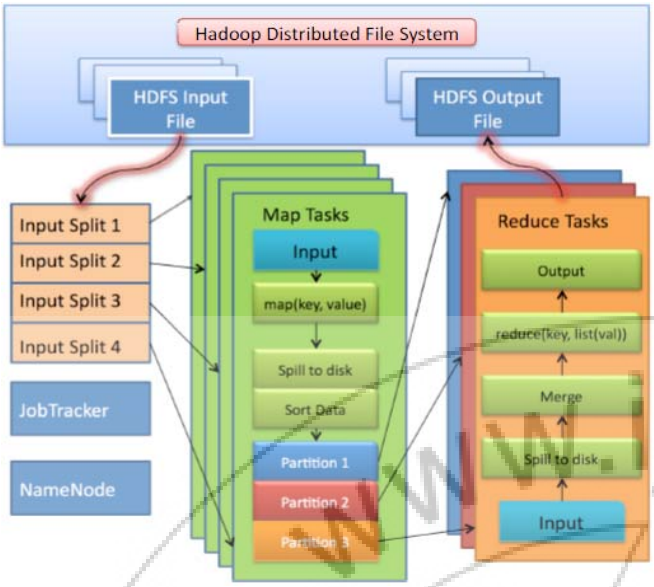

Figure 4: Hadoop MapReduce and HDFS

Hadoop MapReduce[1] jobs read their input data from HDFS and also write their output to it. HDFS has been very popular because of its scalability, reliability and capability of storing very large files. HDFS [10] applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed. This assumption simplifies data coherency issues and enables high throughput data access. A Map/Reduce application or a web crawler application fits perfectly with this model. There is a plan to support appending-writes to files in the future.

B. HBase

Apache HBase[8] is distributed column based database - like layer built on Hadoop designed to support billions of messages per day, HBase is massively scalable and delivers fast random writes as well as random and streaming reads. It also provides row-level atomicity guarantees, but no native cross-row transactional support. From a data model perspective, column-orientation gives extreme flexibility in storing data and wide rows allow the creation of billions of indexed values within a single table. HBase[12] is ideal for workloads that are write-intensive, need to maintain a large amount of data, large indices, and maintain the flexibility to scale out quickly.

C. Hive

Hive [8] is a technology developed at Facebook that turns Hadoop into a data warehouse [8] complete with a dialect of SQL for querying. Being a SQL dialect, HiveQL[11] is a declarative language. The architecture of Hive is shown in fig 5. In PigLatin[6], you specify the data flow, but in Hive we describe the result we want and Hive figures out how to build a data flow to achieve that result. Unlike Pig, in Hive a schema is required, but you are not limited to only one schema. Like PigLatin and the SQL, HiveQL[4] itself is a relationally complete language but it is not a Turing complete language. It can also be extended through UDFs just like Piglatin to be a Turing complete. Hive is a technology for turning the Hadoop into a data warehouse, complete with SQL dialect for querying it. Hive works in terms of tables. There are two kinds of tables you can create: managed tables whose data is managed by Hive and external tables whose data is managed outside of Hive. Another option Hive provides for speeding up queries is bucketing. Like partitioning, bucketing splits up the data by a particular column, but in bucketing you do not specify the individual values for that column that correspond to buckets, you simply say how many buckets to split the table into and let Hive figure out how to do it.

D. PIG

Pig was initially developed at Yahoo Research around 2006 but moved into the Apache Software Foundation in 2007. Pig consists of a language and an execution environment. Pig's language, called as PigLatin[6], is a data flow language - this is the kind of language in which you program by connecting things together. Pig can operate on complex data structures, even those that can have levels of nesting. Unlike SQL, Pig does not require that the data must have a schema, so it is well suited to process the unstructured data. But, Pig can still leverage the value of a schema if you want to supply one. PigLatin is relationally complete like SQL, which means it is at least as powerful as a relational algebra. Turing completeness requires conditional constructs, an infinite memory model, and looping constructs. PigLatin is not Turing complete on itself, but it can be Turing complete when extended with User--Defined Function.

