Data-Intensive Computing with Hadoop

Thanks to:
Milind Bhandarkar <milindb@yahoo-inc.com>
Yahoo! Inc.

Agenda

- Hadoop Overview
- HDFS
- Programming Hadoop
  - Architecture
  - Examples
  - Hadoop Streaming
  - Performance Tuning
- Debugging Hadoop Programs

Hadoop overview

- Apache Software Foundation project
  - Framework for running applications on large clusters
  - Modeled after Google's MapReduce / GFS framework
  - Implemented in Java
- Includes
  - HDFS - a distributed filesystem
  - Map/Reduce - offline computing engine
  - Recently: Libraries for ML and sparse matrix comp.
- Y! is biggest contributor
- Young project, already used by many

Hadoop clusters

It's used in clusters with thousands of nodes at Internet services companies

Who Uses Hadoop?

Amazon/A9
Facebook
Google
IBM
Intel Research
Joost

Last.fm
New York Times
PowerSet
Veoh
Yahoo!

Hadoop Goals

- Scalable
  - Petabytes (1015 Bytes) of data on thousands on nodes
  - Much larger than RAM, even single disk capacity
- Economical
  - Use commodity components when possible
  - Lash thousands of these into an effective compute and storage platform
- Reliable
  - In a large enough cluster something is always broken
  - Engineering reliability into every app is expensive

Sample Applications

- Data analysis is the core of Internet services.
- Log Processing
  - Reporting
  - Session Analysis
  - Building dictionaries
  - Click fraud detection
- Building Search Index
  - Site Rank
- Machine Learning
  - Automated Pattern-Detection/Filtering
  - Mail spam filter creation
- Competitive Intelligence
  - What percentage of websites use a given feature?

Problem: Bandwidth to Data

- Need to process 100TB datasets
- On 1000 node cluster reading from remote storage (on LAN)
  - Scanning @ 10MB/s = 165 min
- On 1000 node cluster reading from local storage
  - Scanning @ 50-200MB/s = 33s-8 min
- Moving computation to the data enables I/O bandwidth scaling
  - Network is the bottleneck
  - Data size is reduced by the processing
- Need visibility into data placement

Problem: Scaling Reliably is Hard

- Need to store Petabytes of data
  - On 1000s of nodes, MTBF < 1 day
  - Many components disks, nodes, switches, ...
  - Something is always broken
- Need fault tolerant store
  - Handle hardware faults transparently
  - Provide reasonable availability guarantees

