International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-4, Issue-6 E-ISSN: 2347-2693

Optimization of Map Reduce Using Maximum Cost Performance Strategy

A. Saran Kumar[1], V. Vanitha Devi[2]

[1,2] *Dept. of CSE, Kumaraguru College of Technology, Coimbatore, India*

www.ijcseonline.org

Received: Jun/02/2016 Revised: Jun/10/2016 Accepted: Jun/24 /2016 Published: Jun/30 / 2016

Abstract – Big data is a buzzword, used to describe a massive volume of both structured and unstructured data that is so large that it's difficult to process using traditional database and software techniques. In most enterprise scenarios the data is too big or it moves too fast or it exceeds current processing capacity. Big data has the potential to help companies improve operations and make faster, more intelligent decisions.Parallel computing is a frequently used method for large scale data processing. Many computing tasks involve heavy mathematical calculations, or analysing large amounts of data. These operations can take a long time to complete using only one computer. Map Reduce is one of the most commonly used parallel computing frameworks. The execution time of the tasks and the throughput are the two important parameters of Map Reduce. Speculative execution is a method of backing up of slowly running tasks on alternate machines. Multiple speculative execution strategies have been proposed, but they have some pitfalls: (i) Use average progress rate to identify slow tasks while in reality the progress rate can be unstable and misleading, (ii) Do not consider whether backup tasks can finish earlier when choosing backup worker nodes. This project aims to improve the effectiveness of speculation execution significantly. To accurately and promptly identify the appropriate tasks, the following methods are employed: (i) Use both the progress rate and the process bandwidth within a phase to select slow tasks, (ii) Use exponentially weighted moving average (EWMA) to predict process speed and calculate a task's remaining time, (iii) Determine which task to backup based on the load of a cluster using a cost-benefit model.

Keywords*:* Map reduce, Cost Performance strategy, Big Data, Stragglers, Speculation

1. INTRODUCTION

1.1 MAP REDUCE FRAMEWORK

Big data is high-volume, high-velocity and high-variety information assets which grow exponentially. It is the ocean of information in which vast zeta bytes of data are flowing from multiple sources. It demands cost-effective and innovative forms of information processing for enhanced insight and decision making. With big data solutions, organizations can dive into all data and gain valuable insights that were previously unimaginable. Big data is difficult to work with using most relational database management systems and visualization packages and hence requires parallel processing. Map Reduce is a parallel computing framework for large scale data processing. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. It was originally developed by google which replaced their original web indexing algorithm in 2004. A variant of Map Reduce namely Hadoop is currently being used by yahoo, facebook, amazon and so on. Google processes 20 PB/day, Jet engine produces 10 TB of data every 30 minutes of flight time and Facebook executes 3000 jobs per day**.** This shows that the smallest increase in performance will have a significant impact.Map Reduce is useful for large, long-running jobs that cannot be handled within the scope of a single request [1]. It can be used for tasks like

- Analyzing application logs
- Aggregating related data from external sources
- Transforming data from one format to another
- Exporting data for external analysis and so on

The key features of Map Reduce are:

- **Scale-out Architecture** - Add servers to increase processing power
- **Security & Authentication** - Makes sure that only approved users can operate against the data in the system
- **Resource Manager** - Employs data locality and server resources to determine optimal computing operations
- **Optimized Scheduling** - Completes jobs according to prioritization
- **Flexibility** - Procedures can be written in virtually any programming language
- **Resiliency & High Availability** - Multiple job and task trackers ensure that jobs fail independently and restart automatically

Map and Reduce are higher-order functions in the Map reduce framework. Map function applies an operation to all elements in a list and Reduce function is like "fold" which aggregate elements of a list. The input file is divided into blocks and to each block a map function is applied which parses the input and generates intermediate key/value pairs. The reduce function aggregates the intermediate result by combining the values associated with the same key. The

final output of the reduce function is written to a file or a file system [2].

The following sequences of steps take place when a Map Reduce operation is performed. The number of map tasks is denoted as *M,* the number of reduce tasks by *R,* and the number of computers as C.

- The Map Reduce library of the user program splits the input file into *M* pieces. It also starts several copies of the program on the *C* machines.
- One of the copies of the program is the master and the rest are the slaves. The master selects idle slaves and assigns a map or reduce task to each one of them.
- The slave with an assignment of a map task reads the contents of the associated pieces of input assigned to it. It parses the input data to form the key-value pairs. It passes each key-value pair to a user-defined map function. The map function emits intermediate key-value pairs which are buffered in memory.
- The key-value pairs buffered in memory are saved to the node's local disk and partitioned into *R* pieces by the partitioning function. The locations of the pairs are sent to the master.
- The master notifies the reduce slaves about the locations of the pairs, and the reducers retrieve the buffered pairs from the local disks of the map slaves. Then the pairs are sorted according to their keys, and the pairs with the same keys are grouped together.
- Each reduce slave passes the intermediate key and the associated values to the reduce function. The output produced by the reduce function is returned to the output file specified by the user.
- After completions of all map and reduce tasks, the master notifies the user program.
- The final outputs of the reduce functions are available as partitioned files inside the *R* partitions. These partition files can be used as input for another Map Reduce process or be fed to programs that are designed to handle partitioned input files, in order to combine them.
- Input and final outputs are stored on a distributed file system. Scheduler tries to schedule map tasks "close" to physical storage location of input data. Intermediate results are stored on local FS of map and reduce workers. Output is often input to another map reduce task.

