Scalable and efficient community detection algorithm using GIRAPH Technology

Sayali Shivaji More

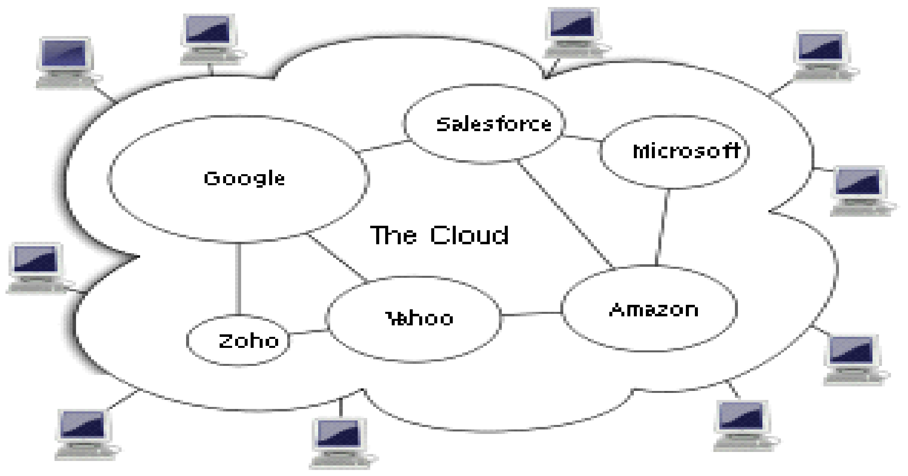
Department of Computer Science and Engineering, SVERI's College of Engineering

Final Year Engineering Student

1] Introduction

Computing in its purest form has changed hands multiple times. First, from near the beginning mainframes were predicted to be the future of computing. Indeed mainframes and large scale machines were built and used, and in some circumstances are used similarly today. The trend, however, turned from bigger and more expensive, to smaller and more affordable commodity PCs and servers.

Most of our data is stored on local networks with servers that may be clustered and sharing storage. This approach has had time to be developed into stable architecture, and provide decent redundancy when deployed right. A newer emerging technology, cloud computing, has shown up demanding attention and quickly is changing the direction of the technology landscape. Whether it is Google's unique and scalable Google File System, or Amazon's robust Amazon S3 cloud storage model, it is clear that cloud computing has arrived with much to be gleaned from.



Cloud computing is a style of computing in which dynamically scalable and often virtualize resources are provided as a service over the Internet. Users need not have knowledge of, expertise in, or control over the technology infrastructure in the "cloud" that supports them.

2] Need for large data processing:

We live in the data age. It's not easy to measure the total volume of data stored electronically, but an IDC estimate put the size of the "digital universe" at 0.18 zettabytes in 2006, and is forecasting a tenfold growth by 2011 to 1.8 zettabytes.

Some of the large data processing needed areas include:-

- The New York Stock Exchange generates about one terabyte of new trade data per day.
- Facebook hosts approximately 10 billion photos, taking up one petabyte of storage.
- Ancestry.com, the genealogy site, stores around 2.5 petabytes of data.
- The Internet Archive stores around 2 petabytes of data, and is growing at a rate of 20 terabytes per month.

3] Challenges in distributed computing --- meeting hadoop:

Various challenges are faced while developing a distributed application. The first problem to solve is hardware failure: as soon as we start using many pieces of hardware, the chance that one will fail is fairly high. A common way of avoiding data loss is through replication: redundant copies of the data are kept by the system so that in the event of failure, there is another copy available. This is how RAID works, for instance, although Hadoop’s filesystem, the Hadoop Distributed Filesystem(HDFS), takes a slightly different approach.

The second problem is that most analysis tasks need to be able to combine the data in some way; data read from one disk may need to be combined with the data from any of the other 99 disks. Various distributed systems allow data to be combined from multiple sources, but doing this correctly is notoriously challenging. MapReduce provides a programming model that abstracts the problem from disk reads and writes transforming it into a computation over sets of keys and values.

This, in a nutshell, is what Hadoop provides: a reliable shared storage and analysis system. The storage is provided by HDFS, and analysis by MapReduce. There are other parts to Hadoop, but these capabilities are its kernel.

