ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 4, October 2013

Sharded Parallel Mapreduce in Mongodb for Online Aggregation

B Rama Mohan Rao,
Dept. of CSE, JNTU College of Engineering,
JNT University, Hyderabad

A Govardhan,
Professor, School of Information Technology,
JNT University, Hyderabad

Abstract*—The Online Aggregation framework proposed to obtain the approximate results for the complex queries more quickly when compared to exact results using the aggregation. The Map Reduce context has evolved as one of the most commonly used parallel computing platforms for processing of large databases. It is one of the widespread programming model for handling of large datasets in parallel with a cluster of machines. This Paradigm permits for easy parallelization on several machines of data computations. The Online Aggregation combined with Map Reduce jobs to improve the performance of Query processing in large databases and to obtain the approximate Results. Sharding is the method of storing data records across multiple machines and is one of the MongoDB's methodology to encounter the demands of increased data set gradually. To minimize the time taken to execute very large database using map and reduce functions shards used for the implementation. This paper proposes a new methodology to improve performance of the online aggregation known as Sharded parallel MapReduce in MongoDB for Online Aggregation to obtain the approximate results in less time compared to the traditional MapReduce paradigm.

I. INTRODUCTION

The capability of generating and gathering a huge amount of data sets for various applications are increasing extremely over the last several years. The users are gradually analyzing these massive datasets using various large parallel database management systems and other parallel data processing infrastructures. Even though several management systems are dramatically increasing the processing speed of the queries significantly in order to obtain the fast response, the queries are still taking millions of hours to process the large input data. Since the individual queries are taking more time to complete the execution, the user is more interested to obtain only the accurate feedback regarding the query execution status [1]. Thus, the accurate feedback for the complex data query processing systems produced exclusively with the combination of MapReduce and Online Aggregation. Online Aggregation proposed by Hellerstein [2] is a methodology that facilitates the users to provide the approximate results to the complex queries more quickly when compared to the exact results. This framework is proposed to obtain the approximate results for the queries using the aggregation where the database is scanned in random order and the results of the aggregate query is updated eventually as the scan proceeds. It is a technique for improving the interactive behavior of database systems processing with expensive analytical queries. This system performs the aggregation query in the online fashion. The basic method of online aggregation is to sample tuples from the input relations and calculate a repeatedly filtering running estimate of the result, along with a "confidence interval" to specify the accuracy of the estimated result. These confidence intervals classically displayed as error bars in a graphical user interface. The precision of the estimated result increases as more and more input tuples handled. In this system, users can both observe the progress of their aggregation queries and control execution of these queries on the fly. It allows interactive exploration of large, complex datasets stored in relational database management systems. Online aggregation interface is a more flexible and satisfying mechanism for data exploration than traditional batch-mode query processing. The Map Reduce is a programming model and a framework for data-intensive distributed computing of batch jobs. MapReduce has emerged as a popular way to harness the power of large clusters of computers. MapReduce allows programmers to think in a data-centric fashion. They focus on applying transformations to sets of data records, allow the details of distributed execution, network communication, coordination, and fault tolerance to handle by the MapReduce framework. To simplify fault tolerance, the output of each Map Reduce task and job materialized to disk before consumed. This framework has evolved as one of the most broadly used parallel computing model for processing on terabytes and petabytes of data set in these recent years. The MapReduce programming model originally designed not only for the batch-oriented systems but also for interactive data analysis. This tendency has enhanced the development of high level query languages that are implemented as MapReduce jobs, such as Hive [3],Pig [4], Microsoft Dryad [5] and Sawzall [6].The Traditional MapReduce express computation as a series of jobs where inputs are the list of records (key-value pairs). The map function is utilized to produce intermediate key-value pairs and reduces function is utilized to call for each distinct key in the map output. The MapReduce paradigms perform both the map and reduce functions parallel to ensure the fault tolerance. The cloud computing community as a support to those cloud-based applications that are data-intensive has also adopted the MapReduce paradigm. The MapReduce Programming model used in a wide variety of

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 4, October 2013

applications and belonging to numerous domains such as analytics, data processing, image processing, machine learning, bio-informatics, astrophysics, etc. One of the significant key element of the MapReduce paradigm is that it is different from previous models of parallel computation as it includes the sequential and parallel computation together. MapReduce is very well suitable for raw parallelization. The MapReduce programming model permits data to pipeline between operators, permits the data to run parallel between the operators, provisions continuous queries, preserves the fault tolerance properties of Hadoop and can run unmodified user-defined MapReduce programs [16]. The Organization of this proposed paper done in this way. Section 1 already discussed about the Introduction for the paper. Section 2 gives the brief discussion on the improvements in the parallel Map Reduce, Section 3 gives the brief discussion about parallelism in MapReduce, Section 4 gives the brief discussion on the proposed methodology, and its implementation, Section 5 discusses about the observed results and its analysis. Section 6 concludes the paper followed by References and Acknowledgement of the paper given in Section 7 and Section 8.

