International Journal of Web-Based Learning and Teaching Technologies
Volume 16 • Issue 6

Data Analytic Models That Redress the Limitations of MapReduce

Uttama Garg, Chandigarh University, Ajitgarh, India

ABSTRACT

The amount of data in today's world is increasing exponentially. Effectively analyzing big data is a very complex task. The MapReduce programming model created by Google in 2004 revolutionized the big-data computing market. Nowadays, the model is being used by many for scientific and research analysis as well as for commercial purposes. The MapReduce model however is quite a low-level programming model and has many limitations. Active research is being undertaken to make models that overcome/remove these limitations. This paper has studied some popular data analytic models that redress some of the limitations of MapReduce, namely ASTERIX, Pregel (Giraph), DraydLINQ, Dremel, and Graph twiddling. The author discusses these models briefly and through the discussion highlights how these models are able to overcome MapReduce's limitations.

KEYWORDS

ASTERIX, DraydLINQ, Dremel, Graph Twiddling, MapReduce, Pregel

INTRODUCTION

Today, a huge amount of data is present all around us; social media, scientific experiments and various organizations generate large quantities of data. Cleaning, analysis and subsequent result generation from this Big Data (IBM, 1911) poses a serious challenge. In 2003, Sanjay Ghemawat *et. al.* (Chen & Steven, 2008) introduced GFS (Google File System) that is scalable DFS (Ghemawat et al., 2003), which laid the foundation of MapReduce (GMR) (Dean & Ghemawat, 2008). It is a programming model to use in generating and processing large datasets. Both of these were landmark developments which revolutionized the entire data analytic industry.

In 2005, Dough Cutting et. al. at Yahoo started building Hadoop (Apache Hadoop, 2006) (named after a toy elephant) based on GFS and GMR. Being an open–source implementation that can run on commodity systems; Hadoop quickly became immensely popular and was being used by big companies like Facebook, Yahoo, Amazon etc.

Other than such fast and parallel processing applications, representation of data in the form of graphs also makes data analysis an easier task. Such an analysis can then be extended to a cloud. Understanding graph generation and manipulation is the basis for graph processing. The Stanford Graphbase (Skiena, 2008; Stanford GraphBase: A Platform for Combinatorial Computing, 1993), by Donald E. Knuth in 1993, is a large and portable collection of programs and data which can used as

DOI: 10.4018/IJWLTT.20211101.oa7

a foundation to study and understand any graph related problems. It is easy to process such graphs locally, but their implementation over a cloud came as a challenge.

Jonathan Conhen in year 2009, in his paper "Graph Twiddling in a MapReduce World" (Microsoft, 2017) makes this idea realistic by decomposing graph operations into a sequence of MapReduce steps, thus processing graphs over a cloud.

In 2010, Grzegorz Malewicz et al. at Google introduced Pregel: system to support large scale graph mining and processing. Giraph,released by the Apache Software Foundation in 2011, which is an open source work of Google's Pregel model for complex graph computation.

In the same year, another distributed system called Dremel, which was orginally in production since 2006, was improved by Sergey Melnik et al. at Google. With the invent and improvement of this interactive query processing system, trillion-row tables can be processed and analyzed within seconds.

In 2011, Alexander Bahm et al. at the University of California discussed about a platform called ASTERIX. This system addresses the problem of limiting and low level programming model of MapReduce and provides a high level storage and analysis of real world data.

Before GMR and Hadoop (Lee et al., 2011), relational DBMS and supercomputers were used to handle storage and computation for such processes. The use of Distributed Systems has increased the performance, scalability, cost-efficiency and fault tolerance of various applications . A distributed system is simply a cluster of systems connected by a network. There is no special hardware requirement for a system to be a part of this network and the computation is distributed among the various components to be done in parallel.

In this paper, we discuss the related underlying technologies and Map Reduce model and the limitations of Map Reduce in Section IV. We then descibe various models that came as an extension, improvement or substitution of the MapReduce framework. ASTERIX in dicussed in section V,Pregel in section VI and Giraph in section VII and respectively.

UNDERLYING TECHNOLOGIES

In this section we include some of the basic technologies/developments that have played a vital role in develop-ment of various models discussed in this paper. These are:

- Google File System
- BigTable

Both Google File System and BigTable were very important for the development of MapReduce. They also form the basis of various other data analytic models.

The Google File System (GFS)

Google file system(GFS) (Ghemawat et al., 2003) is a DFS(distributed file system) created by Google. It aims at providing reliable, fault tolerant and scalable data storage over commodity hardware. In GFS, files larger than 64 Mb are divided into chunks and stored on different computers. Most of the stored files on GFS are either read or modified by appending some new information. GFS setup contains a large number of commodity machines; one of them is known as the Master node and others are Chunkservers. Chunkservers store file chunks (of 64 MB each) whereas the master node is responsible for storing meta-data of the file i.e. creation time, names (64 bit names generated by the master) and location mapping. Master node does not store any file chunks. Chunkservers send small updates to the Master periodically informing it about any changes. Each node in the GFS is usually replicated 3 times.This step is necessary to handle the problem of node failure which is quite a common phenomenon.

Master provides permission for reading and modification in the form of "timed leases". After modification changes are propagated to other replicas by the "Primary Chunkserver". Changes are not made permanent unless all other replicas send an acknowledgement to the Primary Chunkserver.