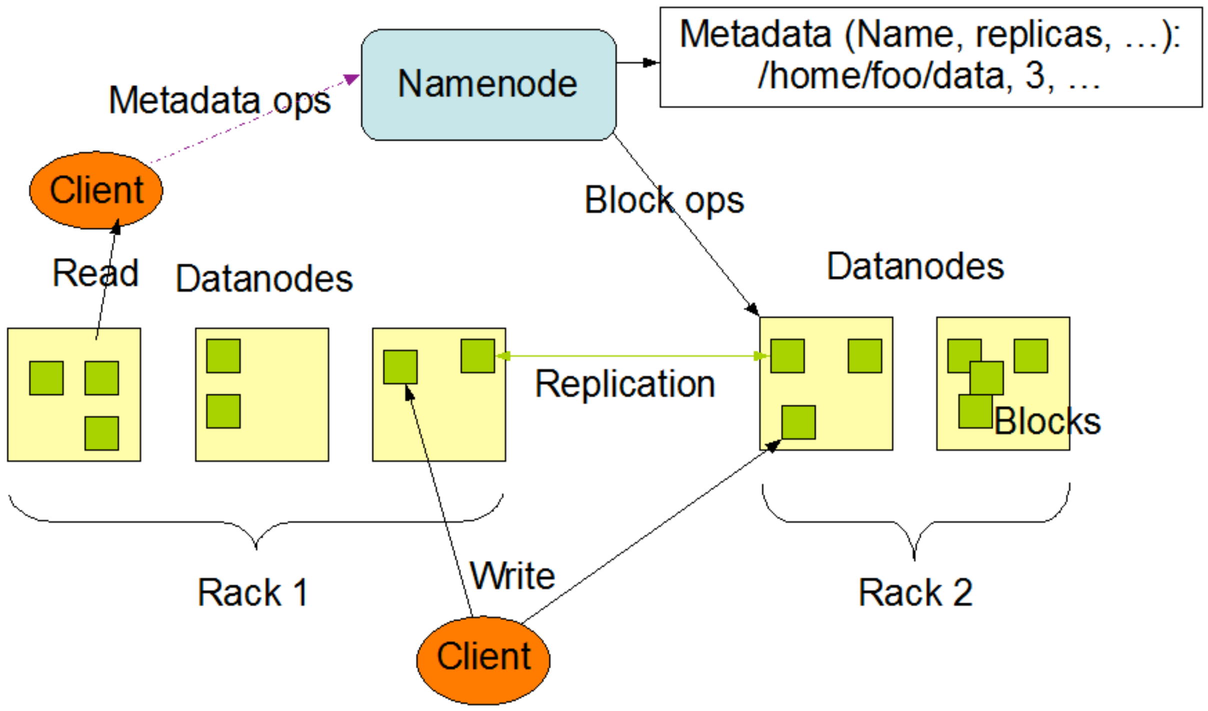
Hadoop Distributed File System

- Fault tolerant, scalable, distributed storage system
- Designed to reliably store very large files across machines in a large cluster
- Data Model
  - Data is organized into files and directories
  - Files are divided into uniform sized blocks and distributed across cluster nodes
  - Blocks are replicated to handle hardware failure
  - Corruption detection and recovery: Filesystem-level checksuming
  - HDFS exposes block placement so that computes can be migrated to data

HDFS Terminology

- Namenode
- Datanode
- DFS Client
- Files/Directories
- Replication
- Blocks
- Rack-awareness

HDFS Architecture



HDFS Architecture

- Similar to other NASD-based DFSs
- Master-Worker architecture
- HDFS Master "Namenode"
  - Manages the filesystem namespace
  - Controls read/write access to files
  - Manages block replication
  - Reliability: Namespace checkpointing and journaling
- HDFS Workers "Datanodes"
