

Performance and Scalability of Broadcast in Spark

Mosharaf Chowdhury, Matei Zaharia, Ion Stoica
Computer Science Division, University of California, Berkeley



Spark Objectives

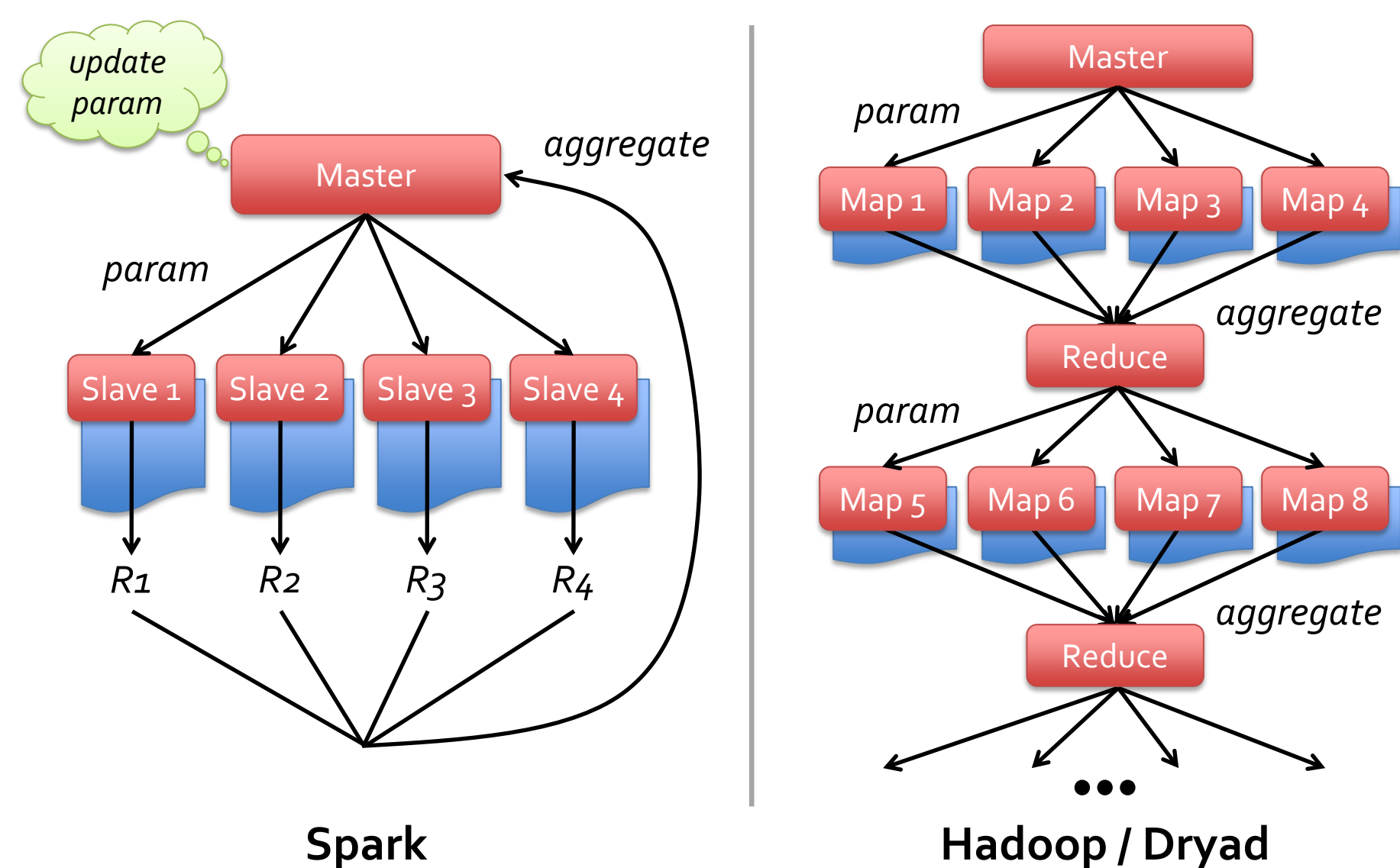
- Support *iterative* jobs
 - » Machine learning researchers in our lab identified this as a workload that Hadoop doesn't perform well on
- Experiment with programmability
 - » Leverage Scala to integrate cleanly into programs
 - » Support *interactive* use from Scala interpreter
- Retain MapReduce's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

