ISSN: 2321-7782 (Online)
Volume 3, Issue 8, August 2015
International Journal of Advance Research in Computer Science and Management Studies
Research Article / Survey Paper / Case Study
Available online at: www.ijarcsms.com

HADOOP Multiple Job Trackers with Master Slave Replication Model

Dr. Shubhra Saggar[1]
Associate Professor
GNIM, India

Dr. Nidhi Khurana[2]
Associate Professor
GNIM, India

Dr. Ravish Saggar[3]
Professor
BCIIT, India

Abstract: ***The Hadoop framework has been designed, in an effort to enhance performances, with a single jobTracker (master node). It's responsibilities varies from managing job submission process, compute the input splits, schedule the tasks to the slave nodes (Task Trackers) and monitor their health .In some environments, like the IBM and Google's Internet-scale computing initiative, there is the need for high-availability, and performances becomes a secondary issue. In this environment, having a system with a Single Point of Failure (such as Hadoop's single Job Tracker) is a major concern. This paper describes the concept of Hodoop and provides a redundant version of Hadoop by adding support for multiple replicated Job Trackers.***

Keywords: Hadoop, JobTracker, TaskTracker, NameNode, DataNode, MapReduce, HDFS, JAR.

I. WHAT IS HADOOP

Apache Hadoop is an open source framework for developing distributed applications that can process very large amounts of data. It is a platform that provides both distributed storage and computational capabilities.

Hadoop has two main layers:

» Computation layer: The computation tier uses a framework called MapReduce.

» Distributed storage layer: A distributed filesystem called HDFS provides storage.

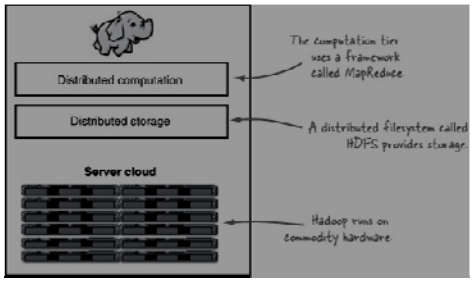

Figure: 1 Hadoop

1.1 HADOOP Distributed File System

The Hadoop distributed file system (HDFS) is a distributed, scalable, and portable file-system written in Java for the Hadoop framework. Each node in a Hadoop instance typically has a single namenode; a cluster of datanodes form the HDFS cluster. The situation is typical because each node does not require a datanode to be present. Each datanode serves up blocks of data over the network using a block protocol specific to HDFS. The file system uses the TCP/IP layer for communication. Clients use Remote procedure call (RPC) to communicate between each other.

HDFS stores large files (typically in the range of gigabytes to terabytes) across multiple machines. It achieves reliability by replicating the data across multiple hosts, and hence does not require RAID storage on hosts. With the default replication value, 3, data is stored on three nodes: two on the same rack, and one on a different rack. Data nodes can talk to each other to rebalance data, to move copies around, and to keep the replication of data high. HDFS is not fully POSIX-compliant, because the requirements for a POSIX file-system differ from the target goals for a Hadoop application. The tradeoff of not having a fully POSIX-compliant file-system is increased performance for data throughput and support for non-POSIX operations such as Append.

HDFS added the high-availability capabilities, as announced for release 2.0 in May 2012, allowing the main metadata server (the NameNode) to be failed over manually to a backup in the event of failure. The project has also started developing automatic fail-over.

The HDFS file system includes a so-called secondary namenode, which misleads some people into thinking that when the primary namenode goes offline, the secondary namenode takes over. In fact, the secondary namenode regularly connects with the primary namenode and builds snapshots of the primary namenode's directory information, which the system then saves to local or remote directories. These checkpointed images can be used to restart a failed primary namenode without having to replay the entire journal of file-system actions, then to edit the log to create an up-to-date directory structure. Because the namenode is the single point for storage and management of metadata, it can become a bottleneck for supporting a huge number of files, especially a large number of small files. HDFS Federation, a new addition, aims to tackle this problem to a certain extent by allowing multiple name-spaces served by separate namenodes.

