CMPSCI 677 Operating Systems **Spring 2017**

Lecture 24: April 25

Lecturer: Prashant Shenoy *Scribe:* ***Abhishek Somani***

24.1 Big Data Application

Nowadays, large volumes of data (Terabytes and Petabytes) are being collected and analyzed at unprecedented scales. The abundance of data and the need for collecting, storing, and processing them have raised many technical challenges for the Big Data applications. Parallel and distributed computing is a matter of paramount importance especially for mitigating scale and timeliness challenges.

Below are the various big data distributed computing platforms:

24.2 Map Reduce Model

MapReduce is a framework for processing parallelizable problems across large datasets using a large number of computers (nodes), collectively referred to as a cluster (if all nodes are on the same local network and use similar hardware) or a grid (if the nodes are shared across geographically and administratively distributed systems, and use more heterogenous hardware). Processing can occur on data stored either in a filesystem (unstructured) or in a database (structured). MapReduce can take advantage of the locality of data, processing it near the place it is stored in order to minimize communication overhead.

Example: Let's say we have huge dataset of number (or may be words in a document) which need to be sorted (or frequency of words). Suppose we have multiple machines. Then each machine could take a chunk of data and sort it. Later, all these sorted chunks chould be merge together similar to merge sort algorithm. However, this requires huge communication between the machines during merge phase. To overcome this problem, let's say we know the range of numbers. In such a case, we could follow the bucket sort paradigm where each machine sorts a specific range of numbers. This way, inter-machine communication can be reduced.

Question : What if the numbers are not evenly distributed. Then some machines would get more data to sort compared to other machines ?
Answer : This is the disadvantage of the approach. So, it is a design tradeoff.

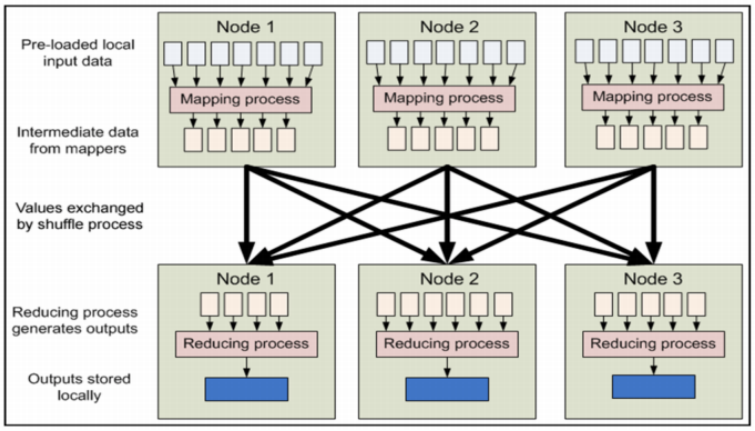

Figure 24.1: Map Reduce example

Map Step : Each worker node applies the "map()" function to the local data, and writes the output to a temporary storage. A master node ensures that only one copy of redundant input data is processed.

Shuffle Step : Worker nodes redistribute data based on the output keys (produced by the "map()" function), such that all data belonging to one key is located on the same worker node.

Reduce Step : Worker nodes now process each group of output data, per key, in parallel.

Its easy for developing distributed parallel processing since developer needs to write only the Map and Reduce functions.

Question : Reduce processes are on certain nodes. Are the map processes only on certain nodes ?
Answer : It is configurable and dynamic. During initial processing, a lot of nodes can be assigned map processes. After Shuffle, the same nodes could also be assigned reduce processes. In general, a node runs a bunch of map and reduce processes.

24.2.1 Other Programming Models

Apache Tez An application framework which allows for a complex directed-acyclic-graph of tasks for processing data.

Microsoft Naiad Naiad is system for data-parallel dataflow computation which attempts to raise the levels of abstraction used by programmers from an imperative sequence of MapReduce-style statements, to involve higher level concepts of loops and streaming.

Spark Apache Spark is a fast and general engine for big data processing, with built-in modules for streaming, SQL, machine learning and graph processing. DAG with in memory resilient data sets.

Flink Apache Flink is an open-source stream processing framework for distributed, high-performing, always-available, and accurate data streaming applications. DAG models extended to cyclic graphs.

24.3 Hadoop Big Data Platform

Hadoop is an implementation of Map-Reduce framework. It is an open-source software framework for storing data and running applications on clusters of commodity hardware. It provides massive storage for any kind

of data, enormous processing power and the ability to handle virtually limitless concurrent tasks or jobs. It has :

- **store managers** : HDFS, HBASE, Kafka, etc (replication for fault tolerence.
- **processing framework** : Map-reduce, Spark, etc
- **resource managers** : Some of the concepts of distributed scheduling are also adpoted since they need to serve multiple users. Example : Yarm, Mesos, etc

24.4 Ecosystem Overview

Based on the requirements different types of frameworks can be used. For example, if user wants to process the data that have lot of graphs then Graph processing framework Giraph can be used. There are machine learning frameworks like MLLib, Oyyx, Tensorflow that are also designed to run to on Hadoop. If a user wants to input data to these distributed processing framework, he could use applications like hive to easily write map-reduce codes. For real time data processing where data is generated continuously by some external source, framework like Spark Storm etc could be used.