1.2 CHALLENGES IN MAP REDUCE FRAMEWORK

- Lack of performance and scalability
- Lack of flexible resource management
- Lack of application deployment support
- Lack of quality of service assurance
- Lack of multiple data source support

1.3 SPECULATIVE EXECUTION

Map Reduce can automatically handle failures. If a node is available but is performing poorly, then the task running on it will take a long time to finish and is called a straggler, Map Reduce runs a speculative copy of its task also called a "backup task" on another machine with the expectation to finish the computation faster. Speculative execution is a common approach for dealing with the straggler problem by simply backing up those slow running tasks on alternative machines. Without this mechanism of speculative execution, a job would be as slow as the misbehaving task. The scheduler schedules backup executions of the remaining in-progress tasks. The task is marked as completed whenever either the primary or the backup execution completes. Slow nodes/stragglers are the main bottleneck for jobs not finishing in time. So to reduce response time, stragglers are speculatively executed on other free machines [3].

1.4 CHALLENGES IN SPECULATIVE EXECUTION

Speculative execution involves the following challenges

- Identifying free resource for the speculative tasks to be executed
- Choosing proper worker nodes to run speculative tasks
- A means to distinguish "stragglers" from nodes that are slightly slower.
- Finding stragglers at the earliest.

2. LITERATURE SURVEY

2.1 RAFTing MAP REDUCE

RAFT focuses on simplicity and also non-intrusiveness, in order to be implementation independent. To efficiently recover from task failures, RAFT exploits the fact that Map Reduce produces and persists intermediate results at several points in time. RAFT piggy-backs checkpoints on the task progress computation. To deal with multiple node failures, query metadata check pointing is used. Therefore the mapping between input key-value pairs and intermediate data are tracked. Thereby, RAFT does not need to re-execute completed map tasks entirely. Instead RAFT only recomputed intermediate data that were processed by local reducers and hence not shipped to another node for processing. A scheduling strategy taking full advantage of these recovery algorithms is introduced. On implementing RAFT on top of Hadoop and evaluating it on a 45-node cluster using three common analytical tasks, the results showed that RAFT outperforms Hadoop runtimes by 23% on average under task and node failures[4].The disadvantages are:

- Not sufficient switching, due to more delay in switching.
- Delay in finding a less workload node.

The techniques used are:

- Local Check pointing (RAFT-LC).
- Remote Check pointing (RAFT-RC).

2.2 DELAY SCHEDULING

Fair scheduler is developed to allocate fair share of capacity to all the users. Two locality problems identified when fair sharing is followed are–head-of-line scheduling and sticky slots. The first locality problem occurs in small jobs (jobs that have small input files and hence have a small number of data blocks to read). The problem is that whenever a job reaches the head of the sorted list for scheduling, one of its tasks is launched on the next slot that becomes free irrespective of which node this slot is on. If the head-of-line job is small, it is unlikely to have data locally on the node that is given to it. Head-of-line scheduling problem was observed at Facebook in a version of HFS without delay scheduling. The other locality problem, sticky slots, is that there is a tendency for a job to be assigned the same slot repeatedly. The problems aroused because following a strict queuing order forces a job with no local data to be scheduled [5].

To overcome the Head of line problem, scheduler launches a task from a job on a node without local data to maintain fairness, but violates the main objective of Map Reduce that schedule tasks near their input data. Running on a node that contains the data (node locality) is most efficient, but when this is not possible, running on a node on the same rack (rack locality) is faster than running off-rack. Delay scheduling is a solution that temporarily relaxes fairness to improve locality by asking jobs to wait for a scheduling opportunity on a node with local data. When a node requests a task, if the head-of-line job cannot launch a local task, it is skipped and looked at subsequent jobs. However, if a job has been skipped long enough, non-local tasks are allowed to launch to avoid starvation. The key insight behind delay scheduling is that although the first slot we consider giving to a job is unlikely to have data for it, tasks finish so quickly that some slot with data for it will free up in the next few seconds. The techniques used are:

- Naive Fair Sharing Algorithm.
- Delay Scheduling in HFS.

2.3 SCARLETT

Scarlett, a system that replicates blocks based on their popularity. By accurately predicting the popularity and working within hard bounds on additional storage, Scarlett causes minimal interference to running jobs. Scarlett improves data locality by 45%, which results in a 20.2% reduction of the job completion times of Hadoop jobs. In addition, by using extensive simulations, Scarlett reduces the number of evictions in the Dryad cluster by 83% and speeds up the jobs by 12.8%.Finally, Scarlett incurs low overhead, as it is able to achieve near-ideal performance by altering replication factors, using less than 10% extra storage space. Scarlett captures the popularity of files and uses that to increase the replication factor of non-accessed files, while avoiding hotspots in the cluster and causing minimal interference to the cross-rack network traffic. To do so, Scarlett computes a replication factor for each file that is proportional to its popularity while remaining within a budget on extra storage due to additional replicas. Scarlett smooth's out placement of replicas across machines in the cluster so that the expected load on each machine (and rack) is uniform. Finally, Scarlett uses compression to reduce the cost of creating replicas [6].The disadvantage is that it mainly focuses on copying the popularity content and fails to schedule them [7].

