International Journal of Scientific Engineering and Research (IJSER)
www.ijser.in
ISSN (Online): 2347-3878, Impact Factor (2014): 3.05

SCahS: Synchronized Caching Strategies for Incremental Computations on Hadoop

Shakil B. Tamboli[1], Smita Shukla Patel[2]

[1]Student of MEIT, Department of Information Technology, Smt.Kashibai Navale College of Engineering, Pune, India
[2]Assistant Professor, Department of Information Technology, Smt.Kashibai Navale College of Engineering, Pune, India

Abstract: ***Data for various applications are evolving continuously either online or offline way. The business communities extract this available big data and search the potential business opportunities using it. For sophistication of this huge data many data processing systems are designed by software companies; but neither fulfills efficient updates or enhance efficiency as expected by business community's demand. These systems lack in performance and do not support compatibility with existing distributed computing environments like Hadoop MapReduce. In addition to these drawbacks it requires to develop only application specific algorithms which need to be replaced dynamically by programmers resulting creating complexity in design and code. The solution is planned to minimize computational time, wastage of input and output bandwidth and optimize resources namely central processing unit, memory utilization for incremental data. Another aspect is also taken care is that the design of proposed system will be simple, easy to understand and execute and fulfill the expectations demanded by users. The system saves intermediately generated data for a specific application by caching it. The result obtained from the proposed system reduces significant computational time and saves memory space.***

Keywords: MapReduce, Hadoop, Caching, Distributed Processing, Big Data, Incremental Computation, Recursive Queries

1. Introduction and Background

The amount of data generated by machines is greater than people. Machine logs, RFID readers, sensor networks, vehicle GPS traces, retail transactions generate lots of data. This volume of data is made publicly available so that organizations extract value from their and other organizations' data to succeed in business to make unexpected and hitherto unimaginable applications. Massive scale data intensive applications like web data analysis, click stream analysis, network monitoring log analysis and analysis of high throughput data from sensors and devices need highly scalable parallel data processing platforms. Some known examples will give us the flood of data becoming available from sources like The New York Stock Exchange generates one terabyte of new trade data per day and Facebook manages uploading of nearly 10 billion photos requires one petabyte of storage [1], [2]. This large volume of data is not simple enough to measure, capture, process, analyze and store electronically. The important attributes of these applications are that they are incremental in nature.

MapReduce framework is implemented on distributed systems as a batch query processor with the ability to run an ad hoc query against whole dataset and get the results in a reasonable time. A MapReduce framework has characteristic as computation should be moved towards data. The ever growing gap between the computation and input output disk is taken at top priority for minimization so that speedup can be achieved. The framework changes the way of computation and unlocks data that was previously archived on tape or disk. It gives opportunity to people for innovation with data irrespective of formats either structured, unstructured or semi structured. The framework implements linearly scalable programming approach in which developer introduces two functions called map and reduce. These functions are unaware about the size of data or cluster on which they are processing so remain unchanged irrespective of data size. The framework has the inherent capability to process data with simple programming way by hiding complexity of infrastructure needed for parallelization, data transfer, scalability, fault tolerance and scheduling [3].

However it is observed that the framework is inefficient for incremental nature of workloads. Some problems need to be solved to improve performance of systems like how to avoid reloading and reprocessing of unchanged data during successive iterations in order to avoid wastage of input, output, processor and memory resources and network bandwidth and how to optimize and decide fix point termination [4]. Therefore a system is proposed to reduce completion time and storage space overhead for Hadoop jobs using synchronized caching strategies or mechanisms. The other objectives kept in mind to improve efficiency, performance, avoid duplication of operations for same data and schedule the jobs efficiently [5].

The proposed system has suggested following techniques to overcome observed drawbacks and challenges. Cache strategies are much in use since long years in computing communities to enhance the performance of operations by applying them in the form of data structure, programming policies, separate speedup frameworks and systems. Cache has properties to access data faster than disk. Cache boosts memory access with optimal architectures and cost in which data is duplicated or partitioned among different layers easily. The important cohesion property of cache retains data consistently on memory. The locality principles retain data within certain regions of time and space.

The system caches the data locally on DataNode and distributed over the clients or dedicated servers or storage devices which forms a single cache for effective management. This local cached data then moved to the NameNode so that it can be accessed from all available locations to form a single cache or global cache. Hence decisions of caching data can be taken by cache manager

Licensed Under Creative Commons Attribution CC BY

available on central coordinator or DataNodes with additional protocols.