An advantage of using HDFS is data awareness between the job tracker and task tracker. The job tracker schedules map or reduce jobs to task trackers with an awareness of the data location. For example: if node A contains data (x,y,z) and node B contains data (a,b,c), the job tracker schedules node B to perform map or reduce tasks on (a,b,c) and node A would be scheduled to perform map or reduce tasks on (x,y,z). This reduces the amount of traffic that goes over the network and prevents unnecessary data transfer. When Hadoop is used with other file systems this advantage is not always available. This can have a significant impact on job-completion times, which has been demonstrated when running data-intensive jobs.

HDFS was designed for mostly immutable files and may not be suitable for systems requiring concurrent write-operations.

Another limitation of HDFS is that it cannot be mounted directly by an existing operating system. Getting data into and out of the HDFS file system, an action that often needs to be performed before and after executing a job, can be inconvenient.

1.2 MapReduce

MapReduce is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster A MapReduce program is composed of a Map() procedure that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and aReduce() procedure that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). The "MapReduce System" (also called "infrastructure" or "framework") orchestrates by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance.

The model is inspired by the map and reduce functions commonly used in functional programming, although their purpose in the MapReduce framework is not the same as in their original forms. Furthermore, the key contributions of the MapReduce framework are not the actual map and reduce functions, but the scalability and fault-tolerance achieved for a variety of applications by optimizing the execution engine once.

MapReduce libraries have been written in many programming languages, with different levels of optimization. A popular open-source implementation is Apache Hadoop. The name MapReduce originally referred to the proprietary Google technology but has since been genericized.

Another way to look at MapReduce is as a 5-step parallel and distributed computation:

» Prepare the Map() input – the "MapReduce system" designates Map processors, assigns the K1 input key value each processor would work on, and provides that processor with all the input data associated with that key value.

» Run the user-provided Map() code – Map() is run exactly once for each K1 key value, generating output organized by key values K2.

» "Shuffle" the Map output to the Reduce processors – the MapReduce system designates Reduce processors, assigns the K2 key value each processor would work on, and provides that processor with all the Map-generated data associated with that key value.

» Run the user-provided Reduce() code – Reduce() is run exactly once for each K2 key value produced by the Map step.

» Produce the final output – the MapReduce system collects all the Reduce output, and sorts it by K2 to produce the final outcome.

Logically these 5 steps can be thought of as running in sequence – each step starts only after the previous step is completed – though in practice, of course, they can be intertwined, as long as the final result is not affected.

II. ARCHITECTURE OF HADOOP

Hadoop consists of the Hadoop Common package, which provides filesystem and OS level abstractions, a MapReduce engine and the Hadoop Distributed File System (HDFS). The Hadoop Common package contains the necessary Java ARchive (JAR) files and scripts needed to start Hadoop. The package also provides source code, documentation and a contribution section that includes projects from the Hadoop Community.

For effective scheduling of work, every Hadoop-compatible file system should provide location awareness: the name of the rack (more precisely, of the 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.

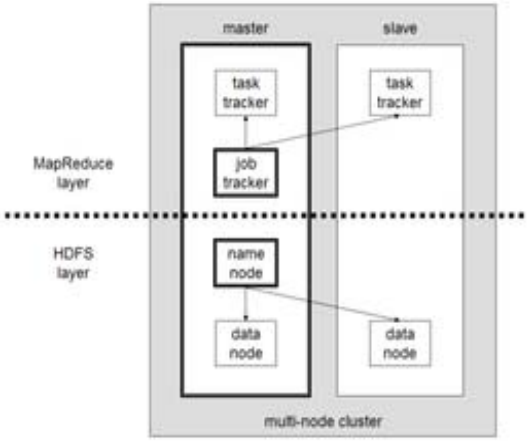

Figure: 2 Architecture of Hadoop

A small Hadoop cluster includes a single master and multiple worker nodes. The master node consists of a JobTracker, TaskTracker, NameNode and DataNode. A slave or worker node acts as both a DataNode and TaskTracker, though it is possible

to have data-only worker nodes and compute-only worker nodes. Hadoop requires Java Runtime Environment (JRE) 1.6 or higher. The standard start-up and shutdown scripts require Secure Shell (ssh) to be set up between nodes in the cluster.

