ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 2, pp: (796-801), Month: April - June 2015, Available at: www.researchpublish.com

DISTRIBUTED DATA CACHE FOR BIGDATA WITH LRU CACHE USING MAPREDUCE TECHNIQUE

[1]R.Saranya, [2]N.Sundaram

[1]M.E. (CSE) Final Year, [2]Assistant Professor, Maha Barathi Engineering College, Vasudevanur, India

Abstract: **In this big data world we can see daily based data which are generated day by day e.g. Google, amazon, Facebook etc. This large amount of data is un-reliable to store, manage and analyze. This large volume of data runs on Commodity hardware, which are parallel arranged. The data that are large in volume takes more time to execute for particular process and causes fail to distributed system because these huge volume data runs on distributed system. To overcome this issue a framework was proposed called HADOOP for big data processing and is being used now days in organizations. It process large amount of data in small amount time rather than large distributed systems. It has automatic fault tolerance capacity to tackle with failed of systems in execution or processing time using Map Reduce programming technique. Here execution time is still an issue for delivering large amount of data and processing it repeatedly for particular oracle. Existing HADOOP system does not have any feature to reduce time for recompilation. I propose a HADOOP distributed system for large data processing, which has two type of cache memory one is local cache, and other is distributed centralized cache memory. I use these cache memories for reducing the recompilation time. Distributed cache consisted with least memory space so LRU cache is implemented to allocate memory space simultaneously. The LRU caching scheme is to remove the least recently used frame when the cache is full and a new page is referenced which is not there in cache.**

Keywords: **big data, distributed, Map Reduce, HADOOP, LRU cache.**

1. INTRODUCTION

Big data is an all-encompassing term for any collection of data sets so large and complex that it becomes difficult to process using on-hand data management tools or traditional data processing applications. Due to such large size of data it becomes very difficult to perform effective analysis using the existing traditional techniques. Since Big data is difficult to work with using most relational database management systems and desktop statistics and visualization packages, requiring instead "massively parallel software running on tens, hundreds, or even thousands of servers. What is considered "big data" varies depending on the capabilities of the organization managing the set, and on the capabilities of the applications that are traditionally used to process and analyze the data set in its domain. "For some organizations, facing hundreds of gigabytes of data for the first time may trigger a need to reconsider data management options. For others, it may take tens or hundreds of terabytes before data size becomes a significant consideration. Big data due to its various properties like volume, velocity, variety, variability, value and complexity put forward many challenges.

Apache's HADOOP is a efficient software tool to access and handle large data set. Hadoop follows the master/slave architecture decoupling system metadata and application data where metadata is stored on dedicated server Name Node and application data on Data Nodes. The Hadoop Distributed File System (HDFS) is a distributed file system 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. It provides high throughput access to application data and is suitable for applications that have large data sets. It is part of the Apache Hadoop Core project. Map Reduce works by breaking the processing into two phases: the map phase and the reduce phase. Each phase has key-value pairs as input and output, the types of which may be chosen by the

programmer. The programmer also specifies two functions: the map function and the reduce function. The input to our map phase is the raw data. I choose a text input format that gives us each line in the dataset as a text value. The key is the offset of the beginning of the line from the beginning of the file.

The purpose of the cache is to duplicate frequently accessed or important data in such a way that it can be recalled with the least effort and delay. The trade-off in distributed caching is network latency and serialization. Distributed memory object caching system is lightweight, scalable, and fast. It speeds up applications by alleviating database, file system and computational load. It saves the time and money by allowing using the existing server memory more efficiently. Dcache offers a simple plug-and-play customization options which allow to use own serialization and logging, allowing for over 70,000 adds per second, 100,000 bulk gets per second and 150,000 removes per second. The various challenges are Capture, Storage, Search, sharing, transfer, analysis, Visualization.

The LRU caching scheme is to remove the least recently used frame when the cache is full and a new page is referenced which is not there in cache.

I use two data structures to implement an LRU Cache.

1. A Queue which is implemented using a doubly linked list. The maximum size of the queue will be equal to the total number of frames available. The most recently used pages will be near front end and least recently pages will be near rear end.
2. A Hash with page number as key and address of the corresponding queue node as value.

When a page is referenced, the required page may be in the memory. If it is in the memory, we need to detach the node of the list and bring it to the front of the queue. If the required page is not in the memory, we bring that in memory. In simple words, we add a new node to the front of the queue and update the corresponding node address in the hash. If the queue is full, i.e. all the frames are full, we remove a node from the rear of queue, and add the new node to the front of queue.