Caching of data is made faster and easier by the reference caching principle in which a HDFS block is formed with two files namely Meta file and block file. Meta file is used for checksum value of data and block file for actual data. So if these files references are cached, it helps in locating the files faster.

The remaining part of paper presented in following manner. The proposed system overview and design is discussed in section II. Section III describes details of proposed system. Section IV and V have a note on implementation and experimental setting procedures. Section VI highlights results obtained during the performance of experiments. The conclusion is given in section VII and in section VIII analyzes related work from earlier systems.

2. Proposed System Overview and Design

The design implemented the concept of self-adjusting computation in Hadoop framework by verifying file names which are uploaded by same names from a single or different machine and maintains transparency with HDFS through an append only file system.

System resources are optimized through keeping the input and intermediate data in memory for MapReduce job's whole life cycle. For example, in Word Count, the intermediate buffer will keep a pointer to a string in the input data as a key, instead of copying the string to another buffer. It is implemented in the RecordReader interface to process a piece of input split generated from the InputSplit interface.

Reusing intermediate data provides opportunities to save the expensive operations such as concurrent memory allocation and need of building additional data structures. Data Locality is improved using temporal and spatial locality principles; in which temporal locality sequentially touches the whole input data only once in the Map phase and randomly touches discrete parts of intermediate data multiple times of Reduce phase for generating final results and spatial locality improves locality for data parallel applications. The proposed system architecture is presented in figure 1.

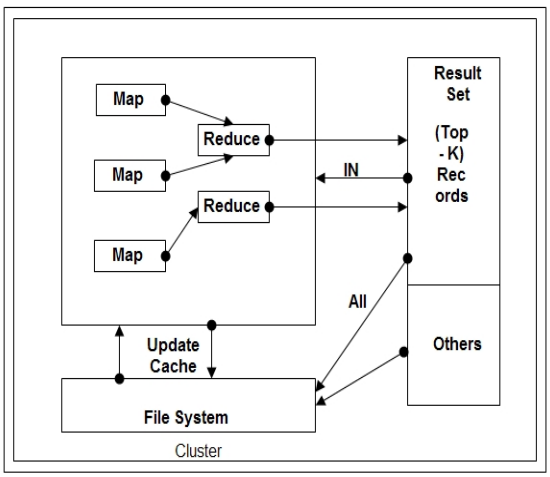

Figure 1: Proposed System Architecture

3. Details of Proposed System

A. Proposed Methodology of System

The system allows caching of data blocks by each DataNode of HDFS and facilitates reads of these blocks by other DataNodes, The custom data structure called synchronized caching strategy is created to speed up dynamic word files by caching memory objects in RAM and hence reducing the number of times the word files must be read. A block cached at a data node is registered in to the hash table. It stores each entry as a key value pair, where key is the block ID to be accessed and value is the DataNode ID where the block is cached. Sufficient amount of RAM memory is reserved at each node to serve as local cache. The system keep a log of the most recently cached blocks in the entire system with the corresponding node where the block is cached.

When a particular node is in need of a data block it generates two simultaneous requests where one is directed at local system which returns the address of the node that has a cached version of the block in question and the other is forwarded to the NameNode that provides the requester with the whereabouts of a replica of the block.

The system implemented synchronized caching strategies which are the combination of two simple greedy caching policies. The first one is to cache an object locally whenever a node needs it and called receiver only greedy caching policy and it helps to serve future block requests for the same block at the same node. The second is for a node to cache an object whenever some other node requests for it and the object is in the NameNode called sender only greedy caching. When compared and evaluated it has been observed that the synchronized caching strategy performs better on existing system and results in less cache misses per job execution.

B. Execution Flow

The data flow for Hadoop job is shown in figure 2. The jobtracker is responsible for scheduling the job to run by assigning map and reduce tasks available on task trackers. The map tasks close to their corresponding InputSplits must next read input data. If the data is in HDFS reading requires network communication with the NameNode. The map tasks de serializes the input data to generate a stream of key/value pairs that is passed into the mapper. The mapper outputs key/value pairs, which are immediately serialized and placed in a buffer. While in the buffer, Hadoop may run the user's combiner to combine values associated with equivalent keys. When the buffer fills up, they are sorted and flushed out to local disk [6].

