



Efficient Failure Management Approach using Trust Point based Recovery Technique

¹Sabitha M. S, ²Dr. S. Vijayalakshmi, ³Rathikaa Sre R. M

¹Research Scholar, Research & Development Centre, Bharathiar University, Coimbatore, Tamilnadu, India

²Assistant Professor, Thiagarajar College of Engineering, Madurai, Tamilnadu, India

³Student, Mepco Schlenk Engineering College, Sivakasi, Tamilnadu, India

Abstract – Big data is used for very huge and complex datasets. Big data is now penetrating into various fields like medical and engineering domains due to its large data management and variety of data set. Big data mining facilitates in filtering useful insights from the huge datasets. Big data provides better results with the integration of cloud computing and the data mining system. Big data computation is required to process large amount of data. Most of the current Big Data computing approaches are using Map Reduce tool. The main drawback in the current approach is handling the task failures. This paper proposes an efficient system to handle the failures by recovering the process from the point where it has stopped. The proposed approach has proved that the job execution time has been reduced compare to the normal Hadoop system.

Index Terms—Big Data, Map Reduce, Failure management, Trustpoint, Recovery

I. INTRODUCTION

Now a day's voluminous data are being generated in day to day activities of humans and machines. Lots of data are being collected from Web, E-commerce, purchases at stores, sensor automation, healthcare environment, bank transactions and mainly the Social Network. Big data is a collection of huge, structured and unstructured data streams. Biggest challenge is handling of Big Data using traditional database management system and data mining tools. Efficient Big data mining cannot be done due of its volume, variability and velocity in a traditional method. But the need of intelligent insights from the large data sets is an important requirement. Scalable data mining tools required to handle the challenges in storage, management, retrieval and processing of huge volume of data in cloud platform.

MapReduce [7], [8], [9], [10], is a Big Data programming model used for writing applications to process very huge amount of data in parallel on various clusters of commodity hardware. It is a parallel programming model for large clusters. Map and Reduce are the two major functions available in MapReduce. MapReduce initially split the input jobs into various independent jobs which are processed by the map tasks in a parallel manner. MapReduce will sort the output of the map and this will become the input for the reduce tasks. A MapReduce file system helps to keep track of both input and output jobs.

The framework takes care of smooth execution of the task by scheduling, monitoring and re executing the failed tasks. Achieving very high bandwidth across the cluster is due to its configuration and allows the framework to schedule the task effectively where data exists. Each cluster node in Mapreduce consists of JobTracker and TaskTracker. JobTrackers acts as master and TaskTracker acts as slave. The scheduling, monitoring and re-execution of the failed tasks taken care by the master and the slave executes the tasks as per the direction of the master. Map is a function that directs the work to different nodes in the distributed cluster. Reduce is another function that collates work and resolves the results into a single value. Most of the execution time used for initialization, scheduling, monitoring and re-execution of failed tasks coordination. It does not support any data streaming or execution of Map and Reduce phase at a time. Overall execution time contributed to input output operations and shuffling among phases.

Basically failure management technique is available in Hadoop to some extent. HDFS contains a storage layer which holds a copy of every data block. In case of failure of any node, it uses the storage system to get back the data. If any failure happens in the job level, it simply re-executes all the Map and Reduce tasks. It increases the computation time and cost especially when we have more number of tasks and the failure happens at the end of the process. The complete set of tasks will be executed if it fails at the end.

This paper proposes a technique to avoid the re-execution of the complete set of tasks. A trustpoint based job recovery mechanism using backtalk values is suggested to re- execute the tasks which are not executed so far.

Section II of this paper discusses the views of various researches. Various steps of the proposed system discussed in Section III. The results of the performance analysed in the Section IV. Finally the research work concluded in Section V.

II. RELATED WORK

This section discusses the views of various researchers and works related to this topic.

Checkpointing is a popular fault tolerance method used in many applications. It keeps a track of the running position with its checkpoint. At the time of failure in the system, the state can be rolled back to its failure position instead of restarting the complete set of operations.

