Journal of Engineering Science and Technology
Vol. 16, No. 6 (2021) 4864 - 4878

IMPROVING THE EFFICIENCY OF SPECULATIVE EXECUTION STRATEGY IN HADOOP USING AMAZON ELASTICACHE FOR REDIS

KAVITHA C.[1,*], ANITA X.[2], SHIRLEY SELVAN[3]

[1]Department of Computer Science and Engineering, Sathyabama Institute of Science and Technology, Chennai-600119, TamilNadu, India
[2]School of Computer Science and Engineering, Vellore Institute of Technology, Chennai Campus, Chennai-600127, TamilNadu, India
[3]Department of Electronics and Communication Engineering, St. Joseph's College of Engineering, Chennai-600119, TamilNadu, India
*Corresponding Author: kavitha4cse@gmail.com

Abstract

MapReduce has become pervasive for processing a huge volume of jobs in a distributed environment. Hadoop, an open source widely used implementation of MapReduce can be set up on massive computing clusters. Hadoop parallelizes a job into multiple chunks and handles the execution of tasks on large commodity clusters. This may result in a straggling or slow task which slows down the overall job execution time. Hadoop enables the speculative execution feature to handle such stragglers, but such a feature is not more efficient and affects the cluster efficiency. Moreover, the speculative task starts from the very beginning of the original task. This paper aims to improve the native speculative execution in Hadoop and ensures that speculative execution brings maximum benefit to the overall execution of a job. The proposed system allows the speculative task to resume the execution from the state where the original tasks has left off by skipping the already computed data. Once the task is identified as slow, the processed data of the task is checkpointed in some sort of external storage system and kills the original task to avoid the wastage of cluster resources. The speculative task takes the checkpoint information and resumes the execution from the state where the original tasks have left off by skipping the already computed data. The source code is modified to make changes to the structure of MapReduce computations for speculative tasks checkpointing. The node processing capability is computed to schedule the speculative task to the faster node. The external source used here is an Amazon ElastiCache for Redis which an in-memory key is-value store for a faster recovery of the checkpointed information. The proposed approach is evaluated by varying the different size datasets with different benchmarking programs It is implemented on top of Hadoop 2.6.5 and the experimental results show that the job execution time is superior to that in the native Hadoop when handling the speculative execution.

Keywords: Backup task, In-memory cache, Original task, Slow tasks, Speculative execution, Stragglers.

1. Introduction

MapReduce is a parallel data processing framework designed for data-intensive applications [1]. It parallelizes the execution of a job across multiple compute nodes. Hadoop, an open-source implementation of MapReduce executes jobs on commodity hardware. The execution time of a job is considered as the main performance metric. The main problem with the standard Hadoop is that when tasks are divided across many nodes, there is a possibility for a few slow nodes. Map tasks that run on those nodes will slow down the completion time of a job [2]. Reduce tasks can start their execution only when all the intermediate results get available. This delays the execution of the reduce tasks. Also, when reduce tasks runs on a slower node, it may delay the overall job final result. Therefore, the job completion time determines the slow task or stragglers. Hadoop runs speculative execution for such a slow task by scheduling it to faster nodes to make the computation faster. The speculative tasks are launched for those tasks which have not made any progress on average. It is just the backup of an original task which starts the execution from scratch on another node. Hadoop kills the speculative task if the original task has completed the execution before the speculative tasks. i.e., the task which has completed its execution first is accepted and the execution of another task is stopped. [3]. There are two types of speculative execution strategies such as cloning scheme and straggler detection method. In Cloning based strategy, several copies of a task are executed in parallel with the initial task until the resource utilization becomes low. Under straggler detection methods, task progress is monitored, and backup copies are created after stragglers are identified [4-6]. Straggler detection is suitable for lightly and heavily loaded clusters whereas cloning scheme is suitable only for lightly loaded cluster [4]. Load balancing can be done dynamically to determine lightly or heavily loaded clusters. Speculative execution can also be enhanced for cluster performance. [7].

In Hadoop's native scheduler, few considerations that affect the performance of MapReduce execution are:

1. Running duplicate tasks increases the cluster load.
2. Killing the speculative task when the original task completes its execution first results in the wastage of cluster resources.
3. All speculative task has to re-execute the partially processed data of the original tasks from disk [8].

In this paper, the effectiveness of the speculative execution feature is improved by utilizing the checkpoint restart to improve the overall execution time. Also, the original task is killed after scheduling the speculative task to another node and resuming to the current point of execution. This avoids redundant computation and the wastage of cluster resources. The partially processed data of an original task i.e., slow task is backed up to the external storage system-Redis Amazon ElastiCache [1] which is an in-memory key-value store that can be used as a cache.

The contributions of this work are listed,

1. Hadoop's native speculative execution is modified and implemented the optimized approach on top of the Hadoop 2.6.5: this prototype includes changes to the structure of MapReduce computations for speculative tasks checkpointing.

