Hadoop Distributed File System for Load Balancing Using Data Declustering Techniques

Devwrat Kumar, Chaudhari Mayur, Joshi Piyush, Kuchekar Vikash

***Abstract*— The Big-data refers to the large-scale distributed data processing applications that operate on unusually huge amounts of data. Google's MapReduce and Apache's MapReduce,its open-source implementation, are the defacto software systems for Large Scale data applications. Study of the MapReduce framework is that the framework produces a large amount of intermediatedata. Such existing information is thrown away after the tasks finish, because MapReduce is not able to utilize them. In this paper, we propose, a data-aware cache framework for large data applications. In this paper, tasks submit their intermediate results to the cache manager. A job queries the cache manager before executing the actual evaluation work. A novel cache depiction scheme and a cache request and reply protocols are designed. We implement Data aware caching by extending Hadoop.**

***Index Terms*— BigData, Hadoop, JobTracker, MapReduce, TaskTracker.**

I. INTRODUCTION

MapReduce is a programming model and a software framework for Large -scale distributed Evaluation on huge amounts of data. Figure 1 represents the highlevel work flow of a MapReduce Task. Application developers specify the evaluation in terms of a map and a reduce function, and the underlying MapReduce Task enrolling system automatically parallelizes the computation across a clusters. MapReduce is popular for its simple programming interface and excellent interpretation when implementing a large spectrum of applications. Since most such applications take a huge amount of input data, known as "Bigdata applications".

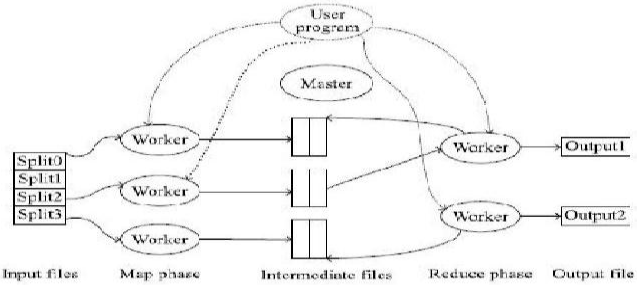

Figure 1: The MapReduce programming model architecture.

as shown in Figure 1, input data is first splitted and then given to workers in the map stage. separate data items are called records. The MapReduce system parses the input chunks to each worker and output the records. After the map phase, intermediate overcome generated in the map phases are stumbled and arranged by the MapReduce system and are then given into the reduce phase to workers. Final result is evaluated by multiple reducers and written back to the disk.

Hadoop is an open source software framework of the Google MapReduce programming model. Hadoop consist of the Hadoop Common, which makes availaible access to the file systems supported by Hadoop. Hadoop Distributed File System (HDFS) provides distributed file storage and is optimized for huge unalterable chunk of data. A small Hadoop cluster will contain a single master and multiple worker nodes called as slave. The master node runs various processes, including a TaskTracker and a Name Node. The TaskTracker is having authority for control on running jobs in the Hadoop cluster. Whereas Name Node handles the HDFS. The TaskTracker and the Name Node are normally collected on the same physical machine. different servers in the cluster execute a Task Tracker and a Data Node processes. A MapReduce job is splitted into tasks. Tasks are controlled by the TaskTracker. The Task Tracker and the DataNode are collected on the same servers to make availaible data locality in evaluation. MapReduce makes availaible a standardized framework for achieveing large-scale distributed computation, known as, the big-data applications.

Still, there is a limitations of the system, i.e., the inability in graditional processing. Graditional processing refers to the applications that expansionally promote the input data and regularly apply evaluations on the input in order to achieve output. There are probable duplicate evaluations and operations being performed in this process. However, MapReduce does not have the any other technique to identify such replicate evaluations and accelerate job execution. provoked by this conclusion, In this paper we advance, a data-aware cache system for bigdata applications using the MapReduce framework, which desire at enlarging the MapReduce framework and supplying a cache layer for efficiently identifying and accessing cache elements in a MapReduce job

II. LITERATURE REVIEW

1. Large-scale Incremental Processing Using Distributed Transactions and Notifications [3]

Daniel Peng et al. recommended, a system for additionally processing renovate to a bulky data set, and expandd it to create the Google web search index. By renewing a batchbased indexing system with an indexing system depend on incremental processing flow, Auther process the equal number of records per day.

