IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 8, August 2016

Cache Management for Big Data Applications: Survey

Kiran Grover[1], Surender Singh[2]

Dept. of Computer Science and Engineering, Om Institute of Technology & Management, Hisar, India[1, 2]

Abstract: Big data processing consumes resources at very large scale and its very challenging to manage the memory for each running process. Researchers have developed the some memory management schemes which can synchronize the data flow for each process. Processes use two memories i.e. Main memory which retains only current processing data and cache which retains frequent required data for processes. It is very challenging to replace the cache data with new one because, during the processing of large volume data, process may claim any data block, so due to random block replacement, process may have to wait for a long time which may result unnecessarily delay. In this paper, we will explore the current research work related to cache management.

Keywords: Big Data, cache, memory, resource management

I. INTRODUCTION

Big data can be referred as huge collection of data. Its main resources are Public/Govt. data, internet, social media, news channels, educational institutes, industries etc. Data is stored on the disks and for analysis purpose; it is loaded in to memory which has a limited size. So traditional system's memory is not suitable for big data processing in real time environment. Traditional systems have different types of memory i.e. volatile and non volatile memory and volatile memory consists of main memory and cache. Cache refers to the intermediate data that is produced by worker nodes/processes during the execution of a Map Reduce task. A piece of cached data is stored in a Distributed File System (DFS). The content of a cache item is described by the original data and the operations applied. Formally, a cache item is described by a 2-tuple i.e. Origin and Operation. Origin is the name of a file in the DFS. Operation is a linear list of available operations performed on the Origin file.[1][8][9]

Large-scale distributed systems (e.g., Google, Facebook) operate on streams of key-value pairs with support of large scale computer networks. Due to the huge volume of input data, streaming is necessary, where the job arriving event and job processing event, both are handled simultaneously. MapReduce applications support parallel processing of jobs. For example, caching can be viewed as computation onal streams, where the frequency of duplicate items determines performance (i.e., hit rate). Large-scale graph-processing algorithms that output edges/nodes in bulk (e.g., BFS search)can also be reduced to streams, where performance may be determined by the size of the frontier (i.e., pending nodes), bias in the observed degree, and/or discovery rate of new vertices. It is common to replace keys with their hashes and apply computation that outputs data in random order, either by design (e.g., reversing edges in graphs) or as by product of some previous computation (e.g., sorting by a different key in an earlier stage of MapReduce). This results in realworkload consisting of randomized streams, in which keysare shuffled in some arbitrary order. Understanding statistical properties of these streams is an important area of research as it leads to better characterization of MapReduce, caching, graph exploration, and more general streaming [1]. However, existing analysis is not just scattered across many fields but is also lacking in its ability to accurately model the stochastic properties of random streams. Google's MapReduce was a huge shift in the evolution of big data processing tools. Since then Hadoop, the open source version of Google MapReduce has become the mainstream of big data processing, with many other tools emerging to handle big data problems. Extending the original MapReduce model to include iterative MapReduce, tools such as Twister and Ha Loop can cache loop invariant data in iterative algorithms locally to avoid repeat input data loading in a MapReduce job chain. Spark also uses caching to accelerate iterative algorithms by abstracting computations as transformations on RDDs instead of restricting computations to a chain of MapReduce jobs.[1][2][3]
There are two types of cache items as the map cache and the reduce cache. They have dissimilar complexities when it comes to sharing under diverse scenarios. Cache items in the map phase are simple to share because the operations useful are generally well-formed. When considering each file divided, the cache manager reports the earlier file divided method used in the cache item. The next new MapReduce activity/ job also need to be divided into the files giving to the same division method in order to utilize the cache items. If the new MapReduce job uses a different file splitting order, the map outcomes cannot be used directly. When seeing cache sharing in the reduce phase, two general situations are identified.[16][17][18]
The first is when the several reducers complete different jobs from the cached reduce cache items of the earlier

MapReduce jobs. In this case, after the mappers submit the results gained from the cache items, the MapReduce framework usages the partitioned provided by the new MapReduce job to feed input to the reducers. The protected computation is obtained by removing the processing in the Map phase. Usually, new content is added at the end of the input files, which requires additional mappers to process. However, this does not need additional processes other than those introduced above. Another situation is when the reducers can actually take advantage of the previously cached reduced cache items. The reducers control how the output of the map phase is shuffled. The cache manager routinely identifies the best-matched cache item to feed each reducer, which is the one with the maximum overlap in the original input file in the Map phase. [1][2][3]