2. The changes include,
    - Allows the new speculative task's attempt to be allocated to the faster node and continue the execution from the latest data offset of the original task utilizing the checkpoint restart.
    - Selects the proper backup node by extending MRAppMaster class to compute the processing capability of each node and the speculator class in Hadoop source code is customized for both the map phase and reduce phase for checkpoint restart.
3. Hadoop 2.6.5 is integrated with Amazon ElastiCache for Redis through the client code Jedis.
4. The performance of new speculative execution is evaluated and compared with standard Hadoop using HiBench Benchmarking suit.

The rest of the paper is organized as follows. Section 2 is related works. Section 3 describes the design and implementation of the proposed approach. Section 4 describes the performance of the proposed framework. Section 5 concludes this paper.

2. Related Works

Many research studies have been conducted to improve the performance of speculative execution in different aspects. Some authors have focused on implementing straggler detection to identify the stragglers accurately, while others focused on improving the performance by cloning the speculative task. Moreover, some research areas dealt with optimizing the speculative execution by improving the cluster performance through load balancing in the cluster. Also, node capabilities are found for assigning the speculative task to faster nodes. Hence many of these approaches have been designed mainly to decreases the execution time of a Hadoop MapReduce job.

Kavitha and Anita [1] proposed a Task Failure Resilience Technique (TFR), a new MapReduce prototype system which is designed and implemented on top of Hadoop 2.6.5 using the Amazon ElastiCache for Redis as a backend for faster recovery during the task failures. The main goal of Task Failure Resilience (TFR) technique for a map and reduce phases are to recover the processed bytes of a map/reduce tasks at a faster rate to continue the execution from the state where the failure has occurred. Whenever a map or reduce task generates an intermediate key-value pair, the pair is sent along with the metadata to the redis instance. Hadoop application is integrated with Amazon ElastiCache for Redis by customizing the input and output classes of Hadoop. TFR performs better than the standard Hadoop.

Wang et al. [8] proposed a Partial Speculative Execution (PSE) technique that uses checkpointing information to start the speculative execution from where the original task has left. It presents two checkpointing techniques and corresponding recovery techniques to launch the speculative execution from checkpointing. This improves the efficiency of speculative execution only by 24%. The major drawback is that too much time is taken for checkpointing. PSE, LATE, and no speculation is evaluated on Hadoop 0.21.0 by adopting various kinds of benchmarks such as WordCount, Grep and K means. PSE technique is 7% faster than the LATE technique and 43% with no speculation.

Zaharia et al. [3] proposed a LATE (Longest Approximate Time to End) scheduling technique which improves the speculative execution in heterogeneous

environments. This technique estimates the completion time of a task and executes the tasks speculatively that finish farthest into the future. It calculates the slow task in the static manner. The 70% of the speculative tasks are killed which wastes the cluster resources and brings no benefit to the overall execution time of a job. One more shortcoming of the LATE scheduler is that it does not distinguish Map/Reduce straggler nodes. The LATE technique improves the Hadoop response time by a factor of 2 in the cluster consisting of 200 Virtual machines on Amazon EC2.

Chen et al. [9] proposed a Self-Adaptive MapReduce (SAMR) scheduling algorithm which calculates the weight of each stage of the map and reduce tasks. This technique uses the core idea of LATE scheduler. It calculates the progress rate for one stage and uses it to compute the estimated time to end for overall tasks. Hence this certainly fails to give approximate time to end causing unnecessary backup tasks to fast tasks and leaving the slow task to backup. This situation results in the wastage of resources. SAMR decreases the execution time of a job up to 25% than Hadoop's native scheduler and 14% than the LATE scheduler. The main drawback in the SAMR technique is that it fails to consider the characteristics of a job such as dataset size, execution time, or weight.

Soualhia et al. [10] proposed ATLAS (an Adaptive Failure-aware Scheduler for Hadoop) which predicts task scheduling outcomes and adjusts scheduling decisions to decrease the impact of tasks failed attempts. Log files in Hadoop are analyzed, and predictive models for task failure are designed. It provides better scheduling to improve the MapReduce performance. This requires the tasks to be assigned to the TaskTracker with sufficient resources. In this work, the improvement in reducing the number of failed tasks has decreased only by 39%. One drawback is that the failure of one map task leads to the dependent reduce tasks failure. Dean and Ghemawat [11] proposed a technique, that backup the executions of the remaining in-progress tasks, when a job is close to completion.

Farhang and Safi-Esfahani [12] proposed a SEWANN framework, Speculative Execution with Artificial Neural Network used to identify stragglers effectively in a heterogeneous environment. This technique uses a neural network to calculate the execution time of a task more accurately. SEWANN provides the correct estimation by using the information of a previously executed task. The drawback in this framework is that rerunning identified stragglers begins its computation from the beginning. SEWANN is evaluated with a simple single application 'wordcount program'. SEWANN outperforms LATE, SVR, Decision tree, and ESAMR techniques by 99%, 99%, 81% and 85%.