2. Design and Evaluation of Network-Leviated Merge for

Manuscript received February 24, 2015.

Devwrat Kumar, Devwrat Kumar pursing engineering in computer science from PUNE UNIVERSITY.

Chaudhari Mayur, Chaudhari Mayur pursing engineering in computer science from PUNE UNIVERSITY (Jspm's Jscoe).

Joshi Piyush, Joshi Piyush pursing engineering in computer science from PUNE UNIVERSITY (Jspm's Jscoe).

Kuchekar Vikash, Kuchekar vikash pursing engineering in computer science from PUNE UNIVERSITY (Jspm's Jscoe)

Hadoop Acceleration [7]
Weikuan Yu et al. presented, Hadoop-A, an acceleration framework that enhances Hadoop with plugin components for rapid data movement, overcoming the existing restrictions. A novel network-levitated merge algorithm is planned to merge data without duplication and disk access. In addition, a full pipeline is arranged to overlap the stumble, merge and degrade phases. Our experimental overcome shows that Hadoop-A significantly fast up data movement in MapReduce and increases the output of Hadoop

3. Improving Mapreduce Performance through Data Placement in Composite Hadoop Cluster [5]
Jiong Xie et al. presented that ignoring the data locality problems in composite environments can noticeably reduce the MapReduce output. In this paper, author addresses the issues of how to put data across nodes in a way that every node has a stable data processing load. Given a data explosive application running on a Hadoop MapReduce cluster, our data placement method adaptively increases the amount of data stored in every node to achieve high-level data-processing enhancement. Experimental outcome on two real data-intensive applications show that our data placement scheme can always enrich the MapReduce performance by rebalancing data beyond nodes before performing a data-enhance application in a composite Hadoop cluster.

4. Improving MapReduce Performance in Composite Network Environments and Resource Utilization [6]
Zhenhua Guo et al. proposed, Benefit Aware hypotthetical fulfillment which predicts the benefit of launching new hypothetical tasks and greatly removes unnecessary runs of hypothetical tasks. Finally, MapReduce is mainly improved for consist environments and its inproficient in composite network environments has been examined in their experiments. Authors examine network heterogeneity aware planning of both map and reduce tasks. Overall, the goal is to enhance Hadoop to handle with significant system heterogeneity and advance resource utilization motivation: MapReduce gives a standardized framework for implementing big-scale distributed evaluation,known as, the big-data applications. However, there is a inhibition of the system, i.e., the inefficiency in incremental processing. Incremental processing refers to the applications that additionally grows the input data and normally assign evaluations on the input in order to generate output.There are hypothetical duplicate evaluations being performed in this process. However, MapReduce don't have the technique to identify such duplicate evaluations and accelerate Task execution. inspired by this observation, in this paper we present, a data-aware cache network for big-data applications using the MapReduce framework, which goals at extending the MapReduce framework and makes availaible a cache layer for simply identifying and accessing cache elements in a MapReduce job

III. NEED

3.1 Cache Description:
Data-aware caching requires all data object to be indexed by its gratified. In the context of huge scale data applications, this means that the data contents and the cache description scheme must narrate the application framework. Although most of the big-data applications execute on standardized platforms, their personal tasks perform efficiently distinct operations and produce several intermediate results. The cache description method should gives a customizable indexing that empowers content of their produced biased results. This is a untrivial task. In the situations of Hadoop, It uses the purify proficiency makes availaible by the Java language to recoganise the object that is utilized by the MapReduce to growth the input data.