  - Serve read/write requests from clients
  - Perform replication tasks upon instruction by Namenode

Interacting with HDFS

- User-level library linked into the application
- Command line interface

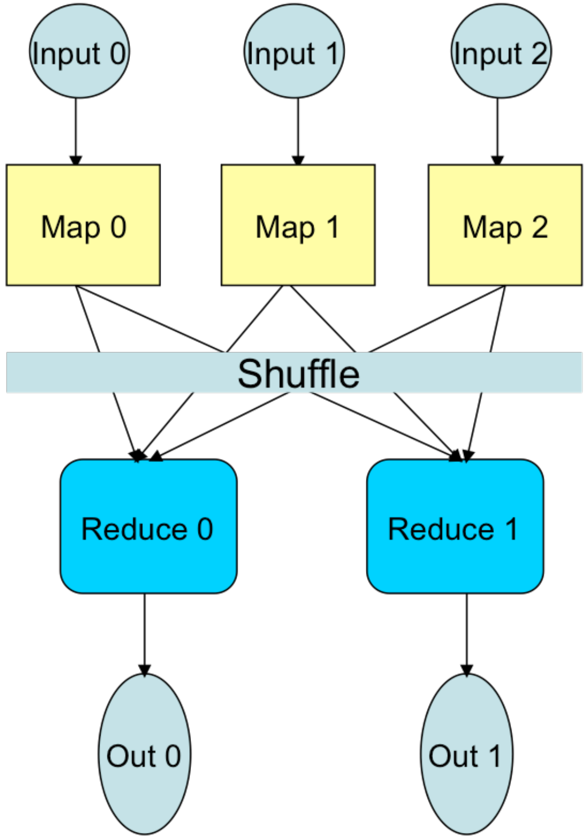
```
hadoop fs [-fs <local | file system URI>] [-conf <configuration file>]
        [-D <property=value>] [-ls <path>] [-lsr <path>] [-du <path>]
        [-dus <path>] [-mv <src> <dst>] [-cp <src> <dst>] [-rm <src>]
        [-rmr <src>] [-put <localsrc> <dst>] [-copyFromLocal <localsrc> <dst>]
        [-moveFromLocal <localsrc> <dst>] [-get <src> <localdst>]
        [-getmerge <src> <localdst> [addnl]] [-cat <src>]
        [-copyToLocal <src><localdst>] [-moveToLocal <src> <localdst>]
        [-mkdir <path>] [-report] [-setrep [-R] [-w] <rep> <path/file>]
        [-touchz <path>] [-test -[ezd] <path>] [-stat [format] <path>]
        [-tail [-f] <path>] [-text <path>]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-chgrp [-R] GROUP PATH...]
        [-help [cmd]]
```