Programming Model

- Reliable distributed datasets (RDD)
 - » 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)

Job Execution Model

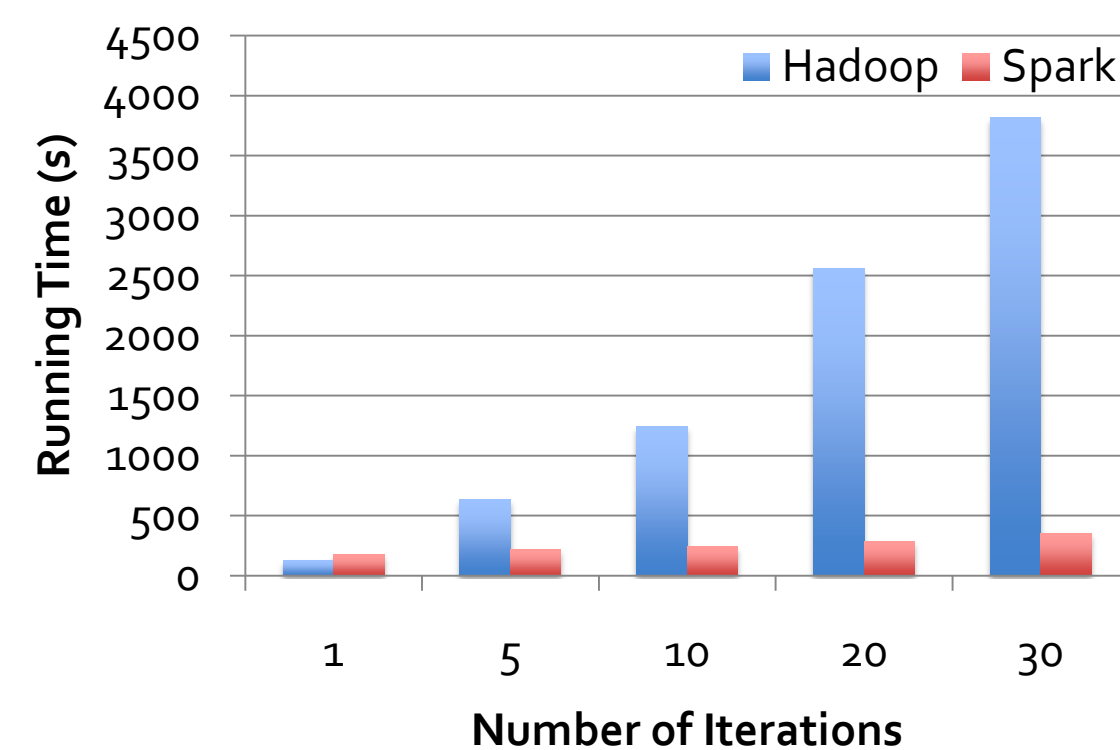
- Master partitions a job into multiple slices and sends them to multiple slaves
- Slave scheduling on remote machines are handled by the underlying Nexus framework



- Users can explicitly cache RDDs in memory for reuse in multiple MapReduce like iterations

