Advanced Data Management

Medha Atre

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

Aug 22, 2016

Announcements

Assignment-2 will be posted on the course webpage by today.

Please complete the Google doc sheet of project groups by today 18:00 IST.

Distributed Systems

- 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.

Map Reduce Philosophy

- Each data item considered as a key-value pair.
- Each unique key is sent (mapped) to a worker. Number of keys are much larger than workers, hence multiple keys will come to one worker.
- Worker's *map* function does processing on this data item, and emits *another* key-value pair. This key-value pair is different from the original mapped key-value pair.
- Again each such unique key is sent to a worker, with multiple keys mapped to one worker.
- The worker combines (*reduces*) values associated with each unique key and emits just a value.

Example – word count


```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

Code taken from MapReduce, OSDI 2004 paper from Google.

Other examples

- Distributed *grep*.
- Count of URL access frequency.
- Reverse web-link graph.
- Term-Vector per host.
- Inverted index – modification of word count.
- Distributed sort.

Join Processing with MapReduce

- Different tables are mapped to the workers based on their table IDs and contents.
- Mappers emit *join-keys (attributes)* and the rest of the tuple contents as respective values along with the table IDs.
- Reducers combine join-keys from different tables and join them.
- Reducers emit join-keys and combined values from different tables.

Pictorial view

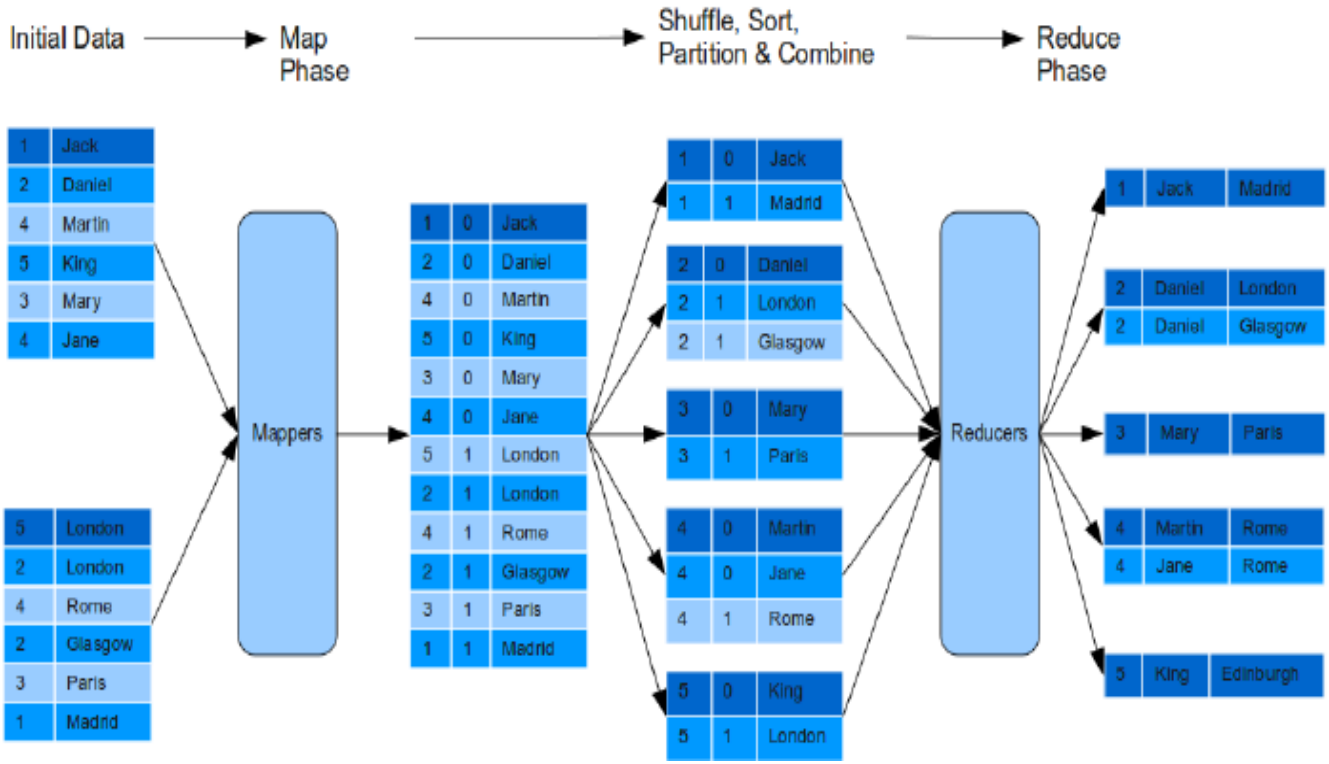

Figure taken from Join Algorithms using Map/Reduce, Jairam Chander, MS thesis, Univ. of Edinburgh.

Map-Reduce with Merge phase

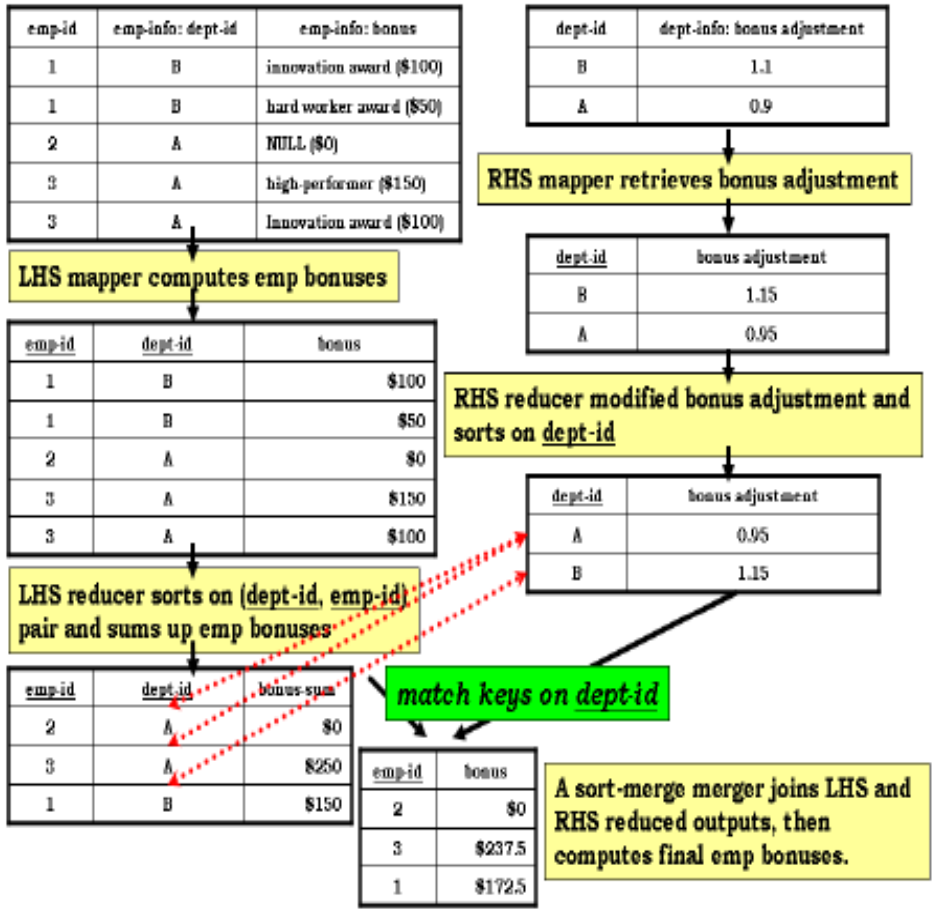

Figure taken from "Map-Reduce-Merge" paper by Yang et al., SIGMOD 2007.

Peer-to-peer

- There is no concept of map and reduce functionality.
- All compute nodes are treated equal, no master, hence every node knows every other node.
