ISSN:0975-9646

Review and Analysis of Straggler Handling Techniques

Ashwin Bhandare[#1], Jitin George[#2], Supreet Deshpande[#3], Yash Karle[#4]

[#]*Department of Information Technology, Pune Institute of Computer Technology*
Savitribai Phule Pune University, Pune

***Abstract*— Distributed processing frameworks split a data intensive computation job into multiple smaller tasks, which are then executed in parallel on commodity clusters to achieve faster job completion. A natural consequence of such a parallel execution model is that slow running tasks, commonly called stragglers potentially delay overall job completion. Straggler tasks continue to be a major hurdle in achieving faster completion of data intensive applications running on modern data-processing frameworks. Such stragglers increase the average job duration by 47% in data clusters of Facebook and Bing even after these companies using state of the art straggler mitigation techniques. This is because current mitigation techniques all involve an element of waiting and speculation. Existing straggler mitigation techniques are inefficient due to their reactive and replicative nature – they rely on a wait-speculate-execute mechanism, thus leading to delayed straggler detection and inefficient resource utilization. Hence a full cloning of small jobs, avoiding waiting and speculation altogether is proposed in a system called as Dolly. Dolly utilizes extra resources due to replication. Therefore Wrangler, a system that proactively avoids situations that cause stragglers was presented which automatically learns to predict such situations using a statistical learning technique based on cluster resource utilization counters. Also predictors for similar nodes or workloads are likely to be similar and can share information, suggesting a multi-task learning (MTL) based approach. Thus we have reviewed some of the approaches of scheduling algorithms like LATE, SAMR and ESAMR, Dolly and Wrangler. Thus we have reviewed some of the approaches of scheduling algorithms like LATE, SAMR and ESAMR, Dolly and Wrangler.**

***Keywords*—Straggler, Speculative Execution, LATE, SAMR, ESAMR, Dolly, Wrangler, Predictive Models, MTL**

I. INTRODUCTION

Today, data is getting generated at an unprecedented scale due to popular Internet-based computer applications that serve millions of users, such as e-commerce websites and social networks. The rate at which this data is growing has rendered parallel processing on commodity compute clusters an inevitable and an attractive option.

A. Motivation

Google originally proposed its MapReduce framework [1] allowing them to process enormous amounts of data. MapReduce is highly scalable to large clusters of inexpensive commodity computers. Hadoop, a popular open source implementation of MapReduce, has been widely adopted by industries of various sizes. For accelerating a job's completion time, MapReduce divides a data intensive computation job into multiple smaller tasks. These tasks are executed in parallel on multiple machines (nodes) in a compute cluster. A job finishes when all its tasks have finished execution. A key benefit of such distributed parallel processing frameworks is that they automatically handle failures, without needing extra e□orts from the programmer. Two basic modes of failures are the failure of a node and the failure of a task. If a node crashes, MapReduce re-runs all the tasks it was executing on a di□erent node. If a task fails, MapReduce automatically re-launches it. However, a tricky situation arises when a node is available but is performing poorly. This causes tasks scheduled on that node to execute slower than other tasks of the same job scheduled on other nodes in the cluster. Since a job finishes execution only when all its tasks have finished execution, such slow-running tasks, called stragglers, extend the job's completion time. This, in turn, leads to increased user costs.

B. Problem Statement

To analyse and review various Straggler mitigation techniques for distributed parallel processing. To study broad sub-categories of Straggler mitigation i.e. Proactive and Reactive approaches.

C. Scope

The existing Straggler mitigation approaches work on minimizing the effect of Stragglers, but not on their complete elimination. They work to maximize performance at the same time minimizing resource utilization. In reality, it is difficult to actually distinguish between temporal and persistent Stragglers. There are many factors that are needed to be considered for identifying a node as a Straggler.