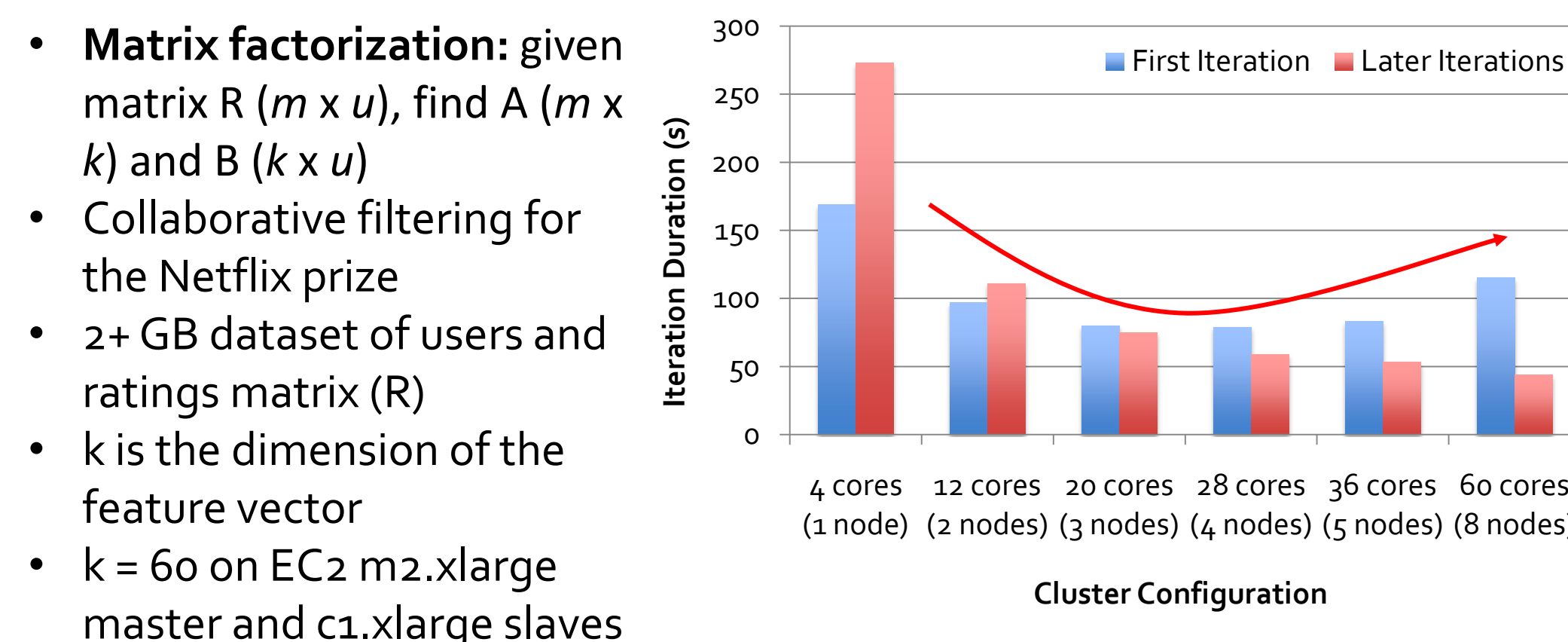
Performance Evaluation

Logistic Regression



- **Goal:** find the best line separating two sets of points
- **Hadoop:** 127 s / iteration
- **Spark:** 1st iteration 174 s; 6s in each of the later iterations
- 29 GB dataset on 20 EC2 m1.xlarge machines, each with 4 cores