Once map output has been flushed out to disk, reducer tasks start fetching their input data. This requires disk and network I/O. Each reducer outputs a sequence of key/value pairs that is sent to the OutputFormat for output. The data is written out to the local DataNode replicated to a configurable number of other DataNodes for further use.

Licensed Under Creative Commons Attribution CC BY

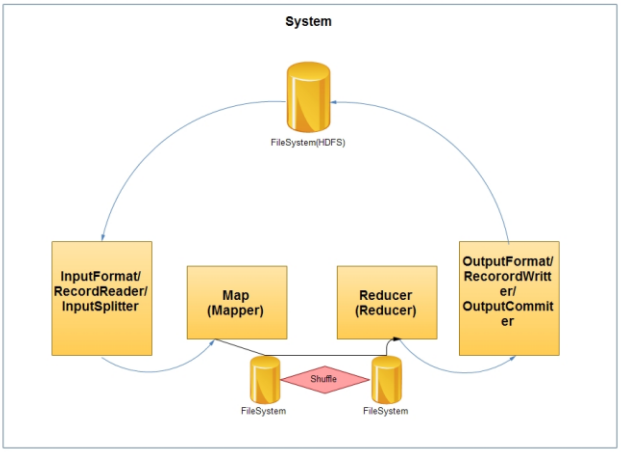

Figure 2: Data flow in a Hadoop system

The data flow in proposed system is shown in figure 3. The cache in proposed system is mostly transparent to the user, as it is intended to work with unmodified Hadoop jobs.

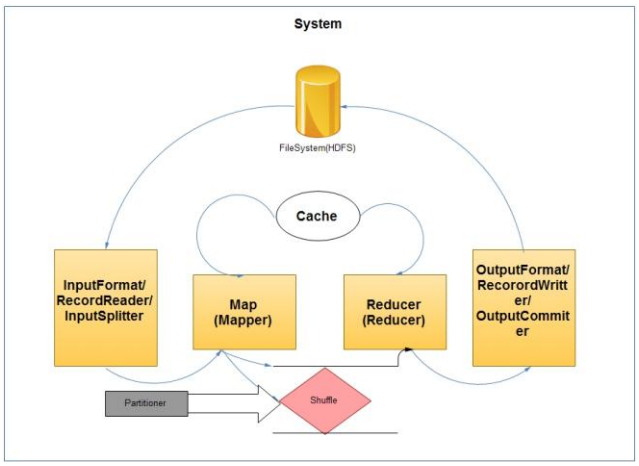

Figure 3: Data Flow in Proposed System

The system provides an in-memory key/value cache between multiple jobs in a job sequence for effective communication. The system provides RecordReader to read in data and de serializes it into a key/value sequence of client. It caches the key/value pairs in memory (related with the input file name) before passing to mapper. Thus in a subsequent job, when the same input is requested, this data will be directly obtained from the cache.

The engine makes effort to avoid the time and space overhead of de serialization by locally shuffling data. The system allows programmer to control keys partitioning among reducers. The default implementation uses a hash function to map keys to partitions. The system provides partition guarantee by mapping partitions to places in deterministic strategy.

C. Models Created in Proposed System

Following models are created to achieve better results.

1. Preprocessing File- In file preprocessing stop words are removed and stemming is performed so that proper collection of words on which operations are performed will be retained.
2. File Vector- When collection of words activity ends in preprocessing, it is very important to evaluate how important a word is to a document in a collection or corpus. The significance increases equivalently to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Tf-idf (Term Frequencies Inverse Document Frequencies) algorithm [7] is a statistical measurement weight of about the importance of word in a document often used in search engine, web data mining, text similarity computation and other applications. So file vector manages above details.
3. Create Signature- To find similar file it should be compared with existing files available among the millions of files to make comparison process faster. To create signature bit vector is used and initialized to zero first then hashed with file vector so that decision will be taken regarding whether existing file to be incremented or decremented.
4. Use Locality sensitive hashing to find nearest neighbor- In large clustering environment to compare file signature; locality sensitive hashing technique is used to ensure that only nearest neighbor need to be checked to place file [8].
5. Store file with related files- Name Node maintains subclustertable which store subclusterid and file placed on that cluster and if subclusterid is not found then new subcluster will be created.