3.2 Cache request and reply protocol:
The amount of intermediate data can be very huge. When such data is demanded by other worker nodes or slave, deciding how to transport this data turns in to very tangled. Simply for processing, programs are moved in to the data node i.e. slave node to run the processing locally. Although, this may not always happen be applicable since the partiality of the worker nodes may not be redially changed. To clarify Data locality is

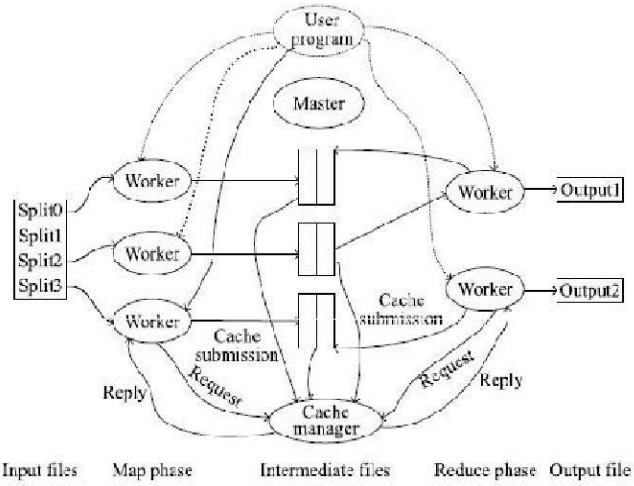

Figure2: High-Level Description Architecture Of The Data Aware Caching

To clarify Data locality issues, the protocol should be able to collect cache elements with the worker processes likely that demand the data, so that the channeling delay and overhead are reduces. In this paper, we propose a novel cache explanation scheme. A high-level explanation is presented in Figure 2.

This scheme analyzes the source input from which a cache member is acquired, and the actions applied on the input, so that a cache elements produced by the workers in the map phase is indexed efficiently. In the reduce phase, we design a another technique to take into consideration the minimal actions are applied on the output in the map phase,also represent a method for reducers to exploit the cached overcome in the map phase to increase the performance of the MapReduce job. We untensil data aware caching in the Hadoop project by advancing the compatible components. Our execution follows a non-invasive approach, so it only requires minimal changes to the application code

IV. MAP PHASE CACHE DESCRIPTION SCHEME

Cache relates to the intermediate data that is formed by worker nodes through the execution of a MapReduce task. A member of cached data is saved in a Distributed File System (DFS). The gratified of a cache members is defined by the actual data and the actions applied. Formally, a cache member is explained by a 2-tuple: fOrigin, Operation. Origin act as same name of a file in the DFS.

Operation is a linear list of available jobs performed on the Origin file. For example,the word count problem, all mapper node/process release a list of fword, countg tuples that record the count of each word in the file that the mapper processes. Data aware caching stores this list to a file. This file becomes a cache elements. Given an actual input data file, word list 013546780.txt, the cache element is explained by fword list 0135467800.txt, elements countg. Here, element invoke to white-space-separated character strings.The new line character is also judged as one of the white spaces, so element specifically put the word in a text file and element count directly agree to the word count operation acheived on the data file. The actual format of the cache explation of several applications varies to their specific semantic contexts accordingly. This can be designed and completed by application developers who are pledged for acheiving their MapReduce tasks. In our model, we present various supported operations:

_ Item Count: This functionality used to calculate of all existance of each item in a file. The items are
separated by a user defined separator.

_ Sort: This functionality sorts the records of the file. The comparison operator is explained on two elements and returns the sequence of priority.

_ Selection: This functionality picks an elements that meets a given criterion. It could be an sequence in the list of elements. A selection action involves choosing the median of a linear list of the given elements.

_ Transform: This functionality transform all elements of the input file into a another format. The transformation is defined further by the other information in the functional explation. This can only be makes availaible by the application developers.