In a larger cluster, the HDFS is managed through a dedicated NameNode server to host the file system index, and a secondary NameNode that can generate snapshots of the namenode's memory structures, thus preventing file-system corruption and reducing loss of data. Similarly, a standalone JobTracker server can manage job scheduling. In clusters where the Hadoop MapReduce engine is deployed against an alternate file system, the NameNode, secondary NameNameand DataNode architecture of HDFS is replaced by the file-system-specific equivalent.

III. ADVANTAGES AND DISADVANTAGE OF HADOOP

Advantages and disadvantage of Hadoop

3.1 Advantages of Hadoop

Following are the list of major areas where Hadoop is found very strong:

- Hadoop is a platform which provides Distributed storage & Computational capabilities both.
- Hadoop is extremely scalable, In fact Hadoop was the first considered to fix a scalability issue that existed in Nutch(-the open source crawler and search engine).
- One of the major component of Hadoop is HDFS (the storage component) that is optimized for high throughput.
- HDFS uses large block sizes that ultimately helps It works best when manipulating large files (gigabytes, petabytes...).
- Scalability and Availability are the distinguished features of HDFS to achieve data replication and fault tolerance system.
- HDFS can replicate files for specified number of times (default is 3 replica) that is tolerant of software and hardware failure, Moreover it can automatically re-replicates data blocks on nodes that have failed.
- Hadoop uses MapReduce framework which is a batch-based, distributed computing framework, Itallows paralleled work over a large amount of data.
- MapReduce let the developers to focus on addressing business needs only, rather than getting involved in distributed system complications.
- To achieve parallel & faster execution of the Job, MapReduce decomposes the job into Map & Reduce tasks and schedules them for remote execution on the slave or data nodes of the Hadoop Cluster.

3.2 Disadvantages or Limitation of Hadoop

Following are the major common areas found as weaknesses of Hadoop framework or system:

- As you know Hadoop uses HDFS and MapReduce, Both of their master processes are single points of failure, Although there is active work going on for High Availability versions.
- Until the Hadoop 2.x release, HDFS and MapReduce will be using single-master models which can result in single points of failure.
- Security is also one of the major concern because Hadoop does offer a security model But by default it is disabled because of its high complexity.
- Hadoop does not offer storage or network level encryption which is very big concern for government sector application data.

» HDFS is inefficient for handling small files, and it lacks transparent compression. As HDFS is not designed to work well with random reads over small files due to its optimization for sustained throughput.

» MapReduce is a batch-based architecture that means it does not lend itself to use cases which needs real-time data access.

» MapReduce is a shared-nothing architecture hence Tasks that require global synchronization or sharing of mutable data are not a good fit which can pose challenges for some algorithms.

IV. THE BUILDING BLOCKS OF HADOOP

4.1 NameNode

The NameNode -Hadoop employs a master/slave architecture for both distributed storage and distributed computation. The distributed storage system is called the Hadoop File System, or HDFS. The NameNode is the master of HDFS that directs the slave DataNode daemons to perform the low-level I/O tasks. The NameNode is the bookkeeper of HDFS; it keeps track of how your files are broken down into file blocks, which nodes store those blocks, and the overall health of the distributed filesystem.

The function of the NameNode is memory and I/O intensive. As such, the server hosting the NameNode typically doesn't store any user data or perform any computations for a MapReduce program to lower the workload on the machine. This means that the NameNode server doesn't double as a DataNode or a TaskTracker.

There is unfortunately a negative aspect to the importance of the NameNode—it's a single point of failure of your Hadoop cluster. For any of the other daemons, if their host nodes fail for software or hardware reasons, the Hadoop cluster will likely continue to function smoothly or you can quickly restart it. Not so for the NameNode.

4.2 DataNode

Each slave machine in your cluster will host a DataNode daemon to perform the grunt work of the distributed filesystem—reading and writing HDFS blocks to actual files on the local filesystem. When you want to read or write a HDFS file, the file is broken into blocks and the NameNode will tell your client which DataNode each block resides in. Your client communicates directly with the DataNode daemons to process the local files corresponding to the blocks. Furthermore, a DataNode may communicate with other DataNodes to replicate its data blocks for redundancy.