A newer version of GFS is called Colossus.

BigTable - Distributed Storage System

BigTable, is another data storage system developed by Google (Chang et. al., 2006 (Chang et al., 2008)) built upon some Google technologies like Chubby, The Google File System and SSTable. It is currently being used by Google in various Google products like Google Earth, Google Finance, YouTube etc. It was designed to scale huge amount of data spread over a large cluster of computers.

Building Blocks

- **GFS:** BigTable storage system is based upon The GFS. Google File System has been discussed in section 2.1.
- **Google SSTable:** It is used to store the BigTable data. It provides us with a map to values from the given keys. (Chang et. al., 2006). Each SSTable contains blocks which helps in mapping/ looking up data. SSTable is vital for data lookup in the BigTable system.
- **Chubby Lock (Burrows, 2006):** It is a lock service designed for distributed systems. It plays a key part in various Google technologies like GFS, BigTable and MapReduce. Chubby consists of 5 replicas in which 1 is elected to be the master. Chubby uses the Paxos algorithm to ensure consistency among its replicas working on the lease-expiration model. BigTable uses Chubby to lock one master that stores the BigTable data, schema information and access control lists. A similar technology to Chubby is ZooKeeper. ZooKeeper (Hunt et al., 2010), a subproject of Hadoop, coordinates different processes in a distributed application. It exploits wait-free data objects to achieve its goals. ZooKeeper provides two features:
  - **Linearizable writes:** all requests for data objects are serialized.
  - **FIFO client order:** All requests from a single client are executed in same order in which they were sent.

Zoopkeeper (Twister, 2021) is similar to Chubby but does not use lock primitives i.e open or close. It is based on wait free data objects. ZooKeeper uses a "watch mechanism" where a client can watch for an update to get notifications when any changes are made to a given data object. Chubby manages the client cache directly by invalidating the caches if some changes occurs.

Data Model and Implementation

The table is indexed by three keys that are row key and column key and timestamp key. Each table in the system is multidimentional in nature and can be split into different tablets (Table segments accross rows of about 200 MBs). Information about these tablets is stored in the Master node (elected and locked by Chubby) which provides lookup services (using SSTable) to locate the data saved on multiple computers managed by The Google File System.

Similar Technologies

There are many alternatives to the BigTable distributed storage system available today. The Boxwood Project (Chang et al., 2008) can be considered as a substitute. Similar services are available with projects like Tapestry, Chord, CAN etc (Chang et al., 2008).

MAPREDUCE

The MapReduce programming model (Dean & Ghemawat, 2008) is based on distributed system. It is well known programming paradigms. An application in this model is implemented as a series of

Map and Reduce operations. Each of them having a Map and a Reduce phase, processing huge no. of data items. And this system supports task management, distribution of computations, automatic parallelization and fault tolerance without burdening the programmer.

Execution Overflow

The Map Reduce model consists of mainly two functions: first Map function and second Reduce function (Dean & Ghemawat, 2008). Map functions take the input data and produce intermediate key-value pairs. These pairs are passed on to Reduce functions. Each Reduce function for every key, processes all the values for that key and generates the output value.

When the user initiates a MapReduce job, the following series of steps take place (Dean & Ghemawat, 2008) (Figure 1).

The input data is partitioned and then distributed across multiple machines or nodes. Different copies of the same program are then started on these nodes.

One of these nodes is called as the master. And rest are called workers. The master node allocates work to workers and manages them.

A worker assigned the map function reads the input. It then generates intermediate key pairs and passes each pair to the Map function.

These intermediate pairs are buffered in the local disk of the machine. The location of these intermediate pairs on the local disk is notified to the Master.

Example

The word count problem (Data mining 2.0: Mine your data like Facebook, Twitter and Yahoo) is considered one of the easiest to implement and understand using the MapReduce model. Here, given a random list of words, the aim is to count the number of each word occurances. The initial list of words is the input data.

Firstly, the initial data is partitioned so that it can be distributed to different map workers.

The master assigns work to different worker nodes. In this example, 3 map worker nodes work as the data is split into 3 parts.

Of the input data (simple words), the Map worker generates intermediate key value pairs. For occurrence of any word the Map function puts the count as 1. The intermediate key value pairs are (Deer-1; Bear-1; River-1, etc.).

Figure 1. MapReduce Execution Overflow (Dean & Ghemawat, 2008)