Hadoop is the popular open source implementation of MapReduce, a powerful tool designed for deep analysis and transformation of very large data sets. Hadoop enables you to explore complex data, using custom analyses tailored to your information and questions. Hadoop is the system that allows unstructured data to be distributed across hundreds or thousands of machines forming shared nothing clusters, and the execution of Map/Reduce routines to run on the data in that cluster. Hadoop has its own file system which replicates data to multiple nodes to ensure if one node holding data goes down, there are at least 2 other nodes from which to retrieve that piece of information. This protects the data availability from node failure, something which iscritical when there are many nodes in a cluster (aka RAID at a server level).

Basic features of HDFS

- Highly fault tolerant
- High throughput
- Suitable for application with large data sets
- Streaming access to file system data
- Can be built out of commodity hard

6] Introduction to MapReduce:

MapReduce is a programming model and an associated implementation for processing and generating largedata sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model.

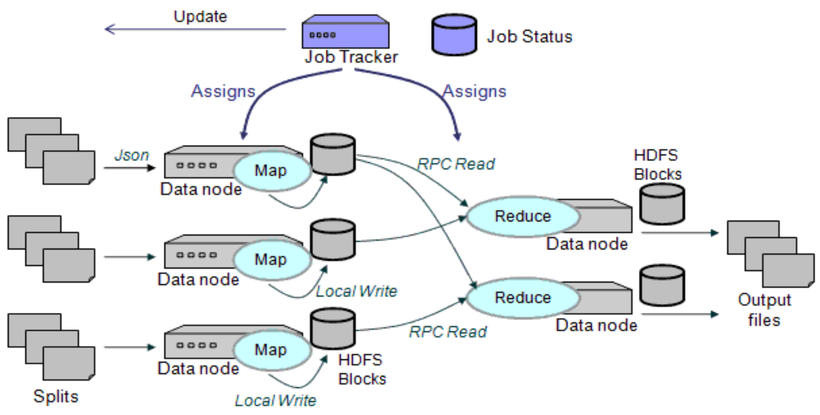
Hadoop MapReduce:

Hadoop Map-Reduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

A Map-Reduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

Typically the compute nodes and the storage nodes are the same, that is, the Map-Reduce framework and the Distributed FileSystem are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.

A MapReduce job is a unit of work that the client wants to be performed: it consists of the input data, the MapReduce program, and configuration information. Hadoop runs the job by dividing it into tasks, of which there are two types: map tasks and reduce tasks. There are two types of nodes that control the job execution process: a jobtracker and a number of tasktrackers. The jobtracker coordinates all the jobs run on the system by scheduling tasks to run on tasktrackers. Tasktrackers run tasks and send progress reports to the jobtracker, which keeps a record of the overall progress of each job. If a tasks fails, the jobtracker can reschedule it on a different tasktracker. Hadoop divides the input to a MapReduce job into fixed-size pieces called input splits, or just splits. Hadoop creates one map task for each split, which runs the userdefined map function for each record in the split.