_ Categorization: This functionality utilized to classify the elements in the input file into multiple groups. This can be an exact categorization, where a deterministic categorization criterion is applied sequentially on every elements, or an approximate categorization, where an iterative categorization process is enforced and the iteration count should be recorded. Cache explanation can be recursive. For example,in sequential processing, a data file could be processed by various worker processes.then a cache elements, produced by the final process, could be from of the intermediate overcome files of previous worker nodes,also its explanation will be gathered together to form a recursive explation. Whereas this recursive explanation could be developed to an iterative one by directly attaching the later functionalites to the older ones. Still, this iterative explanation loses the context information about the later functionalities, if another process is functioning on a later cache elements and is looking for possible cache that could save its own functionalities. By inspecting an iterative explanation, one cannot differentiate among a later on cache elements and a previous one because the origin of the cache elements is the one that was fed by the application developers.The worker processes will not be able to identify the correct cache elements, even if the cache elements is present in cache manager

V. REDUCE PHASE CACHE EXPLATION SCHEME:

The input for the reduce phase is a pairs of key-value,where the value can be a list of values.The scheme utilize for the map phase cache explanation, the actual input and the applied actions are required. The actual input element is restored by storing the intermediate overcome of the map phase in the DFS. The implemented jobs are recoganized by unique IDs that are specified by the users. The cached results, dissimilar those are generated in the Map phase, can't be used as the final throughput. This is because of an additional method, intermediate results are formed in the Map phase are combined in the shuffle phase, which causes a mismatch among the actual input and the currently generated input. A solution of this problem is apply a minimal explanation of the actual input in the reduce stage. The explanation should include the actual data files generated in the Map stage. For example, two data files, “fileX.data” and “fileY.data”, are shuffled to produce two input files, “inputX.data” and “inputY.data”,for two reducers. “inputX.data” and “inputY.data” should include “fileX.data” and “fileY.data” as its shuffling source. As a result, new intermediate data files of the Map phase are formed during additional processing; the shuffling input will be recoganized in a same way. The reducers can recognize new inputs from the shuffling sources by shuffling the currently-generated intermediate overcome from the Map phase to form the final results. For example, assume that “inputZ.data” is currently generated results from Map phase; the shuffling results “fileX.data” and “fileY.data” includes a new shuffling source, “inputZ.data”. A reducer can recoganize the input “fileX.data” as the result of shuffling “inputX.data”, “inputY.data”, and “inputZ.data”. The final results of shuffling the output of “inputX.data” and “inputY.data” are obtained by querying the cache manager. The appended shuffling output of “inputZ.data” is then appended to get the new results. Given the above explanation, the input given to the reducers is not cached wholly. Only a some part of the input is same to the input of the cache elements. The remaining is from the product of the additional processing phase of the map phase. If a reducer can combine the cached partial results with the results aquired from the new inputs and substantially reduce the overall evaluation time, reducers should cache partial results. This property is examined by the jobs executed by the reducers.

VI. HADOOP MAP-REDUCE

6.1 Apache Hadoop

Apache Hadoop is an open-source software platform for storage and handling of large-scale data-sets on clusters of commodity hardware. 'Map-Reduce' is a framework for handling parallelizable issues across large datasets using a large number of nodes, collectively called as a grid or a cluster . Evaluation processing can appear on data stored either in unstructured or in a structured database. Map-Reduce can take favored of data locality, processing it on or adjacent the storage assets in order to reduce the distance of transmitted. "Map" step: Input is given to the master node. Master node divides it into smaller

sub-problems and then distributes them to worker nodes. If required A worker node may again further sub-divide it which leads to a multi-level tree structure. The worker node processes the smaller sub-problem, and gives the response to its master node. "Reduce" step: The master node accumulates the overcome of all the sub-problems and merge them to form the output which is the result to the actual problem it was trying to solve. Map Reduce acknowledges for distributed processing of the map and reduction jobs, where mapping functionality is independent of the others, all maps can be performed in parallel. Similarly, a set of 'reducers' can perform the reduction phase, if and only if all outputs of the map job that share the same key with to the same reducer at the same time.Bigger dataset is used in Map Reduce, commodity server handle peta byte of data in few hours. If one mapper or reducer fails, then rescheduled is used to suppose the input data is still available.

