MapReduce: Simplified Data Processing on Large Clusters

BY:
Nandini Obulappa Gari
Contents

• Map+Reduce
• MR example
• Execution overview
• Fault tolerance
Why not Traditional way:

- Reliability
- Master job
- Scalability
- Critical Path
Map Reduce???

• **What is MapReduce**
  Simple: large –scale data processing

• **Why need of MapReduce.**
  Need of 100’s CPU to process but not to have any inconvenience.

• **MapReduce provides**
  • Automatic parallelization & distribution
  • Fault tolerance
  • I/O scheduling
  • Monitoring & status updates
(Q1) Compared with traditional parallel programming models, such as multithreading and MPI, what are major advantages of MapReduce?
Why Map Reduce!!!!!!!!!!!!

• Highly Scalable.

• Not much knowledge required on parallel and distributed system.

• Fault tolerance.

• Quick processing (less time).

• Reliability.
Map+Reduce

**Mapper**
Input: Input file
Output: Key-value pairs

**Sort**
Input: Key-value pairs
Output: Sorted Key-value pairs

**Reducer**
Input: Sorted KV-pairs
Output: A value

Values
Mapper:
Maps input key/value pairs to a set of intermediate key/value pairs.

Reducer
• Reduces a set of intermediate values which share a key to a smaller set of values.
Small Example: Facebook Common Friends 😊

**INPUT**

- A → B C D
- B → A C D E
- C → A B D E
- D → A B C E
- E → B C D

**For map(A → B C D):**
- (A B) → B C D
- (A C) → B C D
- (A D) → B C D

**For map(B → A C D E):**
- (A B) → A C D E
- (A C) → A C D E
- (A D) → A C D E
- (B D) → A C D E
- (B E) → A C D E

**For map(C → A B D E):**
- (A C) → A B D E
- (B C) → A B D E
- (C D) → A B D E
- (C E) → A B D E

**For map(D → A B C E):**
- (A D) → A B C E
- (B D) → A B C E
- (C D) → A B C E
- (D E) → A B C E

And finally for map(E → B C D):
- (B E) → B C D
- (C E) → B C D
- (D E) → B C D
• Q2. Use Figure 1 to explain a MR program's execution.
Execution Overview

Figure 1: Execution overview
Master

• Responsible for scheduling & managing jobs

• If a task fails to report progress (such as reading input, writing output, etc), crashes, the machine goes down, etc, it is assumed to be stuck, and is killed, and the step is re-launched (with the same input)

• The Master is handled by the framework, no user code is necessary
  • nodes are (or at least should be) deterministic
    • The Master can restart failed nodes.

• If a node is the last step, and is completing slowly, the master can launch a second copy of that node
  • This can be due to hardware issues, network issues, etc.
    • First one to complete wins, then any other runs are killed
(Q3) Describe how MR handles worker and master failures.

Workers Failure:

- MapReduce is resilient towards large scale worker failures.
- The master pings every worker periodically. (no response>failed)
- Map task >>failed worker>>rescheduled>>another worker (even though task executed it reexecutes)
- Completed reduce tasks do not need to be re-executed.

Master Failure:

- A master periodically checkpoints it’s data structures.
- If the master task dies then a new copy is started from the last checkpoint.
- Google MapReduce does not implement any protocol in case of a Master failure since such failures are rare. Therefore the operation will abort in this case.
(Q4) The implementation of MapReduce enforces a barrier between the Map and Reduce phases, i.e., no reducers can proceed until all mappers have completed their assigned workload. For higher efficiency, is it possible for a reducer to start its execution earlier, and why? (clue: think of availability of inputs to reducers)

**NO,** it is not possible for reducer to start executing earlier because:

- There can be failure in any of the Map worker.
- Delay in receiving the key value pair data from any of the worker. (Missing of input to reducers)
References

• http://www.ece.eng.wayne.edu/~sjiang/ECE7650-winter-16/Student-presentations.htm
• www.google.com
Thank you