INTERNATIONAL JOURNAL OF
RESEARCH AND CURRENT DEVELOPMENT
Available Online at http://www.journalijrcd.com
vol. 2, Issue, 1, pp.23-25, January, 2017

IJRCD

ISSN: XXX-XXXX RESEARCH ARTICLE

PARALLEL COMPUTING OF BIG DATA USING MAPREDUCE

Asst.Prof . Swetha, G[1] and Asst.Prof. Radhika, V[2]

Department of Computer Science & Engineering, Teegala Krishna Reddy Engineering College
Hyderabad, T.S-500 097, India

ARTICLE INFO

Article History:
Received 18th October, 2016
Received in revised form 19th November, 2016
Accepted 24th December, 2016
Published online 28th January, 2017

Keywords:

Introduction, Lambda Architecture, Parallel Computing, MapReduce

ABSTRACT

Big data is a term for data sets that are so large or complex that traditional data processing applications are inadequate. Extracting useful information from dataset measuring in gigabytes and tetra bytes is a real challenge for data miners. In this paper we discuss and analyze opportunities and challenges for efficient parallel data processing. Big Data is the next frontier for innovation, competition, and productivity, and many solutions continue to appear, partly supported by the considerable enthusiasm around the Map Reduce paradigm for large-scale data analysis. Map Reduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster. We review various parallel and distributed programming paradigms, analyzing how they fit into the Big Data era, and present modern emerging paradigms and frameworks. With “Big Data” now becoming a reality, more programmers are interested in building programs on the parallel model — and they often find SQL an unfamiliar and restrictive way to wrangle data and write code. The biggest game-changer to come along is Map Reduce, the parallel programming framework that has gained prominence thanks to its use at web search companies.

INTRODUCTION

Big data can be stored, acquired, processed, and analyzed in many ways. Every big data source has different characteristics, including the frequency, volume, velocity, type, and veracity of the data. When big data is processed and stored, additional dimensions come into play, such as governance, security, and policies. Choosing an architecture and building an appropriate big data solution is challenging because so many factors have to be considered. This "Big data architecture and patterns" series presents a structured and pattern-based approach to simplify the task of defining an overall big data architecture. Because it is important to assess whether a business scenario is a big data problem, we include pointers to help determine which business problems are good candidates for big data solutions.

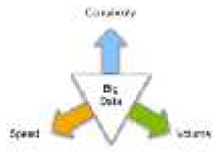

Fig-1

Big Data=Transactions+Ineractions+Observations

Big data is more real-time in nature than traditional DW applications. Traditional DW architectures (e.g. Exadata, Teradata) are not well-suited for big data apps. Shared nothing, massively parallel processing, scale out architectures are well-suited for big data apps.

*✉ ***Corresponding author:*** **Swetha, G**
Department of Computer Science & Engineering, Teegala Krishna Reddy Engineering College Hyderabad, T.S-500 097, India

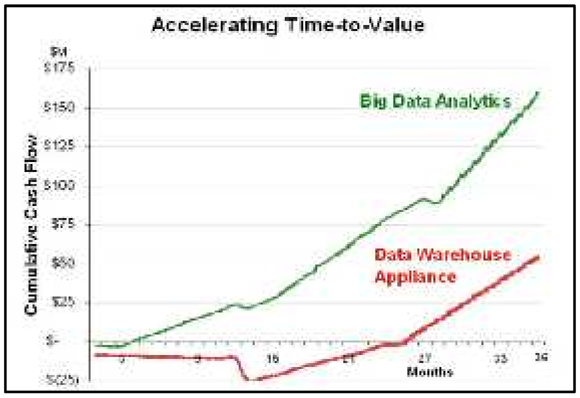

Fig-2

Lambda architecture

The rise of lambda architecture is correlated with the growth of big data, real-time analytics, and the drive to mitigate the latencies of map-reduce.[1]Lambda architecture depends on a data model with an append-only, immutable data source that serves as a system of record.[2] It is intended for ingesting and processing time stamped events that are appended to existing events rather than overwriting them. State is determined from the natural time-based ordering of the data. Lambda architecture describes a system consisting of three layers: batch processing, speed (or real-time) processing, and a serving layer for responding to queries.[3].The processing layers ingest from an immutable master copy of the entire data set.