Map-Reduce overview

- Programming abstraction and runtime support for scalable data processing
- Scalable associative primitive: Distributed "GROUP-BY"
- Observations:
  - Distributed resilient apps are hard to write
  - Common application pattern
    - Large unordered input collection of records
    - Process each record
    - Group intermediate results
    - Process groups
  - Failure is the common case

Map-Reduce

- Application writer specifies
  - A pair of functions called Map and Reduce
  - A set of input files
- Workflow
  - Generate *FileSplits* from input files, one per Map task
  - *Map phase* executes the user map function transforming input records into a new set of kv-pairs
  - Framework *shuffles & sort* tuples according to their keys
  - *Reduce phase* combines all kv-pairs with the same key into new kv-pairs
  - *Output phase* writes the resulting pairs to files
- All phases are distributed among many tasks
  - Framework handles scheduling of tasks on cluster
  - Framework handles recovery when a node fails



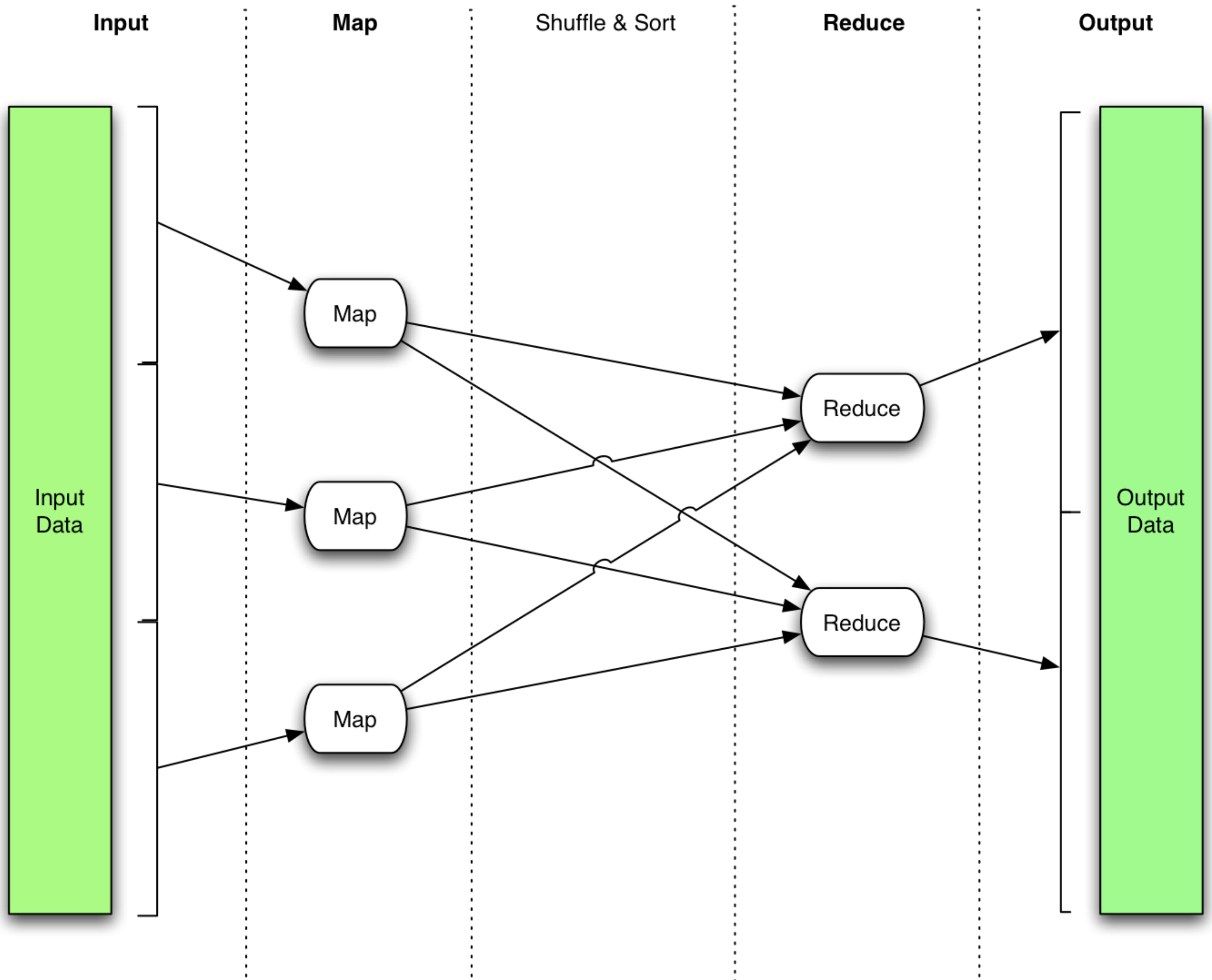
Hadoop MR - Terminology

- Job
- Task
- JobTracker
- TaskTracker
- JobClient
- Splits
- InputFormat/RecordReader

Hadoop M-R architecture

- Map/Reduce Master "Job Tracker"
  - Accepts Map/Reduce jobs submitted by users
  - Assigns Map and Reduce tasks to Task Trackers
  - Monitors task and Task Tracker status, re-executes tasks upon failure
- Map/Reduce Slaves "Task Trackers"
  - Run Map and Reduce tasks upon instruction from the Job Tracker
  - Manage storage and transmission of intermediate output

Map/Reduce Dataflow



M-R Example

- Input: multi-TB dataset
- Record: Vector with 3 float32_t values
- Goal: frequency histogram of one of the components
- Min and max are unknown, so are the bucket sizes