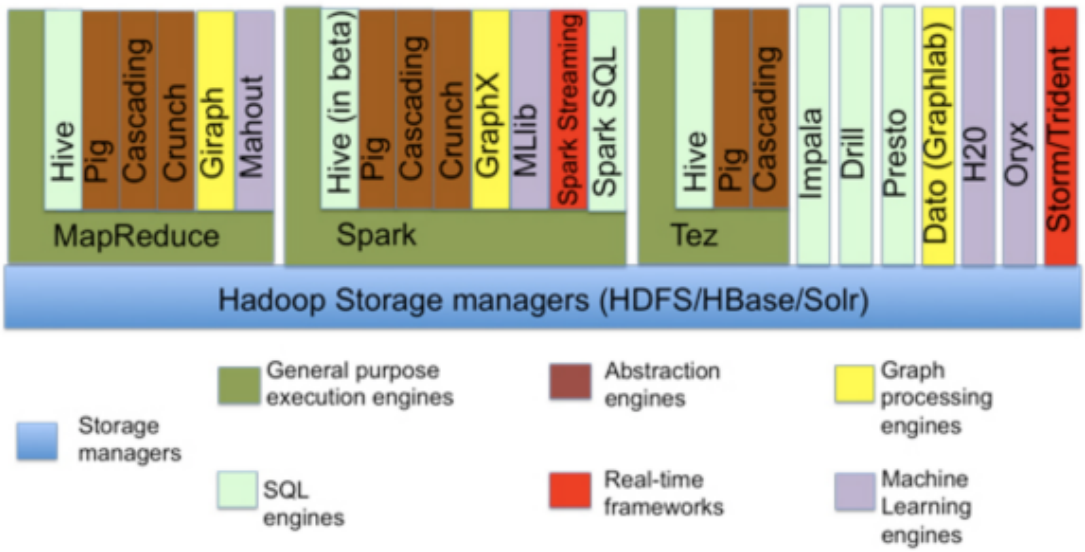

Figure 24.2: Ecosystem overview

24.4.1 Spark Platform

Apache Spark is a powerful open source processing engine built around speed, ease of use, and sophisticated analytics. It was originally developed at UC Berkeley in 2009.

Figure 24.3: Spark Platform

Spark SQL : Spark SQL is a Spark module for structured data processing. It provides a programming abstraction called DataFrames and can also act as distributed SQL query engine. It enables unmodified Hadoop Hive queries to run up to 100x faster on existing deployments and data. It also provides powerful integration with the rest of the Spark ecosystem (e.g., integrating SQL query processing with machine learning).

Spark Streaming Many applications need the ability to process and analyze not only batch data, but also streams of new data in real-time. Running on top of Spark, Spark Streaming enables powerful interactive and analytical applications across both streaming and historical data, while inheriting Sparks ease of use and fault tolerance characteristics. It readily integrates with a wide variety of popular data sources, including HDFS, Flume, Kafka, and Twitter.

MLlib Built on top of Spark, MLlib is a scalable machine learning library that delivers both high-quality algorithms (e.g., multiple iterations to increase accuracy) and blazing speed (up to 100x faster than MapReduce). The library is usable in Java, Scala, and Python as part of Spark applications, so that you can include it in complete workflows.

GraphX GraphX is a graph computation engine built on top of Spark that enables users to interactively build, transform and reason about graph structured data at scale. It comes complete with a library of common algorithms.

Adavatages of Spark

- **Speed :** Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.Apache Spark has an advanced DAG execution engine that supports acyclic data flow and in-memory computing.
- **Ease of Use:** Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it interactively from the Scala, Python and R shells.
- **Generality :** Combine SQL, streaming, and complex analytics.Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.
- **Runs Everywhere:** Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, and S3.

24.4.2 Resilient Distributed Datasets (RDDs)

The Resilient Distributed Dataset is a concept at the heart of Spark. It is designed to support in-memory data storage, distributed across a cluster in a manner that is demonstrably both fault-tolerant and efficient. Fault-tolerance is achieved, in part, by tracking the lineage of transformations applied to coarse-grained sets of data. Efficiency is achieved through parallelization of processing across multiple nodes in the cluster, and minimization of data replication between those nodes. Once data is loaded into an RDD, two basic types of operation can be carried out:

Transformations create a new RDD by changing the original through processes such as mapping, filtering, and more;

Actions, such as counts, which measure but do not change the original data. The original RDD remains unchanged throughout. The chain of transformations from RDD1 to RDDn are logged, and can be repeated in the event of data loss or the failure of a cluster node.