Cache management architecture

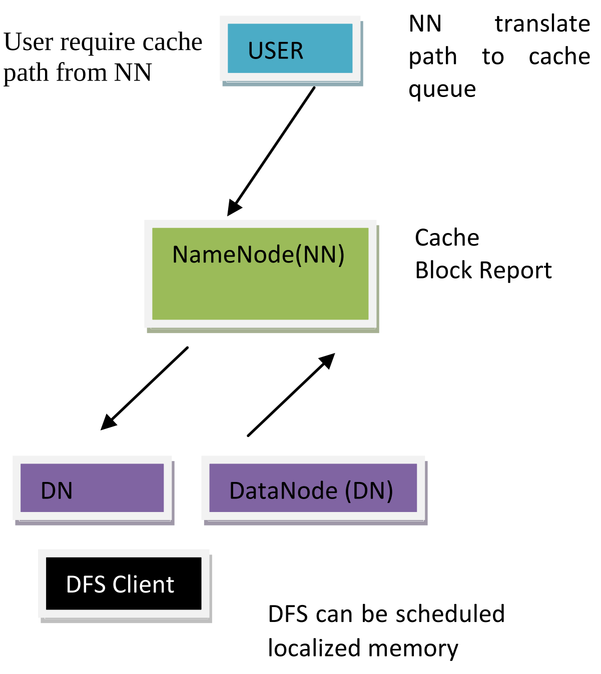
User require cache path from NN
USER
NN translate path to cache queue
NameNode(NN)
Cache Block Report
DN
DataNode (DN)
DFS Client
DFS can be scheduled localized memory

Figure: Cache management architecture [19]

<path>	Contains cache path
<pool-name>	cache pool for directives.
-force	cache pool resource limits audit.
<replication>	Cache Replication Factor
<time-to-live>	Set Directive validation over a time.

TABLE: CACHE DIRECTIVE.

II. LITERATURE SURVEY

Yaxiong Zhao et al. This paper evaluated a data aware cache framework that requires minimum change to the original MapReduce programming model for provisioning incremental processing for Bigdata applications using the MapReduce model. Dache, a data-aware cache description scheme, protocol and architecture. This method requires only a slight modification in the input format processing and task management of the MapReduce framework. As a result, application code only requires slight changes in order to utilize Dache. Experiments show that it can eliminate all the duplicate tasks in incremental MapReduce jobs and does not require substantial changes to the application code.

S. Tamboli et al. presented an efficient one way caching method for big data applications based on MapReduce framework. Cache supports is provided to intermediate data that is created by worker nodes processes during the execution of a MapReduce task. A part of cached data is then kept in DFS. The content of a cache item is well-defined by the original data and the operations applied. Formally, a cache item is described by a 2-tuple i.e. Origin and Operation. Origin is the name of a file in the distributed file system. The operation performed is a data structure in the form of linear list of existing operations done on the original file. They used cache mechanism efficiently to optimize computational time and reduce storage overhead for real time data over the distributed file system.

S. T. Ahmed et al. presented stochastic model framework to analyze LRU caching, MapReduce overhead, and different crawl properties i.e. node-degree bias, frontier size in random graphs. Proposed scheme is used for characterizing applications that process random data streams, including properties as the probability of uniqueness for discovered keys, number of unique values accumulated over a certain time period, the average degree of seen nodes, and the size of the frontier during crawls on large-scale graphs under three different strategies. These models were applicable not just to synthetically generated streams, such as those produced by BFS on random graphs, but also real workloads stemming from LRU caching and Map Reduce processing of IRLbot and WebBase graphs.