System level checkpointing – The processes holds the independent checkpoints. It doesn't consider the other processes.

Complete / Incremental Checkpoints – Incremental checkpointing doesn't store the unmodified portion of the checkpoint image. It uses specialized data structure to hold the informations.

Checkpoint mechanisms are the traditional transparent method. Various non transparent methods

Fagg and Dongarra proposed a process to handle the failures in MPI-2 API system [20]. They suggested an improvised model of FT-MPI. If it receives an error while communicating to the failure process, this system handles the failure process. Before it proceeds the jobs, the process repair the communicator.

A parallel programming model - Global Arrays [21] provided an array data structure to keep the track of data distributed across various machines. Regular put,get and accumulate operations used for accessing the data.

In [22], Ali et al., introduced a redundant data concept to reduce the overhead from failures. It keeps track of two copies of the array structure in different nodes. The idea behind this is if one process fails, it takes the copy from other nodes and proceeds further.

Jeffrey Dean and Sanjay Ghemawat of Google Inc [17] discussed about the MapReduce programming model and shared their experiences in the implementation process for large data sets. Basically this paper discussed about the partitioning methods, scheduling, machine failure handling and managing inter machine communication. This is very much useful for the programmers to utilize the resources in a large scale. The implementation challenges and how the MapReduce programs jobs are executed in the Google's clusters.

Fabrizio Marozzo, Domenico Talia and Paolo Trunfio in the paper [18] "A Framework for Managing MapReduce Applications in Dynamic Distributed Environments" discussed about the architecture of MapReduce system. It proposed a P2P model for effective management of master failures and job recovery. An effective internet based dynamic distributed environment explained.

"Themis: An I/O-Efficient MapReduce" [19] proposed by Alexander Rasmussen, Michael Conley and Rishi Kapoor. Key proposal in this paper is reducing the number of I/O operations to improve the performance. A new design Themis proposed a click log analysis , DNA read sequence alignment and Page Ranking.

III. PROPOSED SYSTEM

This section discusses about the trustpoint based job recovery mechanism. Various stages in the proposed system discussed in this section. First one is the detecting the failures based on backtalk value and the job recovery through recovery process.

Fig.1 shows the general MapReduce architecture.

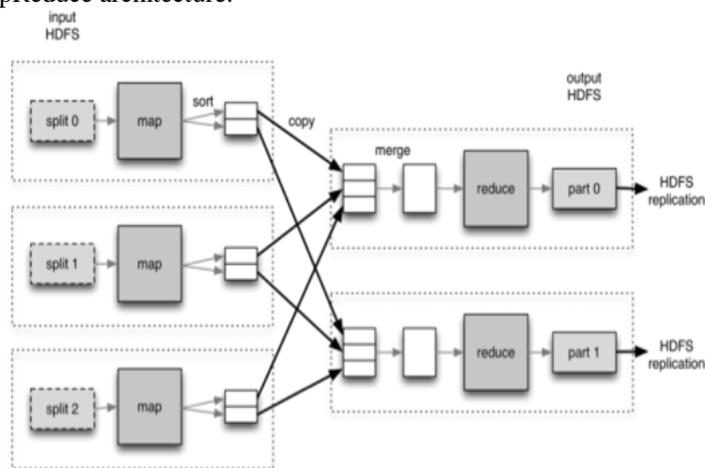


Fig 1. MapReduce Framework

The job handling in MapReduce constitute of three stages. In the first stage, it saves the immediate results in the local storage. Then in the shuffle stage, the results are transferred to the Reduce task and finally the Reduce task transfer the results to the HDFS. In case of any failure in the last phase (ie Reduce phase). The system has to re-execute the complete set of tasks. It completely depends on the timeout The timeout parameter is considered as one of the important factor. If the failure detected at the later stage, it has to re-execute the complete set of tasks. To overcome this problem, Backtalk and Revival system recommended below.