Batch layer

The batch layer precomputes results using a distributed processing system that can handle very large quantities of data. The batch layer aims at perfect accuracy by being able to process *all* available data when generating views. This means it can fix any errors by recomputing based on the complete data set, then updating existing views. Output is typically stored in a read-only database, with updates completely replacing existing precomputed views.[3]

Speed layer

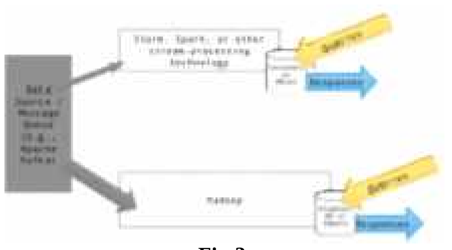

Fig-3

Diagram showing the flow of data through the processing and serving layers of lambda architecture.

The speed layer processes data streams in real time and without the requirements of fix-ups or completeness. This layer sacrifices throughput as it aims to minimize latency by providing real-time views into the most recent data. Essentially, the speed layer is responsible for filling the "gap" caused by the batch layer's lag in providing views based on the most recent data. This layer's views may not be as accurate or complete as the ones eventually produced by the batch layer, but they are available almost immediately after data is received, and can be replaced when the batch layer's views for the same data become available.[3] Stream-processing technologies typically used in this layer include Apache Storm, SQLstream and Apache Spark. Output is typically stored on fast NoSQL databases.[5]

Serving layer

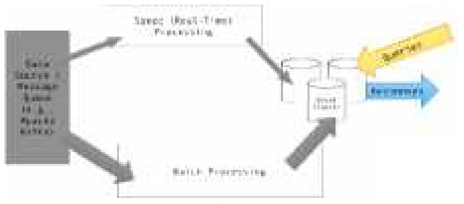

Fig-4

Diagram showing a lambda architecture with a Druid data store.
Output from the batch and speed layers are stored in the serving layer, which responds to ad-hoc queries by returning precomputed views or building views from the processed data. Examples of technologies used in the serving layer include Druid, which provides a single cluster to handle output from both layers.[7] Dedicated stores used in the serving layer include Apache Cassandra or Apache HBase for speed-layer output, and Elephant DB or Cloudera Impala for batch-layer output.[2]

Parallel Computing

Most high performance platforms are created by connecting multiple nodes together via a variety of network topologies. Specialty appliances may differ in the specifics of the configurations, as do software appliances. However, the general architecture distinguishes the management of computing resources (and corresponding allocation of tasks) and the management of the data across the network of storage nodes, as is seen in the figure below:

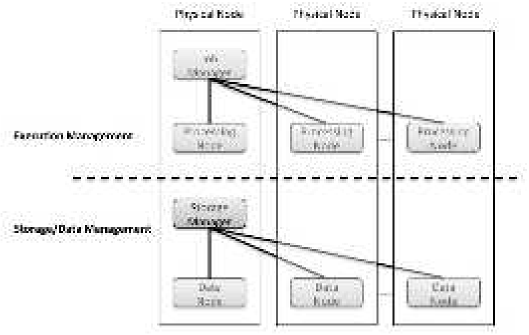

Fig-5

Typical organization of resources in a big data platform.
In this configuration, a master job manager oversees the pool of processing nodes, assigns tasks, and monitors the activity. At the same time, a storage manager oversees the data storage pool and distributes datasets across the collection of storage resources. While there is no *a priori* requirement that there be any colocation of data and processing tasks, it is beneficial from a performance perspective to ensure that the threads process data that is stored in a way that is directly local to the node upon which the thread executes, or is stored on a node that is close to it. Reducing the costs of data access latency through co-location improves performance speed.

MapReduce

The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The Algorithm

- Generally MapReduce paradigm is based on sending the computer to where the data resides!
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
    - ***Map stage****:* The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
    - ***Reduce stage****:* This stage is the combination of the **Shuffle** stage and the **Reduce** stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.