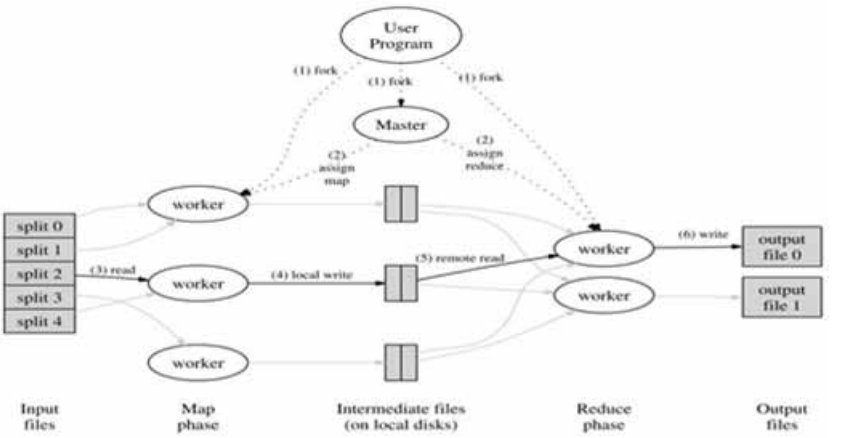
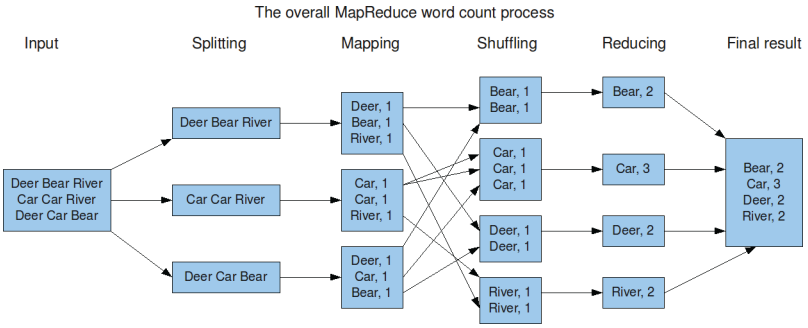


Figure 2. Example of a MapReduce execution (Data mining 2.0: Mine your data like Facebook, Twitter and Yahoo) data to the Reduce workers



These intermediate key-value pairs are stored into the workers local disk. The location of these intermediate key-value pairs is communicated to the Master.

Once all the Map workers have completed their work, the Master distributes work to Reduce workers. It starts this by sending the location of the intermediate.

The Reduce workers read this data through remote procedure calls. As shown in Figure 2 the Reduce workers first sort/shuffle the data by keys so that all similar occurrences of keys (words) are grouped together.

Once sorted, the reduce worker iterates over the key-value pairs and upon discovering a unique word, it runs the Reduce function for that set of intermediate values. The key - value pairs are Bear-2; Car-3; Deer-2; River-2 are thus obtained.

Finally, the output of the different reduce functions is compiled in a single global file. With this, the work of Reduce workers end and the master node notifies the user.

Features

- **Fault Tolerance:** A fault in the MapReduce system can occur mainly in two cases - a worker node fails, or if the master node fails (Dean & Ghemawat, 2008).
- **Worker Node Failure:** The master pings the workers periodically. If a node doesnt respond in a stipulated time, the node is considered to be lost and is set to its initial idle state. The work is then scheduled to some other node. In case of worker failure after completed Map tasks, all the data is lost as the intermediate data is stored in the map worker's local disk and is inaccessible to the master node. For completed Reduce tasks, the output generated is written/appended in a global file. So no data is lost for completed Reduce tasks.
- **Master Node Failure:** Since there is a single master, there is nothing much that can be done if the master node fails. In case of master failure, the program stops running and the user is notified. A better way to handle this is to have periodic checkpoints for the master node. Master node logs are backed up periodically, so if the master node fails, some other node (usually the most powerful) is selected as the master node and the work resumes from the last checkpoint.

Easy Coding Writing programs for Distributed Systems is difficult (Dean & Ghemawat, 2008). A programmer has to keep in mind various parameters like concurrency, parallelisation, performance, data distribution, load balancing, fault tolerance, etc. The original simple computation is obscured by large amounts of complex code. The MapReduce model can be considered as an abstraction, which hides the messy details of parallelisation and other complex issues and allows the user to express the problem as a set of simple computations. The code needed for the complex tasks is kept in various libraries.

Flexibility

The MapReduce model is applicable to a vast variety of problems as it is highly flexible (Melnik et al., 2010). This model can also be easily tweaked to suit the needs of the problem. The power to do optimization is limited by knowledge of system and about a particular computation task. (Chen & Steven, 2008)

Optimisations (Dean & Ghemawat, 2008) are operations done on the input/intermediate data to ensure quick execution. Some example are:

- **Sampling:** The system can take a random sample (Apache Hadoop, 2006) from the dataset and run it to take note of the skew characteristics of the dataset. Such skews can then be taken care of in the full-scale run.
- **Statistics Collection:** One initial run at the beginning would provide us with the characteristics and statistics of the dataset (Apache Hadoop, 2006) or the computations on the dataset. If, then, the computations or the dataset change slightly, we can use the statistics to improve the performance of the subsequent runs.
- **Stragglers:** Stragglers avoidance (Data mining 2.0: Mine your data like Facebook, Twitter and Yahoo) is one of the optimizations proposed to the MapReduce model to ensure timely execution. Stragglers are the machines that take up an unusually long time to complete one of the last few map or reduce tasks in the computation. As mentioned, Reduce function is executed only after all the Map workers have finished. So if a few Map workers are working and are taking considerable time to generate results, the whole of the Reduce function is delayed. To optimize this, the master node duplicates the tasks given to the stragglers to other worker nodes and accepts the result of whichever node that finishes first.