The techniques used are:

- Computing File Replication Factor
- Smooth Placement of Replicas

2.4 MOON

Specifically, the data and task replication scheme adopted by existing Map Reduce implementations is woefully inadequate for resources with high unavailability. MOON extends Hadoop, an open-source implementation of Map Reduce, with adaptive task and data scheduling algorithms in order to offer reliable Map Reduce services on a hybrid resource architecture, where volunteer computing systems are supplemented by a small set of dedicated nodes. The adaptive task and data scheduling algorithms in MOON distinguish between (1) different types of Map Reduce data and (2) different types of node outages in order to strategically place tasks and data on both volatile and dedicated nodes. When a large number of volatile nodes are supplemented with a much smaller number of dedicated nodes, providing scalable data access is challenging. As such, MOON prioritizes the I/O requests on the different resources. Specifically, for files with replicas on both volatile and dedicated Data Nodes, read requests from clients on volatile Data Nodes will always try to fetch data from volatile replicas first. By doing so, the read request from clients on the volatile Data Nodes will only reach dedicated Data Nodes when none of the volatile replicas are available. The disadvantage is that it doesn't support multiple scheduling when some tasks fail. The technique used is I/O throttling on dedicated Data Nodes.

2.5 SPECULATIVE EXECUTION IN GOOGLE

In this method, Google used Map Reduce as a programming model and an associated implementation for processing and generating large data sets. The Map invocations are distributed across multiple machines by automatically partitioning the input data into a set of M splits. The input splits can be processed in parallel by different machines. Reduce invocations are distributed by partitioning the intermediate key space into R pieces using a partitioning function (e.g., hash(key) mod R). The number of partitions (R) and the partitioning function are specified by the user.

Speculative execution was first introduced in Google in which the last few running map and reduce tasks are simply backed up which showed a performance improvement of around 44% in job response time [8].

2.6 SPECULATIVE EXECUTION IN HADOOP

When a node has an empty task slot, Hadoop chooses a task for it from one of three categories. First, any failed tasks are given highest priority. This is done to detect when a task fails repeatedly due to a bug and stop the job.

Second, non-running tasks are considered. For maps, tasks with data local to the node are chosen first. Finally, Hadoop looks for a task to execute speculatively. To select speculative tasks, Hadoop monitors task progress using a progress score 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:

- The copy phase, when the task fetches map outputs.
- The sort phase, when map outputs are sorted by key.
- The reduce phase, when a user-defined function is applied to the list of map outputs with each key.

Hadoop looks at the average progress score of each category of tasks (maps and reduces) to define a threshold for speculative execution. When a task's 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. All tasks beyond the threshold are considered "equally slow," and ties between them are broken by data locality. The scheduler also ensures that at most one speculative copy of each task is running at a time. Finally, when running multiple jobs, Hadoop uses a FIFO discipline where the earliest submitted job is asked for a task to run, then the second, etc. There is also a priority system for putting jobs into higher-priority queues [9].

2.6.1 ASSUMPTIONS IN HADOOP'S SCHEDULER

Hadoop's scheduler makes several implicit assumptions:

- Nodes can perform work at roughly the same rate.
- Tasks progress at a constant rate throughout time.
- There is no cost to launching a speculative task on a node that would otherwise have an idle slot.
- A task's progress score is representative of fraction of its total work that it has done. Specifically, in a reduce task, the copy, sort and reduce phases each take about 1/3 of the total time.
- Tasks in the same category (map or reduce) require roughly the same amount of work.

2.6.2 HETEROGENEITY

The first two assumptions are about homogeneity. Hadoop assumes that any detectably slow node is faulty. However, nodes can be slow for other reasons. Heterogeneity seriously impacts Hadoop's scheduler. Because the scheduler uses a fixed threshold for selecting tasks to speculate, too many speculative tasks may be launched taking away resources from useful tasks (assumption 3 is also untrue). Also, because the scheduler ranks candidates by locality, the wrong tasks may be chosen for speculation first. For example, if the average progress was 70% and there was a 2x slower task at 35% progress and a 10x slower task at 7% progress, then the 2x slower task might be speculated before the 10x slower task if its input data was available on an idle node.

2.6.3 OTHER ASSUMPTIONS

Assumptions 3, 4 and 5 are broken on both homogeneous and heterogeneous clusters, and can lead to a variety of failure modes.

Assumption 3, that speculating tasks on idle nodes costs nothing, breaks down when resources are shared. For example, the network is a bottleneck shared resource in large Map Reduce jobs. Also, speculative tasks may compete for disk I/O in I/O-bound jobs. Finally, when multiple jobs are submitted, needless speculation reduces throughput without improving response time by occupying nodes that could be running the next job.

Assumption 4, that a task's progress score is approximately equal to its percent completion, can cause incorrect speculation of reducers. In a typical Map Reduce job, the copy phase of reduce tasks is the slowest, because it involves all-pairs communication over the network. Tasks quickly complete the other two phases once they have all map outputs. However, the copy phase counts for only 1/3 of the progress score. Thus, soon after the first few reducers in a job finish the copy phase, their progress goes from 1/3 to 1, greatly increasing the average progress. As soon as about 30% of reducers finish, the average progress is roughly $0.3 \cdot 1 + 0.7 \cdot 1/3 = 53\%$, and now all reducers still in the copy phase will be 20% behind the average, and an arbitrary set will be speculatively executed. Task slots will fill up, and true stragglers may never be speculated executed, while the network will be overloaded with unnecessary copying.

Assumption 5, that progress score is a good proxy for progress rate because tasks begin at roughly the same time, can also be wrong. The number of reducers in a Hadoop job is typically chosen small enough so that they they can all start running right away, to copy data while maps run. However, there are potentially tens of mappers per node, one for each data chunk. The mappers tend to run in waves. Even in a homogeneous environment, these waves get more spread out over time due to variance adding up, so in a long enough job, tasks from different generations will be running concurrently. In this case, Hadoop will speculatively execute new, fast tasks instead of old, slow tasks that have more total progress.

