IT architecture while big data influences business decision making. The IoT model has huge amount of networking sensors rooted into various devices and machines in the real world. The sensors are deployed in different fields for collecting various kinds of data like environmental data, geographical data, astronomical data, and logistic data. Mobile equipment's, transportation facilities, public facilities, and home appliances could all be data acquisition equipment's in IoT. The data center platform concentrates storage of data and undertakes responsibilities like acquiring data, managing data, organizing data, and leveraging the data values and functions.

The remaining paper is prepared as follows. A brief description of MapReduce, Hadoop and its ecosystem is given in section 2. The literature survey is presented in section 3. The existing system architecture is explained in section 4. Section 5 presents proposed system architecture to optimize computational time and reduce space of storage system.

2. Related work

Google MapReduce is a programming model and a software framework for large scale distributed processing work on enormous amounts of data. Application designers decide the computation in terms of a map and a reduce function, and the original MapReduce job arranging method routinely parallelizes the computation through a cluster of machines. MapReduce accepted widely because of its simple programming interface and excellent performance when executing a huge range of applications. In this model input data is first divided and then supplied to worker systems in the map phase location. Separated items of data are called records. The MapReduce configured machine divides the input parts to each worker and produces records. When record is produced in the map phase, in-between results are shuffled and sorted by the MapReduce system and are then served into the workers in the reduce phase. Last results are calculated by multiple reducers and written to the disk [1]. The design of the MapReduce framework has main principles as Low cost reliable commodity hardware, extremely scalable RAIN cluster, fault tolerant yet easy to administer, highly parallel yet abstracted.

Apache Hadoop is another important technology used to handle big data with its analytics and stream calculating techniques. Apache Hadoop is an open source software project that allows the distributed processing of large data sets through clusters of commodity servers. It can be mounted up from a single server to thousands of machines and with a very high grade of fault tolerance. Instead of trusting on high end hardware, the effectiveness of these clusters comes from the software's ability to sense and handle failures at the application layer. A small Hadoop cluster will include a single master and multiple worker nodes. The master node runs multiple procedures, including a JobTracker and a NameNode. The JobTracker is answerable for managing successively jobs in the Hadoop cluster. The NameNode, on the other hand, manages the HDFS. The JobTracker and the NameNode are usually collocated on the same physical machine. Other servers in the cluster run a TaskTracker and a DataNode processes. A MapReduce job is divided into tasks. Tasks are managed by the TaskTracker. The TaskTrackers and the DataNode are collated on the same servers to provide data locality in computation [1].

Hadoop subprojects or Ecosystem include Hadoop Common which is the common utilities that provision other Hadoop modules. HDFS is a distributed file system that offers high throughput access to application data discussed earlier with some detail. Hadoop YARN is a framework for job scheduling and cluster resource management. Hadoop MapReduce is a YARN-based system for parallel processing of large data sets. Avro is a data serialization system. Cassandra is a scalable multi-master database with no single points of failure. Chukwa is a data collection system for handling big distributed systems. HBase is a scalable, distributed database that provisions structured data storage for large tables. Hive is a data warehouse infrastructure that offers data summarization and ad hoc querying. Mahout is a Scalable machine learning and data mining library. Pig is a high-level data-flow language and execution framework for parallel computation and ZooKeeper is a high-performance management facility for distributed applications.

Distributed file system or network file system allows client nodes to contact files with the help of network developed of computers. Because of this a number of users working on multiple machines are able to share files and storage resources.

The Google File System is planned and applied to see the rapidly growing demands of Google's data handling needs. GFS promises many goals as distributed file systems such as performance, scalability, reliability, and availability. The design looks after component failures rather than the exemption. The file system consists of hundreds or even thousands of storage machines built from cheap service parts and are accessed by a similar number of client machines. The errors are created by application, operating system, human's and disks, memory, connectors, networking, and power supplies. As a result of this continuous observation, bug detection, management of fault tolerance, and programmed recovery are fundamental part of the system. Files are enormous by traditional standards. Multi GB files are common. Most files are changed by appending new data rather than overwriting current data. Accidental writes within a file are virtually does not exist. When data is written, reading of the files is started. A variety of data share these features. Co designing the applications and the file system API benefits the overall system by increasing our flexibility [14].