Use and Variants

Scientific analysis involves analysis of big volume of data collected from the various sources. State of the art research in fields like cryptography, image analysis, machine learning, data-Mining etc. requires analysis of huge collections of data. Furthermore, scientific or industry processes like distributed sorting, distributed pattern-based searching, web link-graph reversal, web access log stats, user recommendations, clustering, etc. (Dean & Ghemawat, 2008) require tools like Map Reduce for data analysis. The most common approach to solve this problem is to use Map Reduce and its variants (Lee et al., 2011). Map Reduce model is a potent tool making analysis easier. MapReduce technique benefits analysis by providing features like better speed, fault tolerance, load balancing and scalability.

Variants of the MapReduce model to address specific problems can be easily created by modifying/adding/deleting various stages of the original MapReduce model proposed by Google (Dean & Ghemawat, 2008).

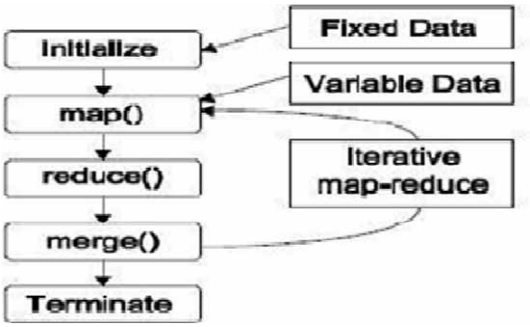
Many variants are being used in the industry. Some popular variants are DisCo, CGL-Mapreduce, Pig, Hive, SCOPE, Twister, PREGEL, Clustera, Dryad. Three have been discussed here.

Apache Hadoop Hadoop (Apache Hadoop, 2006) is the plain vanilla, opensource implementation of the MapReduce model. Hadoop was derived from Google's MapReduce(Dean & Ghemawat, 2008) and Google File System (GFS) papers (Vinayak et al., 2010) (Figure 3).

CGL-MapReduce CGL-MapReduce (Lee et al., 2011) is an opensource implementation of MapReduce in which its creators have added a feature called Iterative MapReduce Stage. Iterative MapReduce stage is different from a normal Map-Reduce model as it allows the programmer to call the Map function again after the completion of the Reduce function. This facility is not present in other popular MapReduce implementations like Hadoop, Disco etc.

As a result of this Iterative MapReduce stage, CGL-MapReduce is good for clustering (specially K-Means cluster-ing), and is also better and faster than Hadoop.

Figure 3. Stages of CGL- MapReduce (Chen & Steven, 2008)



Twister Twister Data Framework (DISCO, 2005), released in 2010, is designed to improve the overall user experience with the existing Hadoop framework. Twister's major task is to transform, extract, load, and manage massive Hadoop clusters. Twister manages the execution of Map and Reduce operations. It also alleviates the burden of writing custom code for data input and output from the Hadoop cluster.

Thus it becomes easy to use and manage Hadoop by utilizing by the abstraction provided by Twister.

Clustera Clustera, is a data management system. It is designed for a wide variety of computationally intensive jobs, which have low I/O and require complex queries on massive tables. Experimental results show that Clustera is highly scalable for SQL processing, it is comparable to Hadoop in performance (Figure 4).

Disco Disco (Pei, 2020) is an open-source platform that is used to analyse large scale data. Disco framework supports parallel computa-tions over large amount of data sets that are running on unreliable cluster of computers.An implementation of MapReduce is also available in the framework.

The Disco core is written in Erlang that is a functional language, designed for building robust, fault-tolerant and distributed applications. Disco users typically write jobs in Python, that makes it possible to express complex algorithms only in tens of lines of code. This implies that its easy and fast to write programs in Disco.

Pig Apache Pig (Apache Software Foundation, 2007) is a platform for analyzing massive data sets, consisting a high-level language that express data analysis programs .Pig can handle very large data sets since it allows substantial parallelization.

Figure 4. Clustera Architecture

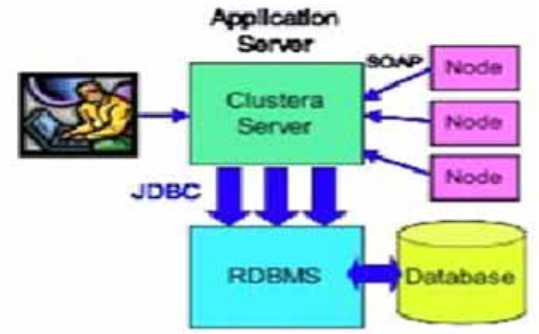
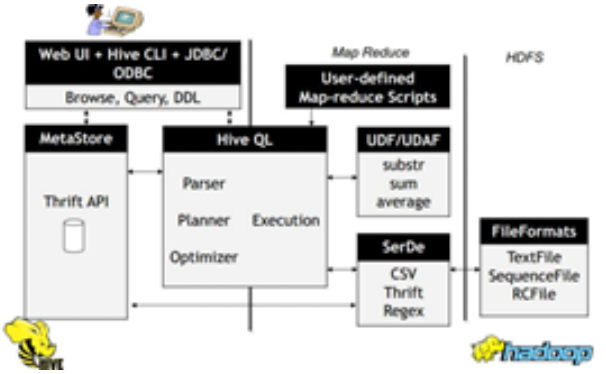


Figure 5. Hive System architecture (Huang et al., 2010)



Pig is a research project at Yahoo! Inc. and then it became an independent subproject of Apache Software Foundation's Hadoop project.