Finally, the 20% progress difference threshold used by Hadoop's scheduler means that tasks with more than 80% progress can *never* be speculatively executed, because average progress can never exceed 100%.

2.6.4 ISSUES IN SPECULATIVE EXECTION

- Too many backups, thrashing shared resources like network bandwidth
- Wrong tasks backed up
- Backups may be placed on slow nodes

2.7 LONGEST APPROXIMATE TIME TO END (LATE) SCHEDULER

In this method proposed by Zaharia, a new speculative task scheduler is proposed which starts from principles and adding features needed to behave well in a real environment. The task that will finish farthest into the future will be selected for speculative execution, because this task provides the greatest opportunity for a speculative copy to overtake the original and reduce the job's response time. A simple heuristic is used to estimate time left which finds the progress rate of each task as Progress Score / T, where T is the amount of time the task has been running for, and then estimate the time to completion as (1 – Progress Score) / Progress Rate. This assumes that tasks make progress at a roughly constant rate. There are cases where this heuristic can fail but it is effective in typical Hadoop jobs [10].

Speculative tasks are launched only on fast nodes - not stragglers. This is achieved by not launchinga speculative task on nodes that are below some threshold, slow node threshold, of total work performed (sum of progress scores for all succeeded and in-progress tasks on the node). This heuristic leads to better performance than assigning a speculative task to the first available node. Another option would be to allow more than one speculative copy of each task, but this wastes resources needlessly.

Finally, to handle the fact that speculative tasks cost resources, the following two heuristics are augmented.

- Speculative cap which indicates the number of speculative tasks that can be running at once
- A Slow task Threshold that a task's progress rate is compared with to determine whether it is "slow enough" to be speculated upon. This prevents needless speculation when only fast tasks are running.

LATE algorithm works as follows:

- If a node asks for a new task and there are fewer than Speculative Cap speculative tasks running:
  - Ignore the request if the node's total progress is below Slow Node Threshold.
  - Rank currently running tasks that are not currently being speculated by estimated time left.
  - Launch a copy of the highest-ranked task with progress rate below Slow Task Threshold.

Like Hadoop's scheduler, a task has to run atleast for 1 minute before evaluating it for speculation. In practice, a good choice for the three parameters to LATE is

- Speculative Cap - 10% of available task slots and
- Slow Node Threshold - 25th percentile of node progress
- Slow Task Threshold - 25th percentile task progress rates

LATE does not take into account data locality for launching speculative map tasks, although this is a potential extension.

The LATE algorithm has several advantages:

- LATE takes into account node heterogeneity when deciding where to run speculative tasks.
- It focuses on estimated time left rather than progress rate. LATE speculatively executes only tasks that will improve job response time, rather than any slow tasks.
- It is robust to node heterogeneity.

LATE scheduler has following demerits:

- As the end time for a task is calculated using the averaged out progress rate against the current progress rate, the end time predicted is likely to be incorrect.
- Initial evaluation time required by the LATE scheduler is high (1 minute) before a task can be marked a straggler. This essentially leads to longer response times.

3. SYSTEM REQUIREMENTS

3.1 HARDWARE REQUIREMENTS

Processor	:	Intel Pentium CPU B950 2.10 GHZ
RAM	:	51MB
Hard Disk Drive	:	80 GB
Keyboard	:	101 Keys
Mouse	:	Optical Mouse
Monitor	:	SVGA/color

3.2 SOFTWARE REQUIREMENTS

Operating System	:	Windows7
IDE used	:	Net Beans
Database used	:	MySql
Platform	:	Hadoop 0.18.0
Language Used	:	Java

3.2.1 JAVA

Java is an object-oriented language similar to C++, but simplified to eliminate language features that cause common programming errors. Java source code files (files with a .java extension) are compiled into a format called byte code (files with a .class extension), which can then be executed by a Java interpreter. Compiled Java code can run on most computers because Java interpreters and runtime environments, known as Java Virtual Machines (VMs), exist for most operating systems, including UNIX, the Macintosh OS, and Windows. Byte code can also be converted directly into machine language instructions by a just-in-time compiler (JIT).

3.2.2 JAVA PLATFORM

One characteristic of Java is portability, which means that computer programs written in the Java language must run similarly on any hardware/operating-system platform. This is achieved by compiling the Java language code to an intermediate representation called Java byte code, instead of directly to platform-specific machine code. Java byte code instructions are analogous to machine code, but are intended to be interpreted by a virtual machine (VM) written specifically for the host hardware. End-users commonly use a Java Runtime Environment (JRE) installed on their own machine for standalone Java applications, or in a Web browser for Java applets. Standardized libraries provide a generic way to access host-specific features such as graphics, threading, and networking.

3.2.3 NET BEANS

The Net Beans Platform is a reusable framework for simplifying the development of Java Swing desktop applications. The Net Beans IDE bundle for Java SE contains what is needed to start developing Net Beans plug-in and Net Beans Platform based applications; no additional SDK is required.

The features of the platform are:

- User interface management (e.g. menus and toolbars)
- User settings management
- Storage management (saving and loading any kind of data)
- Window management
- Wizard framework (supports step-by-step dialogs)
- Net Beans Visual Library
- Integrated Development Tools

3.2.4 J2EE

A Java EE application or a Java Platform, Enterprise Edition application is any deployable unit of Java EE functionality. This can be a single Java EE module or a group of modules packaged into an EAR file along with a Java EE application deployment descriptor. Java EE applications are typically engineered to be distributed across multiple computing tiers.