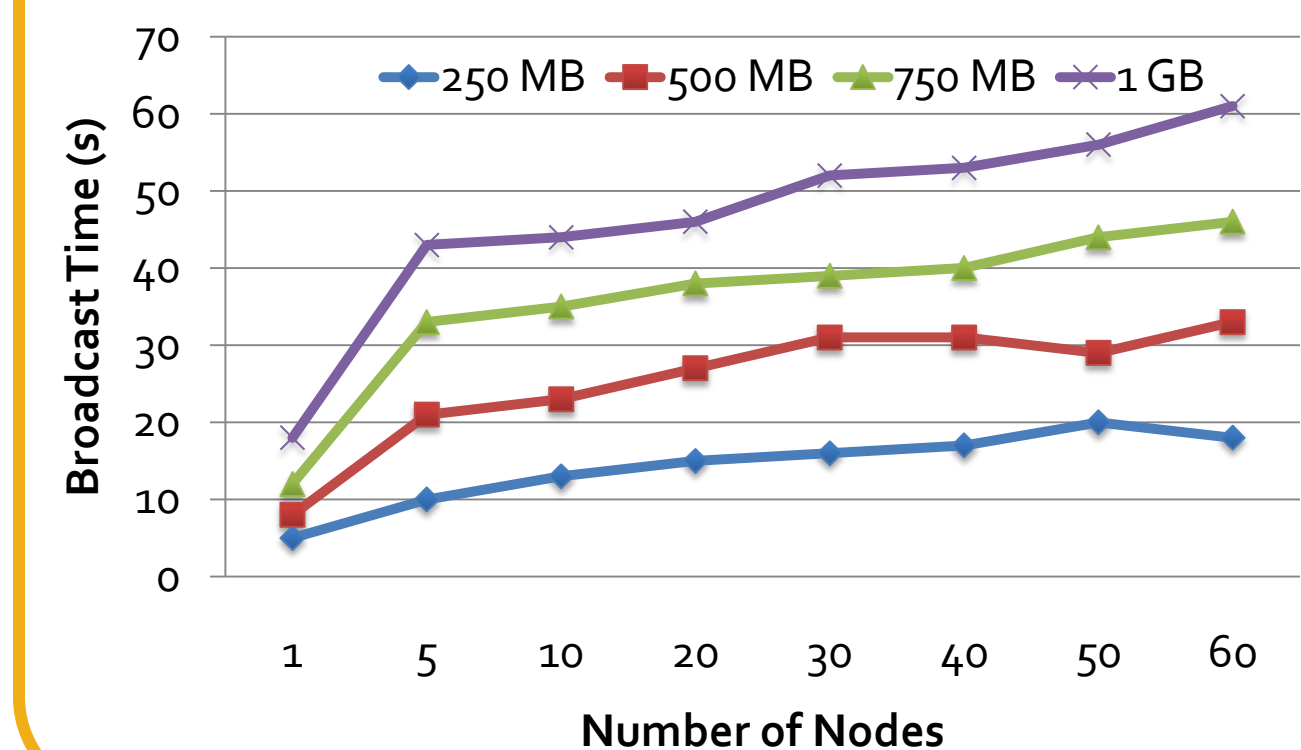
Alternating Least Squares



- **Matrix factorization:** given matrix $R (m \times u)$, find $A (m \times k)$ and $B (k \times u)$
- 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