A. Backtalk system

This system is proposed to track the failures in the submitted tasks.

Let $T = \{t_1, \dots, t_n\}$, $D = \{d_1, \dots, d_m\}$ and $B = \{b_1, \dots, b_m\}$

Where T – set of all tasks

D – Set of all nodes/processors

B - set of all backtalk values associated for every node
 n - total number of tasks
 m – total number of nodes/processors
 $0 \leq b_j \leq 1, j = 1, 2, \dots, m$

A backtalk value b_j is assigned for every processor to specify the reliability of the nodes. The initial backtalk value is 1. The backtalk value reflects the error while fetching the tasks. The backtalk value will be reset to $b_j - \rho$ when the Reduce task throws an error while fetching data from the Map task d_j . The penalty value indicated as ρ . If $b_j < \xi$, then j would be detected as failure. Backtalk threshold is indicated as ξ . This backtalk system described in the below algorithm Fig 2.

```

1: for every  $d_j$  in D do
2:   set  $b_j = 1$ 
3:   for every  $t_i$  in T do
4:     if  $b_j < \xi$  then
5:       return  $j$  is failure
6:     end if
7:     if  $t_i$  gets a fetch error from  $d_j$  then
8:        $b_j = b_j - \rho$ 
9:     end if
10:  end for
11:  return no failure
12: end for
    
```

Fig 2. Algorithm for the proposed Backtalk system

B. Setting trustpoints

The backtalk value concept introduced in the above section to handle the failures while submitting the tasks. There are various types of recovery techniques are available for database systems. This system uses the trustpoint method. The adoption of trustpoint strategy is not only to go back to the state before failure occurs. Also it avoids the complete re-execution of all tasks. This trustpoint related information will be maintained for further recoveries.

In Hadoop, the data is distributed and copied in different nodes. Let $S = \{s_1, \dots, s_r\}$ are list of stored copies of data blocks and s_k is executed by d_j . While processing, the data s_k stored on the node j . The nodes communicate by sending and receiving messages. The messages contains $\{TI_j, TS_j, SS_j\}$ where TI_j – Trustpoint identity, TS_j – Current status of Trustpoint and SS_j – State of the stored copies.

If the process d_j has any trustpoint, the TI_j value will be incremented by 1 else it assigns the value 0. TS_j indicated the current status as No Trustpoint (NTP) or Trustpoint (TP). SS_j is pair of s_k (list of replica) and TS_k . TI_j is the common sequence counter for all replicas of the data block.

In MapReduce, master node takes care of Schedules, monitor and failed task. Master node has a meta data information that can store the place where it stopped and the stored copy. The messages passed to other stored copies and sets the trust point.

For example, if D_j has the trustpoint id TI_j , the message with the format $(TI, TP)_j$.

```

1: While True do
2:   if  $d_j$  sets a checkpoint then
3:     master receives  $(TI_j, TP_j)$ 
4:      $TI_j ++$ 
5:     for the stored copy  $s_k$  on  $d_j$  do
6:        $d_k \in w_k$  gets message as  $(TI_j, \dots, [(s_1, TS_1), (s_2, TS_2), \dots])$ 
7:       if  $TI_k \geq TI_j$  then
8:          $TI_k ++$ 
9:         send the message  $(TI_k, TP_k)$  to master
10:      else
11:         $TI_k = TI_j$ 
12:      end if
13:    update local  $SS_k$ 
14:  end of
15:  update local  $SS_j$ 
16:  update master's meta data
17: end if
18: End while
    
```

Fig 3. Algorithm for setting trustpoint

C. Recovery procedure

This section describes how it recovers the data during failure of the nodes. Initially it gets the saved copies from the master. The recovery procedure completely explained in the Fig 4.

```

1: get saved copies of s1,...,sk-1 from master
2: temp is a a node temp is randomly assigned as a
   permanent storage by master
3:   for i in 1,2,...,k-1 do
4:     if  $TS_j = TP$  then
5:       send the message  $(\Pi_k, TP, SS_j)$  to master
6:     end if
7:   end for
8: reassign data on temp

```

Fig 4. Algorithm for recovery procedure

The above proposed system handles the failures and it avoids the re-execution of complete tasks having the stored copies of the jobs. Since it handle the failures the performance of the system increased a a lot.