Hadoop HDFS represents a distributed file system that is designed to house very large amounts of data (TB or PB) and to deliver high-throughput (streaming) access to the data sets. In any IT environment, HW failures do happen. A single HDFS occurrence may contain of thousands of server nodes, each storing and maintaining part of the file system data. Fault detection and rapid, automated recovery features are at the core of the HDFS design architecture. The HDFS design is fundamentally concentrated the pattern write once, read often input output in which a dataset is generated, and where numerous analysis cycles are performed on the

dataset over time. The HDFS design incorporates the notion of a block size, just at a much larger unit size of 64MB (default) compared to other file system solutions [15].

3. Literature Survey

In paper [1] designed a data aware cache framework in which tasks surrenders its intermediate outcomes to the cache manager. A task asks the cache manager before processing the actual computing work. Cache algorithm for customizable indexing of data objects enables the applications to describe their operations and the content of their partial results is written. A cache request and reply protocol is designed. This Dache protocol is aimed to extend the MapReduce framework and made provision of cache layer for efficiently identifying and accessing cache items in a MapReduce task. In Map phase intermediate data produced by worker nodes processes during the execution of a MapReduce task is stored in a Distributed File System (DFS). The content of a cache item is described by the original data and the operations applied. Relationship between job types and cache organization is managed by creating a protocol with the help of two types of cache items as the map cache and the reduce cache.

The paper[2] focuses on MapReduce applications with huge amount of intermediate key value pairs and relatively low amount of computations situation. Here run time is not dominated by application code in map and reduce. The overhead is of library itself on commodity multicore computer. The core challenge is the organization of MapReduce intermediate data. The paper has presented a new MapReduce library called Metis whose intermediate data structure is a hash table with b+ tree in each entry. Tiled-MapReduce (TMR) [3] has extended the general MapReduce programming model with tiling strategy. It is more effective for MapReduce to iteratively process small pieces of data in turn than processing a large pieces of data at one time on shared memory multicore platforms. Tiled-MapReduce presented number of different methods; executed optimizing techniques which are directed on multicore to improve the memory, cache and CPU resources.

In[4] to minimize the gap in disk access time and bandwidth for large cluster based systems, the efforts are made to design a proactive fetching and caching mechanism based on Memcached distributed caching system and integrated with Hadoop. Two Level greedy caching strategy is adopted for cache policy with two ways. The Incoop system [5] is presented as a MapReduce implementation for incremental computations for finding changes on the input datasets. It allows the automatic update of the outputs of the MapReduce jobs by providing a fine-grained output recycle mechanism. It permits incremental processing to be implemented clearly in an incremental manner. The design of Incoop introduced new techniques such as Inc-HDFS (Incremental HDFS) that offers mechanisms to recognize likenesses in the input data of consecutive job runs. Incoop also controls the granularity with new contraction phase. As well as Incoop increases the usefulness of memoization technique by implementing an affinity based scheduler. That applies a work stealing algorithm to minimize the amount of data movement across machines.

In [6] proposed the extension of MapReduce programming model with EFind, an Efficient and Flexible index access method to better support big data applications. EFind collects index statistics and performs cost-based adaptive optimization to improve index access performance. In [7] XTrie and ETrie extended partitioning techniques are developed to increase load balancing for distributed applications. Increasing load balancing helps MapReduce programs become well-organized at handling tasks by reducing the overall computation time consumed processing data on each node. The HaLoop system [8] is designed to make facility for iterative processing on the MapReduce framework. It uses a new task scheduler that leverages data locality. It also caches and indices application data on slave nodes. HaLoop relies on the earlier file system and has the same task queue structure as Hadoop but the task scheduler and task tracker modules are improved whereas loop control, caching, and indexing modules are freshly presented to the architecture. In [9] suggested a distributed storage middleware HyCache to allow input output efficiently influence the high bi section bandwidth of the high speed interconnect of massively parallel high end computing systems. In this also proposed and analyzed new caching technique named as 2 layer scheduling for optimizing network coast and heuristically reduce the disk input/output cost and evaluated system along with caching mechanism at large scale.