II. RELATED WORK

The MapReduce framework originally developed at Google [7], but has recently seen wide adoption and has become the de facto standard for large-scale data analysis. Publicly available statistics indicate that MapReduce used to process more than 10 petabytes of information per day at Google alone [8]. MapReduce [7] (with its open-source variant Hadoop [9]) is a programming model that used for the processing and implementation of large massive-scale datasets. The Amazon released Elastic MapReduce [10], a web service that facilitates users too easily and economically process large amounts of data. The service comprises of accommodated Hadoop framework running on Amazon's Elastic Compute Cloud (EC2) [11]. The Amazon's Simple Storage Service (S3) [12] functions as storage layer for Hadoop. The Azure MapReduce [13] is an implementation of the MapReduce programming model, built on the infrastructure services, the Azure cloud [14] offers. Nowadays, online aggregation renewed in the context of cloud computing, and some studies conducted based on MapReduce. [17] Propose an alternative implementation of an online MapReduce framework under the shared-memory architecture. The focus of this study is not on large-scale framework architecture but on the challenges of parallel data analytics. Pansare et al implements OLA over MapReduce based on Bayesian framework [15]. The first parallax non-trivial time-based evolutionary indicator for Pig Latin scripts presented in [18] that interprets into a sequence of MapReduce jobs. During the query execution, changing processing speeds and degrees of parallelism handled by Parallax. The implementation of parallax in Pig and overtakes present replacements on typical workloads. Parallax breaks queries into pipelines in single-site SQL query progress estimation [19], [20], that is collections of interconnected operators performed instantaneously. For large problems, parallelism appears to be another promising approach to scaling up, particularly since multi-core, cluster and cloud computing are becoming increasingly the norm. Among parallel computing frameworks, MapReduce has recently been attracting much attention in both industry and academia. There are numerous successes of MapReduce [21], [22] including applications for machine learning and data mining problems. Parallel databases in the 80's and 90's [24], [25] are hard to measure since it requires special hardware and lacked satisfactory solutions to fault tolerance. The large-scale parallel databases [26], [27] are rapidly emerging now a day started to engage the MapReduce for the benefits of parallelism and fault tolerance. Most of the research aligned with these efforts but focuses on one-pass analytics. Hyracks is a new parallel software platform that proposes a DAG-based programming model, Even though Hayracks are not so good for the effective incremental computation than Hadoop, it is more general than MapReduce programming mode. [28] Refers to the techniques where the new MapReduce model used for enormous segregation parallelism and outspread it to incremental one-pass processing, which later used to support stream processing.

III. PARALLELISM IN MAPREDUCE

In the recent research work, the MapReduce is one of the widespread programming model for handling of large datasets in parallel with a cluster of machines. The Map-Reduce programming model has recently become a primary choice for fault-tolerant and massively parallel data crunching [39].This programming model uses the sort-merge technique with the aim to support the parallel processing systems. Hadoop is a familiar open-source software implementation that uses the parallelism in MapReduce paradigm. The Hadoop uses block-level scheduling and a sort-merge technique [29] to implement the group-by functionality for parallel processing of data. The Hadoop causes excessive CPU and I/O overheads with the use of block-level scheduling and a sort-merge technique that blocks reduce operation particularly when multi-pass merge is applied. The MapReduce parallelization unlikely from previous serial implementation, it permits to improve the data size reach by about two to three orders of magnitude (from 20K to 8M vertices). The MapReduce model is best suited for parallelizing because of its capacity to handle large disk-resident data. To accomplish the concept of parallelization into the MapReduce programming model, it essentially

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 4, October 2013

implements the map function to group the data with the help of key and then perform reduce function on each group. The MapReduce Paradigm agrees for parallelism in both the extraction of (key, value) pairs known as the map function and the use of reduce function to each group that is executed in parallel on many nodes. This working model also referred as the MapReduce group-by paradigm. The system of MapReduce performs this computation model including additional functionality such as load balancing and fault tolerance. The parallelism concept applied to various stages in MapReduce programming paradigm as follows:

1. *Parallelized mapping over input data set*: Both the key and value pairs of input data sets processed one by one. This form of a list map is affable to total data parallelism [30], [31]. The order of processing the key and value pairs does not affect the result of the map phase since map is a pure function. The communication between the different threads also avoided.

