Outline

- Overview
- Pipelined MapReduce
- Online Aggregation
- Continuous Queries
- Conclusion
Overview

- MapReduce was originally designed for batch jobs
- Based on Hadoop framework
- Pipeline data between operators to extend MapReduce model beyond batch processing
- Extra options/functions based on pipelining
- Modified fault tolerance mechanism
Pipelined MapReduce

• Map tasks
  - Read input data and perform Map function
  - Use combiners to sort the intermediate output
  - Send intermediate output to Reduce tasks through TaskTracker

• Reduce tasks
  - Read intermediate data and sort it
  - Apply Reduce function to generate final output
Pipelined MapReduce

• Original MapReduce
  ➢ Accumulate outputs of Map tasks and send them to corresponding Reduce tasks

• MapReduce Online
  ➢ Pipeline output of Map tasks to Reduce tasks soon after they are produced
  ➢ Separate Map function and output in different thread
  ➢ Straightforward approach, needs rate adaption
Pipelined MapReduce

• Rate adaption
  - Reduce tasks may be unable to accept input at the moment
  - Balance the workload of combiners and Reduce tasks
  - Reduce transmission overhead
Pipelined MapReduce

• Pipelining scheme
  ➢ Enables early utilization of Reduce tasks
  ➢ Reduce the effect of combiners by moving sorting work from combiners to Reduce tasks
  ➢ May reduce overall performance if Reduce tasks are the bottlenecks
Pipelined MapReduce

10 GB Blocking (5 Reduces)

10 GB Pipelining (5 Reduces)

10 GB Blocking (1 Reduce)

10 GB Pipelining (1 Reduce)
Pipelined MapReduce

• Modifications on fault tolerance
  ➢ Split intermediate data into more files
  ➢ Reduce tasks keep intermediate data as “tentative” until informed
  ➢ Map tasks retain intermediate data in disk until job finishes
  ➢ More complicated scheme but more robust to task failure
Pipelined MapReduce

• Pipelining between jobs
  ➢ Final result cannot be generated before job finished
  ➢ Used for online aggregation
  ➢ Needs task scheduling on high level
Online Aggregation

• Generate rough approximation in a much shorter period of time
• Progress metric can only be estimated
• Approximation metric should be defined by users, otherwise the error would be too large
Online Aggregation

(a) Relative approximation error over time

(b) Example approximate answer
Online Aggregation

• Rely on users to provide proper metric
• Multi-job online aggregation is possible and can be easily supported
• Fault tolerance in multi-job online aggregation needs storage of approximations to recover from failure
Continuous Queries

• Used to analyze constantly arriving data stream
• Original MapReduce model introduces large latency and has to re-compute all data
• Modified version runs continuously and make use of previous results
Continuous Queries

- No major modification to MapReduce Online
- Minor modifications:
  - Force Map tasks to send output to Reduce tasks promptly
  - Invoke Reduce tasks periodically
  - Reduce tasks should be able to utilize previous results
Continuous Queries

• Modifications on fault tolerance
  ➢ Map tasks can no longer retain all output
  ➢ Recovering from failure can only rely on finite history
  ➢ Need to checkpoint states of the tasks periodically
  ➢ Cannot apply to all functions
Continuous Queries

Application Example: Monitoring system
Conclusion

• Pipelining scheme can only reduce completion time when reduce tasks are not the bottleneck
  ➢ Provide pipelining scheme as an option
  ➢ Automatically determine the number of tasks

• Fault tolerance needs more states and checkpoints, but could reduce repetitive work

• Online aggregation and continuous queries are potential research areas
Discussion

• Is optimal scheduling feasible?
• To what extent would scheduling improve the performance?
• Is MapReduce the ideal framework for continuous work?