2. PROBLEM STATEMENT

To reduce the execution time for large application by caching the data.

To coordinate all the local caches in a given cluster.

To evict the unused cached item from cache memory.

3. SCOPE

The scope of the Distributed HADOOP cache memory is to accelerate HADOOP cluster and faster execution. It can be used large scale development of large amount of data. The centralized distributed cache memory is used for the scope.

4. SYSTEM ANALYSIS

Existing System:

In existing they use local cache for execution that is not efficient for faster execution. It can only use cache in the local systems. As an example, Terabytes of posts generated on Facebook or 400 billion annual twitter tweets could mean Big Data! This enormous amount of data will be stored somewhere to analyse and come up with data science reports for different solutions and problem solving approaches.

Big data requires fast processing. Time factor plays a very crucial role in several organizations. For instance, millions of records are generated in the stock market which needs to be stored and processed with the same speed as its coming into the system.

There is no specific format of Big Data. It could be in any form such as structured, unstructured, text, images, audio, video, log files, emails, simulations, 3D models, etc. Until now I have been working with only structured data. It might be difficult to handle the quality and quantity of unstructured or semi-structured data that we are generating on a daily basis.

Proposed System:

The motivation over this system is to use intermediate results for further execution of the applications, where I do not want to execute results which are previously processed. In this system I proposed a distributed cache strategy which keeps

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 2, pp: (796-801), Month: April - June 2015, Available at: www.researchpublish.com

previously processed data in cache memory at local and centralized. So there are two cache memories will be used for faster execution of applications, one will be local cache and another will be centralized distributed cache.

In DFS, I can divide a large set of data files into smaller blocks and load these blocks into multiple number of machines which will then be ready for parallel processing. For example, if I have 1 Terabyte of data to read with 1 machine and 4 Input / Output channels with each channel's reading speed id 100MB/sec, the whole 1 TB data will be read in 45 minutes. On the other hand, if I have 10 different machines, I can divide 1 TB of data into 10 machines and then the data can be read in parallel which reduces the total time to only 4.5 minutes.

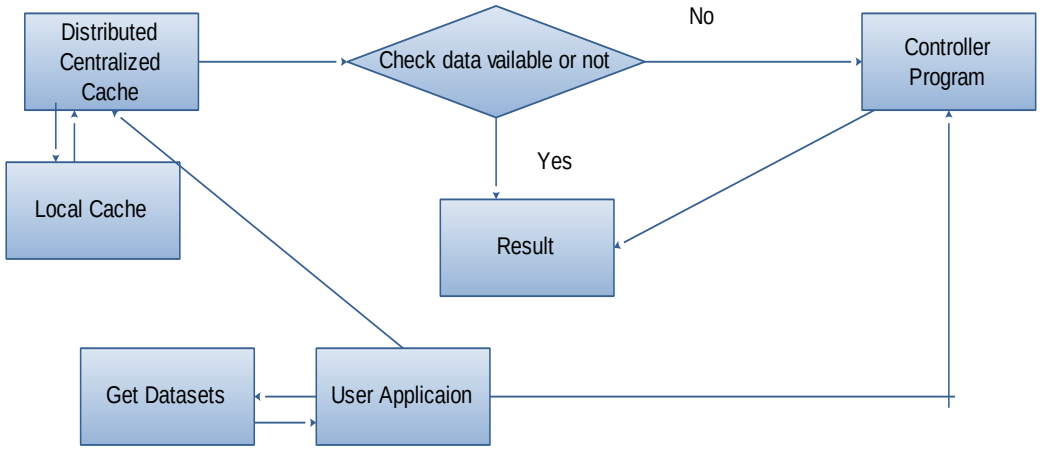
Parallel Processing: When data resides on N number of servers and holds the power of N servers, then the data can be processed in parallel for analysis, which helps the user to reduce the wait time to generate the final report or analysed data.

Fault Tolerance: The Fault tolerance feature of Big Data frameworks (like Hadoop) is one of the main reason for using this framework to run jobs. Even when running jobs on a large cluster where individual nodes or network components may experience high rates of failure, BigData frameworks can guide jobs toward a successful completion as the data is replicated into multiple nodes/slaves.