Having many splits means the time taken to process each split is small compared to the time to process the whole input. So if we are processing the splits in parallel, the processing is better load-balanced if the splits are small, since a faster machine will be able to process proportionally more splits over the course of the job than a slower machine. Even if the machines are identical, failed processes or other jobs running concurrently make load balancing desirable, and the quality of the load balancing increases as the splits become more fine-grained. On the other hand, if splits are too small, then the overhead of managing the splits and of map task creation begins to dominate the total job execution time. For most jobs, a good split size tends to be the size of a HDFS block, 64 MB by default, although this can be changed for the cluster (for all newly created files), or specified when each file is created. Hadoop does its best to run the map task on a node where the input data resides in HDFS. This is called the data locality optimization. It should now be clear why the optimal split size is the same as the block size: it is the largest size of input that can be guaranteed to be stored on a single node. If the split spanned two blocks, it would be unlikely that any HDFS node stored both blocks, so some of the split would have to be transferred across the network to the node running the map task, which is clearly less efficient than running the whole map task using local data. Map tasks write their output to local disk, not to HDFS. Map output is intermediate output: it's processed by reduce tasks to produce the final output, and once the job is complete the map output can be thrown away. So storing it in HDFS, with replication, would be overkill. If the node running the map task fails before the map output has been consumed by the reduce task, then Hadoop will automatically rerun the map task on another node to recreate the map output. Reduce tasks don't have the advantage of data locality—the input to a single reduce task is normally the output from all mappers. In the present example, we have a single reduce task that is fed by all of the map tasks. Therefore the sorted map outputs have to be transferred across the network to the node where the reduce task is running, where they are merged and then passed to the user-defined reduce function. The output of the reduce is normally stored in HDFS for reliability. For each HDFS block of the reduce output, the first replica is stored on the local node, with other replicas being stored on off-rack nodes. Thus, writing the reduce output does consume network bandwidth, but only as much as a normal HDFS write pipeline consume. The dotted boxes in the figure below indicate nodes, the light arrows show data transfers on a node, and the heavy arrows show data transfers between nodes. The number of reduce tasks is not governed by the size of the input, but is specified independently.

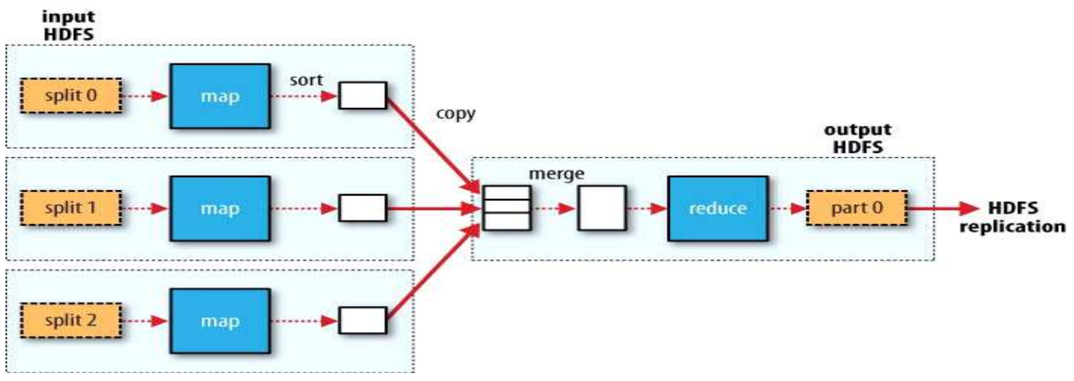


Fig 1: MapReduce data flow with a single reduce task

When there are multiple reducers, the map tasks partition their output, each creating one partition for each reduce task. There can be many keys (and their associated values) in each partition, but the records for every key are all in a single partition. The partitioning can be controlled by a user-defined partitioning function, but normally the default partitioner—which buckets keys using a hash function—works very well. This diagram makes it clear why the data flow between map and reduce tasks is colloquially known as “the shuffle,” as each reduce task is fed by many map tasks. The shuffle is more complicated than this diagram suggests, and tuning it can have a big impact on job execution time. Finally, it’s also possible to have zero reduce tasks. This can be appropriate when you don’t need the shuffle since the processing can be carried out entirely in parallel.