In [10] the Redoop infrastructure is demonstrated. Redoop introduces an incremental processing model in which Window Semantic Analyzer is the optimizer, Dynamic Data Packer is the partition executor, Execution Profiler collects the statistics after the completion of each query recurrence, Local Cache Manager installed on each task node in Redoop maintains the Redoop caches on the node's respective local file system and Window Aware Cache Controller is a new module housed on the Redoop master node that maintains window aware metadata of reduce input and output data cached on any of the task nodes' local file systems. The major innovations include adaptivity to load fluctuations, cache based processing and cache aware scheduling.

In [11] PACMan is developed which is an in memory synchronized caching system for data intensive similar jobs. Similar jobs run numerous tasks concurrently in a wave, and have the all-or-nothing property, i.e., a job is zoomed up only when inputs of all similar parallel tasks are cached. On highest of its coordinated arrangement, PACMan implements two cache replacement policies LIFE and LFU-F that are intended to minimize average accomplishment time of jobs and maximize efficiency of the cluster.

Table 1: Comparative Table of Cache Mechanism Variations

Sr. No.	*Paper Name*	*Major Contribution*	*Optimization Type*	*Computational Time*	*Space Overhead*	*Drawback*
1	Dache: A Data Aware Caching for Big-Data Applications Using the MapReduce Framework	Dache, a data-aware cache description scheme, protocol, and architecture	Caching framework	Moderate	Extra	Two way cache
2	Optimizing MapReduce for Multicore Architectures	The "hash+tree" data structure used, library Metis introduced	Data structure	Better	Moderate	Not attractive, doesn't support restarting
3	Tiled-MapReduce: Optimizing Resource Usages of Data-parallel Applications on Multicore with Tiling	Optimizations on memory, cache	Programming model	Speed up 1.2X to 3.3X	85% memory saved	Data locality difficult
4	A Dynamic Caching Mechnism for Hadoop using Memcached	Proactive fetching and caching, Two Level greedy caching strategy	Integration with Hadoop	Better	Average	Increased traffic overhead when added new components
5	Incoop: MapReduce for Incremental Computations	Incremental HDFS, Memoization, Contraction phase and new scheduling algorithm	New architecture	5%-22%	More space	Difficult to manage
6	Efficient and Flexible Index Access in MapReduce	EFind index access solution	Index access interface	2X-8X	adequate	Maintaining indices is difficult
7	An improved partitioning mechanism for optimizing massive data analysis using MapReduce	XTrie and ETrie partitioning techniques	Partitioning mechanism	Moderate	better	Micro partitioning not supported.
8	HaLoop: Efficient Iterative Data Processing on Large Clusters	Loop aware task scheduling, caching for loop invariant data, caching to support fix point evaluation	New programming model and architecture	Speed up by 1.85	4% memory load	A single pipeline in the loop body rather than DAG
9	HyCache+: Towards Scalable High-Performance Caching Middleware for Parallel File Systems	The new caching technique 2 layer scheduling is designed	Distributed storage middleware HyCache+	Speed up by 29X	High	Real time applications are not supported.
10	Redoop Infrastructure for Recurring Big Data Queries	Window aware optimization, cache aware scheduling and inter window caching mechanism	New Infrastructure	Better	Average	Unbounded caches not controlled.
11	PACMan: Coordinated Memory Caching for Parallel Jobs	In-memory synchronized caching system, two cache replacement policies LIFE and LFU-F	Coordinated Infrastructure	Speed up by 53% - 51%	Space efficiency by 47% - 54%	Small jobs not supported
12	An efficient data caching mechanism for big data application *	One way cache implementation and utilization	Performance based optimization	Low	Low	Text based application supported

4. Existing System architecture

Application developers decide the computational work in terms of a map and a reduce function, and the fundamental MapReduce job arranging method routinely parallelizes the computation across a cluster of machines. 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. The first is when the several reducers complete different jobs from the cached reduce cache items of the earlier MapReduce jobs, as shown in Fig. 1.