6.2 Task Tracker: the Map-Reduce engine

The Map-Reduce engine contains one JobTracker, to which client applications submit MapReduce jobs. The JobTracker transmits work request to number of TaskTracker nodes in the group of nodes. works are process near to worker node to achieve high data locality. A rack-aware file system is used, the JobTracker maintain information about node which contains the data, and which is nearby machines. If the work can't be performed on the actual node where the data resides, priority is given to nodes in the same rack in rack aware file system. This reduces network traffic on the main backbone network.Task is rescheduled if the tasktracker fails or times out. To check the status of TaskTracker, TaskTracker send a heartbeat to the JobTracker in every few minutes.

6.3 Map cache:

Apache Hadoop is an open-source application of originally started by Google, is the MapReduce distributed parallel processing framework. In Map phase input is splitted into multiple file splits which are processed by an equal number of Map worker process, who achieve a data-parallel processing procedure. As explained in Figure. 3, a file splited according to users specification.

999

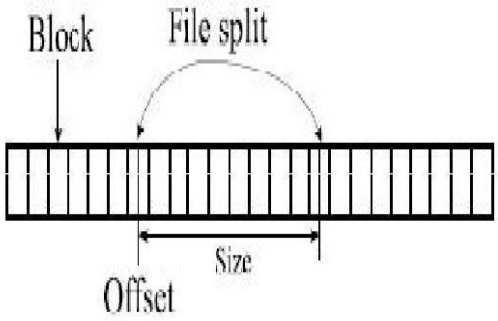

Figure 3: Map Phase A file in a DFS.

The intermediate results obtained by processing file splits are then cached. Each file split is identified by the original file offset,size and name. This originates complications in describing cache members. Further this scheme is slightly altered to work for the general situation, In which The original field of a cache item is changed to a 3-tuple of file name, offset, size. A file split cannot cross file borders in Hadoop MapReduce, which simplifies the explanation scheme of cache elements. Map cache elements can be aggregated by grouping file splits. Original input file generate Multiple cache members from the same original file in the DFS are grouped under the path of the original file, i.e. offset,file name, sizeg.Using this approach it optimize the actual storage of aggregated cache items .So Map cached item can be placed on a single data node in the HDFS cluster which refrain costly queries to multiple data nodes

6.4 Reduce cache

Cache depiction contain the file splits from the map phase. The input given to the reducers is from the whole input of the MapReduce job. thus, further clarify the description by applying the file name with a version number to describe the original file to the reducers. The version number of the input file is used to differentiate incremental changes to input file. A genuine approach is to encrypt the size of the input file is included with the file name. Since incremental changes,attaching new data at the end of the file.the file size is enough to recognize the changes made during different MapReduce jobs. Note that even the entire output of the input files of a MapReduce Task is used in the reduce phase, splited file can still be aggregated, i.e., by using the form of ffile name, split, splitg. As shown in Figure. 4,

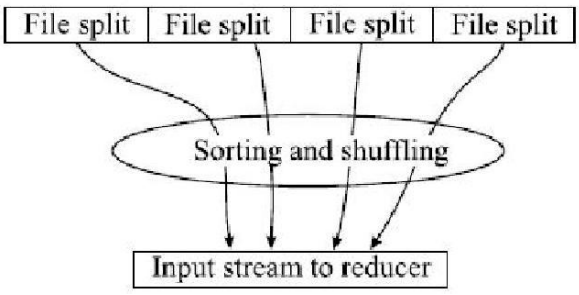

Figure 4:Architecture Of Reducer.

input for the reducers is produced by file spliting, sorting and shufflng. fundamentally this process is implicitlycontrolled by the MapReduce framework, the users specify a shuffling method by supplying a partitioner, which is enforced as a Java object in Hadoop.