IV. PERFORMANCE ANALYSIS

The performance analysis of the proposed Backtalk system explained in this section. Hadoop is an open source implementation of the Mapreduce programming strcture. The failure management system is available in the Hadoop system. In HDFS, the copy of every data block is stored in the storage layer. In case of any failure happens, it re-executes the complete set of Map and Reduce jobs. Which leads to lower performance time and higher cost. This paper proposed a simple technique to handle the failure tasks. It avoids the execution of complete task and it restarts from the place where the task is stopped.

For analysis purpose, two different types of jobs considered and their performance evaluated. The first job is the product wise sales for the current year which has 12 Map Tasks and 16 Reduce asks. The sample job is to find the regional wise produce wise sales for the past 5 years which contains 83 Map tasks and 182 Reduce tasks. The execution time of the two jobs considered for performance analysis. The performance of the system improved for bigger tasks and especially wherever the failure happens. The performance of the sample tasks executed with the Hadoop system and with the proposed system. The performance of the proposed system reduced in time which is indicated in the below chart.

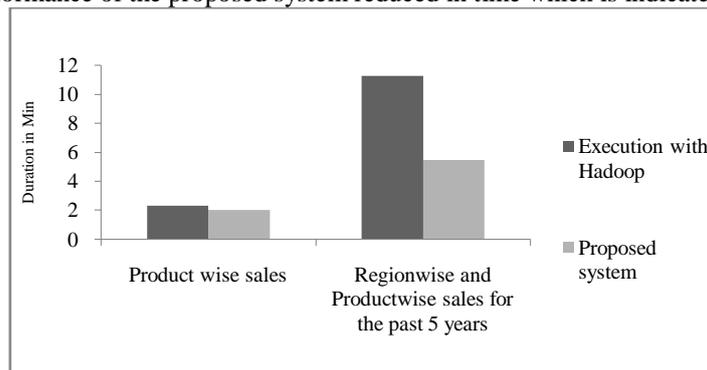


Fig 5. Performance analysis

V. CONCLUSION

Even though the inbuilt failure management technique is available in the Hadoop system, this paper proposes a back talk based job recovery. This avoids the re-execution of all jobs in the system in case of any failure happens. The initial task is to set the backtalk for every processor which is used to specify the reliability of the nodes. It uses trust point method for recovery of tasks. It keeps the replicas of all task as a stored copies of data blocks. It passes the information of current status of trust point and the unique id. This is used to restart the task from the place where it stopped. Two different types of tasks taken for testing purpose. One has more number of Map and Reduce tasks and another one has less number of Map and Reduce tasks. The performance increased for the tasks which has more number of Map and Reduce tasks.

Limitation in this system is considering the complete replica of the tasks. Memory management to be addressed especially for huge jobs. Further research can be extended in this direction.

REFERENCES

- [1] R. Hu, W. Dou, and J. Liu, "ClubCF: A Clustering-based Collaborative Filtering Approach for Big Data Application," 2014.
- [2] T. H. Dao, S. R. Jeong, and H. Ahn, "A novel recommendation model of location-based advertising: Context-Aware Collaborative Filtering using GA approach," *Expert Systems with Applications*, vol. 39, pp. 3731-3739, 2012.
- [3] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE Transactions on Knowledge and Data Engineering*, , vol. 26, pp. 97-107, 2014.
- [4] C. K.-S. Leung and Y. Hayduk, "Mining frequent patterns from uncertain data with MapReduce for Big Data analytics," in *Database Systems for Advanced Applications*, 2013, pp. 440-455.