Wang et al. [13] proposed a new speculative execution scheme eSplash used to efficiently identify the stragglers accurately. eSplash classify the Hadoop cluster nodes based on the node computing capabilities. To implement this solution, Hadoop YARN version 2.7.1 source code is changed by modifying the Speculator class and RMContainerAllocator component in MRAppMaster class. Authors conducted experiments for eSplash on cloud platform and compared this scheme with native speculative execution and LATE scheme. This technique is evaluated with different workloads such as TeraSort, WordCount, Grep, and Wordmean. eSplash outperforms better than the existing approach LATE by 65.7% and 76.7% for non-speculation. Xu et.al. [14] proposed a unifying optimization model for speculative execution, called Chronos. Probability of Completion before Deadline (PoCD) metric is defined in this framework to measure the probability of jobs that meet their deadlines. Cloning,

Speculative restart, and Speculative resume are popular strategies, used to systematically analyse PoCD. These three strategies are used by Chronos as a scheduling strategy to mitigate stragglers. Chronos outperforms standard speculative execution by 50% for net utility increase, PoCD by 80%, and cost improvement by 88%.

Jiang et al. [15] proposed an ORSE, Optimized Resource Scheduling Speculative Execution technique. It uses non-cooperative game theory to assign backup tasks without affecting the original task execution. Both the original task and backup task can be assigned evenly to the nodes in the Hadoop cluster which eventually utilize a greater amount of cluster resources. This technique allows the backup task to get executed at faster node and there may be multiple backup task gets executed at the same time. ORSE considers all backup tasks as participants to game theory since the game concentrates on multi-participation. The execution rate of the node is estimated to assess the node processing capability which helps in assigning the backup task to a fast node. ORSE outperforms the LATE and MCP technique by improvement rate 25.8% and 9.7%.

Jin et al. [16] proposed an optimized speculative strategy based on local data prediction in a heterogenous Hadoop environment. It collects information about the execution of a task in real-time and predict the remaining time of the current task using a local regression model. They also propose a benefit model to calculate the effectiveness of speculative execution. The task is identified as slow when it has the longest remaining time. The benefit model first calculates the benefits and cost of launching a backup task and secondly calculates the profits of the cluster in launching and not launching the backup task. They select the appropriate backup node with high computing performance. The process rate is taken as a node's capability and compare it with the standard deviation of progressing. Ananthanarayanan et al. [17] entailed a system that monitors the tasks and eliminates slow tasks based on the resources, and [5] presented an algorithm to alleviate slow tasks in small jobs by initiating multiple copies of every task. Blacklisting and speculative execution are two processes done by these techniques. Partitioning and balancing the tasks' workload are done to handle the slow tasks.

Quiané-Ruiz et al. [18] proposed a Checkpointing techniques- Recovery Algorithms for Fast-Tracking (RAFT) MapReduce. This technique requires each of the tasks to checkpoint the progress for fast recovery in case of failures. Query Metadata checkpointing scheme is proposed to deal with multiple node failures. Checkpointing the task progress on stable storage is not efficient and creates overhead. In this work, the RAFT technique outperforms standard Hadoop only by 23% under task and node failures.

3. Proposed Methodology

In this section, the speculative execution used by standard Hadoop is discussed and proposed the improved version of speculative execution.

3.1. Speculative execution in Hadoop

In large clusters where hundreds or thousands of nodes are involved, there may be few nodes that do not perform as fast as other. When one node does not perform well, this may cause delay in completion of overall job execution. To avoid this situation, speculative execution is launched in Hadoop which runs multiple copies of same tasks on different nodes. This results in increase in number of tasks than the number of splits. Doubling the tasks leads to doubling the amount of cluster

resources. Also, the execution is carried from the scratch leaving the partially processed data. Speculative execution happens due to following reasons:

- When the hardware is in unequal size, it causes the delay in execution of few tasks.
- Network failure
- Hardware failure
- Slow node

In a busy cluster, speculative execution may affect the overall throughput of data and wastage of resources. When too many speculative tasks are launched, it costs cluster resources.

The speculative execution is enabled by setting true to the following two properties:

- mapred.map.tasks.speculative.execution for map tasks
- mapred.reduce.tasks.speculative.execution for reduce task.

A speculative task is run based on an approach which compares each progress of a tasks to the average. The progress of each task is monitored by the JobTracker. It uses the progress score for each task to identify the stragglers. This progress score is set between 0 and 1. The threshold is maintained for task's average progress score for enabling the speculative execution: the task is considered as stragglers, when the task's progress score is less than the average progress score minus 0.2. Hadoop always monitors the progress of a task using a progress score which is between 0 and 1. The execution of a reduce tasks is divided into three phases [3]

- Copy phase: The task fetches the intermediate key-value pairs of map tasks output.
- Sort phase: Sorting the map results using the key.
- Reduce phase: A user-defined function can be used to the list of map results with each key.