II. LITERATURE SURVEY

Today's most popular computer applications are Internet services with millions of users. The sheer volume of data that these services work with has led to interest in parallel processing on commodity clusters. The leading example is Google, which uses its map-reduce framework to process 20 Petabytes of data per day. Other internet services like e-commerce websites and social networks also cope with enormous volumes of data. These services generate click stream data from millions of users everyday which is a potential gold mine for understanding access patterns and increasing ad revenues. Furthermore, for each user action a web application generates 1 or 2 orders of magnitude more data in system logs which are the main resource that developers and operators have for diagnosing problems in production. Map-reduce model popularised by Google is very attractive for ad-hoc parallel processing of arbitrary data. Map-reduce breaks a computation into small tasks that run in parallel on multiple machines, and scales easily to very large clusters of inexpensive commodity computers. A key benefit of map-reduce is that it automatically handles failures hiding the complexity of fault tolerance from the programmer. If a node crashes, map-reduce runs its task on a different machine.

Equally importantly if a node is available but is performing poorly a condition that is called Straggler, map-reduce runs a speculative copy of its task on another machine to finish its computation faster. Without this mechanism of speculative execution a job would be as slow as the misbehaving task. Stragglers can arise from many reasons including faulty hardware

and misconfiguration. Google has noted that speculative execution can improve job response times by 44%.

Hadoop's scheduler starts speculative tasks based on a simple heuristic comparing each task's progress to the average progress. Although this heuristic works well in homogeneous environments where stragglers are obvious, it can be showed that it can lead to severe performance degradation when its underlying assumptions are broken.

In some experiments, as many as 80% of tasks were speculatively executed. Naıvely, one might expect speculative execution to be a simple matter of duplicating tasks that are sufficiently slow. In reality, it is a complex issue for several reasons. First, speculative tasks are not free – they compete for certain resources, such as the network, with other running tasks. Second, choosing the node to run a speculative task on is as important as choosing the task. Third, in a heterogeneous environment, it may be difficult to distinguish between nodes that are slightly slower than the mean and stragglers. Finally, stragglers should be identified as early as possible to reduce response times.

Existing approaches, whether based on replication or modeling, aren't enough to solve this problem. Speculative execution is a replication-based reactive straggler mitigation technique that spawns redundant copies of the slow running tasks, hoping a copy will reach completion before the original. This is the most prominently used technique today, including production clusters at Facebook and Microsoft Bing. However, without any additional information, such reactive techniques can not differentiate between nodes that are inherently slow and nodes that are temporarily overloaded. In the latter case, such techniques lead to unnecessary over-utilization of resources without necessarily improving the job completion times. Though proactive, Dolly is still a replication-based approach that focusses only on interactive jobs and incurs extra resources. Being agnostic to the correlations between stragglers and nodes' status, replication-based approaches are wasteful.

III. REACTIVE STRAGGLER MITIGATION TECHNIQUES

A. Speculative Execution

In Hadoop, if a node is available but is performing poorly, the condition is called a straggler, MapReduce runs a speculative copy of its task (also called a backup task) on another machine to finish the computation faster. The goal of speculative execution [1] is to minimize a jobs response time. A speculative task is run based on a simple heuristic comparing each tasks progress to the average progress. Hadoop monitors task progress using a parameter called Progress Score which has value between 0 and 1. For a map, the Progress Score is the fraction of input data read. For a reduce task, the execution is divided into three phases, each of which accounts for 1/3 of the score. Hadoop looks at the average Progress Score of each category of tasks (maps and reduces) to define a threshold for speculative execution. When a tasks Progress Score is less than the average for its category minus 0.2, and the task has run for at least one minute, it is marked as a straggler. The speculative task scheduling in Hadoop is based on multiple assumptions, one is that data center is homogeneous, all tasks progress at same rate (while some may be local, some remote, some more compute intensive etc), and all reduce tasks process same amount of data. So if any of these is invalidated, their execution can cause competition and may cause Hadoop to perform poorly. Thus scheduling of speculative tasks which actually help minimize delay is complex. First because it is di□cult to select the task for which to run speculative task as it would be di□cult to distinguish between nodes that are slightly slower than the mean and stragglers especially in heterogeneous environment. Then it is useful in decreasing response time only if stragglers are identified as early as possible, so it needs to scheduled at right time. Few other points that need to be considered while deciding and scheduling them would be the competition of resources (network, cpu etc) created by speculative (nothing but duplicate) tasks and selecting node to run them.