(Bingjing Zhang et al.2015): This paper introduced a collective communication layer for communication optimizations required by the applications. They used Map Collective programming model on top of collective communication abstractions to enhance expressiveness and performance of big data processing. Harp is an implementation designed in a pluggable way to bridge differences between Hadoop ecosystem and HPC system and bring high performance to the Apache Big Data Stack through a clear communication abstraction, which did notexist before in the Hadoop ecosystem. Simple modifications of Mahout library that enhances its low parallel performance this shows that value of building new abstractions into Hadoop rather than developing a totally new infrastructures authors did in their prototype, called Twister system. With three applications, the analysis shows that with Harp these applications can scale up to 128 nodes with 4096 CPU son the Big Red II super computer, where the speedup in most tests is close to linear. Current work can be extended for development of high performance communication libraries for simulation (exascale). It will also support fault tolerance to evaluate the current best practices in MPI, Spark and Hadoop.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 8, August 2016

Authors will also consider the data abstractions to include those needed in pixel and spatial problems.
D. Huang et al. shows a enhanced MapReduce implementation of FIM algorithm by developing a cache layer and a selective online analyzer to analyze effectiveness and efficiency of Smart Cache via extensive experiments on four public datasets. Authors explored new improvement space on top of the state-of-the-art solution, and presented a new regression based method to find optimal cache size. Analysis result shows that smart Cache can be reduced from45.4%, to 97.0% for execution time as compared with the state-of-the-art solution E. Park et al.2015 investigated the memory optimization for big data processing to reduce the energy consumption .They developed a function called memory fast-forward (MFF) which can process the graph computations with optimal memory requests. Simulation results show that MFF unit can reduce 54.6% energy consumption due to low memory traffics. Proposed work can be extended to support large scale systems i.e. multi-GPU.
A.D. et al. explored Hadoop framework and compared its performance against I MapReduce and HaLoop for graph based iterative algorithms. Ha Loop offers better performance as it stores intermediate results in cache and reuses those data on the next successive iteration. For using cache invariant data (inter-iteration locality) it schedules the tasks onto the same node that might occur in different iterations. Hadoop has to reload the data from HDFS in each iteration performing the same operation, thereby wasting a huge amount of resources like CPU jobs, I/O cycles and network bandwidth. And it requires an additional MapReduce job for checking the termination condition. It has been seen from the experimental results that HaLoop offers better performance as it loads the static data only once and by keeping those invariant data in cache. To facilitate the caching of invariant data the scheduler must ensure that tasks are assigned to the same nodes across multiple iterations allowing the use of cache. Proposed scheme can be extended to support parallel concurrent transactions. Tak-Lon Wu et al. provided a fast cache solution for big data processing application by using the combination of Hadoop and PIG frameworks. They used use K-means clustering and Page Rank that has three components: a python control-flow script, a Pig data-transformscript for a single iteration, and two K-means user-defined functions written with a Pig-provided Java interface.

During each iteration, Loader in each Mapper loads the aggregated centroids into memory as vector objects from the distributed cache on disk before computing the Euclidean distances for data points in the Loader stage. Each loader outputs assigned centroids and data points as fields in a single bag, each field in bag is defined as string data type which further splits into tuples for matching Pig's GROUP operation to collect partial centroid vectors from mappers. It takes the average of all partitions, emits to a final centroids file and saves it to HDFS. Results show that proposed scheme is able to provide the fast cache response for Mapreduce.

D. Wei et al. proposed a multi-granularity content tree model and pay-as-you-go mode to support evolvement data modeling. These features help to split the data model,, position data content precisely and to expand the dimensions of the main features that described according to the data subject, and then gradually discover data contained information and relationships among the information. Considering the large size of the data features, this paper designs data persistence mode based on HBase, so as to achieve the purpose of data processing by using technologies within the Hadoop system. Authors also presented data content extraction and content tree initial state algorithms under MapReduce framework, and dynamic loading and local caching algorithms of content tree, thus forming a basic extract-store-load process. This work can be extended for geosciences information and knowledge discovery using platform which is based on big data technology, and construct the value chain of geological data results; construct geosciences information resource sharing and value-added demonstration services to create favorable conditions for information integration, resource sharing and knowledge innovation for the whole society

D. Yang et al. [12] investigated the limitations and strength of Map reduce framework using HiBench workloads. Experimental results show that the speedup Native Task can extend the range of HiBench upto 160%.. Current work can be extended to provide the support for Hive or Tez tools.