Currently, Pig's infrastructure layer consists of a compiler, produces sequences of Map-Reduce programs, for which large-scale parallel implementations already exist (e.g., the Hadoop subproject). Thus the platform overcomes various limitations of Hadoop by providing high level programming language framework.

Hive

Hive (Thusoo et al., 2010) is also an open source data warehouse solution developed on the top of Hadoop(Apache Hadoop, 2006). Hadoop, a MapReduce implementation provides a low level programming model. In order to overcome this limitation, Hive was developed to facilitate querying and manage large datasets over a distributed storage. Hive has its own SQL like programming (Figure 5).

Model and a declarative language called as HiveQL, which can be compiled into Map Reduce job and then executed using Hadoop. It also provides a mechanism to plug in the custom MapReduce jobs into these queries. In addition to this, variety of data formats, primitive types and collections (like maps, arrays etc) are also supported. Designed to work the best with batch jobs over large datasets, Hive helps in improving scalability, extensibility, fault-tolerance and loose-coupling.

SCOPE

In order to improve the performance of a Map Reduce system, it is important to analyze the log files generated during Hadoop execution. Visualization and analysis of these log files can help us understand as to how a Hadoop process behaves. Structured Computations Optimized for Parallel Execution (SCOPE) is a real time MapReduce tracing tool used to keep a track on MapReduce jobs by capturing the details of progress of all the ongoing tasks. It helps to understand the health of Hadoop cluster nodes, displays the distribution of file system blocks, and also views the content of these blocks. It also has a SQL like declarative scripting language and uses an optimizer to convert scripts into an efficient execution plan. SCOPE is therefore a very important performance tracking tool for MapReduce.

Dryad

Dryad (Isard et al., 2009) is a system similar to MapReduce with low level programming support. A Dryad job is a directed acyclic graph where each vertex is an operation performed on the data and a channel represents the graph edges. At run time, vertices are processes communicating with each other through these channels, and each channel is used to pass data.

Figure 6. Dryad System architecture (Isard et al., 2009)

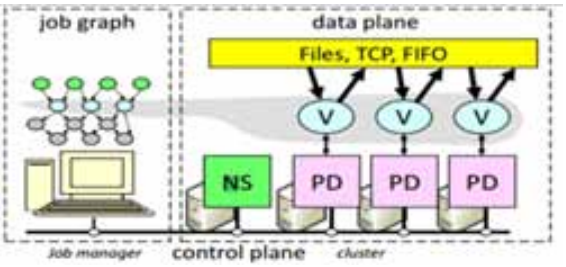


Figure 6 shows Dryad system architecture. Here, NS is name of server which maintain the cluster membership. And the job manager is responsible for spawning of vertices (V) on available computer with help of remote execution and a monitoring daemon (PD). Vertices exchange data through les, TCP pipes, shared-memory channels. The grey shape indicates vertices in job that are currently running and correspondence with the job execution graph.

LIMITATIONS OF MAPREDUCE

Low-Level Programming Model

MapReduce is quite low level for the programming tasks.The programmers have to divide a complicated algorithms into Maps and Reductions manually. They also have to write complex scripts to sequentially execute a series of MapReduce jobs. Hence, many high level systems like ASTERIX (Vinayak et al., 2010), Giraph, Dremel (Apache, 2010) etc. were created to make programming easier.

Many high-level wrappers on MapReduce were also created, like HIVE(Franco et al., 2014), Pig(Apache Software Foundation, 2007) etc. these help users by providing them with many features.

Similarly, in the Dryad project DAGs need to be created as they serve as input to the model. DAG creation is quite a tedious task. Hence one can say that the process is too low-level. This in turn paved the way for DryadLINQ which is a high-level programming model.

Non Recursive Approach

MapReduce cannot be applied to recursive problems. For example, In Fibonacci series previous values are required to compute the subsequent values i.e., $f(k+2) = f(k+1) + f(k)$ in such a case MapReduce can't be applied. It is impossible to break complex computations into smaller counterparts. For such computations MapReduce will require the user to re-initiate the system with previous values. Also, if the data is small in size, it will be better to process it as a single operation on one machine, than running an entire MapReduce process which will require many overheads like synchronization, data communication etc.

Large Scale Graph Processing

MapReduce doesn't work well when we have a large set of graphs to process. Although graph algorithm can be implemented as series of Map Reduce invocations but it requires passing of the graph from one stage to another. MapReduce may lead to a suboptimal performance as passing the state of a graph from one phase to another generates too much of I/O. Moreover it also has some usability issues as it doesn't provide support to do any per-vertex calculations. Then Google came up with Pregel which provides scalability, fault-tolerance, and flexibility to express arbitrary algorithms and handle around billions of graphs together.

Another challenge is to be able to represent data in the form of graphs such that the power of cloud can be analyzed by understanding the problems of very large graphs. For this, it is required to decompose the useful operations asso-ciated with graphs into a series of MapReduce jobs which MapReduce alone does not provide . Graph Twiddling in a MapReduce World (Graph Twiddling in a MapReduce World, 2009) addresses these missing features and shows that the representation of graphs in the form of MapReduce processes can lead to an efficient implementation of such graphs over cloud.