The various data structures implemented are locality sensitive hashing function, subclustering and storing mapping information, cachetable, storing intermediate result in the form of either array of structure or linked list or object of classes.

D. Mathematical Model of Proposed System

Initialization of cache is done as per following procedure. Consider there are n assigned cache nodes and the average size of cache on each node is a, clearly, the total cache size S in the cluster is:

$$S = n * a \quad (1)$$

Let the size of file be l and k the unique words to be cached are computed as:

$$k = S / l \quad (2)$$

Algorithm for Updating the Cache is as per following formula. Existing known cache updating algorithms only take some single issues into consideration which make them replace the cache in an incompetent way. Therefore, it is recommended a methodology to calculate the value degree Value i of a cache tuple i:

$$\text{Value } i = Fi \times Ti / (Tci - Tli) \quad (3)$$

where Fi is the frequency of tuple i being accessed, Ti is the delay time of fetching tuple i from the disk, Tci is current time and Tli is the last access time. As per Formula 3, it is observed that Value i increases as the access frequency Fi and delay time Ti increase and decreases as the interval of fetching data (Tci-Tli) increases.

Licensed Under Creative Commons Attribution CC BY

The data saved on a node modifies endlessly in the real situation, so as to timely cache the aggregated data it requires regularly update the cache. At the time of update, we substitute the last n tuples ranking with the value degree computed with Formula 3. The last n tuples cached on the nodes but seldom used in aggregate computation. These tuples are substituted by some other data stored on the disk. This operation is executed during process of periodic update [9].

The theorems described here prove and compares the initial run time and memory and for fresh updated dynamic reduced time and memory overhead. The various notations used in theorems are t_h for the time it takes to hash data and t_m for the time it takes to send a short message (one that does not contain entire inputs and outputs of tasks). The bounds depend on the total number of map tasks; written N_M and the total number of reduce tasks written N_R. Similarly n_i and n_O denotes the total size of the input and output respectively, n_m to denote the total number of key-value pairs output by Map phase, and n_{mk} to denote the set of distinct keys emitted by the Map phase [10].

Theorem 1 (Initial Run: Time and Overhead). Assuming that Map, Combine, and Reduce functions take time asymptotically linear in their input size and that Combine functions are monotonic, the total time for performing an incremental MapReduce computation with an input of size n_i, where n_m key-value pairs are emitted by the Map phase is

$O(t_m \cdot (N_M + N_R + N_C))$.

Theorem 2 (Initial Run: Space). The total storage space for performing computation with an input of size n_i, where n_m key-value pairs are emitted by the Map phase, and where Combine is monotonic is $O(n_i+n_m+n_o)$.

Theorem 3 (Dynamic Update: Space and Time). The dynamic update with fresh tasks F requires time

$O(t_m (N_M + N_C + N_R) + \sum t(a))$. The total storage requirement is the same as an initial run.

Theorem 4 (Number of Fresh Tasks). If the Map function generates k key-value pairs from a single input record, and the Combine function is monotonic, then the number of fresh tasks, $|F|$, is at most $O(k \log n_m + k)$.

Maintenance of the Cache Coherency is managed as per following procedure. Cache the same data on a backup node and put the backup node into the cluster as a reserve by adding a flag in the aggregation result files to label whether a word has already been cached. If a node is in fail state, then find the backup of data files on the failed node in the cluster and re cache it as per the flag. With this approach, the cached data can be recovered in a short time.

E. Improvements in Hadoop

The data is stored after its initial operation in cache created at local as well as global level. The contributions added to improve the system are basically remote memory caching, more data local jobs and reference caching. In remote memory caching, caching of input data at the DataNode level lowers job execution time. A distributed cache structure is adopted so that DataNodes caches are maintained by Cache Managers. Performance is improved by adding more slots instead of nodes which helps in more map and reduce tasks to be scheduled parallel. Better execution time is obtained with more data local tasks. TaskTracker contacts JobTracker to check availability of slot. If slot is available task is scheduled. If input for task is on a different node, then it is rack local and data is streamed from other node. But with caching, tasks scheduled completed earlier. And in reference caching, initial request attended by the references cached contributed to the improvement because these references assist the system to find the data into cache faster.