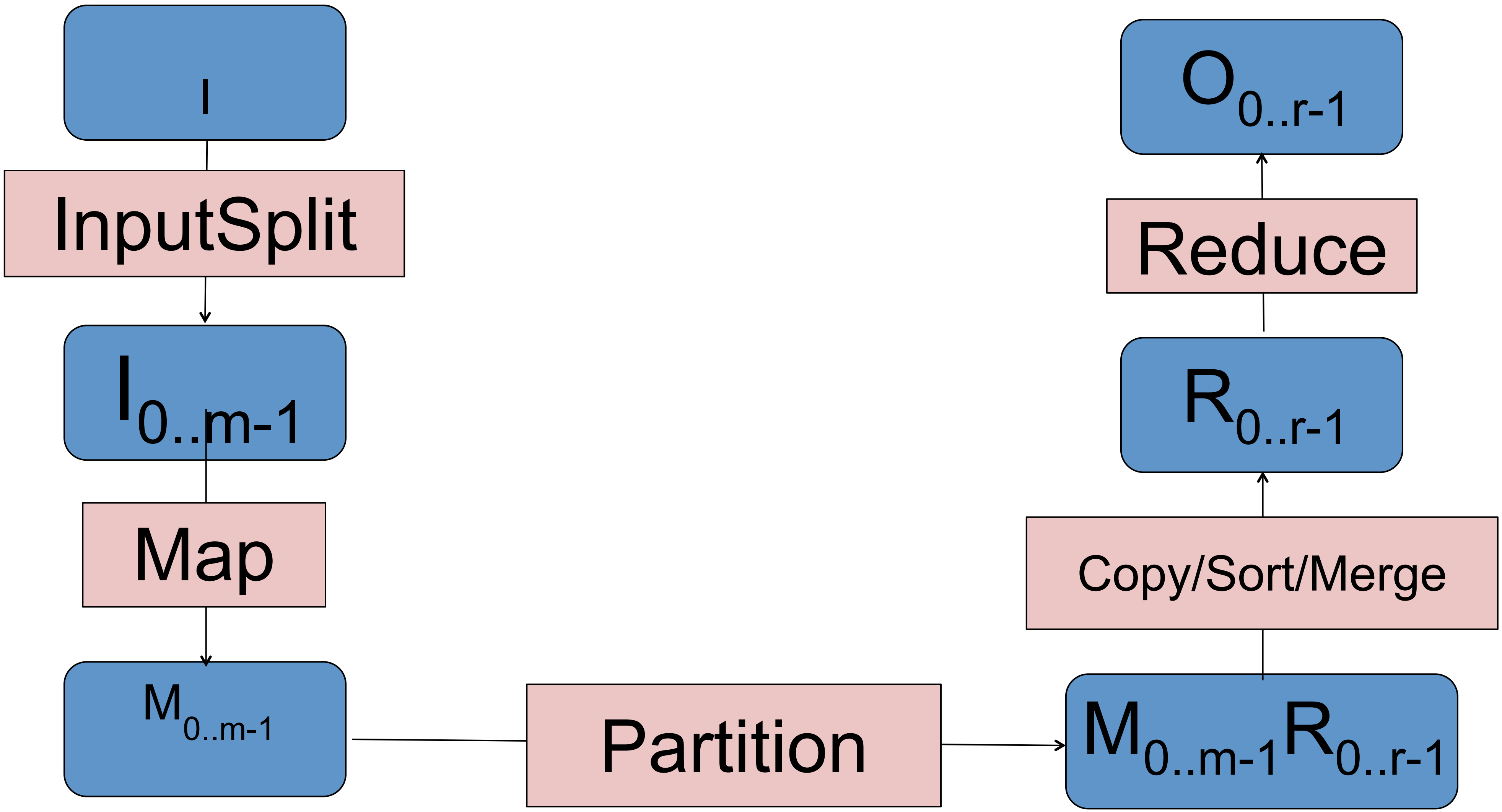
M-R Example (cont.)

- Framework partitions input into chunks of records
- Map function takes a single record
  - Extract desired component v
  - Emit the tuple (k=v, 1)
- Framework groups records with the same k.
- Reduce function receives a list of all the tuples where for a given k
  - Sum the value (1) for all the tuples
  - Emit the tuple (k=v, sum)

M-R features

- There's more to it than M-R: Map-Shuffle-Reduce
- Custom input parsing and aggregate functions
- Input partitioning & task scheduling
- System support:
  - Co-location of storage & computation
  - Failure isolation & handling

Hadoop Dataflow (I_2O)

