Volume: 5 Issue: VI Month of publication: June 2017

DOI:

www.ijraset.com
IC Value: 45.98

Volume 5 Issue VI, June 2017
ISSN: 2321-9653

International Journal for Research in Applied Science & Engineering Technology (IJRASET)

# Checkpointing Methods for Fault Tolerance on MapReduce: A Review

Shani Sunny[1], Jemily Elsa Rajan[2]
[1]PG Scholar, Computer Science and Engineering, Kerala Technological University
[2]Asst.Prof, Computer Science and Engineering, Kerala Technological University

***Abstract*:  *Big Data is a term for information sets that are huge and complicated in which the traditional data processing applications are not capable to manage with them. MapReduce is a programming model for processing and producing these huge data sets. Programs written in this are naturally parallelized and runs on a huge cluster of commodity machines. Mapreduce executes fault tolerance at the task level, so when a task failure happens the whole task will be re-executed once more. In the case of long time running subtasks and then a single task failure is very large and cannot be avoided. The effect of failures can be considerable in terms of performance. As a result fault tolerance is significant for MapReduce. This Paper summarizes the checkpointing methods for fault tolerance on MapReduce.***
***Keywords:  BigData, MapReduce, Hadoop, Fault Tolerance, Checkpointing***

## I. INTRODUCTION

Big data [12] is an evolving term that describes any voluminous amount of structured, semistrutured and unstructured data that is so large and difficult to process using traditional database and software techniques. Big data is characterized by the extreme volume of data, the wide variety of data types and the velocity at which the data must be processed. Although big data doesn't equate to any specific volume of data, often used to describe terabytes, petabytes and even hexabytes of data captured over time. In most enterprise scenarios the volume of data is too big or it moves too fast or it exceeds current processing capacity. Data is processed with advanced tools to reveal meaningful information.

MapReduce [1] is a programming model and an associated implementation for processing and generating large data sets. 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. MapReduce runs on a large cluster of commodity machines and is highly scalable. The computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user of the MapReduce library expresses the computation as two functions: Map and Reduce. Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key and passes them to the Reduce function. The Reduce function, also written by the user, accepts an intermediate key and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per Reduce invocation. The intermediate values are supplied to the user's reduce function via an iterator. This allows to handle lists of values that are too large to fit in memory.

There are three fundamental types of faults [13] in parallel systems, such as byzantine faults, fail-stop faults and fail-shutter faults. MapReduce can allow the latter two kinds of faults. Byzantine fault model can characterized as a random system failures and malicious attacks by a hacker. Fail-stop faults are generally characterized as master failures, worker failures and task failures. MapReduce acquired a centralized architecture, with one node called the master and others are called workers. The master is used for arranging tasks and workers are used to run map and reduce tasks. A MapReduce cluster contains only one master, which makes it easy to set up a standby master to permit master failures. MapReduce reschedules failed tasks from the beginning without bothering the others for task and worker failures. Fail-stutter faults will appear as slow running tasks. MapReduce dynamically detects slow tasks and provide speculative attempts and executing  the same input data.

## II. CHECKPOINTING METHODS

### *A. BFT MapReduce*

J. Dean et.al [2] proposed a byzantine fault-tolerant (BFT) MapReduce runtime system. It tolerates the faults that corrupt the results of computation of tasks such as DRAM and CPU errors/faults. It uses a MapReduce algorithm and prototype that tolerate these faults.  The fault tolerance methods of current MapReduce implementations, namely Hadoop cannot deal with accidental Byzantine faults. They cannot be identified using checksums and often do not crash the task they affect, so they can silently corrupt the result of a task . This system is composed by a set of distributed processes. The clients that request the execution of jobs composed by map and reduce tasks, the JobTracker that manages the execution of a job, a set of Task-Trackers that execute tasks, the NameNode that manages access to data stored in HDFS, and a set of DataNodes where HDFS stores file blocks. If a process is correct if it follows the algorithm, otherwise it is faulty.

### *B. Stream processing*

J.-H.Hwang et.al [3] proposed a method called stream processing in which a query is represented in the form of a directed acyclic graph of operators that describes how to transform data. Some stream operators are directly borrowed from relational algebra and others are adapted to operate over continuous data streams. In a distributed stream processing system, developing more servers increases the system performance. However, it increases the risk of failure. This paper proposed a checkpoint-based collaborative, self-configuring high availability (HA) solution which addresses the needs of distributed stream processing through a parallel fine-grained backup and yields shorter recovery times. Stream-oriented HA solutions differ in how the backup servers operate. The main idea is to divide the query which is located at a given server into units. Each unit can be checkpointed independently, thereby decreases the overall recovery time.