VII. PROTOCOL

7.1 Relationship among Task types and cache organization

The fractional results generated in the map and reduce phases can be used in different scenarios. There are two kind of cache items: the map cache and the reduce cache. They have different types of complexities under different scenarios.Cache items in the map phase are very easy to share because the operations applied are well-designed. When processing each file split, the cache manager reports the previous file splitting scheme used in its cache elements. The new

MapReduce Task urgency to split the files according to the same splitting scheme in way to utilize the cache items. However, if the new MapReduce Taskuses different file

splitting scheme, the map results cannot be used directly,lest the operations applied in the map phase are context
free. By context free, we mean that the operation only produces results based on the input records which does not consider the file split scheme. This is normally true. When considering cache sharing in the reduce phase, we identify two normal situations. The first when the reducers complete different jobs from the cached reduce cache items of the earlier MapReduce jobs are shown in Figure. 5. In this case, after the mappers submit the results obtained from the cache items, the MapReduce framework uses the practitioner handover by the new MapReduce
Taskto feed input to the reducers. The saved evaluation is obtained by discarding the processing in the Map phase.Usually, new content is added at the end of the input files, which requires further mappers to process.However, this does not require supplementry processes other than those introduced above.
The second condition is when the reducers can actually take benifit of the previously-cached reducing cache items as illustrated in Figure. 6. Using those description scheme, the reducers determine how the output of the map phase is shuffled. The cache manager automatically identifies the best-matched cache elements to feed each reducer, which is the one with the maximum overlap in the genuine input file in the Map phase.

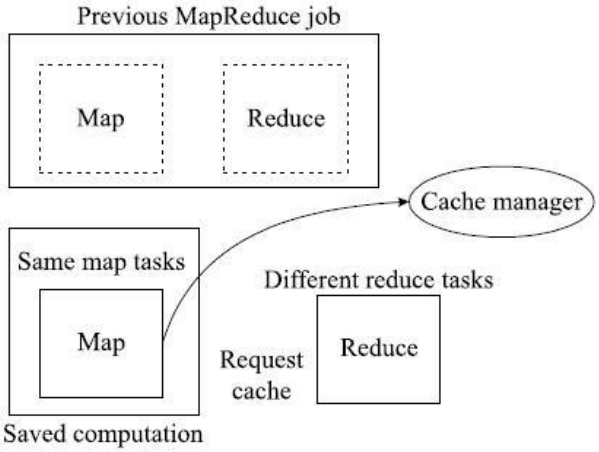

Figure 5: Map with same map task and different reduce tasks

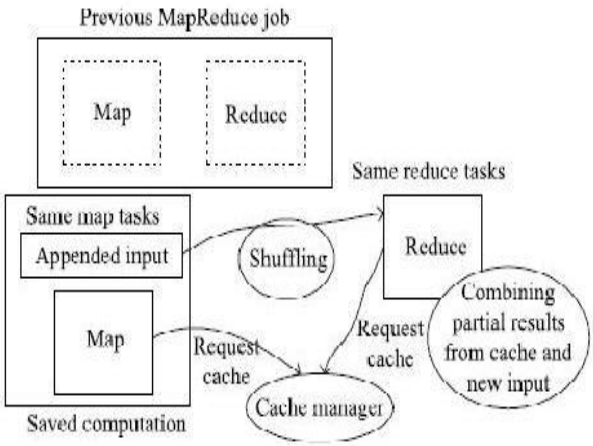

Figure 6: The situation where two MapReduce jobs have the same map and reduce tasks.