Once the straggler task is identified, speculative execution can be launched for such long running tasks. Hadoop's native speculative execution clones the original tasks to another node which has completed the task execution faster. Both the original task and speculative tasks gets executed parallelly wasting the cluster resources. The execution of a speculative tasks begins from the scratch. The execution time of the job depends on the slowest map and reduce tasks. Figure 1 shows the native speculative execution in Hadoop.

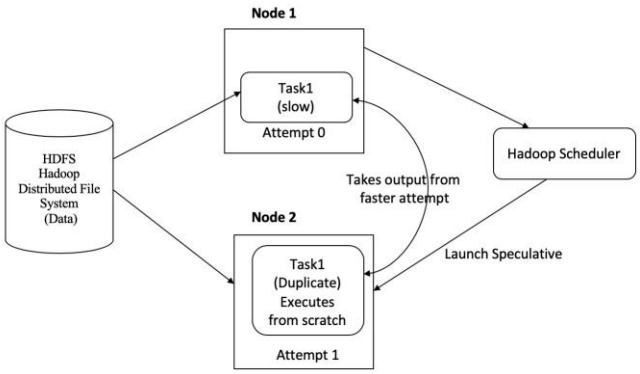

Fig. 1. Native speculative execution.

3.2. Proposed speculative execution framework

MapReduce model breaks the jobs into tasks and runs it in parallel to make the overall execution time of a job to be smaller. This makes the execution time of a job more sensitive to slow-running tasks. An optimized speculative execution is proposed for MapReduce jobs by improving the effectiveness of speculative execution, preventing the speculative tasks from recomputing the partially processed data. The partially processed data of a slow task (original task) is persisted in an external in-memory cache- Amazon ElastiCache for Redis. In-memory layer [1] can be added to the Hadoop application. When hundreds or thousands of map tasks are 'long' running' persisting such huge partially computed intermediate key-value pairs in the file system may lead to the disk failure and drag the performance of MapReduce execution. Amazon ElastiCache is an in-memory key-value store which accelerates the high-volume workload. It is fast and retrieves the data in a sub-millisecond.

By using a Java client- Jedis, the partially processed data of the original is ingested and retrieved with Redis. This client code is combined with the MapReduce framework for writing and reading data to and from the Redis instance parallelly. MapReduce programming is used to pull and push this partially processed data of the original tasks to any number of Redis instances. Data is read and written from a Redis hash. String fields are mapped to string values in Redis, much like a Java HashMap.

The speculator is instantiated in MRAppMaster (MapReduce Application Master). If the speculative execution for map or reduce task is turned on, speculator is created inside the MRAppMaster. Hadoop speculator uses the estimator to estimate the progress of each task. yarn.app.mapreduce.am.job.task.estimator.class is an estimator class which gives input to the speculator for estimating the run time of a task. The following class is modified to enable the custom speculative execution.

yarn.app.mapreduce.am.job.speculator.class: This speculative class is modified to implement the proposed speculative execution policy. org.apache.hadoop.mapreduce.v2.app.speculate has two interfaces speculator and TaskRuntimeEstimator. DefaultSpeculator class implements the Speculator interface. The implementation of improved speculative execution strategy employs a key idea is that the progress of original task attempt's data is recorded by updating it to the Redis storage. The speculative execution is customized for both the map phase and reduce phase. It is ensured that the new speculative attempt is executed fetching the latest data offset from the redis storage. The new speculative execution did not process the already processed bytes and can resume its execution from the state where the original task has left off. The proposed approach does not allow both the original and speculative attempt to run competing each other because initiating multiple identical task's execution and allowing these tasks to compete requires more computing resources in Hadoop. Hence this leads to the wastage of cluster resources. Once the stragglers are detected, speculative copy for the task is launched and the straggler task (original task) is killed to save the cluster resource and to avoid the redundant execution of the same tasks. The scheduler is made to ensure that for each original task, only one speculative copy is launched and is running at a time. For a reduce tasks, the proposed speculation execution strategy can be applied only when the reduce task is in the reduce phase since it needs to collect output data from all mappers. When a reduce tasks are identified as

stragglers, speculative copy of it will be launched and the original reduce tasks is killed after creating the checkpoints to the redis storage. The reduce tasks checkpoint information includes the unprocessed original reduce task key, offset of the sorted reduce inputs.

Checkpoints are stored in the redis instances. Checkpoint records includes hashKey, <key, value> pairs, input_offset, timestamp.

- hashKey is the unique key used to identify to which redis instance the checkpoint data are stored.
- <key,value> pairs are the partially processed intermediate key-value pairs of the slow map and reduce task.
- timestamp to record the time of occurrences of the event.
- For slow map task, recording the input_offset indicates the total number of bytes processed from the input blocks.
- For slow reduce task, recording the input_offset indicates the offset of the sorted reduce inputs.