### *C. Self-Adaptive MapReduce (SAMR)*

Q. Chen et.al [4] proposed a technique that evaluates the improvement of tasks dynamically and modifies it to the continuously varying environment automatically. When a new job is committed, SAMR splits the job into several fine-grained map and reduce tasks and put them to a series of nodes. It reads historical information that is saved on every node and updated after every execution. SAMR adjusts the time weight of each stage of map and reduce tasks according to the historical information respectively. Thus, it gets the progress of each task accurately and detects which tasks need backup tasks. SAMR categorize slow nodes into map slow nodes and reduce slow nodes further. It gets the final results of the fine-grained tasks when either slow tasks or backup tasks finish first. SAMR significantly improves MapReduce in terms of saving the execution time as well as system resources.

### *D. Longest Approximate Time to End*

M. Zaharia et.al [5] proposed a simple, robust algorithm called Longest Approximate Time to End (LATE) to speculatively execute tasks that break the response time. LATE performs better than Hadoop's default speculation algorithm in real workloads on Amazon's Elastic Compute Cloud (EC2). The MapReduce model by Google is attractive for parallel processing arbitrary data. MapReduce will breaks the whole task into small sub tasks that runs in parallel and can moves to large clusters of commodity clusters. Hadoop's scheduler starts executing speculative tasks based on each tasks progress to the average progress. The main goal of speculative execution is to minimize the job's response time. Response time is very important for job's such that when a user wants an answer such as queries on data for monitoring and debugging. Hadoop's scheduler will cause degradation in performance in heterogenous environments.

### *E. RAFT*

J.-A. Quiane-Ruiz *et.al* [6] proposed a family of Recovery Algorithms for Fast-Tracking (RAFT) MapReduce. Fault-tolerance is an important aspect in large clusters because the probability of node failures increases with the growing number of computing nodes. MapReduce has several performance issues in the presence of task and worker failures. A query metadata checkpointing algorithm (RAFT-QMC) that deals with several worker failures. To achieve this, mappers produce query metadata checkpoints of task progress calculation which contains all offsets of input key-value pairs that produce an intermediate result, and the identifier of the reducers that will consume such results. As a result, re-scheduled mappers only recomputed intermediate results required by local reducers. This paper also proposed a scheduling strategy that delegates the responsibility to workers for rescheduling tasks failed due to task failures, and preassigns reducers to workers in order to allow mappers to push intermediate data to reducers.

### *F. Adaptive Optimization*

R.Vernica et.al [7] proposed a number of adaptive optimization techniques for the MapReduce framework that improve its performance and especially performance stability. That is, these adaptive techniques can bring significant performance improvements and also it make use of "Situation-Aware Mappers" that are able to cooperatively make global optimization decision.

It describes a set of techniques for making the MapReduce framework more adaptive to the input data and runtime conditions. These techniques affect different parts of a MapReduce job and all of them are implemented inside the map tasks. The techniques mentioned in this paper are described as follows: (a) Adaptive Mappers dynamically "stitches" a set of input splits into a single virtual split assigned to a mapper, thus changing the checkpoint interval as the mapper is running. (b) Adaptive Combiners improves he hash-based aggregation for combining map outputs with frequent keys, (c) Adaptive Sampling and Partitioning piggybacks on the main job, dynamically decides when to stop sampling, based on a global sampling condition and dynamically decides the partitioning function while the job is running.

### G. *Passive Replication*

Q. Zheng [8] proposed a passive replication on top of re-execution to improve the MapReduce fault tolerance in the cloud. Allocates a few backups besides the running copy for each map task. Backup is only executed when its corresponding task fails. The value of these backups is decided by the Mapreduce users or by cloud providers based on failure information. Thus the failed task is run by one of its backups and this fail over can reduce completion time. Proposed methods to schedule backups, move backup instances, and select backups upon failure for fast recovery. It reduces the latency for individual high-priority user jobs.

### H. *Chained Declustering*

C. Yang et.al [9] proposed MapReduce Fault tolerance in a shared nothing database. Osprey, a middleware implementation of MapReduce fault tolerance for a SQL database. Osprey gives fault tolerance and load balancing based on some key ideas: (a) dividing the queries into subqueries that can be executed in parallel, (b) Dynamic scheduling of the execution of those subqueries on the workers, (c) Re-execution of straggler subqueries. These strategies are adapted from MapReduce, but the Osprey will differ from it because it uses SQL based database systems. Here the data will appear in the form of tables and then the data is partitioned across database nodes by using a technique called chained declustering. A coordinator machine divides the Query into several subqueries. After finishing the current one, new subquery are assigned to the worker nodes. It shows that the system shows good load balancing and fault tolerance properties.

### I. *MRPerf*

G. Wang et.al [10] proposed an efficient Hadoop simulator called MRPerf, which focuses on accurate execution of detailed MapReduce setups, and validates it using measurements on actual Hadoop setups and used to study the impacts of different network topologies on the performance of Hadoop application.adopt a simulation method to understand the overall performance of MapReduce setups. The input configuration in MRPref is determined in a set of files and processed by different processing modules. There are two assumptions in MRPref, a node's resources and it does not model the OS-level asynchronous prefetching. Concerned with the design of cluster, parameters in run time, multi-tenancy and application performance. The goal of MRPerf is to provide fine-grained simulation of MapReduce setups at sub-phase level.