7.2 Cache item submission

Mapper and reducer processes record cache items into their local storage space. When these operations are completed, the cache items are forwarded to the cache manager, which acts like a broker in the publish/subscribe paradigm. The cache manager records the description and the file name of the cache item in the DFS. The cache element should be bring on the same machine as the worker process that generates it. This requirement improves data locality.The cache manager maintains a replica of the mapping among the cache descriptions and the file names of the cache elements in its main **memory** to accommodate fastest reply to queries. It also takes backup of the mapping file into the disk periodically to escape permanently losing data.A worker process contacts the cache manager eachtime before it begins processing an input data file. Theworker process sends the file name and the procedures that it plans to apply to the file to the cache manager. The cache manager get this message and compares it with the stored mapping data. If there is a exact match to a cache item, i.e., its origin is the same as the file name of the request and its procedures are the same as the proposed operations that will be performed on the data files, then the manager will send back a reply containing the temporary description of the cache item to the worker process. The worker process receives the temporary description and fetches the cache item. For more processing, the worker urgency to send the file to the next-stage worker processes.The mapper needs to notify the cache manager that it already processed the input file splits for this job. Then the cache manager reports these results to the next phase reducers. If the reducer do not utilize the cache service, the output in the map phase can be directly shuffled toform of the input for the reducers. Otherwise, more entangled process is executed to obtain the required cache items; this will be explained in next part. If the
proposed procedures are different from the cache items in the manager's records, there are situations where the source of the cache item is the same as the requested file,and the operations of the cache item are a rigorous subset of the proposed operations. The concept of a strict super set indicate to the fact that the item is obtained by applying some additional operations on the subset item. For example, an item count operation is a strict subset operation of an item count obeyed by a selection operation. This case means that if we have a cache item for the first procedure, we could just add the selection operation, which guarantees the correctness of the
operation. One of the benefits of Data aware caching is that it automatically supports the incremental processing.Incremental processing indicates that we have an input that is partially different or only has a less amount of additional data. To perform a previous operation on this new input data is troublesome in traditional MapReduce, because MapReduce does not provide the tools for eagerly expressing such incremental operations. Usually the operation needs to be executed again on the new input data, or the application developers need to manually cache the saved intermediate data and pick them up in the incremental processing. In Data aware caching, process is standardized and specified. Application developers have power to express their intentions and operations by using cache explanation and to request

intermediate outcome through the dispatching service of the cache manager.

7.2.1 Lifetime management of cache item:

The cache manager needs to compute how much time a cache item could be kept in the DFS. Holding a cache elements for an undefined amount of time will waste storage space when no another MapReduce task utilizing the intermediate results of the cache item. There are two types of schemes for determining the lifetime of a cache element, as listed below. The cache manager also can promote a cache item to a permanent file and store it in the DFS, which happens when the cache elements is used as the final result of a MapReduce task. In this case, the lifetime of the cache item is no longer managed by the cache manager. The cache manager still maintains the mapping among cache explanations and the actual storage location.

7.2.2 Fixed storage quota:
Data aware caching allocates a fixed volume of storage space for storing cache items. Old cache items need to be discarded when there is no enough storage space for storing new cache items. The removal policy of old cache items can be shaped as a classic cache replacement problem. In this paper preliminary implementation, the Least Recent Used (LRU) is occupied. The cost of allocating a fixed storage quota can be determined by a pricing model that captures the budgetary expense of using that amount of storage space. Such pricing models are available in a public Cloud service.

7.2.3 Optimal utility:
Increasing the storage space of cache elements, a utility-based measurement can be used to determine by an optimal space allocated for cache items which maximize the advantage of Data aware caching and tribute the constraints of costs.This approch estimates the saved evaluation time, ts, by caching a cache elements for given amount of time, ta. These two variables are used to derive the budegtary gain and cost. The net profit, i.e., the difference of subtracting cost from gain, should be formed positive. To achieve this, an accurate pricing model of evaluational resources is required. Although traditional computing infrastructures do not offer such a model, cloud computing offer. Budgetary values of computational resources are well captured in existing cloud computing services, for example, in Google Compute Engine and Amazon AWS.For severl organizations that rely on a cloud service provider for their IT infrastructure, that would be a perfect model. According to the official report from AmazonAWS, the amount of organizations that are actively using their services is huge, which help them to achieve near billion dollar revenue. Therefore, this cost model should be very useful in real-world utilization. On the other hand, for organizations that rely on their own private IT infrastructure, this model will be inaccurate and should only be used as a reference.
Expensets= Pstorage × Scache × ts(1)
Savets = Pcomputation × Rduplicate × ts(2)
Equations (1) and (2) show how to compute the expense of storing cache and the corresponding stored expense in computation. The details of computing the variables introduced above are as follows. The profit of storing a cache item for ts amount of time is computed by accumulating the charged expenses of all the stored computation tasks in ts. The number of the same task that is submitted by the user in ts is estimated by an exponential distribution. The mean of this exponential distribution is obtained by sampling in history. A freshly generated cache elements requires a bootstrap time to do the sampling. The cost is directly computed from the charge expense of storing the item for ta amount of time. The optimal lifetime of a cache item is the greatest ta, such that the profit is positive. The overall benefits of this scheme are that the user will not be charged more and at the same time the computation time is decreased, which in turn decreases the response time and increases the user satisfaction.