Chained Streaming Broadcast

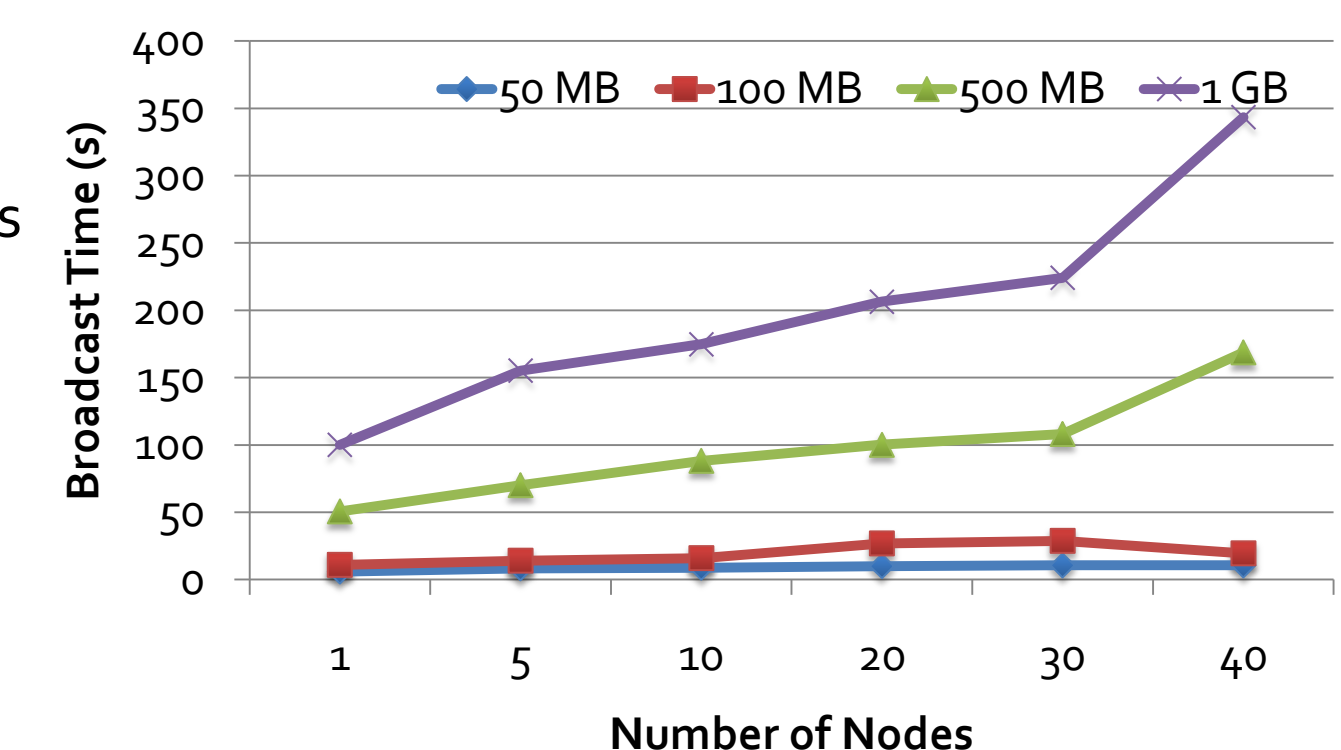
- Create forest of reliably streaming chains
 - » Master starts as a seed, divides the broadcasted variable into blocks, and starts broadcasting
 - » Slaves join the chain to receive broadcast
 - » Whoever is done starts a new chain as a seed



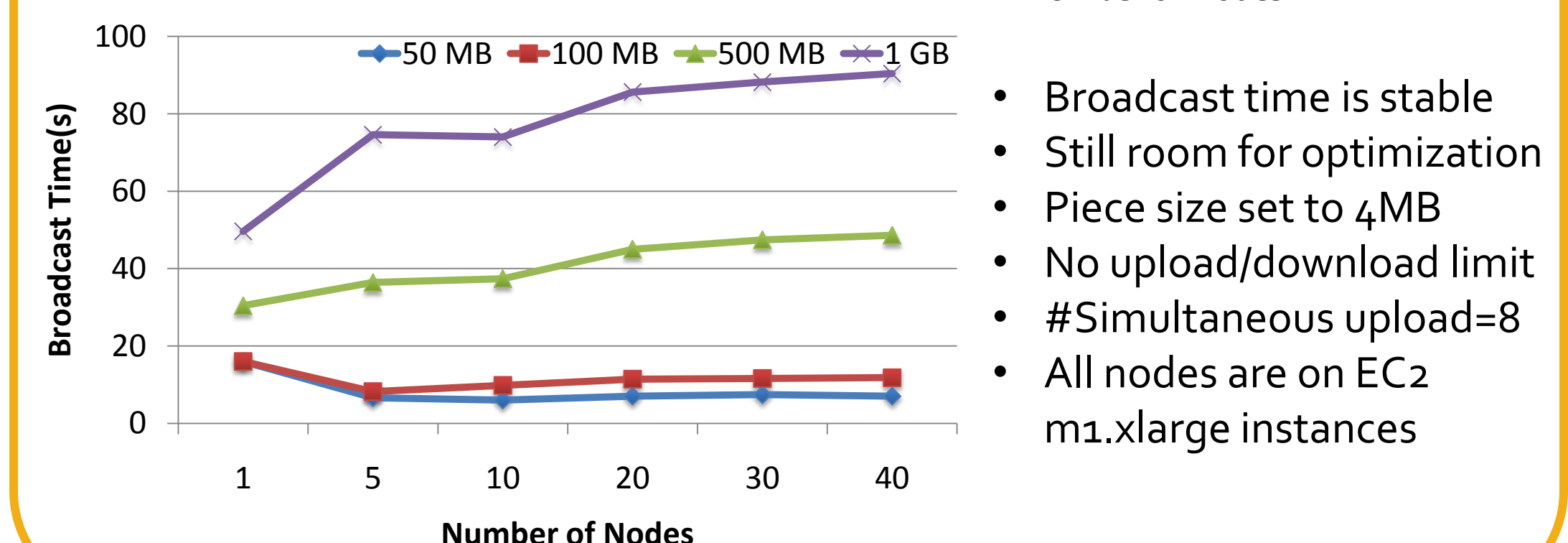
- Broadcast time increases sub-linearly
- Still room for optimization
- Block size is set to 4MB
- All nodes are on EC2 m1.large instances