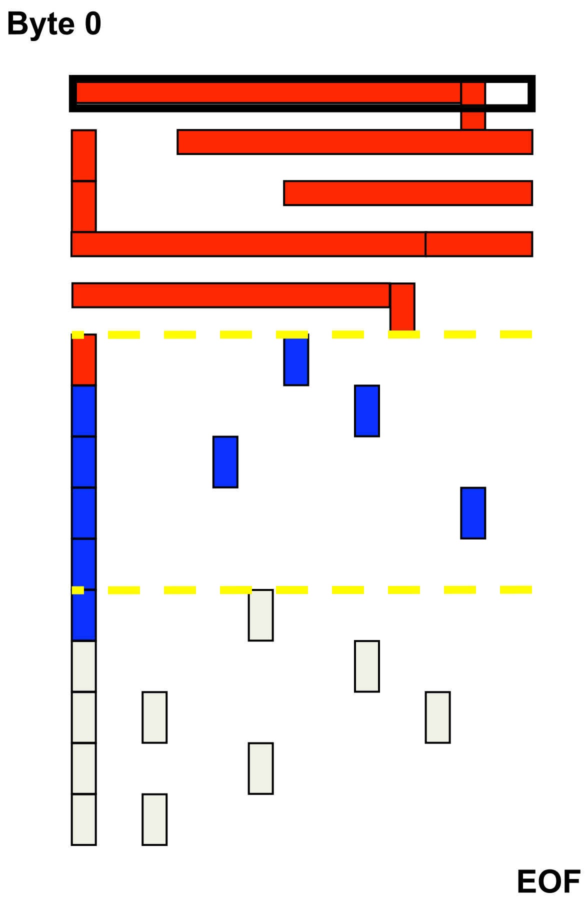


Input => InputSplits

- Input specified as collection of paths (on HDFS)
- JobClient specifies an InputFormat
- The InputFormat provides a description of splits
- Default: FileSplit
  - Each split is approximately DFS's block
    - mapred.min.split.size overrides this
  - Gzipped files are not split
  - A "split" does not cross file boundary
- Number of Splits = Number of Map tasks

InputSplit => RecordReader

- Record = (Key, Value)
- InputFormat
  - TextInputFormat
  - Unless 1st, ignore all before 1st separator
  - Read-ahead to next block to complete last record



Partitioner

- Default partitioner evenly distributes records
  - hashcode(key) mod NR
- Partitioner could be overridden
  - When Value should also be considered
    - a single key, but values distributed
  - When a partition needs to obey other semantics
    - Al URLs from a domain should be in the same file
- Interface Partitioner
  - int getPartition(K, V, nPartitions)

Producing Fully Sorted Output

- By default each reducer gets input sorted on key
- Typically reducer output order is the same as input
- Each part file is sorted
- How to make sure that Keys in part i are all less than keys in part i+1 ?
- Fully sorted output

Fully sorted output (contd.)

- Simple solution: Use single reducer
- But, not feasible for large data
- Insight: Reducer input also must be fully sorted
- Key to reducer mapping is determined by partitioner
- Design a partitioner that implements fully sorted reduce input
- Hint: Histogram equalization + Sampling

Streaming

- What about non-Java programmers?
  - Can define Mapper and Reducer using Unix text filters
  - Typically use grep, sed, python, or perl scripts
- Format for input and output is: ***key \t value \n***
- Allows for easy debugging and experimentation
- Slower than Java programs

```
bin/hadoop jar hadoop-streaming.jar -input
in_dir -output out_dir -mapper
streamingMapper.sh -reducer
streamingReducer.sh
```

- Mapper: `sed -e 's| |\n|g' | grep .`
- Reducer: `uniq -c | awk '{print $2 "\t" $1}'`

Key-Value Separation in Map Output

```
$HADOOP_HOME/bin/hadoop  jar $HADOOP_HOME/hadoop-streaming.jar \
    -input myInputDirs \
    -output myOutputDir \
    -mapper org.apache.hadoop.mapred.lib.IdentityMapper \
    -reducer org.apache.hadoop.mapred.lib.IdentityReducer \
    -jobconf stream.map.output.field.separator=. \
    -jobconf stream.num.map.output.key.fields=4
```

Secondary Sort

```
$HADOOP_HOME/bin/hadoop  jar $HADOOP_HOME/hadoop-streaming.jar \
    -input myInputDirs \
    -output myOutputDir \
    -mapper org.apache.hadoop.mapred.lib.IdentityMapper \
    -reducer org.apache.hadoop.mapred.lib.IdentityReducer \
    -partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner \
    -jobconf stream.map.output.field.separator=. \
    -jobconf stream.num.map.output.key.fields=4 \
    -jobconf map.output.key.field.separator=. \
    -jobconf num.key.fields.for.partition=2 \
    -jobconf mapred.reduce.tasks=12
```

Pipes (C++)

- C++ API and library to link application with
- C++ application is launched as a sub-process
- Keys and values are std::string with binary data
- Word count map looks like:

```
class WordCountMap: public HadoopPipes::Mapper {
public:
  WordCountMap(HadoopPipes::TaskContext& context){}

  void map(HadoopPipes::MapContext& context) {
    std::vector<std::string> words =
      HadoopUtils::splitString(context.getInputValue(), " ");
    for(unsigned int i=0; i < words.size(); ++i) {
      context.emit(words[i], "1");
    }
  }
};
```