B. Longest Approximation Time to End Scheduling Algorithm

M. Zaharia proposed the Longest Approximate Time to End (LATE) [5] scheduling algorithm for speculative execution, which is highly robust to heterogeneity. LATE computes the longest remaining execution time based on the progress score provided by default scheduler, then the scheduler chooses the tasks with the longest remaining time as straggler tasks which will show impact on the overall job response time. Progress rate of task tj, denoted by PRj is used to evaluate the remaining execution time of tj using (1) and TTEj denotes the remaining execution time of task tj is evaluated using (2), where T is the elapsed time.

Longest Approximate Time to End (LATE) algorithm is based on three principles: prioritize tasks to speculate, select fast nodes to run on, and cap speculative tasks to prevent thrashing. To realize these principles LATE algorithm uses following parameters:

SlowNodeThreshold - This is the cap to avoid scheduling on slow nodes. Scores for all succeeded and in-progress tasks on the node are compared to this value. SpeculativeCap - It is the cap on number of speculative tasks that can be running at once.

SlowTaskThreshold - This is a progress rate threshold to determine if a task is slow enough to be speculated upon. This prevents needless speculation when only fast tasks are running.

Progress Rate of a task is given by ProgressScore=ExecutionTime

The time left parameter for a task is estimated based on the Progress Score provided by

Hadoop, as (1 - ProgressScore)=ProgressRate.

1) LATE Scheduling Algorithm

1. a node N asks for a new task
2. if number of running speculative tasks < SpeculativeCap then
3. if nodes total progress < SlowNodeThreshold then
4. ignore the request else
5. rank currently running tasks that are not currently being speculated by estimated time left
6. repeat
7. select next task T from ranked list
8: if progress rate of T < SlowTaskThreshold then
9: Launch a copy of T on node N
10: exit
11: end if
12: until while ranked list has tasks
14: end if
15: end if

One experiment showed that in a cluster with non-faulty nodes experiment (without stragglers), LATE finished jobs 27% faster than Hadoops native scheduler and 31% faster than no speculation. LATE provides gains in heterogeneous environments even if there are no faulty nodes. For Sort with stragglers, on average, LATE finished jobs 58% faster than Hadoops native scheduler and 220% faster than Hadoop with speculative execution disabled. The comparison of worst, best and average-case performance of LATE against Hadoops scheduler and no speculation for runs without and with stragglers are shown below in Figure 6. Sensitivity analysis to SpeculativeCap done in test environment showed that response time drops sharply at SpeculativeCap = 20%, after which it stays low. And a higher threshold value is undesirable because

LATE wastes more time on excess speculation. Experiments for Sensitivity to SlowTaskThreshold (percentile of progress rate below which a task must lie to be considered for speculation) show that small threshold values harmfully limit the number of speculative tasks, values past 25% all work well. Sensitivity analysis to SlowNodeThreshold (percentile of speed below which a node will be considered too slow for LATE to launch speculative tasks on) show that as long as SlowNodeThreshold is higher than the fraction of nodes that are extremely slow or faulty, LATE performs well.

1) Advantages of LATE

LATE primarily focuses on approximating the remaining execution time more willingly than just the progress score since it will speculatively executes only those tasks that will increase overall job response time instead of slow tasks. LATE takes into account node heterogeneity when choosing on which node to run a speculative task.

2) Limitations of LATE

Even though LATE practices better approach to present backup tasks, it cannot always finds the actual straggler tasks since it does not approximate time to end of running tasks correctly. Same as the MapReduce default scheduler, LATE sets the constant values of M1, M2, R1, R2, and R3 as 1, 0, 1/3, 1/3 and 1/3. These values may be altered for different hardware settings and MapReduce applications in real world execution.

C. Self-Adaptive MapReduce Scheduling Algorithm

Q. Chen proposed Self-Adaptive MapReduce (SAMR) [4] scheduling algorithm, which computes the progress of the tasks dynamically and it has implemented the concept of

LATE scheduling algorithm which identifies slow tasks by approximating execution time of a task. To get more accurate progress score than LATE, SAMR uses the historical information recorded on each node in the cluster to tune the weights of map and reduce stages and also it updates the weights after each task execution. Therefore, SAMR scheduler performance is enhanced in heterogeneous environment as compared to MapReduce default scheduler and LATE scheduler.