2. *Parallelized grouping among intermediate data set*: The collection of intermediate data sets using a key is essentially a sorting problem for the reduce phase and numerous parallel sorting models exist [6], [34]. If a distributed map phase is expected, then it is sensible to expect grouping to be associated with distributed mapping. That is, grouping performed for any part of intermediate data. This distributed grouping result could combine centrally, just as in the case of a parallel-merge-all strategy [35].

3. *Parallelized mapping over grouping data set*: A group is nothing but a key with a list of values. The reduction operation performed individually for each group. Again, the arrangement of a mapping operation applied here. The entire data parallelism is acknowledged for the reduce phase just as for the map phase.

4. *Parallelized reduction for each grouping data set*: The Reduce is an operation that separates a list into a distinct value through an associative operation and its components. Then, each application of the Reduce operation immensely parallelized by calculating sub-reductions in a tree-like arrangement while applying the associative operation at the nodes [32], [33]. Apart from these if the binary operation is also commutative, then the order of combining results from sub-reductions can be random.

In the MapReduce programming model, parallelism achieved through a "split/sort/ merge /join" process and described as A MapReduce Job starts from a predefined set of Input data shown in Fig 1. A principal daemon, which is a central coordinator, that starts and the job configuration done.

- According to the job config, the principal daemon will initiate multiple Mapper daemons and Reducer daemons in different equipment. Then the input reader starts reading the data from the DFS directory. This input reader will chunk the read data consequently and send them to the arbitrarily preferred Mapper. This is the split phase where the parallelism originates.
- After the data chunked, the mapper daemon will run a user-supplied map function and produce a collection of key, value pairs. Each element within this collection will sort according to the key and send to the corresponding Reducer daemon. This is the called the sort phase.
- All elements with the similar key will move to the same Reducer daemon. It obtains all the items of that key and raises a user-supplied Reduce function and finally, produce a single entry key, aggregated value as a result. This is called the merge phase.
- The output writer collects the output of reducer daemon. This is effectively called the join phase where the parallelism of mapper and reduce function ends.

Apart from these the drawback of MapReduce to achieve parallelizability in the paradigms are limited to use only map and reduce functions in their programs. Thus, this model trades of programmer flexibility for easy parallelization. This is a difficult trade off, and it is not a priori clear where problems can efficiently solve in the MapReduce paradigm.

A. MongoDB

MongoDB is an open source and a schema-free document-oriented database written in C++ and developed in an open-source project which is mainly driven by the company 10gen Inc. According to the developers of MongoDB, the foremost objective is to decrease the variances between the fast and highly accessible key-value-stores and feature-rich traditional RDBMSs relational database management systems. MongoDB name derived from the adjective humongous [36]. Prominent users of Mongo DB include SourceForge.net, foursquare, the New York Times, the URL-shortener bitsy, and the distributed social network DIASPORA [38]. Mongo DB is an open source NoSQL document store database, commercially supported by 10gen [37]. Even though Mongo DB is a non-relational database, it implements many features of relational databases, such as sorting, secondary indexing and range queries. MongoDB does not organize data in the form of tables with columns and rows; instead, it stores the data in the document form, each of which is an associative array of scalar values, lists, or nested associative arrays. MongoDB documents serialized naturally as Java script Object Notation (JSON) objects, and are in fact, stored internally using a binary encoding of JSON called BSON. If suddenly one of the shard shuts down, the other shards will distribute chunks among them equally in order to maintain a constant and continuous service; this is the best part of using MongoDB. To measure the performance of Mongo DB on a cluster of servers, it uses a technique called sharding, which is nothing but the process of splitting the data uniformly across the cluster to parallelize the access of data.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 4, October 2013