### J. *CPR Technique*

G. Bronevetsky et.al [11] proposed a regularly utilized approach called checkpoint and restart (CPR). Here the state of the calculation will be saved occasionally on the disk. The point when a disappointment occurs, those calculation is restarted from the most recent saved state. It is the obligation of the programmer to instrument flying provisions for CPR. This System need two components: (i) a pre-compiler for Source-to-source change about applications, and (ii) a runtime framework that executes a protocol for facilitating CPR around the threads of the parallel requisition. One of the benefits about this methodology is the capacity to tolerate faults inside the application. In this way applications get to be self-checkpointing and Self-restarting on any platform.

## III. CONCLUSIONS

MapReduce is a programming model and an associated implementation for processing and generating large data sets . There are chances for fault in execution of tasks. The checkpoint strategy is the effective method to overcome this challenge. This paper summarizes various checkpointing strategies for fault tolerance on MapReduce. By using this Checkpoint Strategy, a retrying task doesnot start from the scratch so that it saves time.

## IV. ACKNOWLEDGMENT

I am extremely grateful to my PG Coordinator Asst. Prof. Salitha M K, Assistant Professor of Computer Science and Engineering Department and my guide Asst. Prof. Jemily Elsa Rajan, Assistant Professor of Computer Science and Engineering Department, Believers Church Carmel Engineering College, Perunad for her valuable guidance, timely suggestions and for providing all the vital facilities like providing the Internet facilities and important books, which were essential in the completion of this work. Finally, I would like to thank every individual who gave me even the slightest support to make my paper a success.

## REFERENCES

[1] J. Dean and S. Ghemawat, "Map-Reduce: Simplified data processing on large clusters 0018-9162/95," in Proc. IEEE 6th Conf. Symp. Operating Syst. Des. Implementation, 2004, p. 10.
[2] P. Costa, M. Pasin, A. N. Bessani, and M. Correia,"Byzantine fault tolerant MapReduce: Faults are not just crashes," in Proc. IEEE 3rd Int. Conf. Cloud Comput. Technol. Sci., 2011, pp. 32-39.
[3] J.-H. Hwang, Y. Xing, U. Cetintemel, and S. Zdonik, "A cooperative, self-configuring high-availability solution for stream processing," in Proc. IEEE 23rd Int. Conf. Data Eng., 2007, pp. 176–185.
[4] Q. Chen, D. Zhang, M. Guo, Q. Deng, and S. Guo, "SAMR: A Selfadaptive MapReduce scheduling algorithm in heterogeneous environment," in Proc. IEEE 10th Int. Conf. Comput. Inf. Technol., 2010, pp. 2736–2743.
[5] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in Proc. Symp. Operating Syst. Des. Implementation, 2008, vol. 8, p. 7.
[6] J.-A. Quiane-Ruiz, C. Pinkel, J. Schad, and J. Dittrich, "RAFTing MapReduce: Fast recovery on the raft," in Proc. IEEE 27th Int. Conf. Data Eng., 2011, pp. 589–600.
[7] R. Vernica, A. Balmin, K. S. Beyer, and V. Ercegovac, "Adaptive MapReduce using situation-aware mappers," in Proc. Proc. 15th Int. Conf. Extending Database Technol., 2012, pp. 420–431.
[8] Q. Zheng, "Improving MapReduce fault tolerance in the cloud," in Proc. IEEE Int. Symp. Parallel Distrib. Process. Workshops Phd Forum, 2010, pp. 1–6.
[9] C. Yang, C. Yen, C. Tan, and S. R. Madden, "Osprey: Implementing MapReduce-style fault tolerance in a shared-nothing distributed database," in Proc. IEEE 26th Int. Conf. Data Eng., 2010, pp. 657-668.
[10] G. Wang, A. R. Butt, P. Pandey, and K. Gupta, "A simulation approach to evaluating design decisions in mapreduce setups," in Proc. IEEE Int. Symp. Model., Anal. Simul. Comput. Telecommun. Syst., 2009, pp. 1–11.
[11] G. Bronevetsky, D. Marques, K. Pingali, P. Szwed, and M. Schulz,"Application-level checkpointing for shared memory programs,"ACM SIGOPS Operating Syst. Rev., vol. 38, no. 5, pp. 235–247, 2004.
[12] https://en.wikipedia.org/wiki/Big_data.
[13] Hao Wang, Haopeng Chen, Zhenwei Du, and Fei Hu, "BeTL: MapReduce Checkpoint Tactics Beneath the Task Level" IEEE Transactions on Services Computing., 2016., vol. 9, no. 1, p.1.