. ACM, pp 54–59
75. Malewicz G, Dvorsky M, Colohan CB, Thomson DP, Levenberg JL (2013) System and method for limiting the impact of stragglers in large-scale parallel data processing. U.S. Patent 8,510,538, issued 13 Aug 2013

76. Karakus C, Sun Y, Diggavi S, Yin W (2018) Redundancy techniques for straggler mitigation in distributed optimization and learning. arXiv preprint arXiv:1803.05397
77. Garraghan P, Yang R, Wen Z, Romanovsky A, Jie X, Buyya R, Ranjan R (2018) Emergent failures: rethinking cloud reliability at scale. IEEE Cloud Comput 5(5):12–21
78. Li S, Kalan SM, Avestimehr AS, Soltanolkotabi M (2018) Near-optimal straggler mitigation for distributed gradient methods. In: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) IEEE, pp 857–866
79. Farhat F (2015) Stochastic modeling and optimization of stragglers in MapReduce framework. Thesis, The Pennsylvania State University
80. Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. USENIX Association, pp 2–2
81. Yang H, Lee J (2019) Secure distributed computing with straggling servers using polynomial codes. IEEE Trans Inf Forensics Secur 14(1):141–150
82. Mallick A, Chaudhari M, Joshi G. Rateless codes for straggler mitigation in distributed computing. https://www.andrew.cmu.edu/user/gaurij/18–847F-Lectures/rateless_codes_2018.pdf. Accessed 10 July 2019
83. Chen C, Weng Q, Wang W, Li B, Li B (2018) Fast distributed deep learning via worker-adaptive batch sizing. In: Proceedings of the ACM Symposium on Cloud Computing. ACM, pp 521–521
84. Kapoor R, Porter G, Tewari M, Voelker GM, Vahdat A (2012) Chronos: predictable low latency for data center applications. In: Proceedings of the Third ACM Symposium on Cloud Computing. ACM, p 9
85. Lindsay D, Gill SS, Garraghan P (2019) PRISM: an experiment framework for straggler analytics in containerized clusters. In: Proceedings of the 5th International Workshop on Container Technologies and Container Clouds, pp 13–18
86. Guo Y, Rao J, Jiang C, Zhou X (2017) Moving Hadoop into the cloud with flexible slot management and speculative execution. IEEE Trans Parallel Distrib Syst 3:798–812
87. Wang H, Guo S, Tang B, Li R, Li C (2019) Heterogeneity-aware gradient coding for straggler tolerance. arXiv preprint arXiv:1901.09339
88. Vulimiri A, Godfrey PB, Mittal R, Sherry J, Ratnasamy S, Shenker S (2013) Low latency via redundancy. In: Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies. ACM, pp 283–294
89. Wang D, Joshi G, Wornell G (2015) Efficient straggler replication in large-scale parallel computing. arXiv preprint arXiv:1503.03128
90. Lei L, Wo T, Hu C (2011) CREST: towards fast speculation of straggler tasks in MapReduce. In: 2011 IEEE 8th International Conference on e-Business Engineering (ICEBE). IEEE, pp 311–316
91. Nanduri R, Maheshwari N, Reddyraja A, Varma V (2011) Job aware scheduling algorithm for MapReduce framework. In: 2011 Third IEEE International Conference on Coud Computing Technology and Science. IEEE, pp 724–729
92. Behrouzi-Far A, Soljanin E (2018) On the effect of task-to-worker assignment in distributed computing systems with stragglers. In: 2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton). IEEE, pp 560–566
93. Cipar J, Ho Q, Kim JK, Lee S, Ganger GR, Gibson G, Keeton K, Xing E (2013) Solving the straggler problem with bounded staleness. Presented as part of the 14th Workshop on Hot Topics in Operating Systems
94. Chen F, Wu S, Jin H, Yao Y, Liu Z, Gu L, Zhou Y (2017) Lever: towards low-latency batched stream processing by pre-scheduling. In: Proceedings of the 2017 Symposium on Cloud Computing. ACM, pp 643–643
95. Misra PA, Borge MF, Goiri Í, Lebeck AR, Zwaenepoel W, Bianchini R (2019) Managing tail latency in datacenter-scale file systems under production constraints. In: Proceedings of the Fourteenth EuroSys Conference 2019. ACM, p 17
96. Qureshi NM, Siddiqui IF, Abbas A, Bashir AK, Choi K, Kim J, Shin DR (2019) Dynamic container-based resource management framework of spark ecosystem. In: 2019 21st International Conference on Advanced Communication Technology (ICACT). IEEE, pp 522–526
97. Ouyang X, Garraghan P, Primas B, McKee D, Townend P, Jie X (2018) Adaptive speculation for efficient internetware application execution in clouds. ACM Trans Internet Technol (TOIT) 18(2):15

98. Yan R, Fleury MO, Merler M, Natsev A, Smith JR (2009) Large-scale multimedia semantic concept modeling using robust subspace bagging and MapReduce. In: Proceedings of the First ACM Workshop on Large-Scale Multimedia Retrieval and Mining. ACM, pp 35–42
99. Singh S, Chana I (2016) A survey on resource scheduling in cloud computing: issues and challenges. J Grid Comput 14(2):217–264
100. Zheng P, Lee BC (2018) Hound: causal learning for datacenter-scale straggler diagnosis. Proc ACM Meas Anal Comput Syst 2(1):17
101. Tavakoli N, Dai D, Chen Y (2019) Client-side straggler-aware I/O scheduler for object-based parallel file systems. Parallel Comput 82:3–18
102. Fuerst C, Schmid S, Suresh L, Costa P (2015) Kraken: towards elastic performance guarantees in multi-tenant data centers. ACM SIGMETRICS Perform Eval Rev 43(1):433–434