1) Advantages of SAMR

SAMR takes the two stages of a map task into consideration for the first time and also it categorizes slow nodes into map slow nodes and reduce slow nodes.

2) Disadvantages of SAMR

Although SAMR uses historical data stored on every node in the cluster to determine a more accurate estimate of progress score than LATE, it does not consider that different job types can have different weights for map and reduce stages. In SAMR, the jobs with same type can even have different map stage weights and reduce stage weights when handling the data sets with different sizes.

D. Enhanced Self-Adaptive MapReduce Scheduling Algorithm

Enhanced Self-Adaptive MapReduce (ESAMR) [2] scheduling algorithm is designed to overcome the limitations of the SAMR algorithm by taking many factors into account that could impact the stage weights. The novel contribution of ESAMR is to categorize the historical data stored on each node into K clusters using K-means cluster identification algorithm to tune parameters dynamically and finds slow task accurately. In the map phase, if all the tasks complete their execution then the algorithm will store job's temporary M1 weight and uses this weight to find the cluster whose average M1 weight is the closest. These stage weights will be used to estimate TimeToEnd on that node. In reduce phase, the algorithm follows a similar process of that map phase. It uses temporary R1 and R2 weights to determine the cluster with the closest reduce stage weights. ESAMR detects slow tasks by utilizing these stage weights to approximate TimeToEnd of the reduce tasks of that node. After completion of a job, the algorithm computes the stage weights of map and reduce tasks on each node in the cluster and keeps these new weights as part of the historical data.

Lastly, the algorithm uses K-means cluster identification algorithm to re-categorize the historical data saved on each node into K clusters. This algorithm improves the performance of MapReduce scheduling by launching backup tasks for slow tasks but this algorithm is limited to only K-means algorithm, so classification can be performed using a better clustering algorithm for best results which might further improve the MapReduce performance in the heterogeneous environment.

IV. PROACTIVE STRAGGLER MITIGATION TECHNIQUES

Existing approaches, whether based on replication or modelling, aren't enough to solve this problem. Speculative execution is a replication-based reactive straggler mitigation technique that spawns redundant copies of the slow-running tasks, hoping a copy will reach completion before the original. This is the most prominently used technique today, including production clusters at Facebook and Microsoft Bing. However, without any additional information, such reactive techniques can not differentiate between nodes that are inherently slow and nodes that are temporarily overloaded. In the latter case, such techniques lead to unnecessary over-utilization of resources without necessarily improving the job completion times.

To avoid such problems, a straggler mitigation approach should meet the following requirements:

- It should not wait until the tasks are already straggling.
- It should not waste resources for mitigating stragglers.

A. Wrangler

Wrangler [7] is a system that predicts stragglers using an interpretable linear modeling technique. Wrangler prevents wastage of resources by removing the need for replicating tasks. Wrangler introduces a notion of a confidence measure with these predictions to overcome the modeling error problems; this confidence measure is then exploited to achieve a reliable task scheduling. A prototype implementation of Wrangler demonstrates up to 61% improvement in overall job completion times while reducing the resource consumption by up to 55% for production-level workloads using a 50 node EC2 cluster.

1) Wrangler Architecture

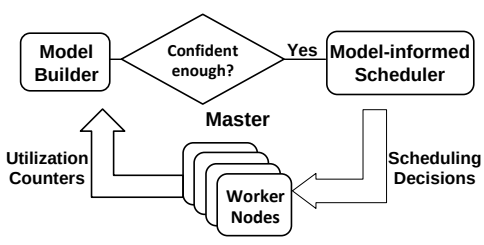
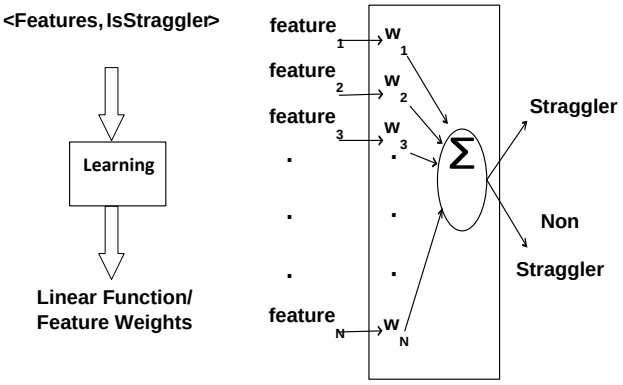


Figure 4.1: Architecture of *Wrangler*.