DataNodes are constantly reporting to the NameNode. Upon initialization, each of the DataNodes informs the NameNode of the blocks it's currently storing. After this mapping is complete, the DataNodes continually poll the NameNode to provide information regarding local changes as well as receive instructions to create, move, or delete blocks from the local disk.

4.3 Secondary NameNode

The Secondary NameNode (SNN) is an assistant daemon for monitoring the state of the cluster HDFS. Like the NameNode, each cluster has one SNN, and it typically resides on its own machine as well. No other DataNode or TaskTracker daemons run on the same server. The SNN differs from the NameNode in that this process doesn't receive or record any real-time changes to HDFS. Instead, it communicates with the NameNode to take snapshots of the HDFS metadata at intervals defined by the cluster configuration.

The NameNode is a single point of failure for a Hadoop cluster, and the SNN snapshots help minimize the downtime and loss of data. Nevertheless, a NameNode failure requires human intervention to reconfigure the cluster to use the SNN as the primary NameNode.

4.4JobTracker

The JobTracker daemon is the liaison between your application and Hadoop. Once you submit your code to your cluster, the JobTracker determines the execution plan by determining which files to process, assigns nodes to different tasks, and monitors all tasks as they're running. Should a task fail, the JobTracker will automatically relaunch the task, possibly on a different node, up to a predefined limit of retries. There is only one JobTracker daemon per Hadoop cluster. It's typically run on a server as a master node of the cluster. The JobTracker is the service within Hadoop that farms out MapReduce tasks to specific nodes in the cluster, ideally the nodes that have the data, or at least are in the same rack.

4.4.1Working of job tracker

Client applications submit jobs to the Job tracker.The JobTracker talks to the NameNode to determine the location of the data.The JobTracker locates TaskTracker nodes with available slots at or near the data.The JobTracker submits the work to the chosen TaskTracker nodes.

The TaskTracker nodes are monitored. If they do not submit heartbeat signals often enough, they are deemed to have failed and the work is scheduled on a different TaskTracker.

A TaskTracker will notify the JobTracker when a task fails. The JobTracker decides what to do then: it may resubmit the job elsewhere, it may mark that specific record as something to avoid, and it may may even blacklist the TaskTracker as unreliable.When the work is completed, the JobTracker updates its status.Client applications can poll the JobTracker for information.

The JobTracker is a point of failure for the Hadoop MapReduce service. If it goes down, all running jobs are halted.

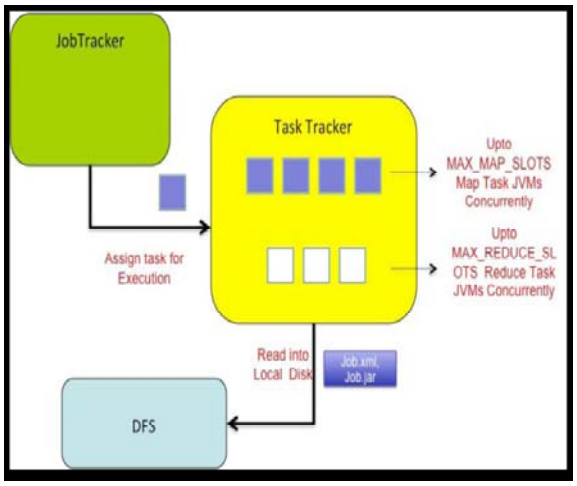

Figure.3: Working of Job Tracker

4.5 TaskTracker

As with the storage daemons, the computing daemons also follow a master/slave architecture: the JobTracker is the master overseeing the overall execution of a MapReduce job and the TaskTrackers manage the execution of individual tasks on each slave node. Figure 4 illustrates this interaction.

Each Task Tracker is responsible for executing the individual tasks that the Job Tracker assigns. Although there is a single Task Tracker per slave node, each Task Tracker can ***spawn*** multiple JVMs to handle many map or reduce tasks in parallel.

One responsibility of the Task Tracker is to constantly communicate with the Job Tracker. If the Job Tracker fails to receive a heartbeat from a TaskTracker within a specified amount of time, it will assume the TaskTracker has crashed and will resubmit the corresponding tasks to other nodes in the cluster. JobTracker Client TaskTracker Reduce Map TaskTracker Reduce Map TaskTracker Reduce Map TaskTracker Reduce Map.