Pipes (C++)

The reducer looks like:

```
class WordCountReduce: public HadoopPipes::Reducer {
public:
  WordCountReduce(HadoopPipes::TaskContext& context){}
  void reduce(HadoopPipes::ReduceContext& context) {
    int sum = 0;
    while (context.nextValue()) {
      sum += HadoopUtils::toInt(context.getInputValue());
    }
    context.emit(context.getInputKey(),
  HadoopUtils::toString(sum));
  }
};
```

Pipes (C++)

- And define a main function to invoke the tasks:

```
int main(int argc, char *argv[]) {
  return HadoopPipes::runTask(
     HadoopPipes::TemplateFactory<WordCountMap,
                                  WordCountReduce, void,
                                  WordCountReduce>());
}
```

Deploying Auxiliary Files

- Command line option: `-file auxFile.dat`
- Job submitter adds file to job.jar
- Unjarred on the task tracker
- Available as $cwd/auxFile.dat
- Not suitable for more / larger / frequently used files

Using Distributed Cache

- Sometimes, you need to read “side” files such as “in.txt”
- Read-only Dictionaries (e.g., filtering patterns)
- Libraries dynamically linked to streaming programs
- Tasks themselves can fetch files from HDFS
  - Not Always! (Unresolved symbols)
- Performance bottleneck

Caching Files Across Tasks

- Specify "side" files via `–cacheFile`
- If lot of such files needed
  - Jar them up (.tgz coming soon)
  - Upload to HDFS
  - Specify via –cacheArchive
- TaskTracker downloads these files "once"
- Unjars archives
- Accessible in task's cwd before task even starts
- Automtic cleanup upon exit

How many Maps and Reduces

- Maps
  - Usually as many as the number of HDFS blocks being processed, this is the default
  - Else the number of maps can be specified as a hint
  - The number of maps can also be controlled by specifying the minimum split size
  - The actual sizes of the map inputs are computed by: max(min(block_size, data/#maps), min_split_size)
- Reduces
  - Unless the amount of data being processed is small: 0.95*num_nodes*mapred.tasktracker.tasks.maximum

Map Output => Reduce Input

- Map output is stored across local disks of task tracker
- So is reduce input
- Each task tracker machine also runs a Datanode
- In our config, datanode uses “up to” 85% of local disks
- Large intermediate outputs can fill up local disks and cause failures
  - Non-even partitions too

Performance Analysis of Map-Reduce

- MR performance requires
  - Maximizing Map input transfer rate
  - Pipelined writes from Reduce
  - Small intermediate output
  - Opportunity to Load Balance

Map Input Transfer Rate

- Input locality
  - HDFS exposes block locations
  - Each map operates on one block
- Efficient decompression
  - More efficient in Hadoop 0.18
- Minimal deserialization overhead
  - Java deserialization is very verbose
  - Use Writable/Text

Performance Example

- Count lines in text files totaling several hundred GB
- Approach:
  - Identity Mapper (input: text, output: same text)
  - A single Reducer counts the lines and outputs the total
- What is wrong ?
- This happened, really!

Intermediate Output

- Almost always the most expensive component
  - (M x R) transfers over the network
  - Merging and Sorting
- How to improve performance:
  - Avoid shuffling/sorting if possible
  - Minimize redundant transfers
  - Compress

Avoid shuffling/sorting

- Set number of reducers to zero
  - Known as map-only computations
  - Filters, Projections, Transformations
- Beware of number of files generated
  - Each map task produces a part file
  - Make map produce equal number of output files as input files
    - How? Variable indicating current file being processed

Minimize Redundant Transfers

- Combiners
  - Goal is to decrease size of the transient data
- When maps produce many repeated keys
  - Often useful to do a local aggregation following the map
  - Done by specifying a Combiner
  - Combiners have the same interface as Reducers, and often are the same class.
  - Combiners must not have side effects, because they run an indeterminate number of times.
  - conf.setCombinerClass(Reduce.class);