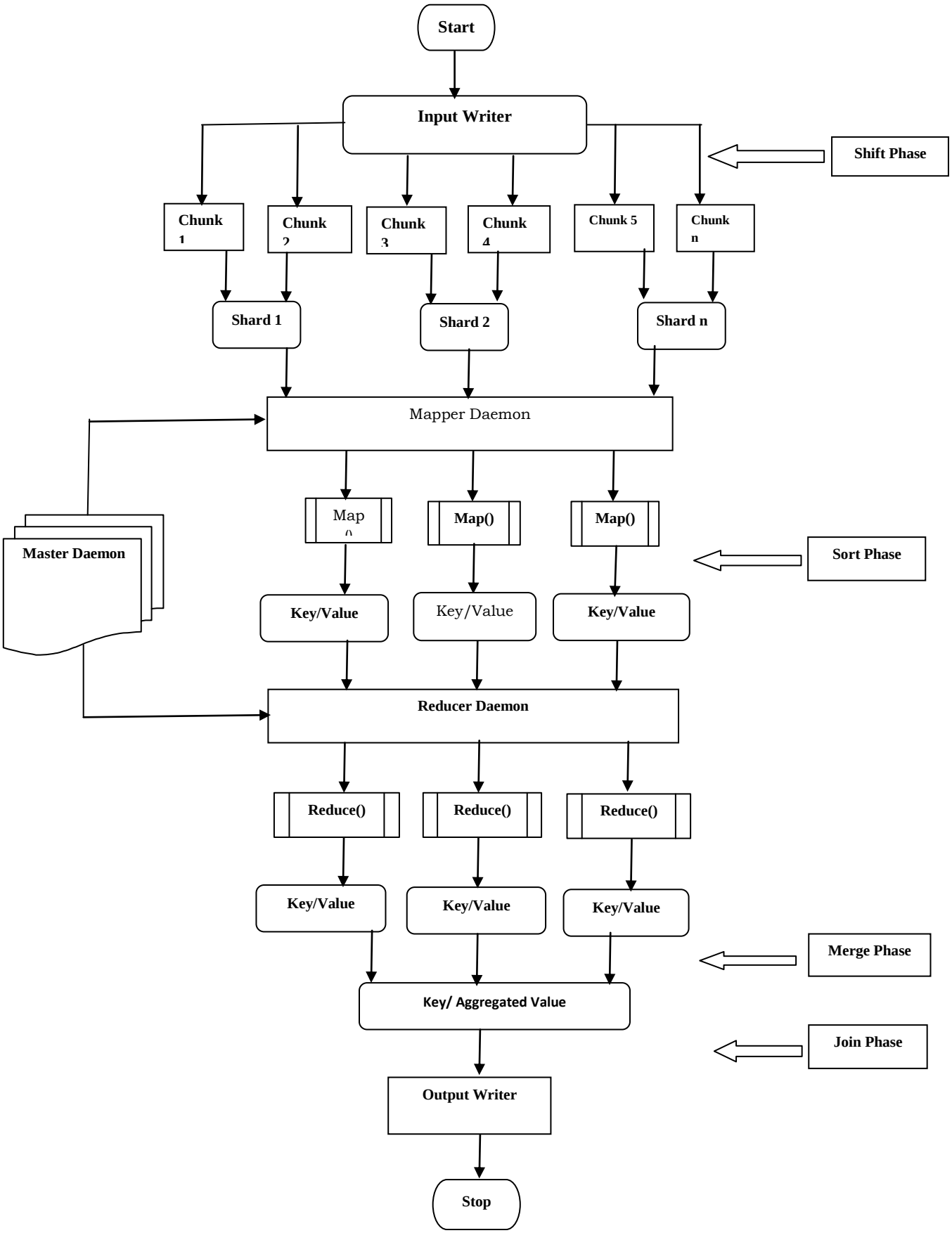

Fig 1: Block diagram of the Map Reduce model using shards

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 4, October 2013

These shards implemented by breaking the Mongo DB server into a set of front-end routing servers (*mongos*), which route operations to a set of back-end data servers (*mongod*). Sharding is the method of storing data records across multiple machines and is one of the MongoDB's methodology to encounter the demands of increased data set gradually. The Concept of sharding in Mongo DB supports the growth in the database and the increasing demands of read and write operations by adding more number of machines into it at a time. Database systems with large data sets and high throughput applications can challenge the capacity of a single server. High query rates can exhaust the CPU capacity of the server. Larger data sets exceed the storage capacity of a single machine. Finally, working set sizes larger than the system's RAM stress the I/O capacity of disk drives. To address these issues of scales, database systems have two basic approaches: *vertical scaling* and *sharding.*

Vertical scaling adds more CPU and storage resources to increase the capacity of the system. The addition of the capacity in the resources has some limitations: high performance systems with large numbers of CPUs and large amount of RAM are disproportionately more expensive compared to smaller systems. There is a *practical maximum* capability for vertical scaling.

Sharding, or *horizontal scaling*, divides the data set and distributes the data over multiple servers, or shards. Each shard is an independent database and collectively the shards make up a single logical database.

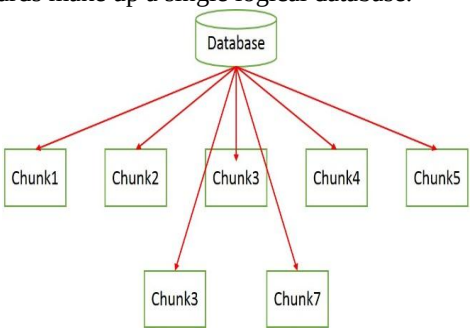

Fig 2: Database split into groups of chunks

Sharding addresses some of the challenge of scaling to support high throughout and large data sets. They are:

- Sharding minimizes the number of operations each shard handles. Each shard processes less number of operations as the cluster grows. As a result, shared clusters can increase capacity and throughput horizontally. For example, to insert data, the application only needs to access the shards that are liable for required records.
- Sharding reduces the amount of data that each server needs to store. Each shard stores less data as the cluster grows. For example, if a database has a 1-terabyte data set, and there are four shards, then each shard might hold only 256GB of data. If there are 40 shards, then each shard might hold only 25GB of data.

