Efficiently distribute files in really large P2P networks in All nodes are on EC2

Support
Broadcast time increases with R3 use from Scala interpreter

30
50
Spark:
20
Later Iterations
10
across parallel operations
5
All nodes are on EC2
5
find the best line
R2
Users can explicitly cache RDDs in memory for reuse in
500 MB
5
10
750 MB
HDFS becomes the
500 MB
100 MB
Experiment with programmability
Goal:
Machine learning researchers in our lab identified this
5
Broadcast time is stable
Hadoop:
Block size is set to 4MB
1 GB
Matrix factorization:
Shared variables
Support
750 MB
29 GB dataset on 20 EC2
SplitStream Broadcast
30
Still room for optimization
Retain MapReduce’s fine-grained fault-tolerance

R4
No upload/download limit
Whoever is done starts a new chain as a seed
Breakdown of later iterations
#Simultaneous upload=8
40
1 GB
Master partitions a job into multiple slices and sends
4 cores
Slaves join the chain to receive broadcast
Utilizes full upload capacity of all the nodes

Performance Evaluation

Logistic Regression

- Goal: find the best line separating two sets of points
- Hadoop: 127 s/iteration
- Spark: 3 rd iteration 374 s/6s in each of the later iterations
- 29 GB dataset on 20 EC2 m2.xlarge machines, each with 4 cores

Alternating Least Squares

- Matrix factorization: given matrix $R (m \times n)$, find $A (m \times k)$ and $B (k \times n)$
- Collaborative filtering for the Netflix prize
- 2+ GB dataset of users and ratings matrix ($R$)
- $k$ is the dimension of the feature vector
- $k = 60$ on EC2 m2.xlarge master and c1.xlarge slaves

Centralized HDFS Broadcast

- Initial implementation of the broadcast primitive
  » Master stores serialized version of the variable in HDFS
  » Slaves read from HDFS and deserializes
  » HDFS becomes the bottleneck

Breakdown of later iterations
- Broadcast time increases with the number of nodes
- Soon becomes the dominant factor

30% of iteration time spent on broadcast

Broadcast-only experiments
- All nodes are running on EC2 m1.large instances
  (2 cores each)

WIP/Future Work

- SplitStream Broadcast
  » Utilizes full upload capacity of all the nodes
  » Working implementation is completed
  » Found to be not completely reliable in certain cases
  » We are working on creating a reliable, dynamic SplitStream protocol

Programmability

» HDFS files, “parallelized” Scala collections
» Can be transformed with map and filter
» Can be cached across parallel operations
» Users can customize persistence
» Parallel operations
  » foreach, reduce, collect
  » No support for “grouped reduce” as of now
» Shared variables
  » Accumulators (add-only)
  » Broadcast variables (read-only)

Spark Objectives

- Support iterative jobs
  » Machine learning researchers in our lab identified this as a workload that Hadoop doesn’t perform well on
- Experiment with programmability
- Leverage Scala to integrate cleanly into programs
- Support interactive use from Scala interpreter
- Retain MapReduce’s fine-grained fault-tolerance