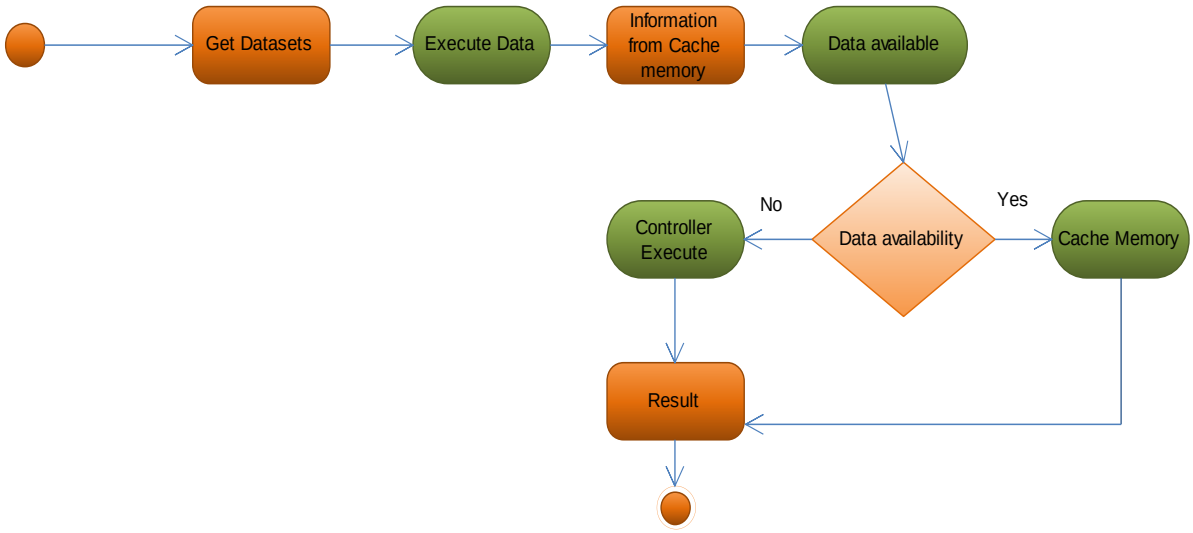
Use of Commodity Hardware: Most of the Big Data tools and frameworks need commodity hardware for its working which reduces the cost of the total infrastructure and very easy to add more clusters as data size increase.

5. DIAGRAMS

Data Flow:

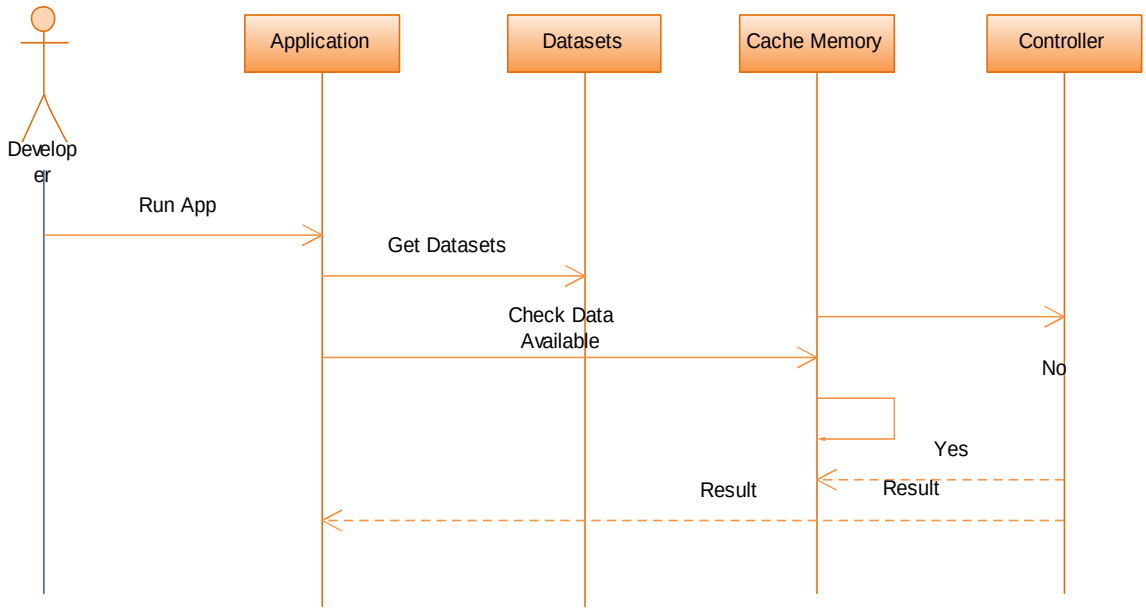


Activity Diagram:

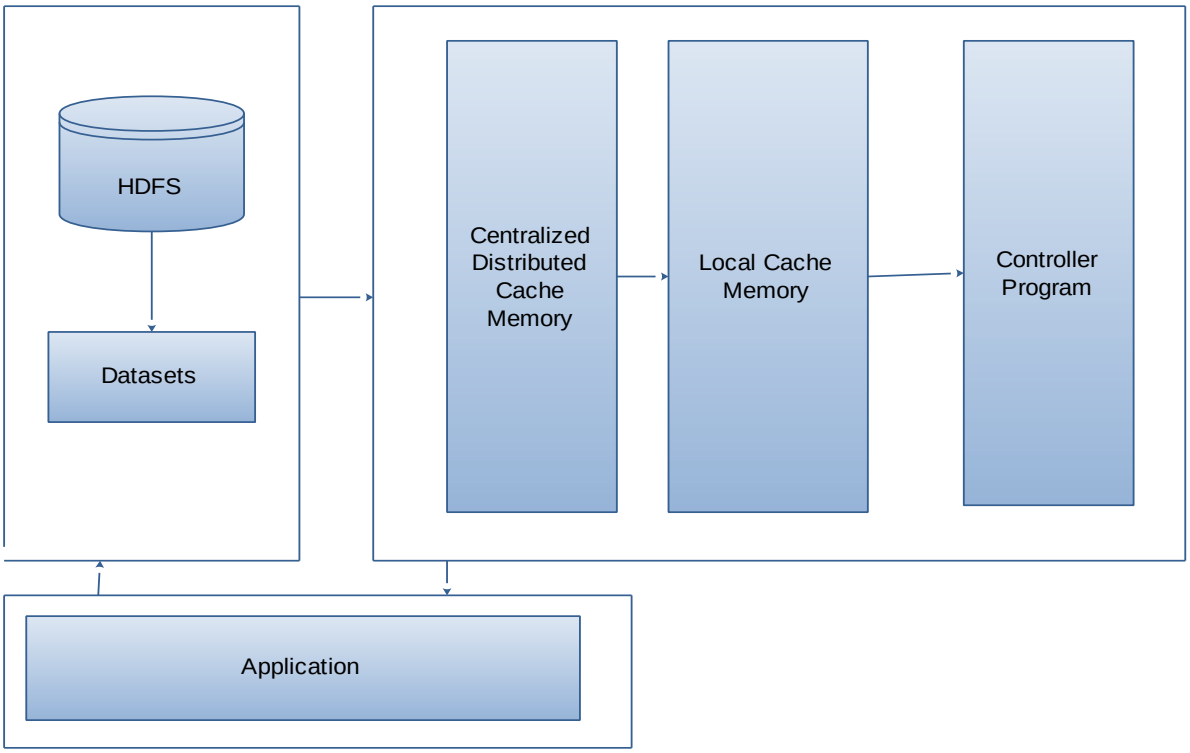


ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 2, pp: (796-801), Month: April - June 2015, Available at: www.researchpublish.com

Sequence Diagram:



Architecture:



Modules Description:

- Cluster and Data Information
- Distributed cache
- Evicting cached item
- Timestamp
- LRU cache

Cluster formation and upload dataset:

Hadoop cluster can be formed by single node or multi -node. The HADOOP cluster mainly consists of Name node, Data node, Job tracker, task tracker. Each node in the cluster is either master or slave. Slave nodes are data \node and task tracker. HDFS is a primary storage in HADOOP application which exposes a file system namespace and allows user to store the data in it. The HDFS file consists of many blocks. Each block consists of 64 MB. Each block is replicated three times .So that the blocks can be processed fastly and easily. Data nodes are responsible for serving read and write request. The name node is the repository for all HDFS metadata. Then massive amount of datasets are uploaded into the file system. The datasets which are cached in cache memory may be structured or unstructured. Execution time for processing the structured and unstructured data will be low.

Configuring Hadoop Cluster:

- I need to configure JDK in Linux system.
- Install Open-SSH Server for Hadoop network.
- Disable IPV6.
- Extract and copy HADOOP tar into Linux system.
- Configure Hadoop xml files.