BitTorrent

- Efficiently distribute files in really large p2p networks in presence of failure and churn
 - » Performance is not well-examined in high-speed reliable data center networks



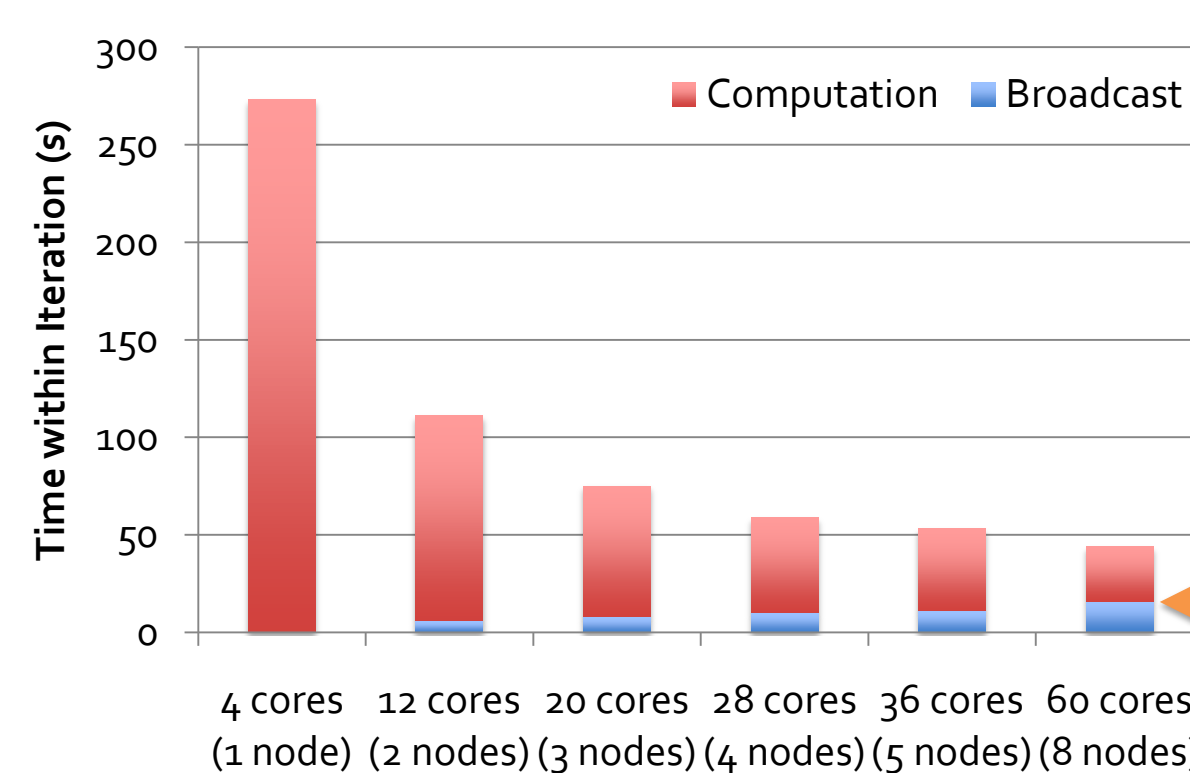
- Default BitTornado settings
- All nodes are on EC2 m1.xlarge instances