3.2.5 WAMP SERVER

WAMPs are packages of independently-created programs installed on computers that use a Microsoft Windows operating system. WAMP is an acronym formed from the initials of the operating system Microsoft Windows and the principal components of the package: Apache, MySQL and one of PHP, Perl or Python.

3.2.6 MySQL

The MySQL development project has made its source code available under the terms of the GNU General Public License, as well as under a variety of proprietary agreements. MySQL was owned and sponsored by a single for-profit firm, the Swedish company MySQL AB, now owned by Oracle Corporation.

3.2.7 HADOOP:

Apache Hadoop is an open-source software framework that supports data-intensive distributed applications licensed under the Apache v2 license. It supports parallel running of applications on large clusters of commodity hardware [2]. The Hadoop framework transparently provides both reliability and data motion to applications. Hadoop implements a computational paradigm named Map Reduce, where the application is divided into many small fragments of work, each of which can execute or re-execute on any node in the cluster. In addition, it provides a distributed file system that stores data on the compute nodes, providing very high aggregate bandwidth across the cluster. Both map/reduce and the distributed file system are designed so that node failures are automatically handled by the framework. It enables applications to work with thousands of computation-independent computers and petabytes of data. The entire Apache Hadoop "platform" is now commonly considered to consist of the Hadoop kernel, Map Reduce and Hadoop Distributed File System (HDFS).

For effective scheduling of work, every Hadoop-compatible file system should provide location awareness: the name of the rack (network switch) where a worker node is. Hadoop applications can use this information to run work on the node where the data is, and, failing that, on the same rack/switch, reducing backbone traffic. HDFS uses this method when replicating data to try to keep different copies of the data on different racks. The goal is to reduce the impact of a rack power outage or switch failure, so that even if these events occur, the data may still be readable. A small Hadoop cluster includes a single master and multiple worker nodes. The master node consists of a Job Tracker, Task Tracker, Name Node and Data Node. A slave or worker node acts as both a Data Node and Task Tracker, though it is possible to have data-only worker nodes and compute-only worker nodes.

In a larger cluster, the HDFS is managed through a dedicated Name Node server to host the file system index, and a secondary Name Node that can generate snapshots of the name node's memory structures, thus preventing file-system corruption and reducing loss of data. Similarly, a standalone Job Tracker server can manage job scheduling. In clusters where the Hadoop Map Reduce engine is deployed against an alternate file system, the Name Node, secondary Name Node and Data Node architecture of HDFS is replaced by the file-system-specific equivalent.

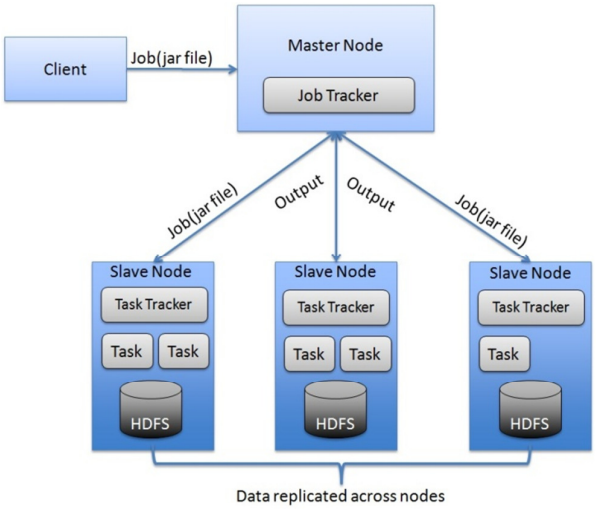


Figure 3.1 Hadoop architecture

4. SYSTEM DESIGN

A new speculative execution strategy named MCP for maximum cost performance is used. The cost is considered to be the computing resources occupied by tasks, while the performance is considered to be the shortening of job execution time and the increase of the cluster throughput. MCP aims at selecting straggler tasks accurately and promptly and backing them up on proper worker nodes. To ensure fairness, we assign task slots in the order the jobs are submitted. Just like other speculative execution strategies, MCP gives new tasks a higher priority than backup tasks. In other words, MCP will not start backing up straggler map/reduce tasks until all new map/reduce tasks of this job have been assigned. MCP chooses backup candidates based on a prompt prediction of the tasks' process speed and an accurate estimation of their remaining time. Then, these backup candidates will be selectively backed up on proper worker nodes to achieve max cost performance according to the cluster load. The modules used are:

- Exponentially Weighted Moving Average (EWMA)
- Maximizing Cost Performance
- Schedule Job
- Map reduce

4.1 EWMA

The task's process speed in the near future is predicted instead of simply using the past average rate. There are many prediction algorithms in the literature, such as EWMA (Exponentially Weighted Moving Average). EWMA scheme which can be expressed as follows:

$$Z(t) = \alpha * Y(t) + (1-\alpha) * Z(t-1);\ 0 < \alpha < 1$$

Where Z(t) and Y(t) are the estimated and the observed process speed at time t, respectively and reflects a tradeoff between stability and responsiveness. The value of α is set to be 0.5 according to the evaluation result. To assure the accuracy of prediction, tasks process speed is not calculated until it has executed for a certain amount of time (speculative lag).