Static vs. Dynamic Data

MapReduce is a batch operation, not an online one. Morever the information is having diverse and rich structures which demand handling a blend of, semistructured, structured and even unstructured information.Also both the data and its structure can keep changing constantly.So the MapReduce cannot handle very large size of evolving world information. So the ASTERIX (Vinayak et al., 2010) came in with the motive to provide high level language support to the semi-structured information over the evolving world models.

Interactive Query Processing

In order to analyze a very large data quickly, large processing power is also required. Many sequences of MapReduce processes are involved while processing such large tables. MapReduce works well on such data, but when there are about trillion of records in a table, it does not provide results immediately. Such kind of data processing requires high degree of parallelism and extremely fast search operations. Dremel(Apache, 2010), an interative ad-hoc query system was then introduced by Google which is capable of querying and processing trillion row tables in seconds and thus complements MapReduce.

Automatic Optimizations

No automatic optimizations take place across the MapReduce boundaries because the underlying MapReduce execution platform is much less fleflxible.Automatic optimizations both static and dynamic optimization for sorting,hierarchial ag-gregation, distribution, analyzing run time statistics etc are required in order to achieve better performance. DryadLINQ(Yu et al., 2011) handles this limitation of MapReduce by providing automatic optimizations and query processing.

ASTERIX

The open source ASTERIX (Vinayak et al., 2010) platform aims to design and develop a highly scalable platform for parallel information storage and a sytem for data-intensive information analysis. ASTERIX is an effort to store, index, query, analyze, enor-mous quantities of semistructured information in parallel. Semistructured data can be defined as the data which is not following formal structure of relational databases and data models and is frequently updated.

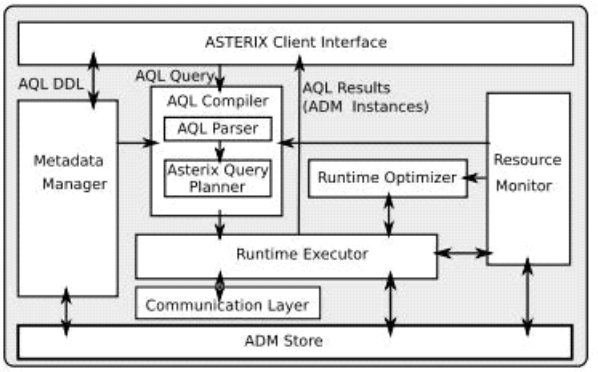
ASTERIX has been built to support high level data languages in itself. It is quite different from wrappers like Hive (Franco et al., 2014), Pig which produce the end result by running a MapReduce job beneath their implimentation.

The author (Vinayak et al., 2010) in their implimentation of ASTERIX are using a selective fault-tolerance mechanism, which takes into account intermediate results of task into consideration rather than the brute force solution used by Hadoop.

ASTERIX Architecture

The authors of the paper(Vinayak et al., 2010) have divided the architecture into two major parts (see Figure 7):

Figure 7. ASTERIX System Architecture (Vinayak et al., 2010)



- **ASTERIX Data Model (ADM):** It is used to store semistructured data. Author of (Vinayak et al., 2010) stress that, "Each individual ADM data instance is typed and self-describing. Datasets may have associated schema information that describes the core content of their instances." ADM is based on various semi-structured formats like JSON and Avro but not XML; as it may have lots of document-oriented query language.
- **ASTERIX Query Language (AQL):** To process, store and modify data, authors designed AQL. In accordance with the structure and the constructs of ADM. AQL has been based on XQuery, but again avoids many XML document related features.

Updates in ASTERIX are done by newer version of data rather than modifying the existing data instances. This results in improved multiuser performance. The storage and indexing in ASTERIX is done using primary and secondary B+ tree indices.

Instead of MapReduce, ASTERIX uses Hyracks, a platform which runs data-intensive jobs in parallel on a cluster of distributed machines. In order to perform AQL queries execution and run time scheduling and coordination, ASTERIX converts AQL requests into Hyracks jobs, Hyracks controls parallelism and resource management on the cluster. Hyrack can also be used to execute MapReduce and Dryad (Isard et al., 2009) tasks. Existing Hadoop(Apache Hadoop, 2006) and Dryad applications can be migrated to a Hyrack cluster which can coexist with an ASTERIX installation.

PREGEL

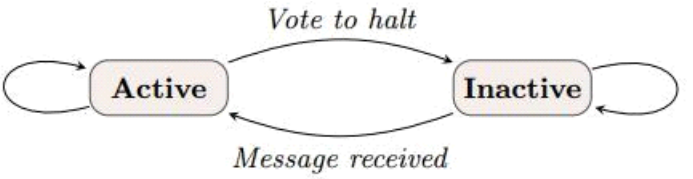
Graphs are of great concern in many computing problems. For example, designing transportation routes, analysing documents on the web, identification of disease outbreak patterns etc. Solving/ Analysing these graph problems require huge processing of graphs. In order to accomplish this Google created a scalable infrastructure, named Pregel (Malewicz et al., 2011), to support mining/ analysis of large scale graphs.

Pregel Architecture

In pregel, the input is a directed graph with its vertices and edges. In each iteration called supersteps each vertex executes a user defined function which defines the logic of the algorithm. This is executed in parallel. Vertices are able to read messages which were sent to them in previous iteration. They (nodes) send messages to be read in next iteration and are also able to modify state of outgoing edges (Figure 8).