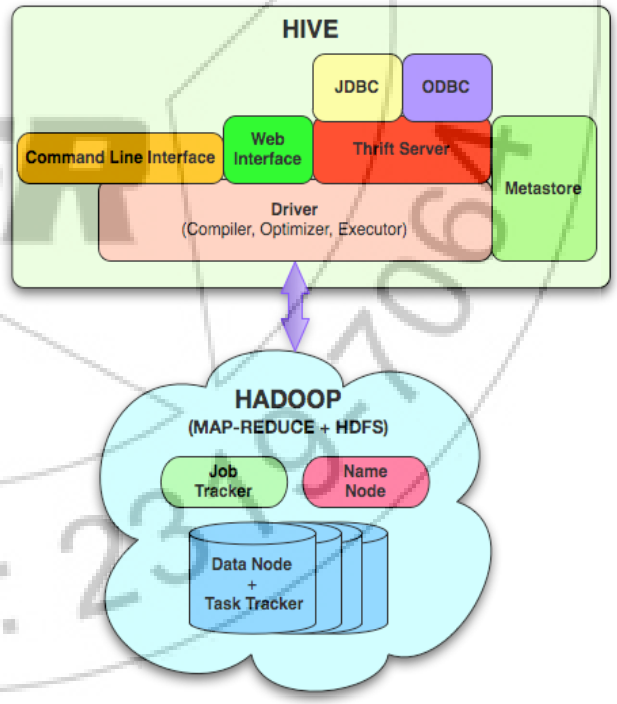

Figure 5: Hive System Architecture

There are three different ways to run Pig. You can run your PigLatin[13] code as a script, just by passing the name of your script file to the pig command. You can run it interactively through the grunt command line launched using Pig with no script argument. Finally, you can call into Pig from within Java using Pig's embedded form.

Licensed Under Creative Commons Attribution CC BY

E. YARN

YARN [4], Yet Another Resource Negotiator, is included in the latest Hadoop release and its goal is to allow the system to serve as a general data-processing framework. It supports programming models [7] other than MapReduce, while also improving scalability and resource utilization. YARN makes no changes to the programming model or to HDFS. It consists of a re-designed runtime system, aiming to eliminate the bottlenecks of the master-slave architecture [4]. The responsibilities of the Job Tracker are split into two different processes, the Resource Manager and the Application Master. The Resource Manager handles resources dynamically, using the notion of containers, instead of static Map/Reduce slots. Containers are configured based on information about available memory, CPU and disk capacity. It also has a pluggable scheduler, which can use different strategies to assign tasks to available nodes. The Application Master is a framework-specific process, meaning that it allows other programming models to be executed on top of YARN, such as MPI or Spark [12]. It negotiates resources with the Resource Manager and supervises the scheduled tasks.

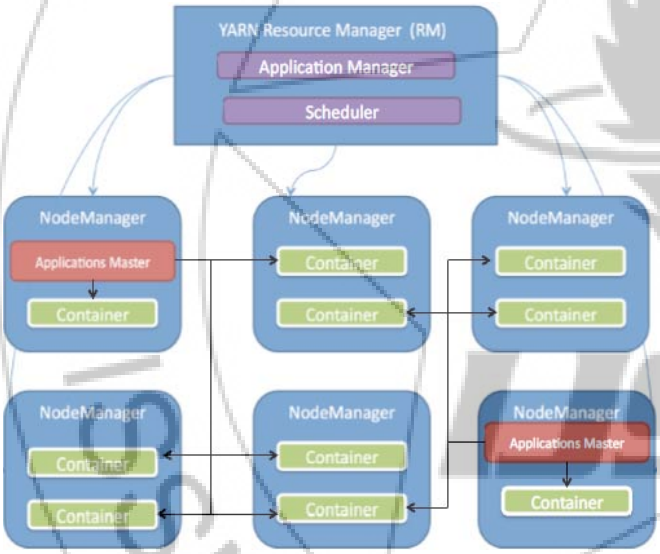

Figure 6: YARN System Architecture

Fig 6 shows the System Architecture of YARN. The lifecycle of a Map-Reduce job running in YARN [7] is as follows:

1. Hadoop MR JobClient submits the job to the YARN ResourceManager (ApplicationsManager) rather than to the Hadoop MapReduce [2] JobTracker.
2. The YARN RM-ASM negotiates the container for the MR Application Master with the Scheduler (RMS) and then launches the MR AM for the job.
3. The MR AM starts up and registers with the RMASM.
4. The Hadoop MapReduce JobClient polls the ASM to obtain information about the MR AM and then directly talks to the AM for status, counters, etc.
5. The MR AM computes input-splits and constructs resource requests for all maps and reducers to the YARN Scheduler.
6. The MR AM runs the requisite job setup APIs of the Hadoop MR Output Committer for the job.
7. The MR AM submits the resource requests for the map/reduce tasks to the YARN Scheduler (RM-S), gets containers from the RM and schedules appropriate tasks on the obtained containers by working with the Node Manager for each container.
8. The MR AM monitors the individual tasks to completion, requests alternate resource if any of the tasks fail or stop responding.
9. The MR AM also runs appropriate task cleanup code of completed tasks of the Hadoop MR Output Committer.
10. Once the entire map and reduce tasks are complete, the MR AM runs the requisite job commit APIs of the Hadoop MR Output Committer for the job.
11. The MR AM then exits since the job is complete.

5. Conclusion and Future Scope

The MapReduce programming model has been successfully used at Google for many different purposes the model is easy to use, even for programmers without experience with parallel and distributed systems, since it hides the details of parallelization, fault tolerance, locality optimization, and load balancing. Second, a large variety of problems are easily expressible as MapReduce computations. MapReduce is easy to parallelize and distribute computations and to make such computations fault tolerant. And there are extensive list of products and projects that either extend Hadoop's functionality or expose some existing capability in new ways.

MapReduce model specifically, are an active research area, still at its infancy. Currently, the interest for MapReduce is at its peak and there exist a lot of problems and challenges still to be addressed. There lies a bright future ahead for Big Data, as businesses and organizations realize more and more the value of the information they can store and analyze. Developing ways to process the vast amounts of data available drives business innovation, health discoveries, science progress and allows us to find novel ways to solve problems, which we considered very hard or even impossible in the past.

References

[1] Apache Hadoop. Available at http://hadoop.apache.org
[2] Apache HDFS. Available at http://hadoop.apache.org/hdfs
[3] Apache Hive. Available at http://hive.apache.org
[4] Apache HBase. Available at http://hbase.apache.org
[5] A community white paper developed by leading researchers across the United States "Challenges and Opportunities with Big Data"
[6] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu and Raghotham Murthy "Hive – A Petabyte Scale Data Warehouse Using Hadoop" By Facebook Data Infrastructure Team
[7] dhruba,jssarma,jgray,kannan,nicolas,hairong,krangana than,dms,aravind.menon,rash,rodrigo,amitanand.s "Apache Hadoop Goes Realtime at Facebook"

Licensed Under Creative Commons Attribution CC BY

SIGMOD .'11, June 12.–16, 2011, Athens, Greece. Copyright 2011 ACM 978-1-4503-0661-4/11/06

[8] Jeffrey Dean and Sanjay Google, Inc." MapReduce: Simplified Data Processing on Large Clusters"

[9] Kyuseok Shim Seoul National University shim@ee.snu.ac.kr "MapReduce Algorithms for Big Data Analysis"

[10] Sanjeev Dhawan1, Sanjay Rathee2, Faculty of Computer Science & Engineering, Research Scholar " Big Data Analytics using Hadoop Components like Pig and Hive" AIJRSTEM 13- 131; © 2013, AIJRSTEM.

[11] Vasiliki Kalavri, Vladimir VlassovKTH The Royal Institute of Technology Stockholm, Sweden kalavri@kth.se "MapReduce: Limitations, Optimizations and Open Issues". TrustCom/ISPA/IUCC,Page1031-1038,IEEE,(2013)

[12] Vinod Kumar, Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal,Mahadev, Konar, Robert Evans,Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, Eric Baldeschwieler "Apache Hadoop YARN: Yet Another Resource Negotiator" SoCC'13, 1–3 Oct. 2013, Santa Clara, California, USA. ACM978-1-4503-2428-1

Licensed Under Creative Commons Attribution CC BY