B. Sharding in MongoDB

MongoDB supports sharding through the configuration of sharded clusters. Sharded cluster has the following components: shards, query routers, and config servers:

- *Shards* store the data. To provide high availability and data consistency, in a production sharded cluster, each shard act as a data set.
- *Query Routers*, or mongos instances, interface with client applications and direct operations to the appropriate shard or shards. The query router processes and targets operations to shards and then returns results to the clients. A sharded cluster can contain more than one query router to divide the client request load. A client sends requests to one query router. Most of the sharded cluster has many query routers.

Config servers store the cluster's metadata. This data contains a mapping of the cluster's data set to the shards. The query router uses this metadata to target operations to specific shards. Production sharded clusters have exactly three config servers.

C. Data Partitioning

MongoDB distributes data or shards at the collection level. Sharding partitions a collection's data by the shard key.

Shard Keys: To shard a collection, you need to select a shard key. A shard key is either an indexed field or an indexed compound field that exists in every document in the collection. MongoDB divides the shard key values into chunks and distributes the chunks evenly across the shards. To divide the shard key values into chunks, MongoDB uses either range based partitioning or hash based partitioning.

Range Based Sharding: For range-based sharding, MongoDB divides the data set into ranges determined by the shard key values to provide range based partitioning.

Consider a numeric shard key: If you visualize a number line that goes from negative infinity to positive infinity, each value of the shard key falls at some point on that line. MongoDB partitions this line into smaller, non-overlapping ranges called chunks where a chunk is range of values from some minimum value to some maximum value. Given a range based partitioning system, documents with "close" shard key values are likely to be in the same chunk, and therefore on the same shard.

Hash Based Sharding: For hash based partitioning, MongoDB computes a hash of a field's value, and then uses these hashes to create chunks. With hash based partitioning, two documents with "close" shard key values are unlikely to be part of the same chunk. This ensures a more random distribution of a collection in the cluster.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 4, October 2013