Developer formats name node using "HADOOP name node -format" command. Then creates a directory under user folder based on user name. Finally create a directory to upload input data. Upload data to input directory

Distributed Cache:

Rather than serializing side data in the job configuration, it is preferable to distribute datasets using Hadoop's distributed cache mechanism. This provides a service for copying files and archives to the task nodes in time for the tasks to use them when they run. To save network bandwidth, files are normally copied to any particular node once per job. Accessing data from cache is much faster compared to disk access. To improve the overall performance and efficiency of both Hadoop and MapReduce, a new system called Cencache have been proposed.

Evicting Cached Item:

Keeping the cached item for a long time will result in wastage of resource and memory. The unused item should be removed from the memory in order to make it available for new cached item. So that Large amount of space will be available. For this, numerous cache replacement algorithms have been designed to increase the hit-ratio. Many well-known cache replacement policies like LRU and LFU alone not effective due to the mobility of the task.

Adaptive replacement cache is the efficient cache replacement policy for cache utilization. This algorithm constantly balances LRU and LFU to improve the combined result. It splitting the cache directory into two lists T1 and T2 for recently and referenced entries. Any entry in L1 that referenced once more, gets another chance and enters L2.Entries entering the cache (T1,T2) will cause to move towards the target maker. If no free space exists in the cache, this marker also determines whether either T1 or T2 will evict an entry.

Timestamp:

After performing distributed cache and evicting unused data item, the final result will be display along with its execution time. The calculated execution time will be present in both time and graph. Time will be display in millisecond. Then the cache execution time is compared with the normal execution (Disk memory). This comparison will be useful for getting a clear view about the centralized cache performance. From that the cache hit ratio and cache misses will be calculate by using the cache oblivious technique.

LRU cache:

Least Recently Used (LRU).Discards the least recently used items first.This algorithm requires keeping track of what was used when, which is expensive if one wants to make sure the algorithm always discards *the* least recently used item. General implementations of this technique require keeping "age bits" for cache-lines and track the "Least Recently Used" cache-line based on age-bits. In such an implementation, every time a cache-line is used, the age of all other cache-lines changes.

6. CONCLUSION

I proposed HADOOP distributed system for large data processing, which has two type of cache memory one is local cache, and other is distributed centralized cache memory. I use these cache memories for reducing the recompilation time. I presented the design and procedures of Distributed cache system for data searching framework in MapReduce. I implemented distributed cache for data searching to save compilation time. MapReduce technique coding will execute the searching of data's in efficient way. So, system processing speed will be improved and compilation time will be saved.

I designed Hadoop Distributed File System (HDFS) 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. It provides high throughput access to application data and is suitable for applications that have large data sets. It is part of the Apache Hadoop Core project. Map Reduce works by breaking the processing into two phases: the map phase and the reduce phase. Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer. The programmer also specifies two functions: the map function and the reduce function. The input to our map phase is the raw data. I choose a text input format that gives us each line in the dataset as a text value. The key is the offset of the beginning of the line from the beginning of the file.

Execution of data's in distributed cache system will be speculated for further more reduction of recompilation timing. Distributed cache memory is consisted with GB capacity space. Once data's completely occupied that memory space leave lack of cache memory will occur. To overcome this problem I have to clear long time unused data's in cache memory. By using LRU cache technique the system should automatically delete that unused data's in regular interval to maintain certain limit of free spaces in cache memory.

REFERENCES

[1] Shafer, J.,Rice Univ., Houston, TX, USA,Rixner, S. ; Cox, A.L.," The Hadoop distributed filesystem: Balancing portability and performance", Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on 28-30 March 2010.

[2] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica University of California, Berkeley "Spark: Cluster Computing with Working Sets" Amawon web services. http://aws.amazon.com/.

[3] Cache algorithms. http://en.wikipedia.org/wiki/Cache algorithms.

[4] Google compute engine. http://cloud.google.com/products/computeengine.html.

[5] Hadoop. http://hadoop.apache.org/.

[6] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. Commun. of ACM, 51(1):107–113, January 2008.

[7] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/ subscribe. ACM Comput. Surv., 35(2):114–131, June 2003

Web address:

[8] http://en.wikipedia.org/wiki/Apache_Hadoop

[9] http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

[10] http://hadoop.apache.org/docs/r2.3.0/hadoop-project-dist/hadoop-hdfs/CentralizedCacheManagement.html