4.2 MAXIMIZING COST PERFORMANCE

Speculative execution has not only benefits, but also costs. In a Hadoop cluster, the cost of speculative execution is task slots, while the benefit is the shortening of the job execution time. A cost-benefit model is established to analyze the tradeoff. In this model, the cost is represented as the time that the computing resources are occupied while the benefit is represented as the time saved by speculative execution.

$$\begin{aligned} rem\_time &= rem\_time_{cp} + rem\_time_{fp} \\ &= \frac{rem\_data_{cp}}{bandwidth_{cp}} + \sum_{p\ in\ fp} est\_time_p * factor_d, \end{aligned}$$

$$factor_d = \frac{data_{input}}{data_{avg}}.$$

4.3 SCHEDULE JOBS

In order to achieve better performance, backup tasks should be assigned to fast worker nodes. This requires an appropriate metric to measure the performance of worker nodes which varies a lot from time to time. To tackle this problem, the moving average process bandwidth of data-local map tasks completed is used on a worker node to represent the node's performance. The data-locality of map tasks is considered when making the backup decisions. The process speed of data-local map tasks can be three times that of non-local map tasks. As a result, if data-locality is not considered, backing up a map task may gain no benefit.

4.4 MAP REDUCE

Map Reduce cluster, after a job is submitted, a master divides the input files into multiple map tasks, and then schedules both the map tasks and the reduce tasks to worker nodes. A worker node runs tasks on its task slots and keeps updating the tasks progress to the master by periodic heartbeat. Map tasks extract key-value pairs from the input, transfer them to some user defined map function and combine function, and finally generate the intermediate map outputs. After that, the reduce tasks copy their input pieces from each map task, merge these pieces to a single ordered (key, value list) pair stream by a merge sort, transfer the stream to some user defined reduce function, and finally generate the result for the job. A map task is divided into map and combine phases, while a reduce task is divided into copy, sort and reduce phases. Reduce tasks can start when only some map tasks complete, which allows reduce tasks to copy map outputs earlier as they become available and hence mitigates network congestion. However, no reduce task can step into the sort phase until all map tasks complete. This is because each reduce task must finish copying outputs from all the map tasks to prepare the input for the sort phase.

5. PERFORMANCE ANALYSIS

5.1 EXISTING SYSTEM

It cannot appropriately handle the situation when there exists data skew among the tasks. Map Reduce cluster, after a job is submitted, a master divides the input files into multiple map tasks, and then schedules both the map tasks and the reduce tasks to worker nodes. Map tasks extract key-value pairs from the input, transfer them to some user defined map function and combine function, and finally generate the intermediate map outputs. A map task is divided into map and combine phases, while a reduce task is divided into copy, sort and reduce phases. No reduce task can step into the sort phase until all map tasks complete. The map or the reduce tasks in Sort jobs, MCP performs much better compared to Hadoop-LATE. The job execution speed and the cluster throughput are improved by 37% and 44% when map skew exists, and by 17% and 19% when reduce skew.

5.2 PROPOSED SYSTEM

Average progress rate is used to identify slow tasks while in reality the progress rate can be unstable and misleading. In a typical Map Reduce job, the master divides the input files into multiple map tasks, and then schedules both map tasks and reduce tasks to worker nodes in a cluster to achieve parallel processing. The main difference between LATE and Mantriis that Mantri uses the task's process bandwidth to calculate the task's remaining time. EWMA is used to predict the process speed of tasks in order to find slow tasks or slow nodes in time. The scenarios which affect the performance of those strategies: data skew, tasks that start asynchronously, improper configuration of phase percentage and abrupt resource competitions.

The Word Count benchmark is run first. Figure 5.1shows the performance comparison of the three strategies. On average, MCP finishes jobs 10 percent faster than Hadoop-LATE and 10 percent faster than Hadoop-None. Moreover, MCP improves the throughput of the cluster by5 percent compared with Hadoop-LATE and 6 percent compared with Hadoop-None. It shows that MCP identifies straggler tasks more accurately and promptly than Hadoop-LATE. In particular, MCP can improve the precision in identifying stragglers in reduce tasks by over 90 percent compared to Hadoop-LATE.

First, an environment that exhibits data skew among map tasks is set up. According to the split strategy in Hadoop, those input files will be divided into two parts, which results in data skew among map tasks. Figure5.2 shows that MCP performs much better than Hadoop-LATE and Hadoop-None. On average MCP increases the job execution speed by 37 percent over Hadoop-LATE and 58 percent over Hadoop-None. Mean while, it improves the throughput of the cluster by 44 percent over Hadoop-LATE and 57 percent over Hadoop-None.MCP can achieve a much bigger improvement than Hadoop-LATE because Hadoop-LATE may conduct many unnecessary backups for the map tasks which occupies the precious slots for other jobs. As a result, the average delay of all jobs in Hadoop-LATE is much longer than that in MCP.

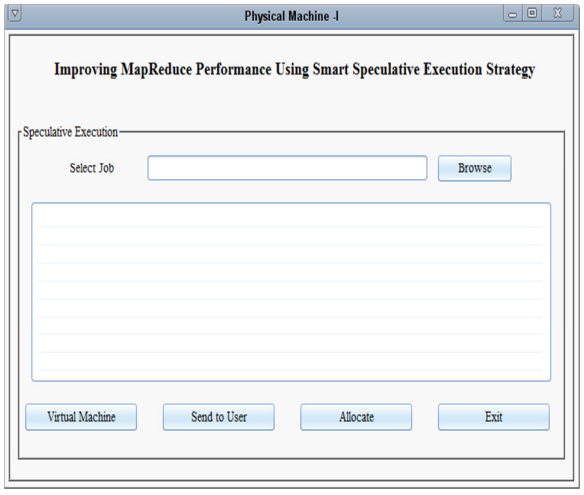
Reduce skew is likely to happen when the distribution of keys in the input data set is skewed and the map output is partitioned by some hash function. This kind of skew is also known as partition skew. Figure5.3 shows that ona verage MCP increases the job execution speed by 17 percent over Hadoop-LATE and by 53 percent over Hadoop-None. Meanwhile, it improves the throughput of the cluster by 19 percent over Hadoop-LATE and by 53 percent over Hadoop-None. MCP achieves less improvement over Hadoop-LATE for reduce skew than for map skew because unnecessary reduce backups do not affect the execution of map tasks from other jobs. It only delays the reduce tasks of other jobs. Therefore, a small delay in launching reduce tasks will not affect the performance of other jobs significantly when those other jobs are still in the map stage.