- Each compute node is associated with a *hash-id* as a part of the *Distributed Hash Table* (DHT).
- Data distribution is done through *hashing* of data values, e.g., columns of a table or nodes of a graph and mapping the hashed values to the compute nodes based on their hash-id.

DHT ring

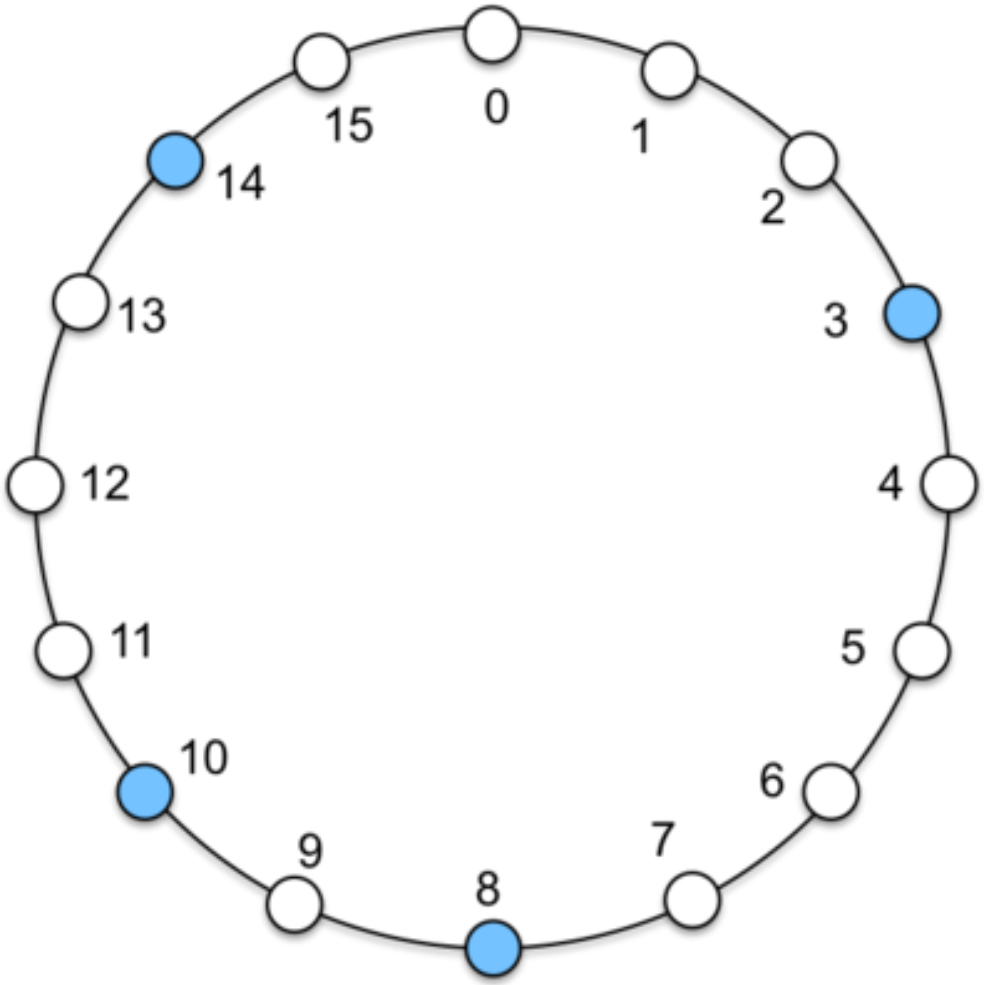

Figure taken from https://www.cs.rutgers.edu/~pxk/417/notes/23-lookup.html

Join Processing in P2P

- Data is assumed to be distributed according to *some* strategy, that the user can choose.
- If the data is not partitioned according to the *join-keys*, it is *re-shuffled* to bring same join keys on the same P2P nodes.
- Reshuffling of the data may continue for queries which have multiple joins, e.g., our running example with two joins.
- Decision of when and how to reshuffle the data during query processing is a part of *query plan generation*.

Next Class

We will learn about some specific map-reduce and DHT based graph processing algorithms and finer details of them.