CMSC 714
Lecture 14
Cloud Computing – Spark and Mesos

Alan Sussman
Notes

• Group research project proposals due Monday
  • Feedback on them next week
• Exam coming up in November
  • May get moved back a few days, to when I am at a conference
Spark

• Single engine for distributed data processing
  • SQL
  • stream processing
  • machine learning
  • graph processing

• Basic idea is to enable composing different types of processing into a single application
  • without copying data, so reuse of data and doing operations in memory is fundamental

• Key abstraction is Resilient Distributed Dataset (RDD)
  • a fault tolerant collection of objects (data items) partitioned across a cluster that can be operated on in parallel

• Functional programming API in Scala, Java, Python, and R
Spark (cont.)

• Users/developers write local functions that operate on RDDs

• RDDs evaluated by Spark runtime lazily
  • that means when they are needed, so only when one needs to be instantiated
  • enables creating an execution plan for a whole set of data transformations (like in an RDBMS)

• User can enable sharing an RDD by making it persistent in memory (spilled to disk if too big)
  • this is a big difference from MapReduce implementations

• Fault tolerance – RDDs can be recomputed if lost by keeping track of lineage (how they were computed)

• Can use different external systems for persistent storage
  • e.g., HDFS, S3, Cassandra
Spark (cont.)

• Additional functionality comes from building libraries on top of basic abstractions
  • SparkSQL for relational queries – but no transactions
  • DataFrames – RDDs of records with a known schema, used for tables in R and Python
  • Spark Streaming for incremental stream processing on discretized streams – split input data into small batches (e.g., data that arrives over 200ms) that is combined with state stored in RDDs to produce new results
  • GraphX – graph computation interface – vertex-based computations for graphs, and graphs partitioned across nodes
  • MLlib – machine learning library

• Claim is that performance is comparable to specialized systems for each kind of processing

• Last note is that they do admit that synchronization in Spark means it does not work well for latency sensitive computations
Mesos

• **A meta-scheduler** – to enable multiple cluster computing frameworks (e.g., Hadoop, OpenMPI) to share cluster resources
  • an alternative to a centralized scheduler

• Basic idea is that the resources register with Mesos, Mesos offers resources to frameworks, frameworks decide whether to accept or reject the resource offers
  • so frameworks do their own scheduling, once they obtain resources from Mesos

• One catch is that someone has to tell Mesos how to decide which resources to offer to which frameworks
  • this is a policy decision (e.g., fair sharing), and there is a Mesos plugin interface for the policy module
  • similar to how HPC cluster schedulers work – SLURM, Torque
Mesos (cont.)

• Basic architecture is one Mesos master/boss that frameworks communicate with, and a Mesos worker daemon on each cluster node
  • each worker process offers resources through its daemon
  • boss offers resources to frameworks, which they can accept or reject
  • frameworks decide which offered resources to use – through a scheduler they register with the boss
  • framework can then launch tasks on acquired resources through their executor process

• Use Zookeeper for fault tolerance
  • a distributed coordination service, to deal with faults in the Mesos master – enables having hot spare copies of the boss – leader election
  • use soft state so new boss can reconstruct internal state from worker daemons and framework schedulers
Mesos (cont.)

• **Efficiency and robustness**
  - Framework can set *filters*, to tell boss which offers it will always reject – so boss won't even try such offers
  - To give incentive for frameworks to respond quickly to offers, Mesos counts outstanding resource offers toward a framework’s allocation of a cluster – so they don't hang onto resources they may not use
  - If a framework does not respond for a while, Mesos rescinds a resource offer

• **Performance**
  - Simulation study shows Mesos provides both good latency to schedulers that need resources, and good cluster utilization, compared to a centralized scheduler
  - Performance best for frameworks that have short tasks to run, and jobs that can scale elastically – so probably not so good for HPC workloads