4. Implementation

Synchronized caching strategies are implemented by extending Hadoop components. Cache manager communicates with task trackers and provides cache items on receiving requests which are implemented in the system. The cache manager uses HDFS, the DFS component of Hadoop, to manage the storage of cache items. In order to access cache items, the mapper and reducer tasks first send requests to the cache manager.

Mapper and Reducer classes only accept key value pairs as the inputs which are fixed by Hadoop interface. An open accessed component InputFormat class allows application developers to split the input files of the MapReduce job to multiple file splits and parse data to key value pairs. The component TaskTracker class is responsible for managing tasks, understand file split and bypass the execution of mapper classes entirely. TaskTracker also manages reducer tasks. Similarly, it could bypass reducer tasks by utilizing the cached results.

5. Experimental Settings

The experimental setup configured for the proposed system is a machine with 3 cores CPU, each core running at 2.10 GHz, 3GB memory, and a SATA disk along with installation of Ubuntu operating system, Hadoop 2.0 framework, Java 6, Net beans editor with Maven build project enviornment. The number of mappers, reducers and replication factors are set by Hadoop framework are default. The application to benchmark the speedup of synchronized caching strategy over Hadoop is word count. It counts the number of unique words in large input text files, searches and sorts the pattern matching words along with their quantity. It is an IO intensive application requires loading and storing a sizeable amount of data during the processing.

6. Results and Discussion

The results obtained from proposed synchronized caching strategy have shown remarkable improvements. The execution time differs between default Hadoop and proposed system as file size increases. Issuing data from cache is faster than disk as requests are served from references. The overall execution time is reduced if the blocks are sent from cache (local or remote) for processing.

Licensed Under Creative Commons Attribution CC BY

Figure 4, 5, 6, 7 and 8 shows the size of data input for processing and reduction in time which are computed as ratio between the input and time of a dynamic run using SCahS i.e. proposed system in this paper and those of Hadoop. The findings of these experimental results are stated in following way: (i) SCahS obtain better performance improvements for all applications when there are incremental changes to the input data. (ii) Higher reduction in computational time for computation intensive applications and for data intensive applications also. (iii) As the size of the incremental data varies there is decrease as in computational time because larger changes allow fewer computation results from previous runs to be reused.