Compress Output

- Compressing the outputs and intermediate data will often yield huge performance gains
  - Specified via a configuration file or set programatically
  - Set mapred.output.compress=true to compress job output
  - Set mapred.compress.map.output=true to compress map output
- Compression types:
  - mapred.output.compression.type
  - “block” - Group of keys and values are compressed together
  - “record” - Each value is compressed individually
  - Block compression is almost always best
- Compression codecs:
  - mapred.output.compression.codec
  - Default (zlib) - slower, but more compression
  - LZO - faster, but less compression

Opportunity to Load Balance

- Load imbalance inherent in the application
  - Imbalance in input splits
  - Imbalance in computations
  - Imbalance in partition sizes
- Load imbalance due to heterogeneous hardware
  - Over time performance degradation
- Give Hadoop an opportunity to do load-balancing
  - How many nodes should I allocate ?

Load Balance (contd.)

- M = total number of simultaneous map tasks
- M = map task slots per tasktracker * nodes
- Chose nodes such that total mappers is between 5*M and 10*M.

Configuring Task Slots

- mapred.tasktracker.map.tasks.maximum
- mapred.tasktracker.reduce.tasks.maximum
- Tradeoffs:
  - Number of cores
  - Amount of memory
  - Number of local disks
  - Amount of local scratch space
  - Number of processes
- Consider resources consumed by TaskTracker & Datanode processes

Speculative execution

- The framework can run multiple instances of slow tasks
  - Output from instance that finishes first is used
  - Controlled by the configuration variable mapred.speculative.execution=[true|false]
  - Can dramatically bring in long tails on jobs

Performance

- Is your input splittable?
  - Gzipped files are NOT splittable
- Are partitioners uniform?
- Buffering sizes (especially io.sort.mb)
- Do you need to Reduce?
- Only use singleton reduces for very small data
  - Use Partitioners and cat to get a total order
- Memory usage
  - Do not load all of your inputs into memory.

Debugging & Diagnosis

- Run job with the Local Runner
  - Set mapred.job.tracker to “local”
  - Runs application in a single process and thread
- Run job on a small data set on a 1 node cluster
  - Can be done on your local dev box
- Set keep.failed.task.files to true
  - This will keep files from failed tasks that can be used for debugging
  - Use the IsolationRunner to run just the failed task
- Java Debugging hints
  - Send a kill -QUIT to the Java process to get the call stack, locks held, deadlocks

Example: Computing Standard Deviation

- Takeaway: Changing algorithm to suit architecture yields best implementation

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

Implementation 1

- Two Map-Reduce stages
- First stage computes Mean
- Second stage computes std deviation

Implementation 1 (contd.)

- Stage 1: Compute Mean
  - Map Input (xi for i = 1 ..Nm)
  - Map Output (Nm, Mean(x1..Nm))
  - Single Reducer
  - Reduce Input (Group(Map Output))
  - Reduce Output (Mean(x1..N))

Implementation 1 (contd.)

- Stage 2: Compute Standard deviation
  - Map Input (xi for i = 1 ..Nm) & Mean(x1..N)
  - Map Output (Sum(xi – Mean(x))2 for i = 1 ..Nm
  - Single Reducer
  - Reduce Input (Group (Map Output)) & N
  - Reduce Output (Standard Deviation)
- Problem: Two passes over large input data

Implementation 2

- Second definition algebraic equivalent
  - Be careful about numerical accuracy, though

$$\sigma = \sqrt{\frac{1}{N}\left(\sum_{i=1}^{N} x_i^{\,2} - N\overline{x}^{2}\right)}$$

Implementation 2 (contd.)

- Single Map-Reduce stage
- Map Input (x_i for $i = 1 .. N_m$)
- Map Output (N_m, [Sum($x^2_{1..Nm}$),Mean($x_{1..Nm}$)])
- Single Reducer
- Reduce Input (Group (Map Output))
- Reduce Output (σ)
- Advantage: Only a single pass over large input

Q&A