Batch Processing

Background

Most modern companies hold an immense volume of data about things such as user activity or usage patterns, and want to gain insights on it through some massive and long computation. While they could do so via a data warehouse, the range of functionality that they could perform is inherently limited by the SQL query language. Instead, sometimes it is useful to be able to run arbitrary code on tons of unstructured data.

Building search indexes, machine learning data aggregation/recommendations, and ETL processes are all great examples of batch computing coming in handy!

Distributed File Systems

File systems such as Hadoop (open source), which are based off of the Google File System, can be replicated around the world and distributed over many nodes to be able to hold a massive amount of data.

MapReduce

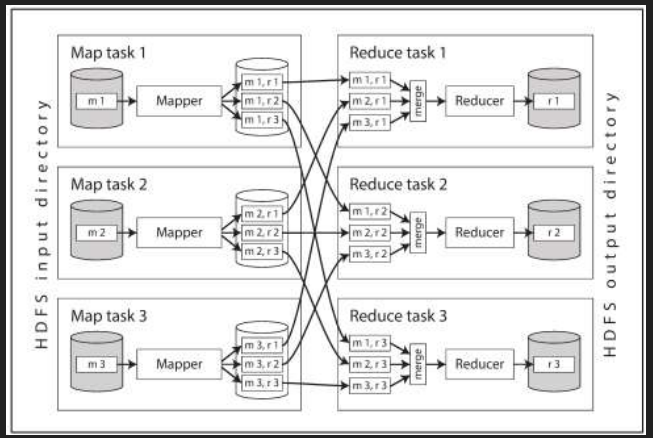
MapReduce is a very simple programming model designed to run on the Hadoop Distributed File System, HDFS, in order to make large computations on tons of data. You only need to write two functions, a mapper and a reducer.

Overview:

- Pass in a ton of input files, return a ton of output files which can be passed into another MapReduce Job
- Computation parallelized among the nodes in the Hadoop cluster, maximize data locality for mappers
- Designed for frequent faults, only restarts a single failed map or reduce job, not all of them
    - Good since these jobs are generally run as background tasks and are often killed

MapReduce Continued

1) Break each input file into a set of records
2) Call the mapper function to extract a key and a value for each record
3) Sort all of the key value pairs by key
    a) Mapper determines which key goes to which reducer partition via a hash function
    b) It then sorts each key within each partition locally
    c) It then sends the list of sorted keys to the proper reducer node
    d) The reducer can then merge together all of the lists that it receives from various mappers
    e) Takes less memory on reducer node to do computation for one key at a time
4) Use the reducer function to take all of the values for a given key and iterate over them handling them in any way you want



Joins in MapReduce

Often times there will be data associations within our input files that we want to resolve, but we do not want to have to reach out to a database every time to make a network call! Let’s look at three different types of joins.

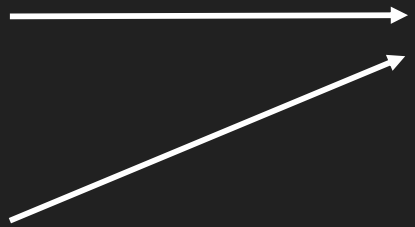
- Sort merge joins
- Broadcast hash joins
- Partitioned hash joins

Sort Merge Joins

Two sets of files that we want to combine to reduce them together. Have mappers on both sets of files, send records of same key to same reducer, reducer merges them together when sorting the keys.

Mapper for browser data
jordan: youtube.com
jordan: xhamster.com
jordan: facebook.com

Mapper for age data
jordan: 21



Reducer for browser/age data
jordan: youtube.com, 21
jordan: xhamster.com, 21
jordan: facebook.com, 21

Broadcast Hash Joins

Same situation as before, but one of the datasets is so small that it can fit in an in-memory hash table on each of the mappers. Just perform the join on the mapper (using the relevant hash table row for the corresponding key) before sending it to the reducer.

Partitioned Hash Joins

Same situation as before, but if the two datasets are partitioned the same way, only load the relevant partition of the smaller dataset into memory on the mapper. Just perform the join on the mapper (using the relevant hash table row for the corresponding key) before sending it to the reducer.

Pitfalls of MapReduce

When chaining MapReduce jobs together, the subsequent one needs to wait for the proceeding one to completely finish in order to start executing.

Bad because:

- Certain nodes can take a really long time to complete their specific map or reduce job compared to other nodes in the MapReduce job
- The result of the MapReduce job is materialized to intermediate state (other files on disk), which is unnecessary and wasteful if those files are never used other than an intermediary between MapReduce jobs (useless writes to disk)
- Every MapReduce job chained together sorts the keys, not always necessary, especially if they've already been sorted by a prior job

DataFlow Engines (i.e. Spark)

- Chain together a bunch of functions called operators, reduces the need for unnecessary mappers
- Entire flow of data (data dependencies) understood from the beginning so that data locality can be maximized
- Use parallel computation
- Do not materialize intermediate state
  - Slightly less fault tolerant than MapReduce as not all intermediate state is written to disk
  - Instead, these engines tend to just recompute it from the most recent prior computations that they still have around, occasionally checkpoint state of computation

Batch Processing Conclusion

Being able to store a ton of data in an unformatted way in a distributed file store and eventually transform it to useful insights makes batch processing extremely useful.

While batch processing was first mainly done via the usage of MapReduce, dataflow engines such as Spark have proven to generally be superior as they do not materialize intermediate state or do unnecessary sorting, as well as optimize for the entire flow of data as opposed to just one map and reduce computation.

While batch processing happens on a bound set of data, we will next cover stream processing, which aims to perform advanced computations on data as it comes in!