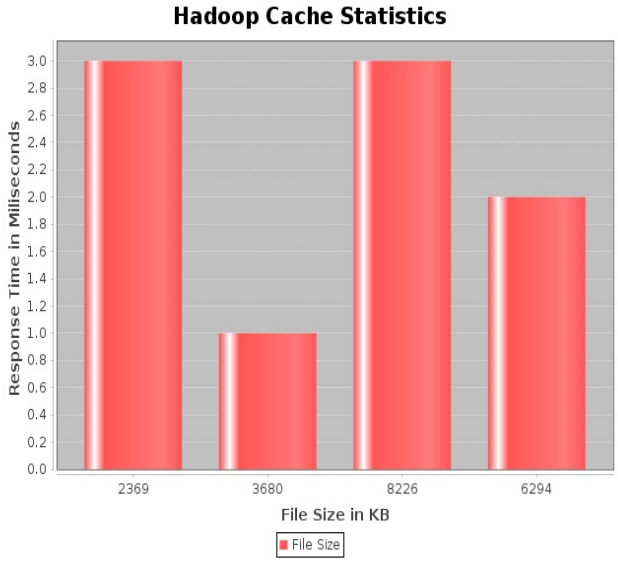

Figure 4: Response time of Hadoop for WC application

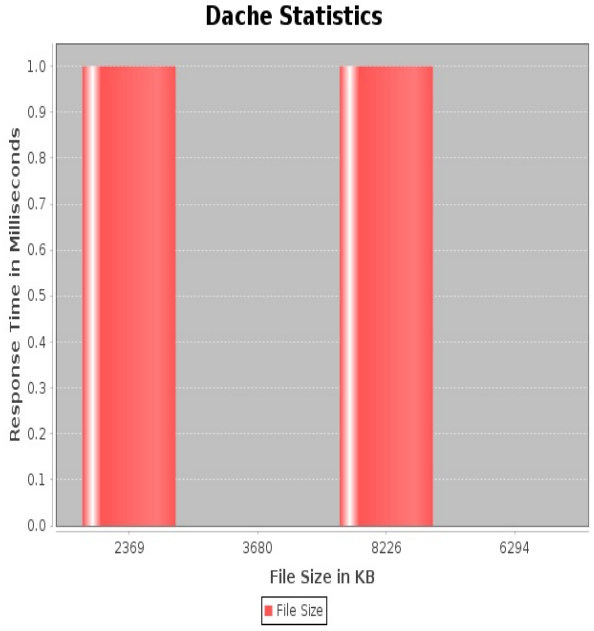

Figure 5: Response time of SCahS for WC application

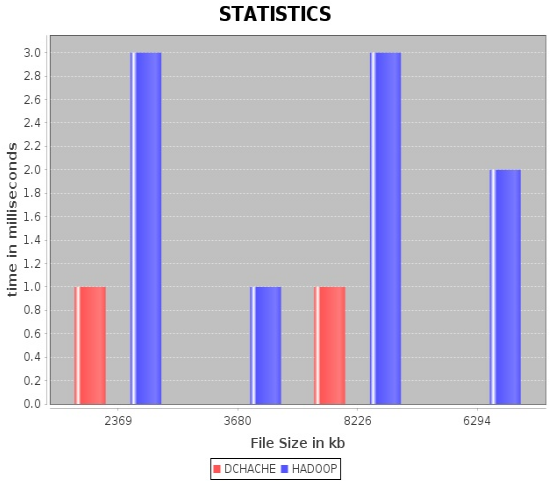

Figure 6: Comparison of Response time between Hadoop and SCahS for WC application

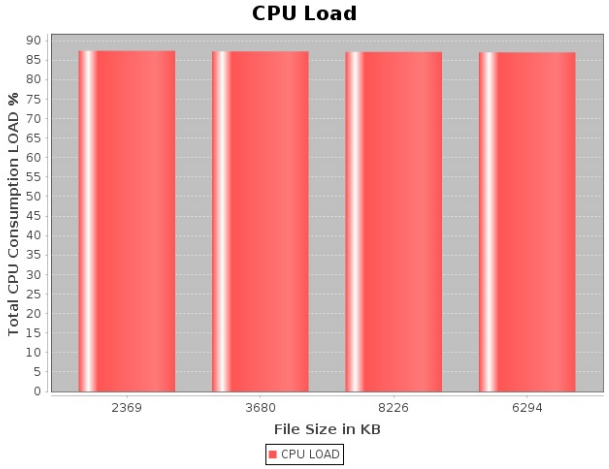

Figure 7: CPU load during operations on SCahS system

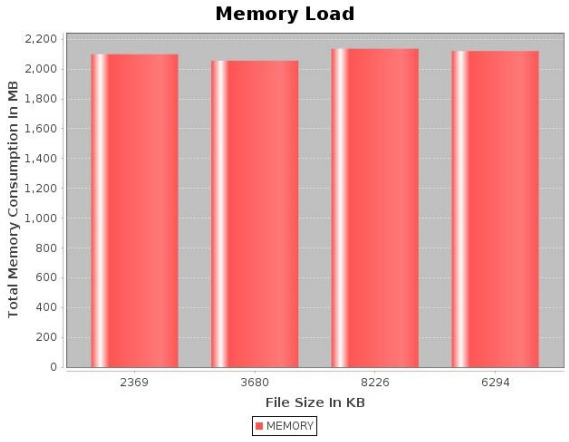

Figure 8: Memory load during operations on SCahS system

7. Related Work

Active research to improve performance of data intensive applications is adopted by scientific and research communities.

The Incoop system [10] notices changes in input automatically update the output by providing an efficient, fine grained result reuse mechanism with the help of programming methods and task level memorization techniques, storage systems, contraction phase for reduce

Licensed Under Creative Commons Attribution CC BY

tasks and affinity based scheduling algorithms. In [11] EFind recommended index structure based on statistical adaptive optimization to improve performance of big data queries. In [12] two partitioning algorithms called XTrie and ETrie developed to improve load balancing for distributed applications.

In paper [13] the cache based design to store intermediate data generated by MapReduce to reduce completion time of big data job's is proposed along with cache request and reply protocol. In the system cache manager customizes the indexing of data objects for applications to describe their operations and contents of their partial results.

