International Journal of Computer Sciences
August'2016 Issue
ISSN Number: 4142 3453

Slot Configuration Technique for Handling Large Data sets in Hadoop Environment

A. Reddy Rekha[1] , Dr. A. David William[2]

1M.Tech (CSE),PG Scholar ,Department of CSE, Bharath College of Engineering and Technology for Women,Kadapa

2 Associate Professor, Department of CSE, Bharath College of Engineering and Technology for Women,Kadapa

Abstract— ***The Hadoop tool is the real platform for optimal analysis for huge amount of data. How to reduce the completion length of a group of MapReduce jobs is one of the main things in Hadoop. The map and reduce are available in Hadoop tool implemented for processing and supply distributed large Terabyte data on capacity clusters. The present open source Hadoop allows only fixed slot configuration, like fastened of map slots and reduce slots entire the cluster lifetime. Such static configuration may lead to more completion time as well reduce the cpu utilizations. Its primary duty is to minimize the completion time of large sets of MapReduce jobs. It can be done by following slot ratio configuration between map and reduce tasks, by updating the workload information of recently completed tasks. Many scheduling methodologies are discussed that aim to improve completion time goal. Propose new schemes which use slot ratio between map and reduce tasks as a tunable knob for minimizing the completion length (i.e., makespan) of a given set. By leveraging the workload information of recently completed jobs, schemes dynamically allocates resources (or slots) to map and reduce tasks.***

Keywords: MapReduce; Makespan; WorkLoad; Dynamic Slot Configuration

I. *INTRODUCTION*

A classic Hadoop cluster has a single name node and multiple data nodes. The name node, which is configured with job tracker, is responsible for job scheduling and job execution co-ordination. Each data node configured with task tracker, which manages MapReduce slots. Hadoop has a static slot configuration, which means a fixed number of map slots and reduce slots which are only used for processing map reduce tasks. Map tasks can run by map slots, and reduce tasks can run in reduce slots. This static slot configuration may lead to poor performance and low resource utilization. Apache Hadoop components are responsible for running large data sets. Main Hadoop parallel processing components are Hadoop Distributed File System (HDFS), Hadoop YARN, and Hadoop MapReduce.

We propose dynamic slot configuration, which dynamically allocates slots for map and reduce tasks. Our aim is to modify name node functionality, which means to increase additional responsibility for monitoring workload information, dynamic slot assignment, and scheduling. Also, we need to modify the task tracker slot allocation policy to dynamically allocate tasks to MapReduce tasks without any slot specification. We can make use of map task slots (map slots) to reduce slots and vice versa. The main idea behind dynamic slot configuration is to avoid idle slot in the MapReduce slots. The job tracker estimates the current workloads in each task tracker using workload monitoring component.

A Hadoop cluster has single master node and multiple slave nodes. The master node runs the JobTracker routine which is responsible for scheduling jobs and coordinating the execution of tasks of each job. Each slave node runs the TaskTracker for hosting the execution of MapReduce jobs. The concept of "slot" is used to indicate the capacity of accommodating tasks on each node. In a Hadoop system, a slot is assigned as a map slot or a reduce slot serving map tasks or reduce tasks, respectively. At any given time, only one task can be running per slot. The number of available slots per node indeed provides the maximum degree of parallelization in Hadoop. Here shown that the slot configuration has a significant impact on system performance. The Hadoop framework use fixed numbers of map slots and reduce slots at each node as the default setting throughout the lifetime of a cluster. The values in this fixed configuration are usually heuristic numbers without considering job characteristics. Therefore, this static setting is not well optimized and may hinder the performance improvement of the entire cluster. In this work, propose and implement a new mechanism to dynamically allocate slots for map and reduce tasks. The primary goal of the new mechanism is to improve the completion times (i.e., the makespan) of a batch of MapReduce jobs while retain the simplicity in implementation and management of the slot- based Hadoop design.

The key idea of this new mechanism, named TuMM, is to automate the slot assignment ratio between map and reduce tasks in a cluster as a tunable knob for reducing the makespan of MapReduce jobs. The Workload

Monitor (WM) and the Slot Assigner (SA) are the two major components introduced by TuMM. The WM that resides in the JobTracker periodically collects the execution time information of recently finished tasks and estimates the present map and reduce workloads in the cluster. The SA module takes the estimation to decide and adjust the slot ratio between map and reduce tasks for each slave node. With TuMM, the map and reduce phases of jobs could be better pipelined under priority based schedulers, and thus the makespan is reduced. Further the dynamic slot assignments in heterogeneous environments, and propose a new version of TuMM, named H TuMM, which sets the slot configurations for each individual node to reduce the makespan of a batch of jobs. The slot assigner component decides the optimum slot for assigning tasks. The schedulers are used to schedule the tasks in the data nodes. The task tracker sends the status report to the job tracker for every 3 minutes. Failure tasks are assigned to the next nodes based on this status report. The job tracker is always monitoring the task execution and slot assignment.