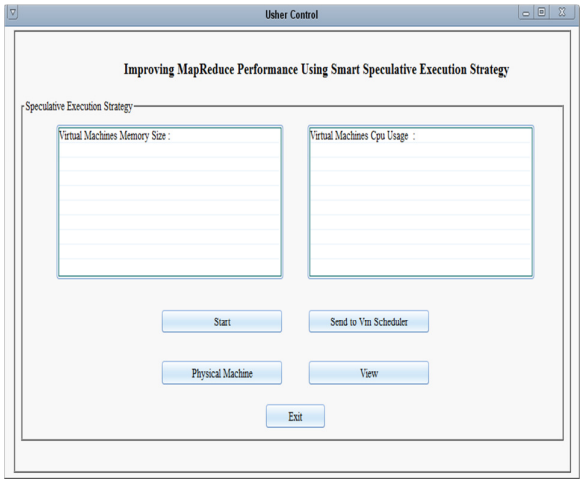
6. CONCLUSION AND FUTURE ENHANCEMENT

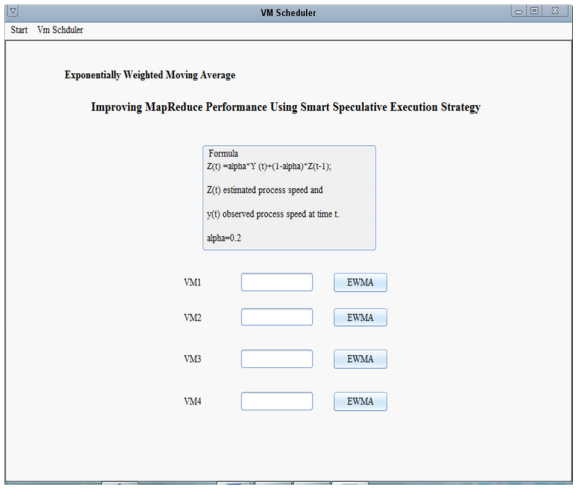
The pitfalls of current speculative execution strategies with respect to data skew, tasks starting asynchronously and abrupt resource competitions in Map Reduce are analyzed. Based on the analysis, new speculative execution strategy called MCP is developed to handle these pitfalls by taking into consideration the cost performance of cluster computing resources. MCP decreases the job execution time and improves the throughput. MCP achieved up to 39 percent improvements over Hadoop-LATE. It fits well in both heterogeneous and homogeneous environments, handles the data skew case and is quite scalable which performs very well in both small clusters and large clusters. Time complexity of MCP is found to be O(n).

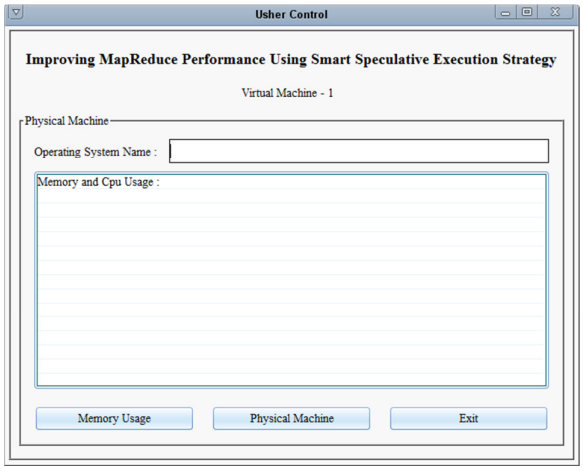
The number of nodes in the distributed environment can be increased to suit the real time requirements. Multiple data sources can be added to improve the accuracy of the output. The resource allocation capabilities of the MCP can be optimized further so that all the tasks are assigned to proper worker nodes and to minimize the execution time of the task.