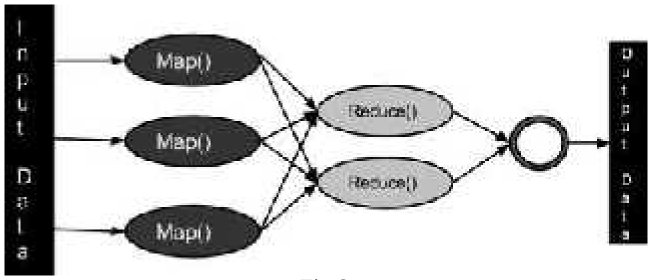

Fig-6

The MapReduce framework operates on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types. The key and the value classes should be in serialized manner by the framework and hence, need to implement the Writable interface. Additionally, the key classes have to implement the Writable-Comparable interface to facilitate sorting by the framework. Input and Output types of a MapReduce job: (Input) <k1, v1> -> map -> <k2, v2>-> reduce -> <k3, v3>(Output).

	Input	**Output**
Map	<k1, v1>	list (<k2, v2>)
Reduce	<k2, list(v2)>	list (<k3, v3>)

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

References

1. Abadi, D.J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., Zdonik, S.: Aurora: a new model and architecture for data stream management. VLDB J. *Int. J. Very Large Data Bases* 12(2), 120–139 (2003)CrossRef
2. Beckhusen, R.: So it begins: Darpa sets out to make computers that can teach themselves. http://www.wired.com/dangerroom/2013/03/darpa-machine-learning-2/all/1 (2013). Accessed 18 Apr 2013
3. Bell, G., Hey, T., Szalay, A.: Beyond the data deluge. Science 323(5919), 1297–1298 (2009)CrossRef
4. Berkan, R.: Big Data: a blessing and a curse. http://www.searchenginejournal.com/big-data-blessing/53528/ (2012). Accessed 15 Apr 2013
5. Cisco: Cisco visual networking index: Global mobile data traffic forecast update, 2011–2016. http://www.cisco.com/ (2012). Accessed 16 Apr 2013
6. Cortes, C., Fisher, K., Pregibon, D., Rogers, A.: Hancock: a language for extracting signatures from data streams. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 9–17. ACM (2000)
7. Darema, F.: The spmd model: past, present and future. In: Recent Advances in Parallel Virtual Machine and Message Passing Interface, pp. 1–1. Springer, Berlin (2001)
8. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM 51(1), 107–113 (2008)CrossRef
9. Dorier, M., Antoniu, G., Cappello, F., Snir, M., Orf, L.: Damaris: how to efficiently leverage multicore parallelism to achieve scalable, jitter-free i/o. In: 2012 IEEE International Conference on Cluster Computing (CLUSTER), pp. 155–163. IEEE (2012)
10. Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.H., Qiu, J., Fox, G.: Twister: a runtime for iterative mapreduce. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, pp. 810–818. ACM (2010)
11. Ekanayake, J., Pallickara, S., Fox, G.: Mapreduce for data intensive scientific analyses. In: IEEE Fourth International Conference on eScience 2008 (eScience'08), pp. 277–284. IEEE (2008)
12. Fox, G., Bae, S.H., Ekanayake, J., Qiu, X., Yuan, H.: Parallel data mining from multicore to cloudy grids. In: High Performance Computing Workshop, vol. 18, pp. 311–340 (2009)
13. Frank, C.: Forbes: Improving Decision Making in the World of Big Data. http://www.forbes.com/sites/christopherfrank/2012/03/25/improving-decision-making-in-the-world-of-big-data/ (2012). Accessed 15 Apr 2013
14. Gainaru, A., Cappello, F., Kramer, W.: Taming of the shrew: modeling the normal and faulty behaviour of large-scale hpc systems. In: 2012 IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS), pp. 1168–1179. IEEE (2012)
15. Ghemawat, S., Gobioff, H., Leung, S.T.: The google file system. In: ACM SIGOPS Operating Systems Review, vol. 37, pp. 29–43. ACM (2003)