Scheduling Hadoop Jobs to Meet Deadlines

Kamal Kc, Kemafor Anyanwu
Department of Computer Science
North Carolina State University
{kkc,kogan}@ncsu.edu

Introduction

- MapReduce
  - Cluster based parallel programming abstraction
  - Programmers focus on designing application and not on issues like parallelization, scheduling, input partitioning, failover, replication
- Hadoop
  - open source implementation of MapReduce framework
  - A Hadoop job is a workflow of Map Reduce cycles

Introduction

- Using Hadoop
  - Cluster infrastructure required
    - costly to maintain
    - sharing cluster resources among users a viable approach
  - Demand based pay-as-you-go model can be attractive to meet user's computation requirement
  - One such user requirement is the time specification: **deadline**
  - But current Hadoop does not support deadline based job execution
- *How to make Hadoop support deadlines?*
  - Develop interface to input the deadline
  - Modify the Hadoop scheduler to account for deadlines

Problem definition

- A user submits a job with a specified deadline D
- Hadoop cluster has fixed number of machines with fixed map and reduce slots
- Hadoop job is broken down into fixed set of map and reduce tasks
- Problem:
  - *Can the job meet its deadline ?*
  - *If yes, then how should we schedule the tasks into the available slots of the machines ?*
- **Constraint Scheduler for Hadoop**: our effort to tackle these problems

Constraint Scheduler

- Extends the real time cluster scheduling approach to incorporate 2 phase(map and reduce) computation style
- Can the deadline be met ?
  - Let n_m^{min}, n_r^{min} be the minimum # of map and reduce tasks that need to be scheduled to meet deadline
  - map tasks can be started as soon as job is submitted *but when should the reduce be started ?* (answer: let reduce should be started at $S\_r(max)$ to finish the deadline)
  - then the job can meet deadline:
    - If map slots >= n_m^{min} is available before $S\_r(max)$
    - if reduce slots > = n_r^{min} is available after $S\_r(max)$
  - But how do we know the values of n_m^{min}, n_r^{min}, $S\_r(max)$?

Constraint Scheduler

- Assume we can know/estimate (data processing tasks)
  - map cost per unit data c_m
  - reduce cost per unit data c_r
  - communication cost per unit data c_d
  - filter ratio f
- Also assume
  - cluster is homogeneous
  - key distribution is uniform
- Then, for a job of size σ with arrival A and deadline D
- s_m and s_r are actual start times for map and reduce resp.

$$s_r^{max} = A + D - \frac{f\sigma c_r}{n_r} - f\sigma c_d$$

$$n_m^{min} = \left\lceil \frac{\sigma c_m}{s_r^{max} - s_m} \right\rceil$$

$$n_r^{min} = \left\lceil \frac{f\sigma c_r}{A + D - f\sigma c_d - s_r} \right\rceil$$

Constraint Scheduler - 2

- How to schedule tasks in cluster machines ?
  - Possible techniques:
    - assign all map and reduce tasks if enough slots are available
    - assign minimum tasks
    - assign some fixed number of tasks greater than minimum
  - Constraint Scheduler's approach:
    - assign minimum tasks
    - intuitive appeal : some empty slots available for other jobs

Design and Implementation

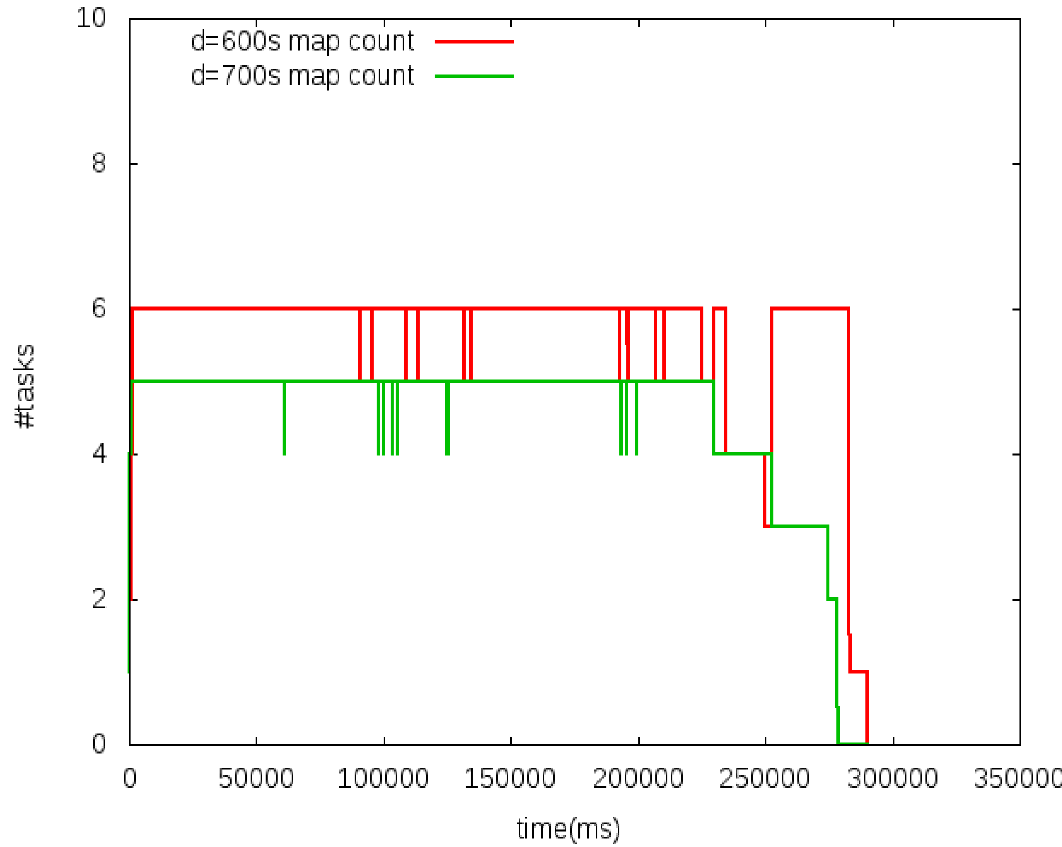
- Developed as a contrib module using Hadoop 0.20.2 version
- Web interface:
  - to specify deadline
  - to provide map/reduce cost per unit data
  - to start job