- [5] B. Lu and S. Wei, "One More Efficient Parallel Initialization Algorithm of K-Means with MapReduce," in *Proceedings of the 4th International Conference on Computer Engineering and Networks*, 2015, pp. 845-852.
- [6] X. Cai, F. Nie, and H. Huang, "Multi-view k-means clustering on big data," in *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, 2013, pp. 2598-2604.
- [7] K. Shim, "MapReduce algorithms for big data analysis," *Proceedings of the VLDB Endowment*, vol. 5, pp. 2016-2017, 2012.
- [8] X. Cui, P. Zhu, X. Yang, K. Li, and C. Ji, "Optimized big data K-means clustering using MapReduce," *The Journal of Supercomputing*, vol. 70, pp. 1249-1259, 2014.
- [9] A. Pal and S. Agrawal, "An experimental approach towards big data for analyzing memory utilization on a hadoop cluster using HDFS and MapReduce," in *First International Conference on Networks & Soft Computing (ICNSC), 2014*, 2014, pp. 442-447.
- [10] I. Triguero, D. Peralta, J. Bacardit, S. García, and F. Herrera, "MRPR: A MapReduce solution for prototype reduction in big data classification," *Neurocomputing*, vol. 150, pp. 331-345, 2015.
- [11] V. López, S. del Río, J. M. Benítez, and F. Herrera, "Cost-sensitive linguistic fuzzy rule based classification systems under the MapReduce framework for imbalanced big data," *Fuzzy Sets and Systems*, vol. 258, pp. 5-38, 2015.
- [12] S. del Río, V. López, J. M. Benítez, and F. Herrera, "On the use of MapReduce for imbalanced big data using Random Forest," *Information Sciences*, vol. 285, pp. 112-137, 2014.
- [13] J. Evermann and G. Assadipour, "Big data meets process mining: Implementing the alpha algorithm with map-reduce," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, 2014, pp. 1414-1416.
- [14] H. Chai, G. Wu, and Y. Zhao, "A document-based data warehousing approach for large scale data mining," in *Pervasive Computing and the Networked World*, ed: Springer, 2013, pp. 69-81.
- [15] L. Ismail, M. M. Masud, and L. Khan, "FSBD: A Framework for Scheduling of Big Data Mining in Cloud Computing," in *IEEE International Congress on Big Data (BigData Congress), 2014*, pp. 514-521.
- [16] A. B. Patel, M. Birla, and U. Nair, "Addressing big data problem using Hadoop and Map Reduce," in *Nirma University International Conference on Engineering (NUICONE), 2012*, pp. 1-5.
- [17] Jeffrey Dean and Sanjay Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, *OSDI 2004*
- [18] Fabrizio Marozzo, Domenico Talia and Paolo Trunfio, *A Framework for Managing MapReduce Applications in Dynamic Distributed Environments*,
- [19] Alexander Rasmussen, Michael Conley and Rishi Kapoor, Themis: An I/O-Efficient MapReduce, *SOCC'12, October 14-17, 2012, San Jose, CA USA*
- [20] G. E. Fagg and J. Dongarra. FT-MPI: Fault tolerant MPI, supporting dynamic applications in a dynamic world. In *Proceedings of the 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, pages 346-353, London, UK, 2000. Springer-Verlag*
- [21] J. Nieplocha, B. Palmer, V. Tipparaju, M. Krishnan, H. Trease, and E. Apr'a. Advances, applications and performance of the global arrays shared memory programming toolkit. *Int. J. High Perform. Comput. Appl.*, 20:203-231, May 2006
- [22] N. Ali, S. Krishnamoorthy, N. Govind, and B. Palmer. A redundant communication approach to scalable fault tolerance in PGAS programming models. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, pages 24-31, February 2011. [4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215(3):403-410, October 1990. [5] S. F. Altschul, T. L. Madden, A. A. Schaffer, J