The speculative task will obtain the checkpoint data from the redis instance and resume its execution from the last recorded input offset.

MRAppMaster class is extended to compute the processing capability of each node to allocate the speculative task to the faster node.

$$Node_{ProcessingCapability} = \frac{Node_{completed}}{Node_{processed}} \quad (1)$$

where the $Node_{completed}$ represents the ratio of tasks completed successfully and $Node_{processed}$ represents the ratio of total processed tasks which includes the tasks that are completed and failed.

Figure 2 shows the proposed speculative execution strategy in Hadoop.

1. Hadoop Scheduler identifies the slow tasks by calculating the progress score and average progress score of each category of tasks.
2. Once slow tasks are identified, the scheduler kills the execution of slow tasks once its progress is updated to the Redis storage. This avoids the resource wastage.
3. Schedulers choose the faster node and launches the speculative tasks to it.
4. The node which runs the speculative task remotely fetches the output results of the slow tasks from the Redis storage. Hence the speculative tasks skip the partial computation.
5. The output from the speculative tasks is considered.

3.3. Workflow of customized speculative map and reduce tasks execution

Algorithm 1 and 2 outlines the workflow of proposed speculative execution framework for map and reduce task and to make the speculative task to obtain the checkpoint data from the redis instance and resume its execution from the last recorded input offset of the original task.

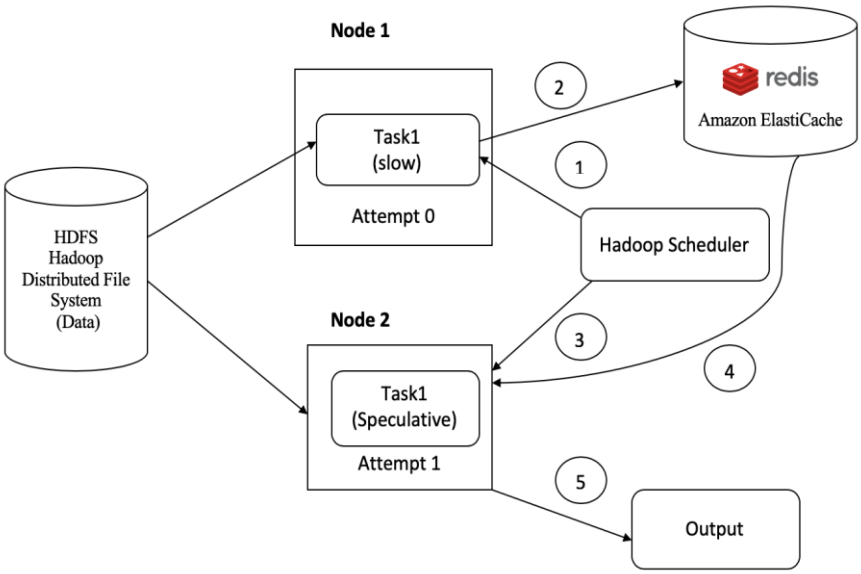

Fig. 2. Proposed speculative execution.

Algorithm 1: Customized Speculation for Map Task

Require:
$Task_i$= {$Task_1$, $Task_2$, $Task_3$, …...} a set of tasks running
$Node_i$ = {N_1, N_2, N_3….} the set of nodes that are available
Begin

[1] for $Task_i \in Task_{running}$ (i=1,2,3,…) do
[2] TaskType Task = taskID.getTaskType();
[3] if ($Task_i$ == TaskType.map)
[4] getEstimator($Task_i$);
[5] if $Task_i$ is slow then
[6] computeMapSpeculation() // Launch Speculation
[7] progress= $Task_i$.getProgress();
[8] /* user defined code to write partially processed data of an original task */
[9] Jedis j= new Jedis(host);
[10] j.connect();
[11] Jedis new= jedisMap.get(Math.abs (key.hashcode()) % jedisMap.size();
[12] new.hset(hashKey, key.toString(),value.toString(), timestamp, input_offset);
[13] End
[14] if $Task_i$.is Complete writing then
[15] retireMap($Task_i$);
[16] End
[17] End
[18] if T_{backup} is worth running then:

```
[19]        Execute T_backup on Node N_i
[20]                new.hget(hashKey,  key.toString(), value.toString(),
                     timestamp, input_offset);
[21]                speculativeMap(Object Key,  Text Value);
End
```