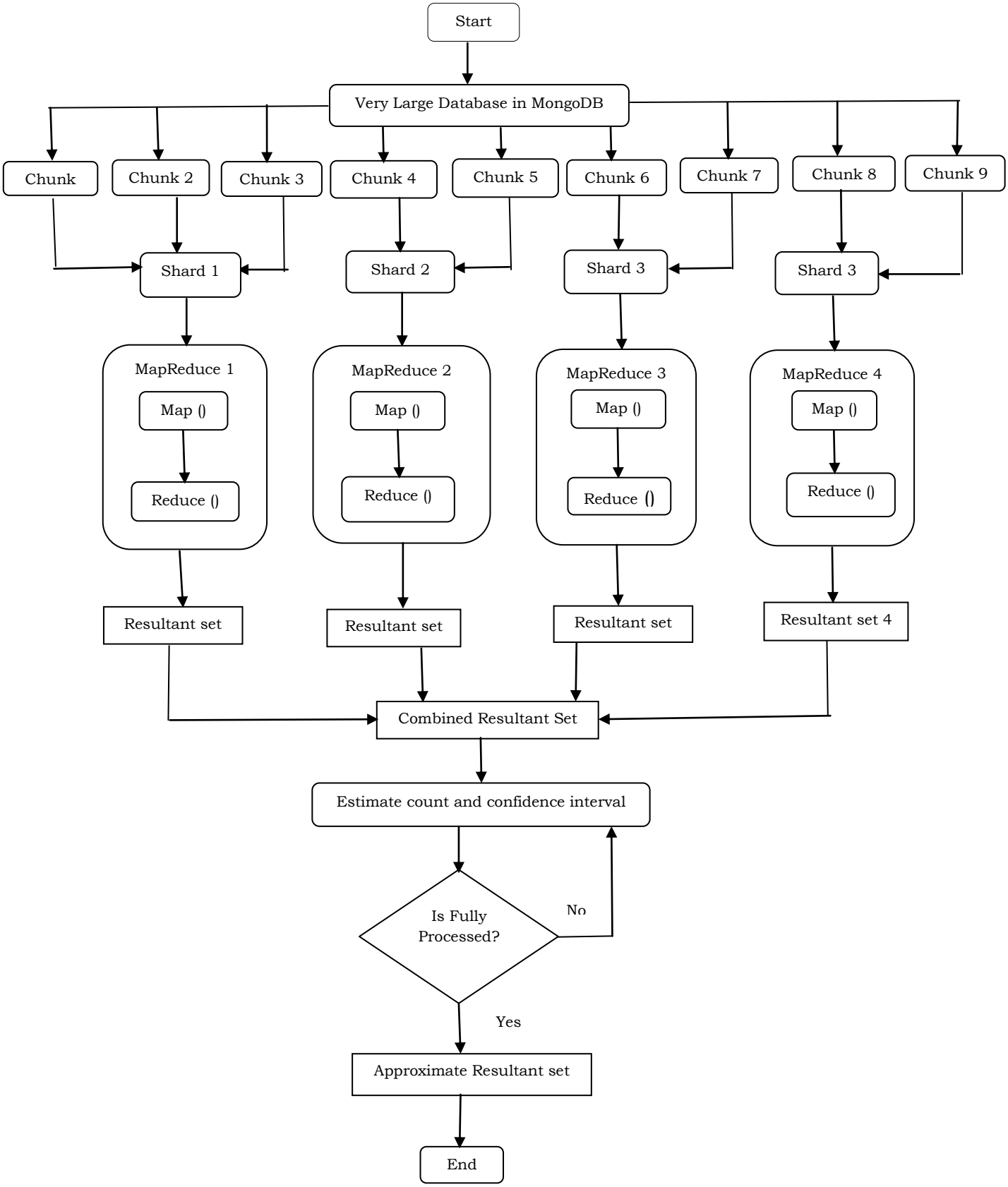

Fig 4: Sharded parallel MapReduce in MongoDB for Online Aggregation

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 4, October 2013

IV. PROPOSED METHODOLOGY

In the Online Aggregation, the large database are scanned in random order during the query processing time and as the scan proceeds sequentially, the approximate result updated for every aggregating query. The MapReduce programming model used along with the online aggregation to obtain the approximate results of the lager database in less time compared to the time taken to compute in the Traditional Query Processing systems, but it is usual that for the very large data set the computation time using the online aggregation interface and MapReduce model is high. As the size of the data increases, a single machine is not enough to store the data and not sufficient to produce a satisfactory read and write throughput. Thus, to minimize the time taken to compute very large data sets this paper proposed a new methodology known as Sharded parallel MapReduce in MongoDB for Online Aggregation. This methodology uses shards to store large data sets across multiple machines in Mongo DB for parallel execution where the map and reduce functions are executed in parallel on two or more servers, computer or terminals at same time. This methodology also improves the performance and efficiency of the online aggregation and MapReduce paradigm.

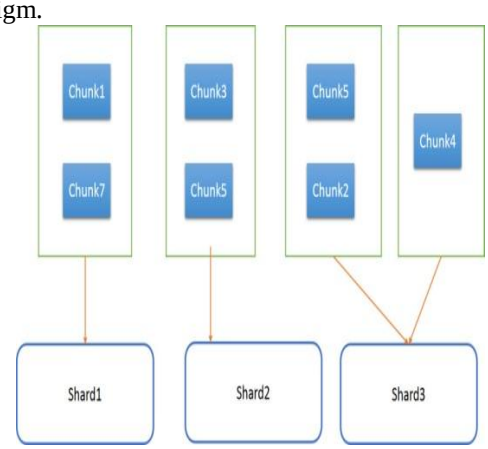

Fig 3: Chunks assigned to each shard in a balanced way (Chunks need not necessarily be in order

A. Algorithm for sharded parallel MapReduce in mongo DB for Online Aggregation

Algorithm 1 and Fig 4 gives the method of implementation of the proposed methodology. In this methodology, very large datasets needed to run on multiple machines considered for implementation. This large database grouped into some equal or unequal number of chunks as shown in Fig 3. Then these group of chunks are combined randomly to form group of datasets known as shards as shown in Fig 2. Grouping of the chunks into shards done in any order depending on the size of the database where each chunk considered for easy execution. The obtained shards send to each MapReduce Programming model for execution. The map and reduce functions are performed in sequence, which in turn executed on different machines parallel. Thus, the time taken to execute all the shards in MapReduce model is similar. The retrieved reduced resultants sets then combined and given to the interface for Online Aggregation to get an efficient approximate result quickly.

