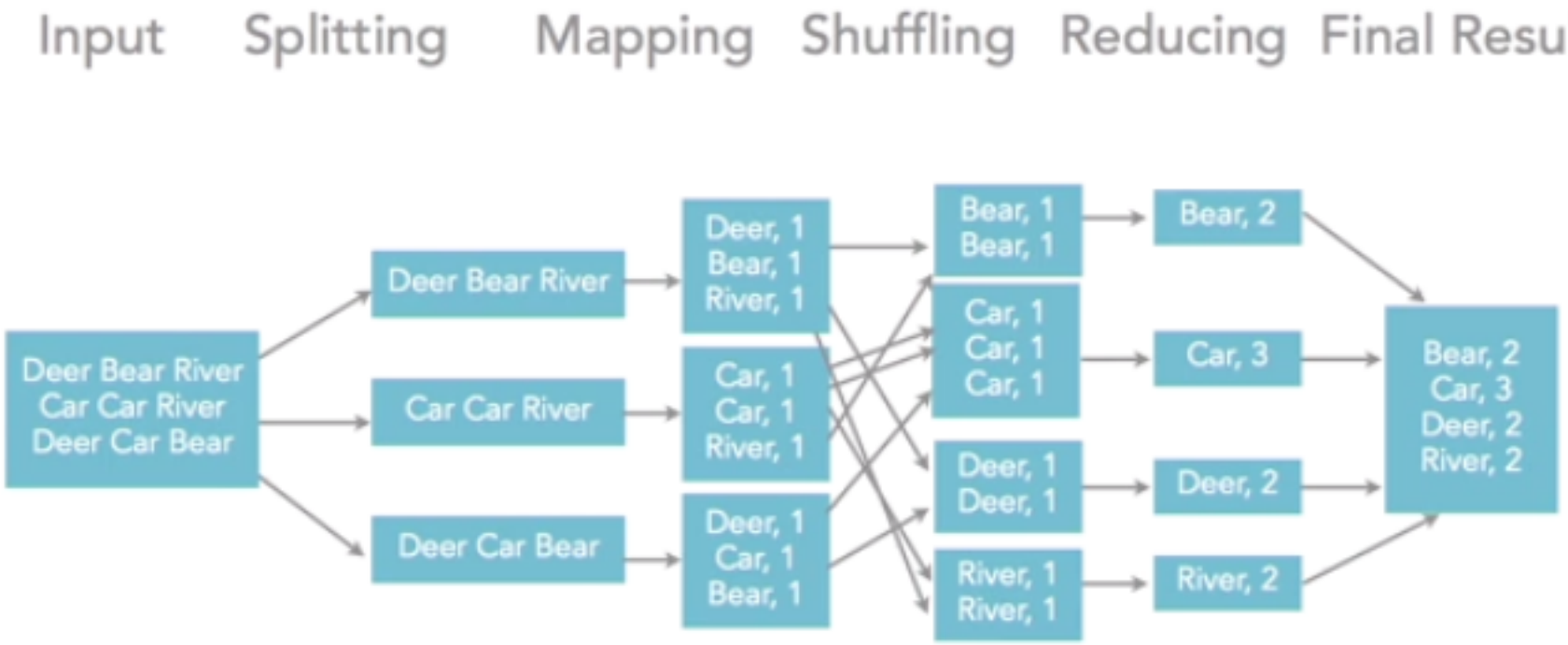
Tuning MapReduce

Jay Urbain, PhD

Tuning by physical methods



Better MapReduce: Optimizations

- Optimize before job runs
  - Preprocessing on data, files size, compression, encryption
- Optimize onload of the data
  - Change compression ratio, different size chunks
- Optimize the map phase of the job
  - Code, hashing
- Optimize shuffle phase of the job
  - Get better distribution on shuffle-sort
- Optimize the reduce phase
  - Additional transformation, combine or separate aggregations
- Optimize after job completes
  - Partition job in to separate jobs, pipeline

- *Other*
  - *Make sure you verify your cluster parameters especially for nonstandard configurations*
  - *Unused resources*
  - *Overstressed resources – spill out to disk – use cluster-monitoring tools, e.g., Ganglia*
  - *Add more nodes – especially when you're using cloud*
  - *Mixed local and cloud Hadoop*

Understanding MapReduce

- Map
  - (k1, v1): input information
  - (k2, v2): intermediate values (one list per node)
    - can use *combiner* (mini-Reducer in Mapper) before emitting intermediate values
- Shuffle/sort
- Reduce
  - (k2, list(v2))
  - (k3, v3)

- Notes:
  - Keep jobs as simple as possible, consider adding steps
  - One type of activity at a time

Mapper Task Optimization

- Sub-divide tasks
  - Chain jobs
  - 1-3 minutes per map task run
- Custom partitioner
  - Default is hash partitioner
- Reduce the amount of data
  - skip unnecessary or bad data, add counter, partition input files
- Logging and counters
- Monitor/tune for optimal spill ratio
- Local Reducers/ combiners
- Set Map-only jobs
  - Photo processing – no aggregation needed

Input file compression

- Vendor implementations differ
- Read the documentation
- Cloudera's version of LZO is 'splittable'
- Use best from vendor for app

Data Types

- Custom types supported – write RawComparator
  - Custom comparator
    - E.g. geospatial data
  - Custom input split
  - Custom partitioner
  - Use the Text type only when you are using text data

Reducer Task Optimization

- Subdividing tasks (chaining jobs)
- Logging/debugging to see what's going on
- Secondary sort
- Set thresholds
- May not need a Reducer (Map only)
  - Photos, music, transformation only apps
- Reduce data in Mapper

org.apache.hadoop.mapreduce.filecache.DistributedCache

Creating a Distributed Cache

- Used to cache files at nodes needed to run application jobs
- Include the path via URL's (hdfs:// ir http://) in JobConf settings
- Files will be copied to each data node on job execution
- E.g.:
  - Using a lookup file for translation, black list, white list
  - Translate numeric code to human values

Summary

- Add more node
  - Especially in cloud environment
- Distributed cache
- Skipping bad records
- Subdividing tasks
- Logging/profiling
- Debugging, unit testing
- Optimized Java processing
- String manipulation can be a bottleneck
  - StringBuffer.append
  - LongWritable and BytesWritable are faster than text parsing

Review

- Which of the following is a Mapper Task Optimization?
  - Subdivide tasks
  - All of the answers
  - Reduce the amount of data
  - Monitor/tune for optimal spill ratio

- Creating a distributed cache is important for optimization; it is used to cache files like text files, archives, and JAR files that are needed to run applications or jobs
  - True
  - False

Review

- Reducer task optimization usually falls into four categories: subdivide tasks, logging/ debugging, secondary sort, and setting thresholds
  - True
  - False