7.3 Cache request and reply

7.3.1 Map cache:
There are distinct complications that are caused by the actual designs of the Hadoop MapReduce framework. The first is, when do map phase issue cache requests? As described above, map cache items are identified by the data chunk and operations performed. In order to protect the original splitting scheme, cache requests must be sent out before the file splitting phase. The jobtracker, which is the significant controller that manages a MapReduce job,issues cache requests to the cache manager. The cache manager replies a list of cache explanations. Then the jobtracker splits the input file on remaining file section that have no corresponding outcomes in the cache items. That is,the jobtracker urge to use the same file split scheme as the one used in the cache elements in order to literaly utilize them. In this scenario, the new appended input file should be split among the same number of map phase tasks, so that it will not quiet slow the entire MapReduce Task down.Then their results are combined together to form an aggregated Map cache item; to achive this nested results MapReduce job is used.

7.3.2 Reduce cache:
The cache request process is more entangled. The first step is to examine the requested cache item with the cached items in the cache manager's database. The cached results in the decrease phase may not be directly used due to the incremental changes. As a outcome, the cache manager needs to recognize the overlaps of the original input files of the requested cache and stored cache. In our initial implementation, this is done by performing a linear scan of the stored cache members to find the one with the maximum overlap with the request. While comparing the request and cache item, the cache manager first identifies the petitioner. The practitioner in the request and the cache item has to be identical, i.e., they should use the same number of reducers and same partitioning algorithm .This requirement is illustrated in Figure. 7. The overlapped part is mechanism that a part of the processing in the reducer could be saved by aquiring the cached results for that part of the input. The incremented part, still, will need to be processed by the reducer itself. The final conclusion are generated by merging both parts. The actual method of merging results is determined by the user.

VIII. CONCLUSIONS

This paper present shows the design and evaluation of a data aware cache framework that requires minimal change to the actual MapReduce programming model for arranging incremental processing for Big data applications using the MapReduce model. This paper propose, a data-aware cache description scheme, protocol, and architecture. This Paper Presented 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 Data in data aware caching. This paper appliance it in Hadoop by extending relevant components. In the future, we plan to adapt our framework to more general application scenarios and appliance the scheme in the Hadoop project.

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 ISSN 1007-0214 05/10 Volume 19, Number 1, pp 39- 50, February 2014

[2] Hadoop, http://hadoop.apache.org/,2013

[3] D. Peng and f. Dabek,"Large Scale incremental Processing using distributed Transaction and notification", in Proc. of OSDI'2010, Berkeley, CA, USA, 2010

[4] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving Mapreduce performance in heterogeneous environments",in Proc. of OSDI' 2008, Berkeley, CA, USA, 2008

[5] Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, "Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters",

[6] Amawon web services, http://aws.amazon.com/, 2013.

[7] Google compute engine, http://cloud.google.com/ Products/computeengine.html, 2013

AUTHORS:

Devwrat Kumar, Devwrat Kumar pursing engineering in computer science from PUNE UNIVERSITY.

Chaudhari Mayur, Chaudhari Mayur pursing engineering in computer science from PUNE UNIVERSITY (Jspm's Jscoe).

Joshi Piyush, Joshi Piyush pursing engineering in computer science from PUNE UNIVERSITY (Jspm's Jscoe).

Kuchekar Vikash, Kuchekar vikash pursing engineering in computer science from PUNE UNIVERSITY (Jspm's Jscoe)