Sharded parallel MapReduce in MongoDB for Online Aggregation Algorithm 1:

1. Consider the collection of very large data set in terms of terabytes or petabytes in the open software Mongo DB for Query Processing.
2. The collection of whole database divided into group of datasets known as chunks.
3. These chunks grouped into number of shards equally or unequally.
4. The total numbers of shards obtained given to each MapReduce Paradigm for parallel execution at a time.
   a. First the map function is performed where the number of values are mapped to the same keys
   b. Then reduce function is performed for the same keys in order to combine the values at one place.
5. The steps (a) and (b) performed until map and reduce operations in each MapReduce model completely execute the data set
6. Then the resultant reduced data set is sent to the online aggregator to estimates the count , confidence and interval for the data set and the results are updated from the previous to new ones.
7. The step 6 continues until all the reduced data set in the online aggregation gives approximate results.

V. EXPERIMENTAL RESULTS

The Fig 5 shows the output results of the implemented in Mongo DB open software using three model set or clusters or shards. The three terminals shown in the Fig 4 are used for implementing the sharding in three clusters while one master terminal takes over the above three terminals to control it as slave terminals.

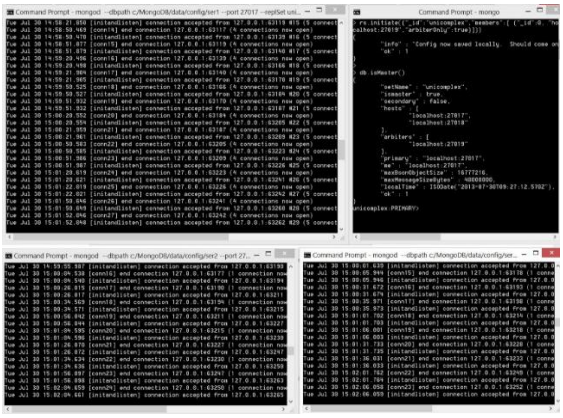

Fig 5: the Output Result for the execution of MapReduce function on 3 terminals

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 4, October 2013

Fig 6: the Output result of Online Aggregation in the Applets

Fig 6 show the online aggregation execution where the MapReduce implemented on 3 terminals for the whole database and the online aggregation results obtained on the applet front end. The Aggregate Function selected for the execution is Average.

1. Average – Average of the values in the field num_tabs in the database.
2. Confidence – The confidence interval for which the online aggregation was default set into.
3. Interval – The Error value that might occur while approximating the values of the MapReduce.

VI. CONCLUSION

The MapReduce Paradigm used in online aggregation has proved to be very efficient methodology for the retrieving approximate results in the large dataset. The use of the parallelism concept into MapReduce helps to further increase the efficiency. The execution of the very large database in two more processors, servers or machines improved the performance. Here for effectively make use of parallelism in MapReduce, the Mongo DB used shards into the database. Thus, executing the MapReduce in various servers with the help of shards has improved the MapReduce framework in Mongo DB, thereby improving the online aggregation performance. This actually works better than the previously proposed methodologies of traditional MapReduce since the fact that the program executed in more than one processor is considered. Using shard a pseudo cluster created, if the desktop configuration is high, as in Huge Ram (16GB) and faster processor, the number of shards increases, thereby increasing the level of parallelism in architecture.

REFERENCES

[1] A. Myers, "The importance of percent-done progress indicators for computer-human interfaces," In Proc. of CHI'85, pp. 11–17, 1985.

[2] J. M. Hellerstein, "The case for online aggregation," Technical Report, EECS Computer Science Division, University of California, Berkeley, CA, 1996.

[3] Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive —a warehousing solution over a map-reduce framework", In VLDB, 2009.

[4] Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: a not-so-foreign language for data processing", In Proc. of the SIGMOD Conf., pp.1099–1110, 2008.

[5] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad, "Distributed Data-Parallel Programs from Sequential Building Blocks," Proc. European Conf. Computer Systems, Mar. 2007.

[6] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan, "Interpreting the data: Parallel analysis with Sawzall", Scientific Programming, vol. 13, no. 4, pp. 277–298, 2005.

[7] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters", In Proceedings of OSDI, pp. 137–150, 2004.

[8] Das, M. Datar, A. Garg, and S. Rajaram, "Google news personalization: Scalable online collaborative filtering", In Proceedings of WWW, pp. 271–280, 2007.

[9] Hadoop, http://hadoop.apache.org/Amazon Elastic MapReduce, http://aws.amazon.com/elasticmapreduce/.

[10] Amazon Elastic Compute Cloud (EC2), http://aws.amazon.com/ec2/.

[11] Amazon Simple Storage Service (S3), http://aws.amazon.com/s3/.

[12] T. Gunarathne, T.-L. Wu, J. Qiu, and G. Fox, "MapReduce in the Clouds for Science," In Procs. of CLOUDCOM '10, pp. 565–572, Washington, DC, 2010.