Figure 4.1 shows Wrangler's system architecture, which extends Hadoop. Job scheduling in Hadoop is handled by a master, which controls the workers. The master assigns tasks to the worker nodes in response to the heartbeat message sent by them every few seconds. The assignments depend upon the number of available slots as well as locality. Wrangler has two basic components.

1. Model Builder: Using the job logs and snapshot of resource usage counters collected regularly from the worker nodes using a Ganglia-based node-monitor and build a model per node. These models predict if a task will straggle given its execution environment; they also attach a confidence measure to each of their predictions.
2. Model-informed Scheduler: Using the predictions from the models built earlier, a model-informed scheduler then selectively delays the start of task execution if that node is predicted to create a straggler. A task is delayed only if the confidence in the corresponding prediction exceeds the minimum required confidence. This avoids overloading of nodes, thus reducing their chances of creating stragglers.

2) Linear Model for Predicting Stragglers



(a) Learning phase (b) Prediction phase Fig 4.2

Finding what actually causes stragglers is challenging due to complex task-to-node and task to-task interactions. Linear modeling techniques from the machine learning domain are appropriate for probabilistic modeling of a node's behaviour, which can be represented through the various resource usage counters. These techniques adapt to dynamically changing resource usage patterns on a node. This alleviates the pains of manual diagnosis of the source of individual straggler appearance. It is necessary to learn the behaviour of each node individually to be robust to heterogeneity in today's clusters. As shown in Figure 4.2(a), during the learning phase, these techniques learn *weights* on the features using labelled data that represents the ground truth. In this context this data is the node's resource usage counters at the time of submission of a task and a label (isStraggler), indicating whether it was a straggler. Using these weights and the node's resource usage counters the model calculates a score for predicting if it will turn out to be a straggler. This prediction phase is depicted in Figure 4.2(b).

3) Support Vector Machine

SVM [8] is a statistical tool that learns a linear function separating a given set of vectors (e.g., node's resource usage counters) into two classes (e.g., straggler class and non-straggler class). This linear function is called the separating hyperplane; each of the two half spaces defined by this hyperplane represents a class. In the model building phase, this hyperplane is computed such that it separates the vectors of node's resource usage counters belonging to one class (stragglers) from those of the other class (non-stragglers) with maximum distance (called margin) between them. Later, a new observed resource usage vector (i.e. a test vector) can be evaluated to see which side of the separating hyperplane it lies, along with a score to quantify the confidence in classification based on the distance from the hyperplane.

4) Features

The node-level features spanning multiple broad categories are as follows:

1. CPU utilization: CPU idle time, system and user time and speed of the CPU, etc.
2. Network utilization: Number of bytes sent and received, statistics of remote read and write, statistics of RPCs, etc.
3. Disk utilization: The local read and write statistics from the data nodes, amount of free space, etc.
4. Memory utilization: Amount of virtual, physical memory available, amount of buffer space, cache space, shared memory space available, etc.
5. System-level features: Number of threads in different states (waiting, running, terminated, blocked, etc.), memory statistics at the system level.

5) Confidence Measures

Simply predicting a task to be a 'straggler' or a 'non-straggler' is not robust to modeling errors. To ensure reliable predictions, the notion of confidence measure along with the prediction of these linear models is introduced. Confidence measure to help decide if our predictions are accurate enough for preventing stragglers by influencing the scheduling decisions is needed. The farther a node counter vector is from the separating hyperplane, higher are the chances of it belonging to the predicted class. To obtain a probability estimate of the prediction being correct, the distance from the separating hyperplane is converted to a number in the range [0, 1]. These probabilities are obtained by fitting logistic regression models to this distance.