II. MAP REDUCE THEORY

MapReduce: MapReduce is a framework for processing parallelizable problems across huge datasets using a large number of computers, collectively referred to as a cluster (if all nodes are on the same local network and use similar hardware) or a grid (if the nodes are shared across geographically and administratively distributed systems, and use more heterogeneous hardware). Computational processing can occur on data stored either in a file system (unstructured) or in a database (structured). MapReduce takes advantage of the locality of data, data processing on or near the storage assets in order to decrease the data transmission. Figure 2 describes Hadoop MapReduce process, which involves input data, split phase, Map phase, Intermediate data, Reduce phase and Output data. HDFS: Hadoop uses Hadoop distributed File System (HDFS) which is an open source implementation of the Google File System (GFS) for storing data. HDFS is a distributed file system that not only stores the data but also ensures fault tolerance through replication designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets.

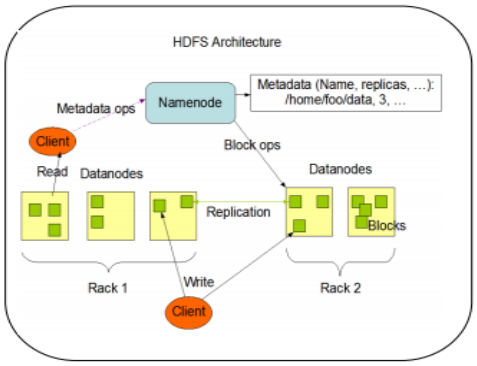

Fig 1.HDFS Architecture

Figure 1 describe HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

III. SYSTEM MODEL AND DYNAMIC SLOT CONFIGURATION UNDER HETEROGENEOUS ENVIRONMENTS

Heterogeneous environments are fairly common in today's cluster systems. For example, system managers of a private data center could always scale up their data center by adding new physical machines. Therefore, physical machines with different models and different resource capacities can exist simultaneously in cloud Servers.

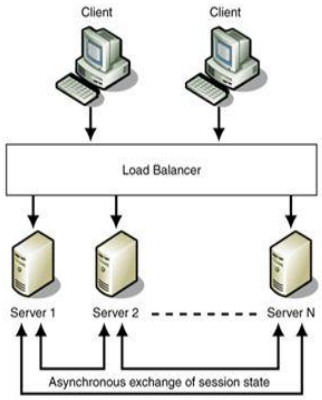

Fig:2 Heterogeneous cluster

IV. ALGORITHM DESIGN

H TuMM shares the similar idea of TuMM, i.e., dynamically assign slots to map and reduce tasks to align the process of map and reduce phase based on the collected workload information. The key difference of H TuMM is to set the slot configurations for each node individually in a heterogeneous cluster, i.e., each of those nodes will have different slot assignment ratio between map and reduce tasks. To accomplish it, H TuMM collects the workload information on the entire cluster and on each individual node as well: when a map/reduce task is finished on node i, the workload collector updates.

1. The average execution time of map/reduce tasks, i.e., t_m/t_r;
2. The average execution of map/reduce tasks that ran on node i, i.e., t^i

0: **Input:** Average task execution time on node i and across the cluster, and the remaining task number of current running jobs;
0: **When** Node i has free slots and ask for new task assignment through the heartbeat message;
1: $s_m^i \leftarrow \lfloor S^i * \frac{\overline{t_m^i * n'_m}}{\overline{t_m^i * n'_m} + \overline{t_r^i * n'_r}} \rfloor$;
2: $s_r^i \leftarrow \lfloor S^i * \frac{\overline{t_r^i * n'_r}}{\overline{t_m^i * n'_m} + \overline{t_r^i * n'_r}} \rfloor$;
3: **if** $s_m^i + s_r^i \leq S^i$ **then**
4: **if** $\frac{t_m^i}{t_m} > \frac{t_r^i}{t_r}$ **then**
5: $s_r^i \leftarrow S^i - s_m^i$;
6: **else**
7: $s_m^i \leftarrow S^i - s_r^i$.
8: **if** $(s_m^i - rt_m^i) > (s_r^i - rt_r^i)$ **then**
9: assign a map task to node i;
10: **else**
11: assign a reduce task to node i;