4.6 Scheduling

By default Hadoop uses FIFO, and optional 5 scheduling priorities to schedule jobs from a work queue. In version 0.19 the job scheduler was refactored out of the JobTracker, and added the ability to use an alternate scheduler (such as the Fair scheduler or the Capacity scheduler).

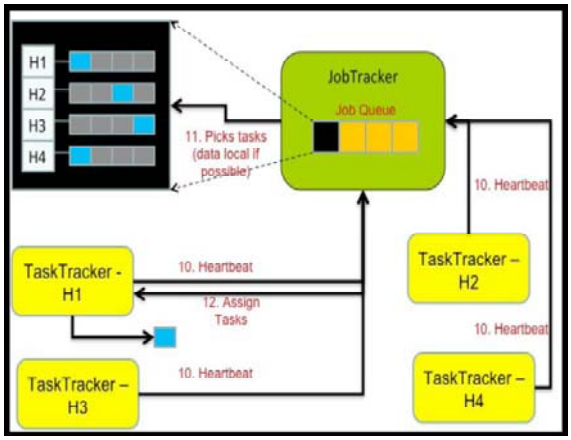

Figure:4:Working of Task Tracker

4.6.1 Fair scheduler

The fair scheduler was developed by Facebook. The goal of the fair scheduler is to provide fast response times for small jobs and QoS for production jobs. The fair scheduler has three basic concepts.

1. Jobs are grouped into Pools.
2. Each pool is assigned a guaranteed minimum share.
3. Excess capacity is split between jobs.

By default, jobs that are uncategorized go into a default pool. Pools have to specify the minimum number of map slots, reduce slots, and a limit on the number of running jobs.

4.6.2 Capacity scheduler

The capacity scheduler was developed by Yahoo. The capacity scheduler supports several features that are similar to the fair scheduler.

» Jobs are submitted into queues.

» Queues are allocated a fraction of the total resource capacity.

» Free resources are allocated to queues beyond their total capacity.

» Within a queue a job with a high level of priority has access to the queue's resources

There is no preemption once a job is running.

V. PROPOSED CONCEPT

To provide the JobTracker scalability, the solution supports running multiple JobTrackers in the system in parallel. A simple load balancing algorithm is implemented on the JobClient side, which distributes users' jobs amongst the JobTrackers currently running in the system. In the same time, TaskTrackers are distributed amongst the JobTrackers and each TaskTracker can be also re-assigned to another JobTracker in case the current JobTrackers load distribution requires that.

5.1 Multiple JobTrackers with master-slave replication model

To implement this model, the JobTracker must provide:

» A facility to accept Job Submissions.

» A Job Processing agent which schedules the tasks to the TaskTrackers and monitor their status.

» Some way to manage/store the current state of the system.

» A coordination management agent that can send/receive status updaterequests/responses.

The Master JobTracker is the only entity that can communicate with theTask Trackers. It has a variable number of Slave JobTrackers, that monitor its health state and, eventually, elect its successor, as a master, in case of a failure. So the coordination agent should have status monitoring capabilities and a fault-proof election protocol should be defined.

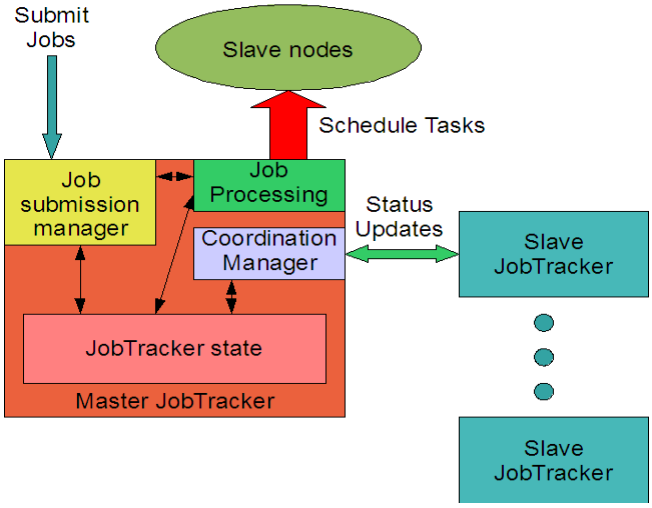

Figure:5 Logical architecture of master-slave replication model

5.1.1State management in master-slave

» Every operation that a_ects the state triggers a status change event, and, consequentially, a status update to slaves (Slaves in Hot Stand- By).