6) Summary

Wrangler proactively avoids stragglers to achieve faster job completions while using fewer resources. Rather than allowing tasks to execute and detecting them as stragglers when they run slow, Wrangler predicts stragglers before they are launched. Wrangler's notion of a confidence measure allows it to overcome modeling errors. Further, Wrangler leverages this confidence measure to achieve a reliable task scheduling; thus eliminating the need for replicating them. Prototype on Hadoop using an EC2 cluster of 50 nodes showed that Wrangler speeds up the 99th percentile job execution times by up to 61% and consumes up to 55% lesser resources as compared to the speculative execution for production workloads at Facebook and Cloudera's customers. Although it serves as a straggler avoidance approach on its own, Wrangler can also be used in conjunction with existing mitigation approaches. In the future, we aim to speed up the training process by (1) reducing the time spent for capturing training data per node in a cluster and (2) training straggler prediction models across workloads.

B. Multi-task Learning

Proactive straggler mitigation techniques attempt to schedule tasks in a way that limits the effect of stragglers by modeling straggler behaviour. Recently, Wrangler showed that incorporating predictive models of straggler behaviour in the scheduler can lead to large improvements in job completion times.

However, to address heterogeneity in the nodes and changing workload patterns, proactive model based approaches have previously modelled each workload and node independently.

Independent models pose two critical challenges: (1) each new node and workload requires new training data which can take hours to collect, delaying the application of model based scheduling, and (2) clusters with many nodes may have only limited data for a given workload on each node leading to lower quality models.

These shortcomings can be addressed if each classifier is able to leverage information gleaned at other nodes and from other workloads. For instance, when there is not enough data at a node for a workload, we can gain from the data collected at that node while it was executing other workloads, or from other nodes running the same workload. Such information sharing falls in the ambit of multi-task learning (MTL) [3], where the learner is embedded in an environment of related tasks, and the learner's aim is to leverage correlations between the tasks to improve performance of all tasks.

In this work, explicit knowledge about the dependencies between tasks to improve the performance of MTL is exploited. In particular, classifiers by workload and by node is grouped. Using our formulation to predict stragglers allows us to reduce job completion times by up to 59% over Wrangler. This large reduction arises from a 7% increase in prediction accuracy. Further, equal or better accuracy can be obtained by using a sixth of the training data, thus bringing the training time down from 4 hours to about 40 minutes. In addition, our formulation reduces the number of parameters by grouping parameters into node-dependent and workload-dependent factors. It is seen that, in the event of a particular task having insufficient data, parameter grouping can lead to significant gains over a naive MTL formulation.

Finally, while it can be shown experiments on straggler avoidance, our learning formulation is general and can be applied to other systems that train node or workload dependent classifiers. For instance, Throughput Scheduler uses such classifiers to allot resources to tasks, and can benefit from such multitask reasoning.

1) Need for Multi-task Learning

Our proposal is to leverage the correlations between the classifiers to reduce data collection time. Concretely, a task executing on a node will be a straggler because of a combination of factors. Some of these factors involve the properties of the node where the task is executing (for instance, the node may be memory-constrained) and some others involve particular requirements that the tasks might have in terms of resources (for instance, the task may require a lot of memory). These are workload-related factors. When collecting data for a new workload executing on a given node, one must be able to use information about the workload collected while it executed on other nodes, and information about the node collected while it executed other workloads.

This kind of sharing of information is precisely the motivation for the machine learning paradigm known as multitask learning (MTL). Each task has its own training data set, although typically all training points of all tasks live in the same feature space. The tasks are related to each other, and the goal of MTL is to leverage this relationship to improve performance or generalization of all the tasks.

2) Summary

In this work, it can be shown the utility of multitask learning in solving the real-world problem of avoiding stragglers in distributed data processing. Our novel MTL formulation captures the structure of our tasks and reduces job completion times by up to 59% over prior work. This reduction comes from a 7 point increase in prediction accuracy. Our formulation can achieve better accuracy with only a sixth of the training data and can generalize better than other MTL approaches for tasks with little or no data. Finally, although use of straggler avoidance as the motivation, our formulation is more generally applicable, especially for other prediction problems in distributed computing frameworks, such as resource allocation.