Algorithm will terminate when all vertices "vote to halt". In the initial step, all vertices are in the active state. Each active vertex participates in the computation. A vertex can deactivate itself

Figure 8. Vertex State Machine (Malewicz et al., 2011)



by "voting to halt" when it has no fur-ther work to do. The algorithm terminates when there is no message in transit and a vertex does not have any other work.

The output of a Pregel program is a "set of values" associated with different vertices. For example, if a program is running a graph mining algorithm the output may be some statistics associated with the data.

The authors of the paper (Malewicz et al., 2011) have given an example illustrating how Pregel works. A strongly connected graph is taken as the input where each vertex having a value, propagates its largest value to every other vertex. In each iteration, the vertex with the largest value (received from various messages), forwards the value to all its neighbors. The algorithm terminates when no vertex is left with any new information.

The authors have also discussed how a Pregel user can use it to solve many problems like Page Rank, Bipartite Matching, Shortest Path computation, and a Semi-Clustering algorithm. We have discussed PageRank computation on Pregel in the subsequent section.

Application on PageRank

The performance, fault tolerance and scalability of Pregel workwell for graphs with millions of vertices. Morever the authors are investigating techniques for scaling Pregel to even larger graphs. Pregel also has an interactive application programming interface. For example, its developers have claimed that implementation of the PageRank computation algorithm (Hunt et al., 2010) in Pregel requires just 15 lines of code.

GIRAPH

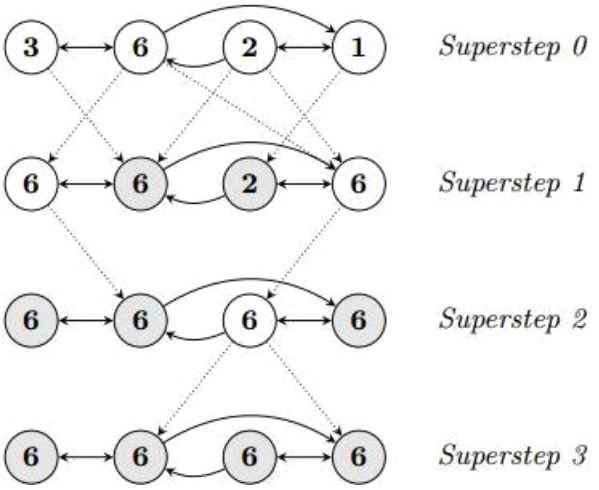
Giraph is an open-source implimentation of Google's Pregel model. It is rapidly being developed by the Apache foundation.

In Figure 9, Dotted lines are messages and Shaded vertices have voted to halt (Malewicz et al., 2011).

Nested Columnar Storage

It provides a lossless columnar format representation by defining Repetition levels and Definition levels. (Apache, 2010) defines these repetition levels as at those repeated field in field's path in the value has repeated. This repetition level is assigned to each value in order to disambiguate the occurrences of various fields in a nested storage. Each value in the field with path p has a definition level that specify the number of fields in p that can be undefined, i.e. either optional or repeated, are actually present in the record. To encode the record structure in columnar format: each column has been stored as set of blocks. Where each block contains repetition and definition levels. Null is not stored explicitly as they are determined by the definition levels. Definition levels are not stored for values which are always defined. Similarly repetition levels are stored only if required.

Figure 9. Maximum Value Example



CONCLUSION

MapReduce limitations were being highlighted and a huge demand existed for a more versatile/powerful MapReduce or creation of an alternative. To improve on the limitations, many substitutes and extensions have been suggested. ASTERIX is one such substitution proposed, which provides a data intensive storage and computing platform for the analysis of large datasets. To compete with Hadoop and other systems, Microsoft created Dryad. To further improve Dryad, DryadLINQ was built which was augmented by combining Dryad with LINQ. In order to address and query very large datasets, another system called DREMEL was developed. DREMEL is designed such that it complements the MapReduce paradigm. Though MapReduce provides some existing graph based algorithms, such algorithms could not have been scaled to very large and complex graphs. Graph Twiddling, an idea proposed by Jonathan Cohen, focused on decomposing the graph operations into a sequence of MapReduce steps. With this idea of large scale graph processing, Google also came up with a model called Pregel. Pregel can be seen to have the combined features of Dremel and Graph twiddling. Looking at all these ideas and models which derive their interest from MapReduce, it can be seen that a lot of work has been done to either improve MapReduce or to replace it altogether. ASTERIX incorporates all the features of MapReduce along with many extensions to it and can be clearly seen as a replacement to MapReduce. Even after so many advancements, what is required is another complete system (ASTERIX does come close) that could address all the issues discussed. This can solve the problem of relying on different systems for different needs, thereby providing a single system to do it all.

REFERENCES

Alexander, B.,Vinayak R. B., Michael, J., Carey, Grover, R., Chen Li, Nicola O.,Rares, V., Alin D., Yannis, P., Vassilis J, Tsotras.(2010). ASTERIX: Towards a scalable, semistructured data platform for evolving-world models.

Apache. (2010). Apache ZooKeeper. https://zookeeper.apache.org/

Brin, S., & Page, L. (1998). The Anatomy of a Large-Scale Hypertextual Web Search Engine In *Proc.7th Intl. Conf. on the World Wide Web*.