In this case, after the mappers submit the results gained from the cache items, the MapReduce framework usages the partitioner 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.
The benefit of Dache is that it routinely supports incremental processing. Incremental processing means that an input that

is partially dissimilar or only has a small amount of additional data. To perform earlier operation on this new input data is difficult in conventional MapReduce, because MapReduce does not offer the tools for readily expressing such incremental operations.

Typically the operation needs to be performed again on the new input data, or the application developers need to manually cache the stored intermediate data and pick them up in the incremental processing. In Dache, this process is normalized and prepared. Application developers can easily put their intentions and operations by using cache description and to request intermediate results through the dispatching service of the cache manager.

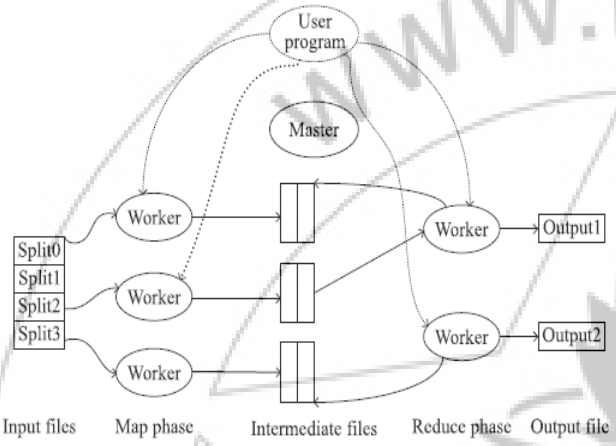

Figure 1: Existing system architecture for Dache

5. Proposed System Architecture

The proposed architecture is shown in Fig.2 along with its functional diagram in Fig.3. Cache supports the intermediate data that is created by worker nodes processes during the execution of a MapReduce task. A part of cached data is then stored in a Distributed File System (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. As an example, the word count application has each mapper node process gives a list of words and count tuples record the count of each word in the file that the mapper processes. The planned system stores such list to a file.

The exact format of the cache description of different applications varies according to their exact purpose which is to be developed and executed by application developers who are responsible for implementing their MapReduce tasks. The supported operations in the system are Item Count which counts of all occurrences of each item in a text file. The items are separated by a user defined separator. Sort is the operation sorts the records of the file. The comparison operator is defined on two items and returns the order of precedence. Selection operation selects an item that meets a given criterion. This may be a sequence in the list of items. The distinctive selection operation contains selecting the average of a linear list of items. Transform operation transforms each item in the input file into a different item. The transformation is described further by the other information in the operation reports and then could only be identified by the site application developers. Classification operation classifies the items in the input file into multiple groups.

5.1 Single Phase Cache Description Scheme

The input provided to the reduce phase is a list in terms of key value combination pairs and the value may be a grouping of values. As stated in the method applied for the map phase cache mechanism, the first unique input and the applied operations are needed. The unique input is acquired by storing the intermediate results of the map phase in the distributed file system. The functionally applicable operations are recognized by unique identifications that are stated by the user in the cache. The cached results, unlike those generated in the Map phase, are not be straight forwardly applied in the final outcome since, in appended slight incremental processing, in-between results generated in the Map phase are likely combined in the provided shuffling phase creates a mismatch between the first unique original input of the cache items and the newly generated input. The reducers can identify new inputs from the shuffling sources by shuffling the newly generated intermediate result from the Map phase to form the final results.