Where possible, these RDDs remain in memory, greatly increasing the performance of the cluster, particularly in use cases with a requirement for iterative queries or processes.

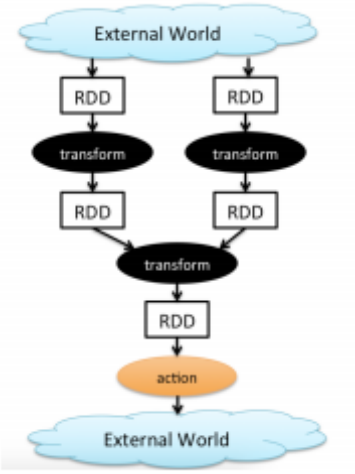

Figure 24.4: RDD

Question : Does Spark do exact as Map-reduce except for caching?
Answer : Spark has potential to do more than that. Map-reduce can be implemented the same way.

24.5 Distributed system Security

The introduction of distributed systems and the use of networks for carrying data between computers is a major factor that has affected security. Security is a complicated business that wasn't given much thought until uses of computer networks increased and the potential for abuse became interesting

24.5.1 Authentication

In simple terms, authentication is identification plus verification. Identification is the procedure whereby an entity claims a certain identity, while verification is the procedure whereby that claim is checked. Thus the correctness of an authentication relies heavily on the verification procedure employed. The entities in a distributed system that can be distinctly identified are collectively referred to as principals. There are three main types of authentication of interest in a distributed system: (A1) message content authentication verifying that the content of a message received is the same as when it was sent; (A2) message origin authentication verifying that the sender of a received message is the same one recorded in the sender field of the message; and (A3) general identity authentication verifying that the a principals identity is as claimed.

24.5.1.1 Authentication Protocols

Authentication Protocol ap1.0 Perhaps the simplest authentication protocol we can imagine is one where Alice simply sends a message to Bob saying she is Alice. The flaw here is obvious - there is no

way for Bob actually to know that the person sending the message I am Alice is indeed Alice. For example,Trudy (the intruder) could just as well send such a message.

Authentication Protocol ap2.0 If Alice has a well-known network address (e.g., an IP address) from which she always communicates, Bob could attempt to authenticate Alice by verifying that the source address on the IP datagram carrying the authentication message matches Alices well-known address. In this case, Alice would be authenticated. IP spoofing can easily fail this protocol.

Authentication Protocol ap3.0 One classic approach to authentication is to use a secret password. We have PINs to identify ourselves to automatic teller machines and login passwords for operating systems. The password is a shared secret between the authenticator and the person being authenticated. Telnet and FTP use password authentication. The security flaw here is clear. If someone eavesdrops on Alices communication, then it can learn Alices password. Lest you think this is unlikely, consider the fact that when you Telnet to another machine and log in, the login password is sent unencrypted to the Telnet server. Someone connected to the Telnet client or servers LAN can possibly sniff (read and store) all packets transmitted on the LAN and thus steal the login password. In fact, this is a well-known approach for stealing passwords

Authentication Protocol ap3.1 for fixing ap3.0 is naturally idea would be to encrypt the password. By encrypting the password, we can prevent someone from learning Alices password. If we assume that Alice and Bob share a symmetric secret key, K_{AB}, then Alice can encrypt the password and send her identification message, I am Alice, and her encrypted password to Bob. Bob then decrypts the password and, assuming the password is correct, authenticates Alice. Bob feels comfortable in authenticating Alice since Alice not only knows the password, but also knows the shared secret key value needed to encrypt the password. The issue here is : the use of cryptography here does not solve the authentication problem. Bob is subject to a replay attack: someone now need only eavesdrop on Alices communication, record the encrypted version of the password, and play back the encrypted version of the password to Bob to pretend that she is Alice.

Authentication Protocol ap4.0 The problem with ap3.1 is that the same password is used over and over again. One way to solve this problem would be to use nonces - one in a life time number. Alice and Bob could agree on a sequence of passwords (or on an algorithm for generating passwords) and use each password only once, in sequence. Ap 4.0 also uses symmetric keys for encryption.

Authentication Protocol ap5.0 A protocol that uses public key cryptograhpy in a manner analogous to the use of symmetric key cryptographty in protocol ap4.0 is protocol ap5.0. Since ap5.0 uses public key techniques, it requires that Bob retrieve Alice's public key. Here, public-key encryption is also known as asymmetric-key encryption. public-key encryption uses two different keys at once – a combination of a private key and a public key. The private key is known only to your computer, while the public key is given by your computer to any computer that wants to communicate securely with it. To decode an encrypted message, a computer must use the public key, provided by the originating computer, and its own private key.