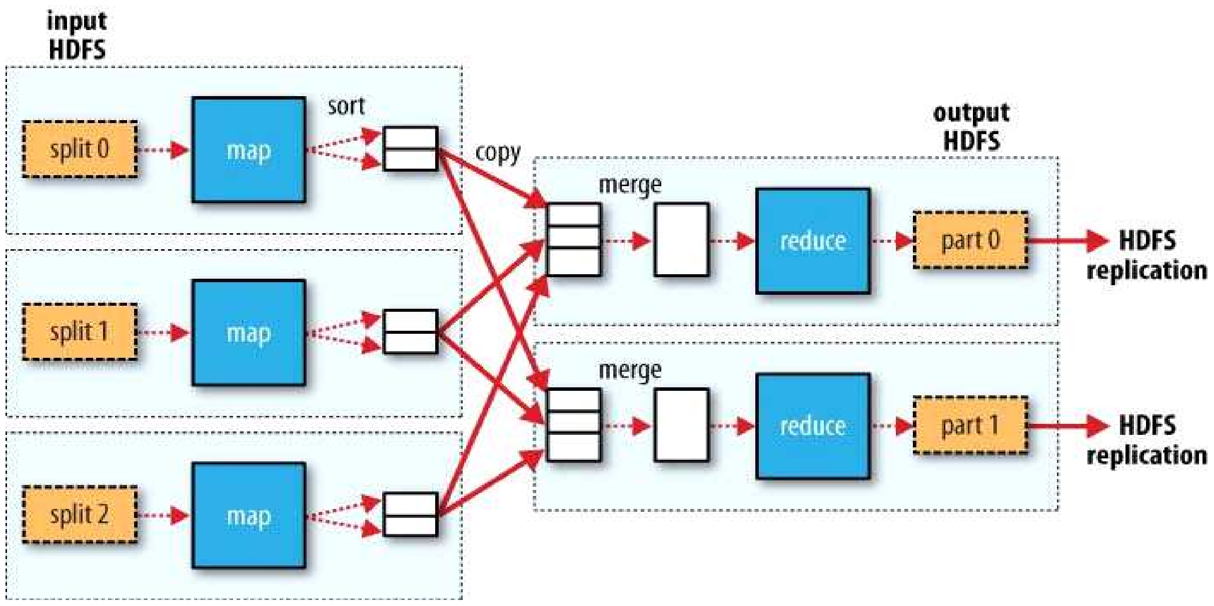


Fig 2: MapReduce data flow with multiple reduce tasks

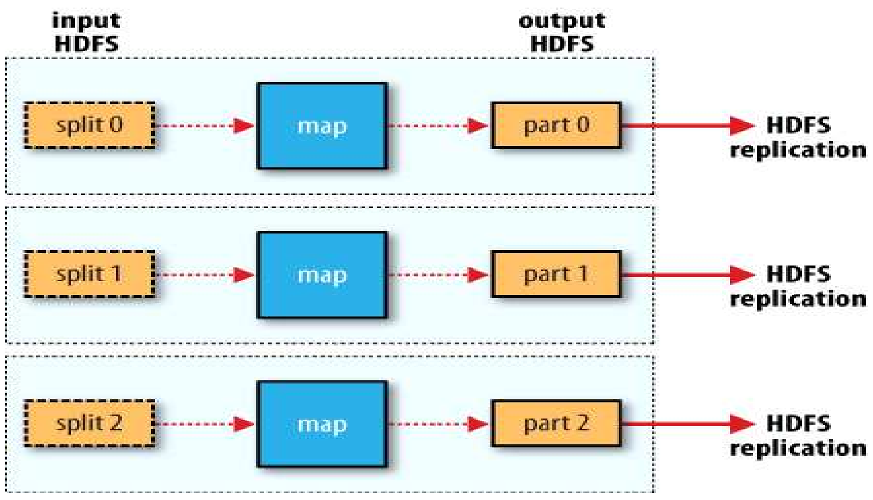


Fig 3: MapReduce data flow with no reduce tasks

7.1] Design of HDFS:

HDFS is a file system designed for storing very large files with streaming data access patterns, running on clusters on commodity hardware. Let's examine this statement in more detail:

- Very large files

"Very large" in this context means files that are hundreds of megabytes, gigabytes, or terabytes in size. There are Hadoop clusters running today that store petabytes of data.*

- Streaming data access

HDFS is built around the idea that the most efficient data processing pattern is a write-once, read-many-times pattern.

Commodity hardware

Hadoop doesn't require expensive, highly reliable hardware to run on. It's designed to run on clusters of commodity hardware (commonly available hardware available from multiple vendors†) for which the chance of node failure across the cluster is high, at least for large clusters. HDFS is designed to carry on working without a noticeable interruption to the user in the face of such failure. It is also worth examining the applications for which using HDFS does not work so well. While this may change in the future, these are areas where HDFS is not a good fit today:

Low-latency data access

Applications that require low-latency access to data, in the tens of milliseconds range, will not work well with HDFS. Remember HDFS is optimized for delivering a high throughput of data, and this may be at the expense of latency. HBase (Chapter 12) is currently a better choice for low-latency access.