Algorithm 2: Customized Speculation for Reduce Task

```
Begin
[1] for Task_i  ∈  Task_running ( i=1,2,3,…) do
[2]   TaskType Task = taskID.getTaskType();
[3]   if (Task_i == TaskType.reduce)
[4]      if Task_i in reduce phase then
[5]         getEstimator(Task_i);
[6]                 if Task_i is slow then
[7]            computeReduceSpeculation () // Launch Speculation
[8]            progress= Task_i.getProgress();
[9]          /* user defined code to write partially processed data of an original
               task */
[10]             Jedis j= new Jedis(host);
[11]             j.connect();
[12]               Jedis new= jedisMap.get (Math.abs (key.hashcode()) %
                   jedisMap.size();
[13]             new.hset(hashKey,  key.toString(),value.toString(),
                 timestamp, input_offset);
[14]         End
[15]      End
[16]   End
[17] End
[18]     if Task_i.is Complete writing then:
[19]        retireReduce(Task_i);
[20] if T_backup is worth running then:
[21]    Execute T_backup on Node N_i
[22]         new.hget(hashKey,  key.toString(), value.toString(), timestamp,
           input_offset);
[23]        speculativeReduce(Object Key,  Text Value);
End
```

Algorithm 1 outlines the workflow of optimized speculative execution for Map Tasks. Application Master manages and supervises task running of a stage. For a running task, get the type of the task (from line no. 1-2). If the type of the running task is map, then the estimator property is used to get the task completion time at runtime. If a map task is running beyond the estimated run time, it is considered as the slow map task (from line no. 3-4). Once the slow task is detected, speculative execution is launched, and the progress of the slow map task (original task) will be written to the redis instance by connecting to the redis server via the jedis client code. After establishing the connection, get the right redis instance where the progress of the slow map task will be written to. The progress includes the partially processed intermediate key-value pairs and number of input bytes processed are recorded to the redis instance.

Jedis set method can be used to write the progress of the original task out to redis from Hadoop application. (From line no. 5-13). The slow map task is killed after copying its progress to the redis instance (from line no. 14- 17). MRAppMaster class is extended to find the faster node by computing the processing capability of a node and to schedule the speculative tasks to the faster node. The node which runs the speculative task gets the necessary information from the redis cache and begins the execution. (From line no. 18-19). This avoids wastage of resources. Speculative task is executed on the faster node by retrieving the last saved bytes of the slow task using the Jedis "get" method by attaching the timestamp to retrieve the data recorded at the time of occurrence of that event and input_offset for the speculative map task to resume execution from the latest bytes. (From line no. 20-21).

Algorithm 2 outlines the workflow of optimized speculative execution for Reduce Tasks. If the type of the running task is reduced, then the estimator property gets the task completion time at runtime. If a reduce task is running beyond the estimated run time, it is considered as the slow reduce task. For a reduce tasks, the proposed speculation execution strategy can be applied only when the reduce task is in the reduce phase since it needs to collect output data from all mappers. (From line no 1-5). The important consideration in the speculative execution for reduce tasks is that it gets its input from more than one map task which are running on different nodes, so the data transfer takes place in case of reduce tasks. Running duplicate reduce task causes the data transfer to happen more than once which affects the network load. Hence, when a reduce tasks are identified as stragglers, speculative copy of it will be launched and the original reduce tasks is killed after recording its progress to the redis storage (from line no. 6-19). The reduce tasks checkpoint information includes the partially processed key-value pairs of the slow reduce task and the offset of the sorted reduce inputs.

To avoid excessive backup task, only one copy per task is allowed at most. This optimized speculative execution computes speculation periodically based on the mapping structure.

4. Results and Discussion

The performance and sensitivity of the improved version of speculative execution is evaluated on a heterogeneous cluster of 10 nodes in a virtual environment: 1 master node and 9 slave nodes which are grouped under different levels with different capacities.

- Level 1: 4 machines with 8 core Intel processor operating at 2.40 GHz having a memory with 8 GB and HDD 1 TB
- Level 2: 3 machines with 4 core Intel processor operating at 3.3 GHz having a memory with 4 GB and HDD 500 GB
- Level 3: 2 machines with 3 core Intel processor operating at 2.8 GHz having a memory with 3 GB and HDD 500 GB.

Hadoop is configured with one node as the master node allowing it to host NameNode and Resource Manager. The rest of each 9 nodes are allowed to run one DataNode and one Node Manager. In Level 1, each NodeManager was configured with 8 cores and hence can run a maximum of 8 map or reduce tasks simultaneously. In Level 2, each NodeManager was configured with 4 cores and hence can run a maximum of 4 map or reduce tasks simultaneously. In Level 3,

each NodeManager was configured with 3 cores and hence can run a maximum of 3 map or reduce tasks simultaneously.

New prototype is built on top of Hadoop 2.6.5 and use the baseline version for comparison. This prototype can work on any higher versions of Hadoop 2.6.5. Redis 2.8.22 is deployed on the Amazon ElastiCache. Jedis is a client library in java which is used to connect Hadoop application and Redis. server. Redis is used as an in-memory data store to store and retrieve the key-value pairs of original tasks to and from the Hadoop. The capacity of Redis is 42.84 GB and the node type is m4.xlarge. The performance is evaluated for the following parameters such as execution time and efficiency of speculative execution.