- Broadcast time is stable
- Still room for optimization
- Piece size set to 4MB
- No upload/download limit
- #Simultaneous upload=8
- All nodes are on EC2 m1.xlarge instances

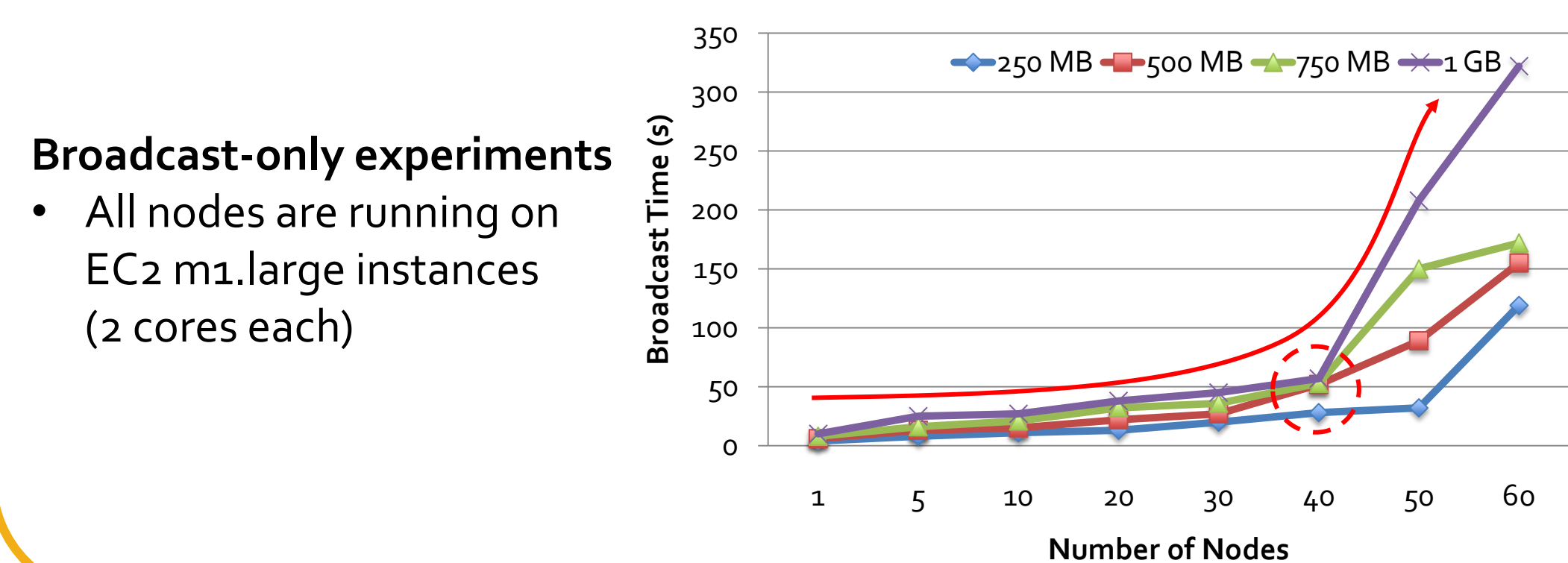
Centralized HDFS Broadcast

- 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

36% 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