Lots of small files

Since the namenode holds filesystem metadata in memory, the limit to the number of files in a filesystem is governed by the amount of memory on the namenode. As a rule of thumb, each file, directory, and block takes about 150 bytes. So, for example, if you had one million files, each taking one block, you would need at least 300 MB of memory. While storing millions of files is feasible, billions is beyond the capability of current hardware.

- Multiple writers, arbitrary file modifications

Files in HDFS may be written to by a single writer. Writes are always made at the end of the file. There is no support for multiple writers, or for modifications at arbitrary offsets in the file. (These might be supported in the future, but they are likely to be relatively ineffic8] Prominent users:

- Yahoo!

On February 19, 2008, Yahoo! Inc. launched what it claimed was the world's largest Hadoop production application. The Yahoo! Search Webmap is a Hadoop application that runs on a more than 10,000 core Linux cluster and produces data that is now used in every Yahoo! Web search query.

There are multiple Hadoop clusters at Yahoo!, and no HDFS filesystems or MapReduce jobs are split across multiple datacenters. Every hadoop cluster node bootstraps the Linux image, including the Hadoop distribution. Work that the clusters perform is known to include the index calculations for the Yahoo! search engine.

On June 10, 2009, Yahoo! made available the source code to the version of Hadoop it runs in production. Yahoo! contributes back all work it does on Hadoop to the open-source community. The company's developers also fix bugs and provide stability improvements internally, and release this patched source code so that other users may benefit from their effort.

- Facebook

In 2010 Facebook claimed that they had the largest Hadoop cluster in the world with 21 PB of storage. On July 27, 2011 they announced the data had grown to 30 PB. On June 13, 2012 the announced the data had grown to 100 PB. On November 8, 2012 they announced the warehouse grows by roughly half a PB per day.

Other users

Besides Facebook and Yahoo!, many other organizations are using Hadoop to run large distributed computations.

- Amazon.com
- Ancestry.com
- Akamai
- American Airlines
- AOL
- Apple
- eBay
- Federal Reserve Board of Governors

8] Advantages

- Distribute data and computation. The computation local to data prevents the network overload.
- Linear scaling in the ideal case. It used to design for cheap, commodity hardware.
- Simple programming model. The end-user programmer only writes map-reduce tasks.
- Flat scalability.
- HDFS store large amount of information
- HDFS is simple and robust coherency model
- That is it should store data reliably.

9] Disadvantages

- Rough manner: - Hadoop Map-reduce and HDFS are rough in manner. Because the software under active development.
- Programming model is very restrictive: - Lack of central data can be preventive.
- Joins of multiple datasets are tricky and slow: - No indices! Often entire dataset gets copied in the process.

- Cluster management is hard:- In the cluster, operations like debugging, distributing software, collection logs etc are too hard.
- Still single master which requires care and may limit scaling
- Managing job flow isn't trivial when intermediate data should be kept
- Optimal configuration of nodes not obvious. Eg: – #mappers, #reducers, mem.limits

10] Conclusion:

In our work, we present a performance evaluation & Map-Reduce implementation details of some complex spatial operations on Hadoop platform. The results demonstrate the feasibility and efficiency of the Map Reduce model and show that small scale cluster has the potential to be applicable to computationally intensive spatial data computations. We showed that executing Map-Reduce version of a spatial Query Involving Spatial join on a small scale cluster clearly outperforms that of single spatial databases in terms of time taken to output desired results. Real world Massive graphs, which can't be manipulated by traditional sequential algorithms on a single machine, can be efficiently processed on a small scale cluster because of their small world property. The Comparative study of Hadoop and Distributed database systems shows that the two technologies are rather complementary with each other and neither of the two is good at what other does.

11] References:

http://www.cloudera.com/hadoop-training-thinking-at-scale

http://developer.yahoo.com/hadoop/tutorial/module1.html

http://hadoop.apache.org/core/docs/current/api/

http://hadoop.apache.org/core/version_control.html