The HiBench Benchmarking suit is used to assess the performance of optimized speculative execution. The experiment is run by varying the different size datasets for different benchmarking programs such as WordCount, Sort, and TeraSort. These benchmarking programs represent different CPU-intensive and data-intensive jobs. WordCount workload is CPU-intensive whereas the Sort and TeraSort workloads are Disk-intensive jobs.

1. WordCount: Given a text input file, WordCount benchmark program is used to counts the occurrences of each word.
2. Sort: Sort benchmark is used to sort the lines in the given input text dataset lexicographically.
3. TeraSort: TeraSort benchmark function is similar to Sort benchmark, but it provides an improved version of sort where the equal load is distributed on all nodes.

These workloads are evaluated with different datasets varied from 5 GB to 20 GB. Experiments are repeated 3 times and the proposed system is compared with standard Hadoop to show the performance improvement. The result of this analysis uses execution time as the primary metric. The average of these experiments is taken for each dataset is shown in the graph. Manually 2 Slave nodes are slowed down to evaluate the optimized speculative execution with stragglers. Figures 3, 4 and 5 shows the job completion time of the WordCount, sort and TeraSort workload with stragglers. The displayed results are obtained using Algorithm 1 and 2. The graph shows that the result for WordCount, sort and TeraSort workload running with different file sizes. It is evident that the larger the input dataset is, the longer the overall job execution time is taken.

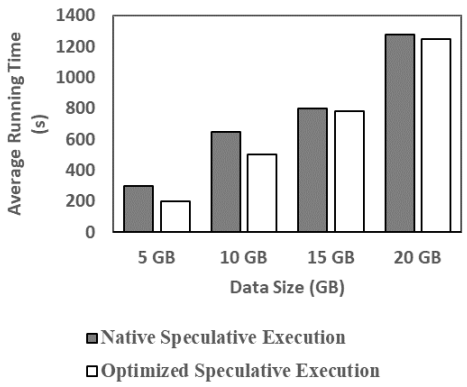

Fig. 3. Execution time of the WordCount benchmark with stragglers.

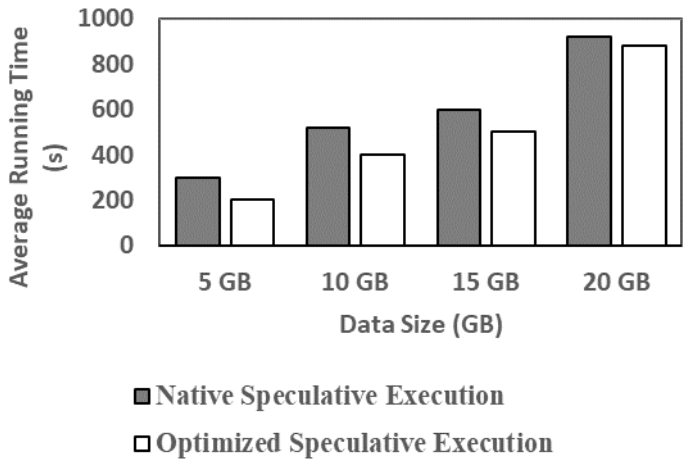

Fig. 4. Execution time of the Sort benchmark with stragglers.

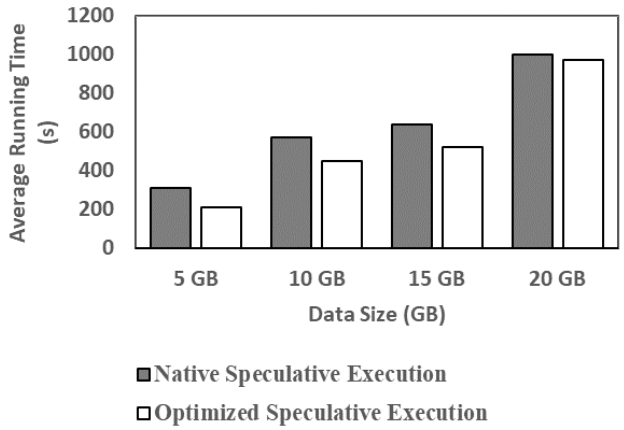

Fig. 5. Execution time of the TeraSort benchmark with stragglers.

5. Conclusions

Speculative execution in the Hadoop framework has been used to alleviate slow tasks in the MapReduce framework. Unfortunately, the speculative execution in standard Hadoop 2.6.5 could not work effectively as expected. It is found that recomputing the partially processed data of an original task can be avoided. An optimized speculative execution is implemented in Hadoop 2.6.5 to eliminate the costs. By persisting the partially processed data in Amazon ElastiCache for Redis, speculative tasks start from the state where the original tasks left instead of starting from scratch. The proposed speculative strategy is experimentally evaluated under the different benchmarking program and compared it to the native speculative execution. This proposed strategy performs better than the native approach in Hadoop 2.6.5.