Tiled-MapReduce (TMR) [14] implemented the prototype called Ostrich for iteratively processing small pieces of data so as to process an enormous amount of data at one time on shared memory multicore platforms at faster speeds. However there are more design constraints regarding the size of iteration window and whether to copy keys/values at large quantity or not.

In[15] to reduce the gap in disk access time and bandwidth for big cluster based systems, the system design provides proactive fetching and caching mechanism based on Memcached distributed caching system. The system adopted two level greedy caching strategy in which initially cache an object locally whenever a node needs it and unavailable in local cache called receiver only greedy caching and later cache an object whenever some other node requests for it and the object is in the file system but not in the local cache called sender only greedy caching.

In [16] the Redoop infrastructure is validated. Redoop presents Window Semantic Analyzer for optimization, Dynamic Data Packer as partition executor, Execution Profiler gathers the statistics, Local Cache Manager and Window Aware Cache Controller to maintain window aware metadata of reduce input and output data which is cached on any of the task nodes' local file systems.
In [17] PACMan implemented two cache replacement policies LIFE and LFU-F which are designed to reduce average completion time of jobs and maximize output of the cluster using all or nothing property described as retaining inputs of jobs with low wave widths and maximizing retention of frequently used inputs.

C-Aware cache management and storage algorithm proposed in paper [18]. The speed of cache media and network load conditions considered for establishing this strategy. The system analyzes historical information of accessed cached data from network and forecasts the future access to the cache and storage server performance based on historical information.

In paper [19] collaborative caching approach adopted to lower job execution times on DataNode. In collaborative caching mechanism cache distributed over the clients, dedicated servers or storage devices form a single cache to fulfil the requests. Effective data local jobs techniques is achieved in this arrangement. Local as well as remote data was cached on DataNodes and supplied as an input to MapReduce jobs. The global cache is formed from all the participating DataNode's machines. Smart cache solution from paper [20] implements two phase structure, but takes care to avoid unnecessary searching overhead in the second phase. SmartCache's success observed from key principle which states that if one itemset is very close to, although not above, the threshold, it might exceed the threshold in other splits.

HaLoop system is presented in paper [21] to efficiently handle the iterative nature of applications. HaLoop insisted on two simple intuitions for better performance by MapReduce. In first iteration a MapReduce cluster can cache the invariant data and then reuse that data in further iterations. And in second stage a MapReduce cluster can cache reducer outputs making checking for a fix point more efficient, without an extra MapReduce job. The paper makes contributions by designing new Programming Model and Architecture for Iterative Programs with the help of programming interface to express iterative data analysis applications. Loop Aware Task Scheduler to enable data reuse across iterations by physically colocating tasks in different iterations. Caching for Loop invariant data by caching and indexing data that are invariant across iterations in cluster nodes during the first iteration of an application. And caching to Support fix point evaluation at the reducer's local output.

8. Conclusion

Synchronized caching strategy is designed, implemented and evaluated by extending Hadoop framework for provisioning incremental processing for big data applications. The proposed synchronized caching strategy is powerful for cache management. The new value degree cache replacement algorithm is implemented to serve as eviction policy. The results obtained during the experiments; shown that there is substantial improvement in performance of Hadoop jobs in the form of reduction in completion time and storage overhead for big data application.

In the future, it is decided to plan proposed system for more general application scenarios.

References

[1] Executive Office of the President " Big Data: Seizing Opportunities, Preserving Values" May 2014.
[2] James Manyika, Michael Chou, Brad Brown, Jacques Bughin, Rihards Dobbs, Charlas Roxburgs, Angela Hung Bayer "Big data: The next frontier for innovation, competition, and productivity" McKinsey Global Institute, May 2011.
[3] Min Chen, Shiwen Mao, Yunhao Liu "Big Data: A Survey" Published online: 22 January 2014 © Springer Science+Business Media New York 2014.
[4] Tom White "Hadoop: The Definitive Guide",Third edition,Oreilly, ISBN: 978-1-449-31152-0.
[5] Venkatesh Nandakumar "Transparent in-memory cache for Hadoop-MapReduce" A thesis submitted in conformity with the requirements for the degree of Master of Applied Science Graduate Department of Electrical and Computer Engineering University of Toronto, 2014.