» A user-con_gurable number of state changes or a time interval triggers a status change event (Slaves in Warm Stand-By).

» A status update event is never triggered, so, in case of a failure, a slave start from scratch (Slaves in Cold Stand-By).

The first choice is the best, despite the introduced overhead, because if the JobTracker fails, a hot slave can be brought on-line, as the new master, without notifying the TaskTrackers (enhancing failure transparency and reducing the amount of modi_es that should be made to TaskTracker's code). The third option cannot be considered in an environment where high-availability is needed, so it's excluded a priori.

5.1.2 State management in master-slave

» Status can be managed in different ways:

» Soft-state with eager coordination protocol (pessimistic).

» Soft-state with lazy coordination protocol (optimistic).

» Soft-state shared between replicas with a shared, reliable cache system (e.g. Ehcache).

» Hard-state stored in the DFS (every operation that changes state triggers a _le system write operation).

5.1.3 Soft-state with eager coordination protocol

This protocol consists of four phases:

1. Status affecting operation submission.

2. Execution and local status update.

3. Status update submission to slaves and two-phase change commit protocol.

4. Conform operation successfully executed (or result return).

In Figure 6, this phases are explained graphically.

This protocol is meant to avoid that potential failures during the status update phase, but it delays every operation the time needed to reach a status agreement between copies.

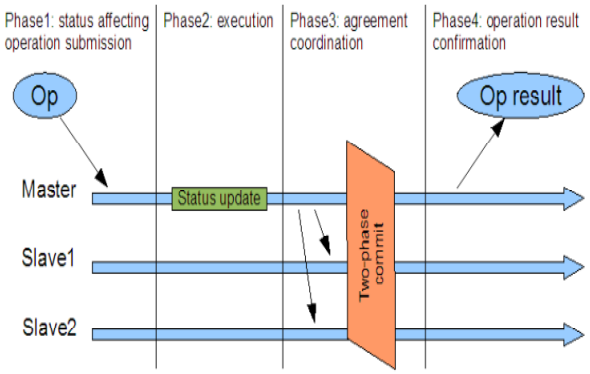

Figure:6: Soft-state with eager coordination protocol

5.1.4 Soft-state with lazy coordination protocol

This protocol consists of four phases:

1. Status affecting operation submission.

2. Execution and status update.

3. Conform operation successfully executed (or result return).

4. Submit status changes to slaves and start a reconciliation phase to synchronize the status.

In Figure 7, this phases are explained graphically.

This protocol assumes no failures can occur between the result return and the reach of a status agreement between copies. It does not delay operation result but if a failure happens during the reconciliation phase it could lead to have an inconsistent state between copies.

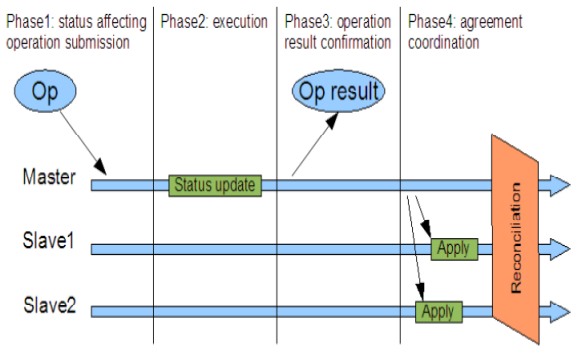

Figure:7 Soft-state lazy coordination protocol

VI. CONCLUSION

The proposed design is based on saving the JobTracker state into the Infinispan data store. The JobTracker is started on a distributed task, and in case the node where it runs fails, the framework performs the JobTracker automatic failover on some other, healthy node. During restart, the JobTracker loads the stored state and continues with the normal operation. While in failover, TaskTrackers can continue executing the running jobs normally and the JobClients and TaskTrackers automatically reconnect to the JobTracker upon the restart. For all these task, the system is available for users to submit new jobs all the time.