Abbreviations

ATLAS	Adaptive Failure-aware Scheduler
LATE	Longest Approximate Time to End
MRApp Master	MapReduce Application Master
ORSE	Optimized Resource Scheduling Speculative Execution
PSE	Partial Speculative Execution
PoCD	Probability of Completion before Deadline
RAFT	Recovery Algorithms for Fast-Trackin
SAMR	Self-Adaptive MapReduce
SEWANN	Speculative Execution with Artificial Neural Network

References

1. Kavitha, C.; and Anita, X. (2020). Task failure resilience technique for improving the performance of MapReduce in Hadoop. *ETRI Journal*, 42(5), 748-760.
2. Wu, H.; Li, K.; Tang, Z.; and Zhang, L. (2014), A heuristic speculative execution strategy in heterogeneous distributed environments. *Proceedings of the Sixth International Symposium on Parallel Architectures, Algorithms and Programming*. Beijing, China, 268-273.
3. Zaharia, M.; Konwinski, A.; Joesph, A.D.; Katz, R.; and Stoica, I. (2008). Improving MapReduce performance in heterogeneous environments. *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation*. San Diego, California, 29-42.
4. Xu, H.; and Lau, W.C. (2015). Optimization for speculative execution in a MapReduce-like cluster. *Proceedings of the IEEE Conference on Computer Communications*. Hong Kong, China, 1071-1079.
5. Ananthanarayanan, G.; Ghodsi, A.; Shenker, S.; and Stoica, I. (2013). Effective straggler mitigation: attack of the clones. *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation*. Lombard, Chicago, 185-197.
6. Ananthanarayanan, G.; Hung, M.-C.C.; Ren, X.; Stoica, I.; Wierman, A.; and Yu, M. (2014). Grass: Trimming stragglers in approximation analytics. *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation*. Seattle, Washington, 289-302.
7. Mathew, J.; Mathew, T.J.; and Scaria, T. (2020). Improved Hadoop cluster performance by dynamic load and resource aware speculative execution and straggler node detection. *International Journal of Engineering and Advanced Technology*, 9(4), 2370-2377.
8. Wang, Y.; Lu, W.; Lou, R.; and Wei, B. (2015). Improving MapReduce performance with partial speculative execution. *Journal of Grid Computing*, 587-604.
9. Chen, Q.; Zhang, D.; Guo, M.; Deng, Q.; and Guo, S. (2010). SAMR: A self-adaptive MapReduce scheduling algorithm in heterogeneous environment. *Proceedings of the 10th International Conference on Computer and Information Technology*. Bradford, United Kingdom, 2736-2743.

10. Soualhia, M.; Khomh, F.; and Tahar, S. (2015). ATLAS: An adaptive failure aware scheduler for Hadoop. *Proceedings of the 34th International Performance Computing and Communications*. Nanjing, China, 1-8.
11. Dean, J.; and Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1), 107-113.
12. Farhang, M.; and Safi-Esfahani, F. (2020). Recognizing MapReduce straggler tasks in big data infrastructures using artificial neural networks. *Journal of Grid Computing*, 18(3), 879-901.
13. Wang, J.; Wang, T.; Yang, Z.; Mi, N.; and Sheng, B. (2016). eSplash: efficient speculation in large scale heterogeneous computing systems. *Proceedings of the IEEE 35th International Performance Computing and Communications Conference*. Las Vegas, NV, USA, 1-8.
14. Xu, M.; Alamro, S.; Lan, T.; and Subramaniam, S. (2018). Chronos: a unifying optimization framework for speculative execution of deadline-critical MapReduce jobs. *Proceedings of the IEEE 38th International Conference on Distributed Computing Systems*. Vienna, Austria, 718-729.
15. Jiang, Y.; Liu, Q.; Dannah, W.; Jin, D.; Liu, X.; and Sun, M. (2020). An optimized resource scheduling strategy for Hadoop speculative execution based on non-cooperative game schemes. *Computers, Materials and Continua*, 62(2), 713-729.
16. Jin, D.-D.; Liu, Q.; Liu, X.-D.; and Linge, N. (2019). An optimized speculative execution strategy based on local data prediction in heterogeneous Hadoop environment. *Journal of Computers*, 30(3), 130-142.
17. Ananthanarayanan, G.; Kandula, S.; Greenberg, A.; Stoica, I.; Lu, Y.; Saha, B.; and Harris, E. (2010). Reining in the outliers in map-reduce clusters using mantri. *Proceedings of the 9th Conference on Operating Systems Design and Implementation*, 265-278.
18. Quiane´-Ruiz, J.A.; Pinkel, C.; Schad, J.; and Dittrich, J. (2011). Rafting MapReduce: Fast recovery on the raft. *Proceedings of the 27th International Conference on Data Engineering*. Hannover, Germany, 589-600.