[13] The Windows Azure Platform, http://www.microsoft.com/windowsazure/.

[14] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie, "Online aggregation for large MapReduce jobs", In VLDB 2011 Conference Proceedings, pp. 1135–1145, August 2011.

[15] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, J. Gerth, J. Talbot, K. Elmeleegy, and R. Sears, "Online aggregation and continuous query support in MapReduce", In SIGMOD 2010 Conference Proceedings, pp. 1115–1118, June 2010.

[16] Bose JH, Andrzejak A and Hogqvist M, "Beyond online aggregation: Parallel and incremental data mining with online Map-Reduce", Proceedings of the workshop on massive data analytics on the cloud, Raleigh, 2010.

[17] Kristi Morton, Abram Friesen, Magdalena Balazinska and Dan Grossman Computer Science and Engineering Department, "Estimating the Progress of MapReduce Pipelines," University of Washington Seattle, Washington, USA.

[18] S. Chaudhuri, V. Narassaya, and R. Ramamurthy, "Estimating progress of execution for SQL queries," In Proc. of the SIGMOD Conf., Jun 2004.

[19] G. Luo, J. F. Naughton, C. J. Ellman, and M. Watzke, "Toward a progress indicator for database queries", In Proc. of the SIGMOD Conf., Jun 2004.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 3, Issue 4, October 2013

[20] R. M. Yoo, A. Romano, and C. Kozyrakis, "Phoenix Rebirth: Scalable MapReduce on a Large-Scale Shared-Memory System," in Proceedings of 2009 IEEE International Symposium on Workload Characterization (IISWC), pp. 198–207, 2009.

[21] J. Dittrich, J.-A. Quian´e-Ruiz, S. Richter, S. Schuh, A. Jindal, and J. Schad, "Only Aggressive Elephants are Fast Elephants", PVLDB, 5, 2012.

[22] J. DeWitt, R. H. Gerber, G. Greak ,M .L. Heytrns, K. B. Kumar and M. Muralikrishna, "Gamma - a high performance dataflow database machine", In Proc. 12th. Conf. on Very Large Database, pp. 228–237, August 1986.

[23] DeWitt and J. Gray, "Parallel database systems: the future of high performance database systems", Commun. ACM, vol. 35, no. 6, pp. 85–98, 1992.

[24] Abouzeid, K. Bajda-Pawlikowski, K.Adabi, D. Silberschatz, A.Rasin, S.A., "Hadoopdb: An architectural hybrid of MapReduce and dbms technologies for analytical workloads", Proc. VLDB Endow, Vol. 2, No. 1, pp. 922–933, 2009.

[25] Friedman, P. Pawlowski, and J. Cieslewicz, "SQL/MapReduce: a practical approach to self-describing, polymorphic, and parallelizable user-defined functions", Proc. VLDB Endow., vol. 2, no. 2, pp. 1402–1413, 2009.

[26] Edward Mazur, Boduo Li, Yanlei Diao, Prashant Shenoy, "Towards Scalable One-Pass Analytics Using MapReduce,"

[27] T. White, "Hadoop: The Definitive Guide", O'Reilly Media, Inc., 2009.

[28] B. Skillicorn, "Architecture-independent parallel computation," IEEE Computer, vol. 23, No. 12, pp. 38–50, 1990.

[29] B. Skillicorn, "Foundations of Parallel Programming," Number 6 in Cambridge Series in Parallel Computation, Cambridge University Press, 1994.

[30] E. Blelloch, "Programming parallel algorithms," Communications of the ACM, vol. 39, no. 3, pp. 85–97, 1996.

[31] D. B. Skillicorn and D. Talia, "Models and languages for parallel computation," ACM Computing Surveys, vol. 30, no. 2, pp. 123–169, 1998.

[32] D. J. DeWitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H.-I. Hsiao, and R. Rasmussen, "The Gamma Database Machine Project," IEEE Transactions on Knowledge and Data Engineering, vol. 2, no. 1, pp. 44–62, 1990.

[33] 10gen, Inc: MongoDB, 2010, http://www.mongodb.org.

[34] Copeland and Rick, "How Python, Turbo Gears, and MongoDB are Transforming Source," Forge.net, Presentation at PyCon in Atlanta, February 2010.

[35] Heymann and Harry, "MongoDB at foursquare", Presentation at MongoNYC in New York, May 2010.

[36] Strozzi and Carlo, "NoSQL – A relational database management system", 2007–2010.

[37] Chen, S., and Schlosser, S. W. Map-reduce meets wider varieties of applications. Tech. Rep. IRP-TR-08-05, Intel Labs Pittsburgh Tech Report, May 2008.