C. Dolly

Small jobs that are typically run for interactive data analyses in data centers continue to be plagued by disproportionately long-running tasks called stragglers. In the production clusters at Facebook and Microsoft Bing, even after applying state-of-the-art straggler mitigation techniques, these latency sensitive jobs have stragglers that are on average 8 times slower than the median task in that job. Such stragglers increase the average job duration by47%. This is because current mitigation techniques all involve an element of waiting and speculation. Cloning of small jobs only marginally increases utilization because workloads show that while the majority of jobs are small, they only consume a small fraction of the resources. The main challenge of cloning is, however, that extra clones can cause contention for intermediate data. A technique called as delay assignment, which efficiently avoids such contention. Evaluation of our system, Dolly [6] , using production workloads shows that the small jobs speedup by 34% to 46% after state-of-the-art mitigation techniques have been applied, using just 5% extra resources for cloning.

1) Explanation

Achieving low latencies for these small interactive jobs is of prime concern to data center operators. The problem of stragglers has received considerable attention already, with a slew of straggler mitigation techniques being developed. These techniques can be broadly divided into two classes: black-listing and speculative execution. However, our traces show that even after applying state-of-the-art blacklisting and speculative execution techniques, the small jobs have stragglers that, on average, run eight times slower than that job's median task, slowing them by 47% on average. Thus, stragglers remain a problem for small jobs.

Blacklisting identifies machines in bad health (e.g., due to faulty disks) and avoids scheduling tasks on them. The Facebook and Bing clusters, in fact, blacklist roughly 10% of their machines. However, stragglers occur on the non-blacklisted machines, often due to intrinsically complex reasons like IO contentions, interference by periodic maintenance operations and background services, and hardware behaviours. For this reason, speculative execution was explored to deal with stragglers. Speculative execution waits to observe the progress of the tasks of a job and launches duplicates of those tasks that are slower. However, speculative execution techniques have a fundamental limitation when dealing with small jobs. Any meaningful comparison requires waiting to collect statistically significant samples of task performance. Such waiting limits their agility when dealing with stragglers in small jobs as they often start all their tasks simultaneously. The problem is exacerbated when some tasks start straggling when they are well into their execution. Spawning a speculative copy at that point might be too late to help. This technique is both general and robust as it eschews waiting, speculating, and finding complex correlations. Such proactive cloning will significantly improve the agility of straggler mitigation when dealing with small interactive jobs.

Cloning comes with two main challenges. The first challenge is that extra clones might use a prohibitive amount of extra resources. However, our analysis of production traces shows a strong heavy-tail distribution of job sizes: the smallest 90% of jobs consume as less as 6% of the resources.

The second challenge is the potential contention that extra clones create on intermediate data, possibly hurting job performance. Efficient cloning requires that we clone each task

and use the output from the clone of the task that finishes first. This, however, can cause contention for the intermediate data passed between tasks of the different phases (e.g., map, reduce, join) of the job; frameworks often compose jobs as a graph of phases where tasks of downstream phases (e.g., reduce) read the output of tasks of upstream phases (e.g., map). If all downstream clones read from the upstream clone that finishes first, they contend for the IO bandwidth. An alternate that avoids this contention is making each downstream clone read exclusively from only a single upstream clone. But this staggers the start times of the downstream clones.

Our solution to the contention problem, delay assignment, is a hybrid solution that aims to get the best of both the above pure approaches. It is based on the intuition that most clones, except few stragglers, finish nearly simultaneously. Using a cost-benefit analysis that captures this small variation among the clones, it checks to see if clones can obtain exclusive copies before assigning downstream clones to the available copies of upstream outputs. The cost-benefit analysis is generic to account for different communication patterns between the phases, including all-to-all (MapReduce), many-to-one (Dryad), and one-to-one (Dryad and Spark).

Dolly, a system that performs cloning to mitigate the effect of stragglers while operating within a resource budget. Evaluation on a 150 node cluster using production workloads from Facebook and Bing shows that Dolly improves the average completion time of the small jobs by 34% to 46%, respectively, with LATE and Mantri as baselines. These improvements come with a resource budget of merely 5% due to the aforementioned heavy-tail distribution of job-sizes. By picking the fastest clone of every task, Dolly effectively reduces the slowest task from running 8× slower on average to 1.06×, thus, effectively eliminating all stragglers.

2) Does Dolly mitigate stragglers?

Unless specified otherwise, the cloning budget β is 5% and utilization threshold τ is 80%. Dolly improves the average completion time of jobs by 42% compared to LATE and 40% compared to Mantri, in the Facebook workload. The corresponding improvements are 27% and 23% in the Bing workload.