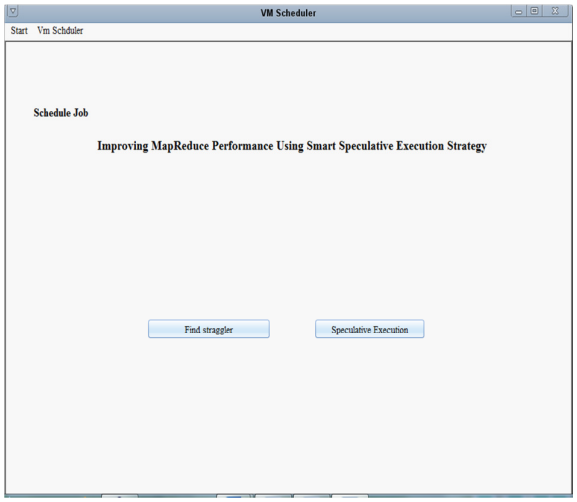
7. SCREEN SHOTS

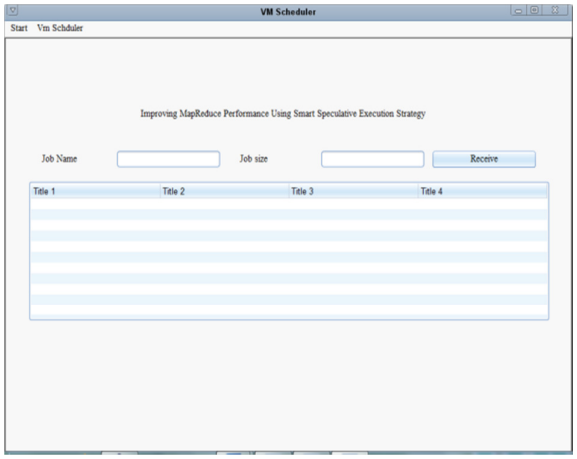












8. REFERENCES

[1] J. Dean and S. Ghemawat, "Map reduce: simplified data processing on large clusters," Commun. ACM, vol. 51, pp. 107–113, January 2008.

[2] "Apache hadoop, http://hadoop.apache.org/."

[3] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in Proc. of the 2nd ACM SIGOPS/Euro Sys European Conference on Computer Systems 2007, ser. Euro Sys '07, 2007.

[4] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving map reduce performance in heterogeneous environments," in Proc. of the 8th USENIX conference on Operating systems design and implementation, ser. OSDI'08, 2008.

[5] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using mantri," in Proc. of the 9th USENIX conference

on Operating systems design and implementation, ser. OSDI'10, 2010.

[6] Y. Kwon, M. Balazinska, and B. Howe, "A study of skew in map reduce applications," in The 5th Open Cirrus Summit, 2011.

[7] P.H and Ellaway, "Cumulative sum technique and its application to the analysis of peri stimulus time histograms," Electroencephalography and Clinical Neurophysiology, vol. 45, no. 2, pp. 302–304, 1978.

[8] K. Avi, K. Yaniv, L. Dor, L. Uri, and L. Anthony, "Kvm: The linux virtual machine monitor," Proc. of the Linux Symposium, Ottawa, Ontario, 2007, 2007.

[9] Quiane-Ruiz,Pinkel, C.,Schad, J. ,Dittrich, J."RAFTing Map Reduce: Fast recovery on the RAFT" Data Engineering (ICDE), 2011 IEEE 27th International Conference in Hannover, Publication Year: 2011.

[10] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I.Stoica, D. Harlan, and E. Harris, "Scarlett: Coping with Skewed Content Popularity in Map reduce Clusters," Proc. Sixth Conf. Computer Systems (EuroSys '11), 2011.

[11] B. Nicolae, D. Moise, G. Antoniu, L. Bouge, and M. Dorier,"Blobseer: Bringing High Throughput under Heavy Concurrency to Hadoop Map-Reduce Applications," Proc. IEEE Int'l Symp. Parallel Distributed Processing (IPDPS), Apr. 2010.

[12] J. Leverich and C. Kozyrakis, "On the Energy (In)Efficiency of Hadoop Clusters," ACM SIGOPS Operating Systems Rev., vol. 44,pp. 61-65, Mar. 2010.

[13] T. Sandholm and K. Lai, "Mapreduce Optimization Using Regulated Dynamic Prioritization," Proc. 11th Int'l Joint Conf. Measurement and Modeling of Computer Systems, (SIGMETRICS '09),2009.

[14] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A.Goldberg, "Quincy: Fair Scheduling for Distributed Computing Clusters," Proc. ACM SIGOPS 22nd Symp. Operating Systems Principles(SOSP '09), 2009.

[15] M. Zaharia, D. Borthakur, J. SenSarma, K. Elmeleegy, S. Shenker,and I. Stoica, "Delay Scheduling: A Simple Technique for AchievingLocality and Fairness in Cluster Scheduling," Proc. Fifth European Conference Computer Systems (EuroSys '10), 2010.
Kala Karun, A ; Chitharanjan, K ; "A review on hadoop — HDFS infrastructure extensions ", IEEE Conference on Information & Communication Technologies (ICT), JeJu Island, April 2013. Page(s): 132 - 137.

[16] D.Deepika1, K.Pugazhmathi, "Efficient Indexing and Searching of Big Data in HDFs", International Journal of Computer Sciences and Engineering (IJCSE) Vol.-4(4), Apr 2016, E-ISSN: 2347-2693.

[17] Tanuja A, Swetha Ramana D, "Processing and Analyzing Big data using Hadoop", International Journal of Computer Sciences and Engineering (IJCSE) Vol.-4(4), PP(91-94) April 2016, E-ISSN: 2347-2693.

AUTHORS PROFILE

Mr. A. Saran kumar received his B.Tech. degree in Information Technology from Coimbatore Institute of Technology (Autonomous), Coimbatore, Tamil Nadu. And he is currently pursuing M.E. Degree in Computer Science and Engineering in Kumaraguru College of Technology (Autonomous), Coimbatore, Tamil Nadu, India. His areas of interest are Data mining, Big Data and Web Technology.

Ms. V. Vanitha Devi received her B.E. degree in Computer Science and Engineering from Sri Shakthi Institute of Engineering and Technology, Coimbatore, Tamil Nadu. And she is currently pursuing M.E. Degree in Computer Science and Engineering in Kumaraguru College of Technology (Autonomous), Coimbatore, Tamil Nadu, India. Her areas of interest are Cloud Computing, Big Data and Data Structures.