Licensed Under Creative Commons Attribution CC BY

[6] Avraham Shinnar, David Cunningham, Benjamin Herta, Vijay Saraswat "M3R: Increased Performance for InMemory Hadoop Jobs**",** roceedings of the VLDB Endowment, Vol. 5, No. 12,38th International Conference on Very Large Data Bases, Istanbul, Turkey, August 27th 31st 2012.
[7] Ritu A.Mundada, Aakash.A.Waghmare "Cache Mechanism To Avoid Duplication Of Same Thing In Hadoop System To Speed Up The Extension", IJRET: International Journal of Research in Engineering and Technology, Volume: 03 Issue: 11, Nov-2014.
[8] Sayali Ashok Shivarkar "Speed-up Extension to Hadoop System", International Journal of Engineering Trends and Technology (IJETT), Volume 12 Number 2, Jun 2014.
[9] Dunlu Peng, Kai Duan and Lei Xie "Improving the Performance of Aggregate Queries with Cached Tuples in MapReduce", International Journal of Database Theory and Application Vol. 6, No. 1, February, 2013.
[10] Pramod Bhatotia, Alexander Wieder, Rodrigo Rodrigues, Umut A. Acar, Rafael Pasquini "Incoop: MapReduce for Incremental Computations" Max Planck Institute for Software Systems (MPI-SWS) SOCC'11, Cascais, Portugal, October 27–28, 2011.
[11] Zhao Cao, Shimin Chen, Dongzhe Ma, Jianhua Feng, Min Wang "Efficient and Flexible Index Access in MapReduce" Published in Proc. 17th International Conference on Extending Database Technology (EDBT), Athens, Greece, March 24-28, 2014.
[12] Kenn Slagter, Ching-Hsien Hsu,Yeh-Ching Chung Daqiang Zhang "An improved partitioning mechanism for optimizing massive data analysis using MapReduce" Published online: 11 April 2013 © Springer Science+Business Media New York 2013.
[13] Yaxiong Zhao, Jie Wu, and Cong Liu "Dache: A Data Aware Caching for Big-Data Applications Using the MapReduce Framework" Tsinghua Science and Technology ISSNl l1007-0214l l05/10l lpp39-50 Volume 19, Number 1, February 2014.
[14] Rong Chen, Haibo Chen, and Binyu Zang "Tiled-MapReduce: Optimizing Resource Usages of Data-parallel Applications on Multicore with Tiling" Parallel Processing Institute Fudan University PACT'10, September 11–15, 2010.
[15] Gurmeet Singh, Puneet Chandra and Rashid Tahir "A Dynamic Caching Mechnism for Hadoop using Memcached" Department of Computer Science, University of Illinois at Urbana Champaign.
[16] Chuan Lei, Zhongfang Zhuang, Elke A. Rundensteiner, and Mohamed Y. Eltabakh "Redoop Infrastructure for Recurring Big Data Queries" Worcester Polytechnic Institute, Worcester, MA USA VLDB '14, Hangzhou, China, September 15, 2014.
[17] Ganesh Ananthanarayanan, Ali Ghodsi, AndrewWang, Dhruba Borthakur, Srikanth Kandula, Scott Shenker, Ion Stoica "PACMan: Coordinated Memory Caching for Parallel Jobs" University of California, Berkeley, Facebook, Microsoft Research, KTH/Sweden.
[18] Zhu Xudong, Yin Yang, Liu Zhenjun, and Shao Fang "C-Aware: A Cache Management Algorithm Considering Cache Media Access Characteristic in Cloud Computing", Research Article, Hindawi Publishing Corporation, Mathematical Problems in Engineering, Article ID 867167, 13 pages, Volume 2013.
[19] Meenakshi Shrivastava, Dr. Hans-Peter Bischof "Hadoop-Collaborative Caching in Real Time HDFS" Computer Science, Rochester Institute of Technology, Rochester, NY, USA.
[20] Dachuan Huang, Yang Song, Ramani Routray, Feng Qin "SmartCache: An Optimized MapReduce Implementation of Frequent Itemset Mining" The Ohio State University, IBM Research – Almaden.
[21] Yingyi Bu, Bill Howe, Magdalena Balazinska, Michael D. Ernst "HaLoop: Efficient Iterative Data Processing on Large Clusters" Department of Computer Science and Engineering University of Washington, Seattle, WA, U.S.A. 36th International Conference on Very Large Data Bases, September 1317, 2010, Singapore

Licensed Under Creative Commons Attribution CC BY