In my opinion, it is better to avoid using active copies due to the high complexity of the coordination protocol and, instead, using a master-slave model with soft-state shared between copies through a distributed cache mechanism or saved on HDFS.

References

1. R. Abbott and H. Garcia-Molina. Scheduling I/O requests with deadlines: A performance evaluation. In Proceedings of the 11th Real-Time Systems Symposium, pages 113–124, Dec 1990.
2. An introduction to Apache Hadoop for big data. http://bigdataanalyticsnews.com/introduction-apache-hadoop-big-data/
3. Hadoop. http://hadoop.apache.org, 2009.
4. HDFS (hadoop distributed file system) architecture. http://hadoop.apache.org/common/docs/current/hdfs design.html, 2009.
5. Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma,Khaled Elmeleegy_ Scott Shenker,Ion Stoica , ―Job Scheduling for MultiUser MapReduce Clusters‖ ,University of California, Berkeley , Facebook Inc _Yahoo! Research Electrical Engineering and Computer Sciences University of California at Berkeley Technical Report No. UCB/EECS-2009-55 http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-55.html April 30, 2009
6. G. Candea, N. Polyzotis, and R. Vingralek. A scalable, predictable join operator for highly concurrent data warehouses. In 35th International Conference on Very Large Data Bases (VLDB), 2009.
7. R.-I. Chang, W.-K. Shih, and R.-C. Chang. Deadline-modificationscan with maximum-scannable-groups for multimedia real-time disk scheduling. In RTSS '98: Proceedings of the IEEE Real-Time SystemsSymposium, page 40, Washington, DC, USA, 1998. IEEE Computer Society.
8. J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In OSDI'04: Proceedings of the 6th Symposium on Opearting Systems Design & Implementation, pages 10–10, 2004.
9. J. Dean and S. Ghemawat. Mapreduce: a flexible data processing tool. Commun. ACM, 53(1):72–77, 2010.
10. K. DeLathouwer, A. Y. Lee, and P. Shah. Improve database performance on file system containers in IBM DB2 universal database v8.2 using concurrent I/O on AIX. Technical report, IBM, 2004.
11. S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, pages 29–43, New York, NY, USA, 2003. ACM.
12. J. M. Hellerstein, M. Stonebraker, and J. Hamilton. Architecture of a database system. Foundations and Trends in Databases, 1(2):141–259, 2007.
13. Richard O. Duda, Peter E. Hart, and David G. Stork. Pattern Classification (2nd Edition). Wiley-Interscience, edition, November 2000.
14. Geoffrey Holmes Bernhard Pfahringer Peter Reutemann Ian H. Witten Mark Hall, Eibe Frank. The weka data mining software. SIGKDD Explorations, 11(1), 2009.
15. Hadoop Distributed File System. http://hadoop.apache.org/ common/docs/current/hdfs design.html.
16. Fair Scheduler. http://hadoop.apache.org/common/docs/r0.20. 2/fair scheduler.html
17. Capacity Scheduler. http://hadoop.apache.org/common/docs/ r0.20.2/capacity scheduler.html.
18. Quan Chen, Daqiang Zhang, Minyi Guo, Qianni Deng, and Song Guo. Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. In Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on, 29 2010.
19. J. Polo, D. Carrera, Y. Becerra, V. Beltran, J. Torres, and E. Ayguade and. Performance management of accelerated mapreduce workloads in heterogeneous clusters. In Parallel Processing (ICPP), 2010 39th International Conference on, pages 653 –662, 2010.
20. Aameek Singh, Madhukar Korupolu, and Dushmanta Mohapatra. Server-storage virtualization: integration and load balancing in data centers. In Proceedings of the 2008 ACM/IEEE conference on Supercomputing, SC '08, pages53:1–53:12, Piscataway, NJ,USA, 2008. IEEE Press.
21. P. J. Shenoy and H. M. Vin. Cello: a disk scheduling framework for next generation operating systems. In SIGMETRICS '98/PERFORMANCE '98: Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems, pages 44–55, New York, NY, USA, 1998. ACM.