V. RELATED WORK

In literature, there was research study on performance optimization of Hadoop MapReduce jobs. An essential way for upgrading the performance of a MapReduce job is dynamic slot configuration and job scheduling. J. Polo et al. calculated the map and reduce task completion time dynamically and update it every minute during job execution. Task scheduling policy was based on the priority of each job. Priority was estimated based on the concurrent allocation of jobs. The dynamic scheduler is pre-emptive. It affects resource allocation of low priority jobs. J. Wolf et al. Apache Hadoop released next generation MapReduce, called YARN. It replaces MRv1 fixed slot configuration. YARN deals with CPU cores and memory requirements. It splits the job tracker into two components; they are resource managements and job scheduling. MapReduce tasks assignment is based on CPU cores and memory requirement of each task. YARN users simply update their MRv1 by installing mrv2 compatibility API and recompile the MRv1 application. J. Wang et al. proposed fair slot setting for dynamically allocate available slots to particular tasks. They used FRESH for static and dynamic slot configuration. The static slot configuration slots are allocated before cluster launch based on previous task execution records. It uses deduct workload function to update current workloads of running jobs in the cluster. The fair scheduler was proposed to achieve fairness metric. The dynamic slot assignment slots are allocated during task execution. It used Johnson indices to represent the level of fairness.

VI. RESULTS

In this section we have shown the working of the proposed system. The data shows the total required time for the completion time and makespan task for proposed system in contrast with the existing system which maximum cost conference.

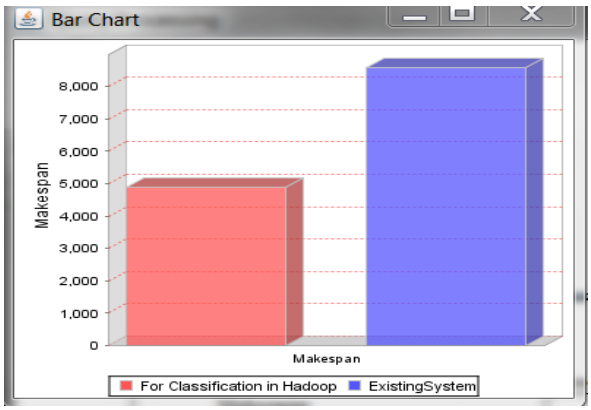

Fig 3: Classification existing system with proposed system using Homogeneous cluster

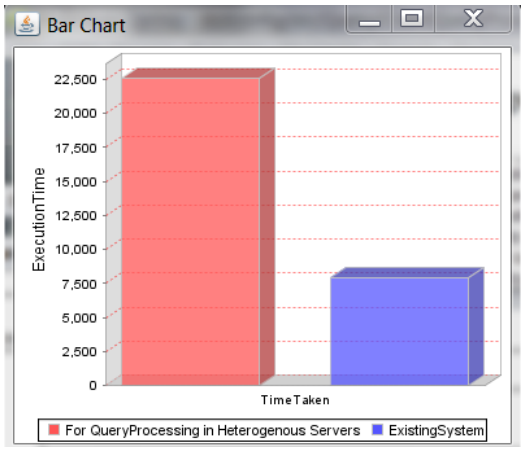

Fig 4: Classification using proposed system with existing system in heterogeneous cluster

VII. CONCLUSION

Dynamic slot configuration is one of the important factors while processing a large data set with MapReduce paradigm. It optimizes the performance of MapReduce framework. Each job can be scheduled using any one of the scheduling policies by the job tracker. The task managers which are present in the task tracker allocate slots to jobs. From the examined paper, it is concluded to prefer a dynamic slot allocation strategy that includes active jobs workload estimation, optimal slot assignment, and scheduling policy.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters", in Communications of the ACM, vol. 51, 2008.

[2] J. Polo, D. Carrera, Y. Becerra et al., "Performance-driven task co-scheduling for MapReduce environments", in NOMS'10, 2010.

[3] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: Automatic Resource inference and allocation for MapReduce environments", in International Conference on Autonomic Computing, 2011.

[4] A. Bansal, A. Deshpande, P. Ghare, S. Dhikale, B. Bodkhe, " Healthcare Data Analysis using Dynamic Slot Allocation in Hadoop", International Journal of Recent Technology and Engineering Vol. 3 Issue 5, November 2014.

[5] Apache Hadoop YARN (yet another resource negotiator) Reference

[6] Link: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop- yarn-site/YARN.html.

[7] Z. Li, Q. Zhang, M. F. Zhani, R. Boutaba, Y. Liu, Z. Gong et al., "DREAMS: Dynamic Resource Allocation for MapReduce with Data Skew", Integrated Network Management (IM), DOI 10.1109/INM.2015.7140272,2015 IFIP/IEEE International Symposium on May 2015.