If a reducer might combine the cached fractional results with the results gained from the new inputs and considerably minimize the overall processing time, reducers might cache fractional results. Basically this stuff is determined by the operations executed by the reducers.

The Cache manager that is enhanced third party cache utility which handles data from the reducer. The HDFS gets job task from the application and submits back to the cache manager. The job is then stored with unique identity value which is unique in nature in the cache manager. This manager proposes the output which is predefined in the manager if already queried and then omits the mapper phase thus reducing the time complexity.

The manager even checks the duplication of the job which is submitted to the system and thus produces unique timeless output, even for batch input. If queried by the manager doesn't exist within the cache system mappers are invoked for completing the job and the output is again generated for the system. The intermediate output of the system is thus stored with unique value in the cache manager for further retrieval. This data is stored for re enforced for further quires, so as when a duplicate relevant job is submitted the pre output from the cache manager is submitted.

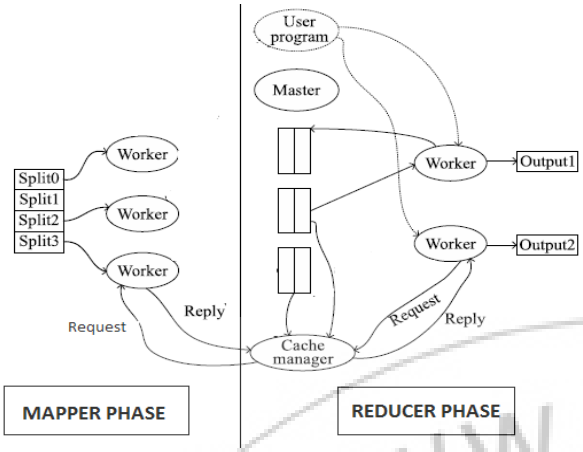

Figure 2: Proposed System Architecture.

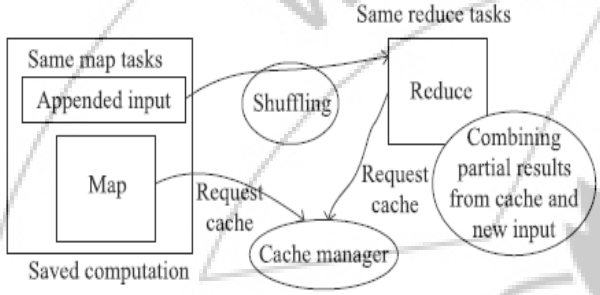

Figure 3: Functional diagram of proposed system.

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 ISSNl l1007-0214l l05/10l lpp39-50 Volume 19, Number 1, February 2014.

[2] Yandong Mao Robert Morris M. Frans Kaashoek "Optimizing MapReduce for Multicore Architectures" Massachusetts Institute of Technology, Cambridge, MA.

[3] 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.

[4] 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.

[5] 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, October 27–28, 2011, Cascais, Portugal.

[6] Zhao Cao, Shimin Chen, Dongzhe Ma, Jianhua Feng, Min Wang "Efficient and Flexible Index Access in MapReduce" (c) 2014, Copyright is with the authors. Published in Proc. 17th International Conference on Extending Database Technology (EDBT), March 24-28, 2014, Athens, Greece.

[7] 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.

[8] 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.

[9] Dongfang Zhao, Kan Qiao, Ioan Raicu "HyCache+: Towards Scalable High-Performance Caching Middleware for Parallel File Systems" Illinois Institute of Technology, USA and Argonne National Laboratory, USA.

[10] 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, September 15, 2014, Hangzhou, China.

[11] 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.

[12] 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.

[13] Min Chen · Shiwen Mao · Yunhao Liu "Big Data: A Survey" Published online: 22 January 2014 © Springer Science+Business Media New York 2014.

[14] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung "The Google File System" SOSP'03, October 19–22, 2003, Bolton Landing, New York, USA.

[15] Dominique A. Heger " Hadoop Design, Architecture & MapReduce Performance" DHTechnologies - www.dhtusa.com