Experimental Evaluation

- Setup
  - Physical cluster
    - 10 tasktrackers, 1 jobtracker
  - Virtualized cluster
    - single physical node
    - 3 guest Vms as tasktrackers, host system as jobtracker
  - Both systems:
    - 2 map/reduce slots per tasktracker
    - 64MB HDFS block size
- Hadoop job
  - Job equivalent to the query: `SELECT userid, count(actionid) as num_actions FROM useraction GROUP BY userid`
  - useraction table contains (userid, actionid) tuples
  - Job translates into aggregation operation which is one of the common form of Hadoop operation

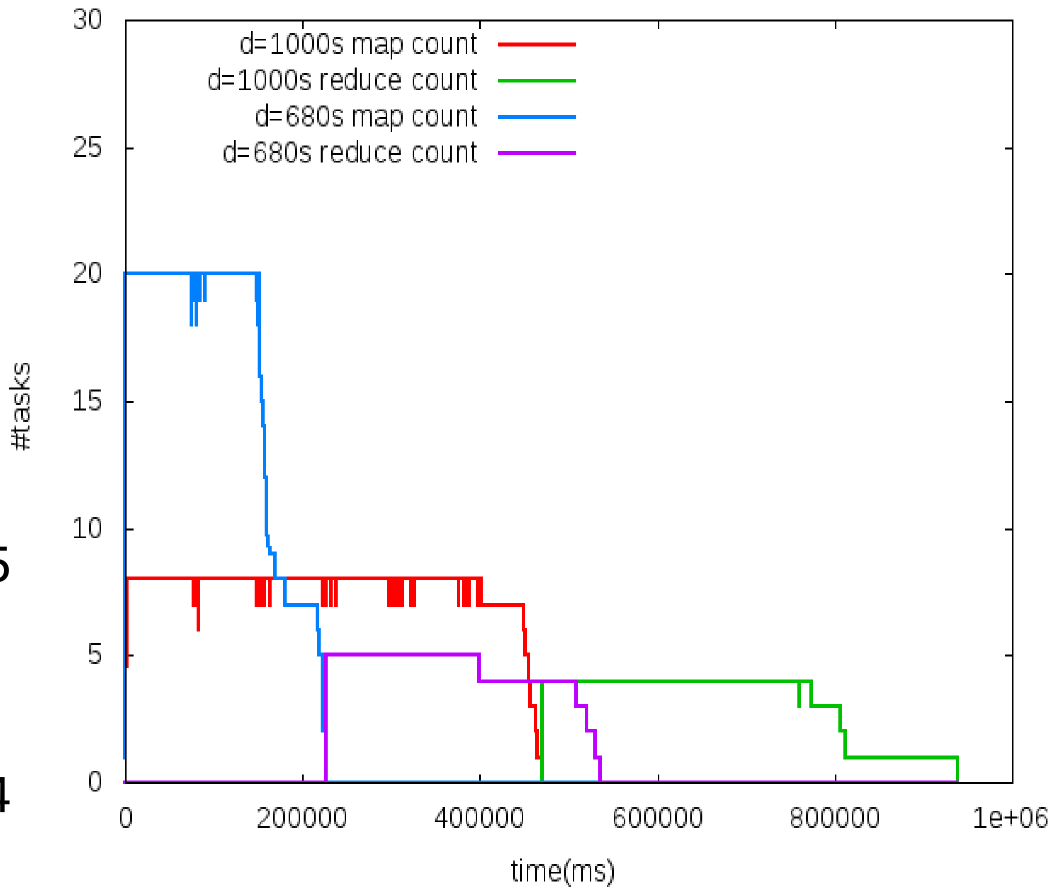
Results

- Virtualized cluster
  - Input size = 975MB
  - 16 map tasks
  - 2 deadlines
    - 600s deadline
      - min map tasks = 6
    - 700s deadline
      - min map tasks = 5
      - finished early due to less task resulting in less cpu load



Results

- Physical cluster
  - Input size = 2.9GB
  - 48 map tasks
  - 2 deadlines
    - 680s
      - min map tasks = 20
      - min reduce tasks = 5
    - 1000s
      - min map tasks = 8
      - min reduce tasks = 4



Future work

- Take into account
  - node failures
  - speculative execution
  - map/reduce computation cost estimation
  - impact of map tasks with non local data

Conclusion

- Extended the real time cluster scheduling approach for MapReduce style computation
- Constraint Scheduler identifies if a Hadoop job can meet its deadline and schedules accordingly if the deadline can be met
- Constraint Scheduler based on general enough model that can be extended to account for the assumed conditions

Thank you