III. PROBLEM FORMULATION

Big Data processing requests consume lot of resources over system. Large scale buffer is required for data processing because there are several input/output operations may exists those can be executed parallel. Frequent required data can be temporarily stored in a buffer, called cache but it is not feasible to store huge volume of random data n cache due to its cost although cache can reduce the number of input and output operations. For large scale data, it is still a major issue that how to optimize the cache in such a way that service consumer process should perform minimum buffer operations by making the use of cache memory.
Common Cache management Issues:
Cache configuration
State Maintenance for the cached objects
Fault Tolerance

IV. CONCLUSION

In this survey paper, cache management issues were investigated. Researchers have developed various solutions to manage the cache for Big Data processing and each has its own merit sand demerits. Now we discuss most relevant solutions for cache management, i.e. Data Aware Caching, Dache is a framework based on Mapreduce MAP Reduce. It is able to identify the duplicate tasks for elimination purpose. HiBench workloads can be used to analyze the performance of Map reduce framework, HBase Tree based data model splits the data and data repositioning is used to process large volume data by dynamic loading and local cache management. It can be further enhanced to process the distributed geological data. K-means clustering and PageRank offers fast cache solution which works on the combination of Hadoop and PIG frameworks and it splits data into multiple tuples which are further processed by Mappers. Optimization of CPU based I/O operations can reduce the requirements of available bandwidth. Memory Fast-Forward reduces the total memory traffic and optimizes energy consumption. Smart cache can reduce the total process execution time. This study can be further utilized to develop a solution for cache management.

REFERENCES

[1] Yaxiong Zhao, Jie Wu, and Cong Liu, "Dache: A Data Aware Caching for Big-Data Applications Using the MapReduce Framework", TSINGHUA SCIENCE AND TECHNOLOGY, IEEE-2014,Vol.19(1), February 2014 pp. 39-50
[2] ShakilTamboli ,Smita Shukla Patel,"Survey on innovative approach for improvement in efficiency of caching technique for Big Data Application", ICPC, IEEE-2015, pp.
[3] SarkerTanzir Ahmed, Dmitri Loguinov, "Modeling Randomized Data Streams in Caching, Data Processing, and Crawling Applications", INFOCOM, IEEEE-2015
[4] Bingjing Zhang, Yang Ruan, Judy Qiu, "Harp: Collective Communication on Hadoop", IEEE-2015, pp.228-233
[5] Dachuan Huang, Yang Song, RamaniRoutray, Feng Qin,"SmartCache: An Optimized MapReduceImplementation of Frequent Itemset Mining", IEEE-2015, pp.16-25
[6] Eunhyeok Park, JunwhanAhn, Sungpack Hong, SungjooYoo, and Sunggu Lee, "Memory Fast-Forward: A Low Cost Special Function Unitto Enhance Energy Efficiency in GPU for Big Data Processing", Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE-2015, pp.1341-1346
[7] AkashdeepDebbarma, Annappa B., Ravi G. Mude, "Performance Analysis of Graph Based Iterative Algorithms on MapReduce Framework", International Conference for Convergence of Technology, IEEE-2014, pp.1-6
[8] Hao Zhang, Gang Chen, Beng Chin Oo, Kian-Lee Tan,Meihui Zhang, "In-Memory Big Data Management and Processing: A Survey", IEEE-,pp.1-24
[9] RupaliPashte, Ritesh Thakur, "A Survey on Optimal Data Storage of Cache Manager for Big Data Using Map Reduce Framework", IJSR-2012, pp.1510-1513
[10] Tak-Lon Wu, Abhilash Koppula, Judy Qiu, "Integrating Pig with Harp to Support Iterative Applications with Fast Cache and Customized Communication", International Workshop on Data-Intensive Computing in the Clouds, IEEE-2014, pp.33-39
[11] Dongqi Wei, Chaoling Li, WumutiNaheman, "Organizing and Storing Method for Large-scale Unstructured Data Set with Complex Content", International Conference on Computing for Geospatial Research and Application, IEEE-2013, pp.70-76
[12] Dong Yang, Xiang Zhong, Dong Yan, Fangqin Dai, Xusen Yin,"NativeTask: A Hadoop Compatible Framework for High Performance", ICBD, IEEE-2013, pp.94-101
[13] https://en.wikipedia.org/wiki/Apache_Hadoop
[14] http://hadoop.apache.org/docs/current/
[15] https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html
[16] https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html
[17] https://pig.apache.org/
[18] https://hive.apache.org/
[19] https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop hdfs/CentralizedCacheManagement.html