WAYNE STATE UNIVERSITY
Scalable and Secure Internet Services and Architecture
ECE7650 (Winter -2016)

Presentation Q&A Report

On

MapReduce: Simplified Data Processing on Large Clusters

By:
Nandini Obulappa Gari
Fy4399
1) Compared with traditional parallel programming models, such as multithreading and MPI, what are major advantages of MapReduce?

Advantages of MapReduce over traditional Parallel Programming Models are:

- **Fault tolerance**: Failure in the workers is identified and replaced by master to work normally i.e. Master keep sending a message to worker periodically to check the status, if no response then master replaces the worker with new one. Master keep checking itself in the data structures with help of checkpoints, Master Failure occurs very rarely in Google’s MapReduce.

- **Automatic parallelization & distribution**: Not much knowledge required on parallel and distributed system since MapReduce library are stored in User Program and distributed with the help of master automatically.

- **Master job**: Presence of Master to take care of the function of workers in both Map and reduce function

- **Reliability**: If there is any function failure is detected by master and replaced and rescheduled the task immediately without any loss or delay in the process.

- **Critical Path**: It is time taken by the worker to complete the task in given time i.e.if any failure occurs in worker it is replaced quickly to proceed with the task to complete in the given time.

2) Use Figure 1 to explain a MR program’s execution.

![Figure 1: Execution overview](image_url)

The above figure shows the Execution Overview of Mapreduce.

1. **The MapReduce library in the user program first splits the input files** into M pieces of typically 16 megabytes to 64 megabytes It then starts up many copies of the program on a cluster of machines.
2. **One of the copies of the program is special – the master.** The rest are workers that are assigned work by the master. There are M map tasks and R reduce tasks to assign. The **master picks idle workers** and assigns each one a map task or a reduce task.

3. A worker who is assigned a **map task reads the contents of the corresponding input split.** It parses key/value pairs out of the input data and passes each pair to the user-defined Map function. The **intermediate key/value pairs produced** by the Map function are buffered in memory.

4. Periodically, the **buffered pairs are written to local disk,** partitioned into R regions by the partitioning function. The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the reduce workers.

5. When a **reduce worker** is notified by the master about these locations, it **uses remote procedure calls to read the buffered data** from the local disks of the map workers. When a reduce worker has read all intermediate data, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together. The sorting is needed because typically many different keys map to the same reduce task.

6. The reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it **passes the key and the corresponding set of intermediate values to the user’s Reduce function.** The output of the Reduce function is appended to a final output file for this reduce partition.

7. When all map tasks and reduce tasks have been completed, the master wakes up the user program. At this point, the **MapReduce call in the user program returns back to the user code.**

**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. If no response received the master assumes the worker as failed.
- If master identifies the failed worker at map task then it reschedule the whole task to the another worker even though the task is completed by previous worker.
- 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.
(4) 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 workers (Missing of inputs to reducers)