Advanced Data Management

Medha Atre

Office: KD-219
atrem@cse.iitk.ac.in

Aug 29, 2016

Announcements

- Assignment-2 is online, it is due on Sept 8 23:59 IST.
- Start working towards definition of your course project. More papers and a few broad level course project ideas will be floated this week.

Recap

- Map-Reduce framework
  - Has one (or more) masters which control other slaves/workers.
  - Applications are typically written as a series of *map* and *reduce* functions.
  - Map-Reduce is a *concept*, not a system. Origins in *functional programming*.
  - Data distribution is governed by master's programming.
- Peer-to-peer framework
  - Flat hierarchy, every compute node in the cluster knows every other node.
  - Based on the principle of Distributed Hash Table (DHT).
  - Data distribution governed by hashing of data values according to the P2P DHT.

Graphs over MapReduce

- Distribution strategies:
  - Vertex-based distribution
  - Locality aware distribution with methods like METIS
  - n-hop distribution
- Distribution often happens as a pre-processing activity and not during the query execution.
- Query planner/optimizer takes the distribution strategy and any indexes into consideration and builds a query plan.
- Each compute node executes that query plan independently.

Data distribution with MapReduce – 1

- Import the data file using standard HDFS *put* or *copyFromLocal* commands.
- Run an *auxiliary* map-reduce job to first fetch this data and redistribute it according to your demand.
- Mapper's output key-value pair what you want to hash the data on.
- Reducer is an identity function.

Data distribution with MapReduce – 2

- Run another data management application on top of HDFS such as HBase, a distributed columnstore.
- Import the data into HBase, which will create a better split.

Data distribution with MapReduce – 3

- Tweak HDFS' *put* functionality to change its way of creating the data splits (for advanced programmers).

Query Processing

- In case of data distribution strategy 1, the output of the previous auxiliary MapReduce job is used as an input to the query execution MapReduce job.
- Joins can be map-side or reduce-side.
- In case of map-side joins, reducers are often identity functions.
- In case of reduce-side joins, mappers are often identity functions.
- Mappers (or reducers) will in turn use standard query processing techniques like sort-merge-joins or hash-joins etc to join over the input data *keys*.

Query Processing

- For cascaded joins, a series of map-reduce jobs are created by the query planner, with any intermediate auxiliary map-reduce jobs to *shuffle* the data required for the next level of join.
- Appreciate why query plan generation and optimization is required even when you do distributed processing of the data, and even when the data is not purely relational or graph shaped.
- E.g., even for the word-count example, having the entire document on one compute node avoids run-time data moving among the nodes.
- Word-count like jobs are less affected because they are run as pre-processor tasks.
- Online queries like relational joins or graph patterns are immediately affected due to perceived time-delays.

Alternate Data Distribution Strategies

- n-hop – a graph vertex and all n-hop vertices from it are stored together.
- METIS – a large graph is partitioned apriori using METIS like tools, and these partitions are then distributed on HDFS.
- For such *non-standard* partitioning strategies, Hadoop's *InputFormat* and *RecordReader* are used in combination to distribute the data according user's strategy – using auxiliary map-reduce phase.
- HDFS' native block-wise data split might not fall on the exact borders of the records. In that case HDFS reads the rest of the block from the other compute node (meta info stored at master) to complete the record read by the map function.

Next Class

We will take an overview of some specific techniques published in the literature, and start learning data management over peer-to-peer networks.