Burrows, M. 2006. The Chubby lock service for loosely-coupled distributed systems. In Proceedings of the 7th symposium on Operating systems design and implementation (OSDI '06). USENIX Association, USA, 335–350.

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2008. Bigtable: A Distributed Storage System for Structured Data. ACM Trans. Comput. Syst. 26, 2, Article 4 (June 2008), 26 pages. DOI:https://doi.org/ 10.1145/1365815.1365816

Chen, S., & Steven, W. S. (2008). *Map-Reduce Meets Wider Varieties of Applications*. IRP.

Data mining 2.0: Mine your data like Facebook, Twitter and Yahoo. (n.d.). Retrieved April 02, 2021, from http://www.rabidgremlin.com/data20/

Dean, J., & Ghemawat, S. (2008, January). MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107–113. doi:10.1145/1327452.1327492

DISCO. (2005). Disco MapReduce. http://discoproject.org/

Franco, T., Kadhi, S., Leonard, J., Adouani, N., Co, D., & Kuhnert, N. (2014). TheHive Project. TheHive Project. https://thehive-project.org/

Ghemawat, S., Gobioff, H., & Leung, S. T. (2003). The Google File System. In SOSP

GoogleDevelopers. Lecture 5: Parallel Graph Algorithms with MapReduce, 28 Aug. 2007; http://youtube.com/watch?v=BT-piFBP4fE.

Graph Twiddling in a MapReduce World,J Cohen,2009.

Hadoop, A. (2006). Apache Hadoop. http://hadoop.apache.org/

Huang, D., Shi, X., Ibrahim,S., Lu Lu, Hongzhang Liu, Wu, S. Jin., H. (2010) MR-Scope: A Real-Time Tracing Tool for MapReduce

Hunt, P. and Konar, M. Yahoo! Grid, Junqueira, F. P., and Reed, B. (2010). ZooKeeper: Wait-free coordination for Internet-scale systems,In Yahoo! Research

IBM. (1911). us-en_software_HP. https://www.ibm.com/products/software

Isard, M., Budiu, M., Yu, Y., Birrell, A., & Fetterly, D. (2009). Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In Microsoft Research Labs.

Lamport, L. (2001, November). Paxos Made Simple. https://lamport.azurewebsites.net/pubs/paxos-simple.pdf

26. Lee, K. H., Choi, H., Chung, Y. D., & Moon, B. (2011). *Parallel Data Processing with MapReduce: A Survey*. ACM SIGMOD.

Malewicz, G., Austern, M. H., Bik, A.J.C., Dehnert, J.C., Horn, I., Leiser, N., and Czajkowski, G.(2011).Pregel: A System for Large-Scale Graph Processing,In Google Inc.,

Melnik, S., Gubarev, A., Long, J.J., Romer, G., Shivakumar, S., Tolton, M., Vassi-lakis, T.(2010) Dremel: Interactive Analysis of Web-Scale Datasets

Microsoft. (2017, June 8). DryadLINQ. Microsoft Research. https://www.microsoft.com/en-us/research/project/dryadlinq/?from=http%3A%2F%2Fresearch.microsoft.com%2Fresearch%2Fsv%2Fdryadlinq%2F

Olston, C., Reed, B., Srivastava, U., Kumar, R., & Tomkins, A. (2008). Pig Latin: a Not-so-Foreign Language for Data Processing. In *Proceeding of ACM SIGMOD*.

Pei, L. (2020). Home - Apache Hive - Apache Software Foundation. Confluence. https://cwiki.apache.org/confluence/display/Hive/Home

Skiena, S. (2008). The Stanford GraphBase. The Stony Brook Algorithm Repository. https://www3.cs.stonybrook.edu/%7Ealgorith/implement/graphbase/implement.shtml

Stonebraker, M., Abadi, D., DeWitt, D. J., Madden, S., Paulson, E., Pavlo, A., & Rasin, A. (2010, January). MapReduce and Parallel DBMSs: Friends or Foes? *Communications of the ACM*, *53*(1), 64–71. doi:10.1145/1629175.162919

The Apache Software Foundation. (2007). The Pig Project. Apache Hadoop. https://pig.apache.org/

The Stanford GraphBase: A Platform for Combinatorial Computing, Knuth, d. E. 1993

Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Zhang, N., Antony, S., Liu, H., Murthy, R. (2010) Hive - A Petabyte Scale Data Warehouse Using Hadoop

Twister, B. (2021). Boeing Twister | Carahsoft. Carahsoft. https://www.carahsoft.com/boeing-twister

Wikipedia contributors. (2021, March 14). Language Integrated Query. Wikipedia. https://en.wikipedia.org/wiki/Language_Integrated_Query

Yu, Y. Isard, M., Fetterly, D., Budiu, M., Erlingsson1, I., Kumar, P., Currey, G.J.(2011). DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language,In Microsoft Research Silicon Valley joint afliation, Reykjavk University, Iceland 2011

Zhou, J., Bruno, N., Wu, M.C., Per-Ake Larson, Chaiken, R., Shakib. D. (2012) SCOPE: parallel databases meet MapReduce

Uttama Garg is an assistant professor at Chandigarh University at Mohali. She has 5+ years of experience in teaching and research.