Small jobs (bin-1) benefit the most, improving by 46% and 37% compared to LATE and 44% and 34% compared to Mantri, in the Facebook and Bing workloads. This is because of the power-law in job sizes and the policy of admission control.

V. ANALYSIS

TABLE I ANALYSIS OF DIFFERENT STRAGGLER HANDLING TECHNIQUES

Name	Method	Factors								
		Memory Utilisation	**Replication**	**Disk Utilisation**	**Straggler Detection**	**Data Set**	**Delay**	**Speed**	**Reliability**	**Confidence Measure**
LATE	Reactive	Medium	Medium	No	Yes	No	High	Low	Medium	Average
SAMR	Reactive	Medium	Medium	Medium	Yes	Yes	Low	Medium	High	Good
ESAMR	Reactive	Medium	Medium	Medium	Yes	Yes	Low	Medium	High	Good
Dolly	Proactive (Cloning)	Highest	High	No	No	No	No	Good-small tasks	High	--
Wrangler	Proactive	High	No	High	Yes	Yes	High	Good	Highest	Good

VI. CONCLUSION AND FUTURE SCOPE

A. Conclusion

In the course of this study we have reviewed and analysed various straggler mitigation approaches. We have studied the broad sub-categories of straggler mitigation i.e. proactive and reactive methods. The proactive techniques proving to have better performance along with lower disk and memory utilization. We have compared LATE, SAMR, ESAMR techniques of Speculative Execution, Dolly technique of Cloning, Proactive Wrangler technique along with Machine learning Approach. Wrangler technique with Machine learning approach is the best amongst the other approaches compared. We have presented a brief analysis of the different mitigation techniques based on various performance parameters.

B. Future Scope

The existing techniques are meant for straggler mitigation or for reducing the impact of stragglers. In future we can have techniques for elimination of stragglers altogether without resource blacklisting and optimal use of resources. Although being the best of the existing straggler mitigation approaches Proactive Machine learning Wrangler technique has its own shortcomings too. For datasets having a huge variance in the numbers obtained across multiple nodes this technique is not much effective.

Thus we could try to generalise the technique for all kinds of datasets eventually improving it's confidence measure too.

REFERENCES

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: *Simplified data processing on large clusters.* In Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation-Volume6,OSDI'04,pages10–10,Berkeley, CA, USA, 2004. USENIX Association.

[2] Ganesh Ananthanarayanan, Srikanth Kandula, Albert Greenberg,IonStoica,YiLu,BikasSaha,andEdwardHarris. *Reining in the outliers in map-reducec lusters using mantri.* InProceedingsofthe9thUSENIXConferenceonOperatingSystems Design and Implementation, OSDI'10, pages 1–16, Berkeley, CA, USA, 2010. USENIX Association.

[3] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy H. Katz, and Ion Stoica. *Improving mapreduce performance in heterogeneous environments*. In OSDI, 2008.

[4] Q. Chen, D. Zhang, M. Guo, Q. Deng, and S. Guo, *"SAMR: A self adaptive mapreduce scheduling algorithm in heterogeneous environment,"* in Proceedings of the 10th IEEE International Conference on Computer and Information Technology, CIT '10, (Washington, DC, USA), pp. 2736–2743, IEEE Computer Society, 2010

[5] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, and I. Stoica. *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.* In USENIX NSDI, 2012.

[6] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. *E□ective straggler mitigation: Attack of the clones*. In NSDI, 2013.

[7] Neeraja J. Yadwadkar, Ganesh Ananthanarayanan, and Randy Katz. Wrangler: *Predictable and faster jobs using fewer resources.* In Proceedings of the ACM Symposium on Cloud Computing, SOCC '14, pages 26:1–26:14, New York, NY, USA, 2014. ACM.

[8] R. Nanduri, N. Maheshwari, A. Reddyraja, and V. Varma, *"Job awarescheduling algorithm for mapreduce framework,"* in Proceedings of the 3rd International Conference on Cloud Computing Technology and Science, CLOUDCOM '11, (Washington, DC, USA), pp. 724–729, IEEE Computer Society, 2011.