Introduction to Hadoop, MapReduce and HDFS for Big Data Applications

Serge Blazhievsky
Nice Systems

SNIA Legal Notice

- The material contained in this tutorial is copyrighted by the SNIA unless otherwise noted.
- Member companies and individual members may use this material in presentations and literature under the following conditions:
  - Any slide or slides used must be reproduced in their entirety without modification
  - The SNIA must be acknowledged as the source of any material used in the body of any document containing material from these presentations.
- This presentation is a project of the SNIA Education Committee.
- Neither the author nor the presenter is an attorney and nothing in this presentation is intended to be, or should be construed as legal advice or an opinion of counsel. If you need legal advice or a legal opinion please contact your attorney.
- The information presented herein represents the author's personal opinion and current understanding of the relevant issues involved. The author, the presenter, and the SNIA do not assume any responsibility or liability for damages arising out of any reliance on or use of this information.

NO WARRANTIES, EXPRESS OR IMPLIED. USE AT YOUR OWN RISK.

You Will Learn About:

- Hadoop history and general information
- Hadoop main components and architecture
- How to work with HDFS
- MapReduce and how it works

Chapter 1: Introduction to Hadoop

- The amount of data processing in today's life
- What Hadoop is why it is important
- Hadoop comparison with traditional systems
- Hadoop history
- Hadoop main components

There is Lots of Data out There!

Real Facts:

- New York Stock Exchange generates 1TB/day
- Google processes 700PB/month
- Facebook hosts 10 billion photos taking 1PB of storage

Amount of Data will Only Grow!

- Machine/System Logs
- Retail transaction logs
- Vehicle GPS traces
- Social Data (Facebook, Twitter, etc…)

Importance of Data

- Business Success today heavily depends on:
  - Ability to Store and Analyze large data sets…
    - Netflix – Folks who purchased movie A are more likely to also purchase movie B and C
  - Ability to extract other organizations' data…
    - Amazon Web Services
    - Mashup Applications

Netflix paid 1 million dollars to solve big data problem !!!!

Data Storage and Analysis – Problem #1

- Problem #1: Read/Write to disk is slow
  - 1TB drives are the norm, transfer speed is still at 100Mb/sec
- Solution: Use multiple disks for parallel reads
  - 1HD = 100Mb/sec
  - 100HD = 10Gb/sec

Data Storage and Analysis – Problem #2

- Problem #2: Hardware Failure
  - Single machine failure
  - Single disk failure
- Solution:
  - Keep multiple copies of Data
  - RAID or HDFS

Data Storage and Analysis – Problem #3

- Problem #3: How do you merge data from different reads?
  - Only complete results needs to be taken into consideration and results of the failed jobs will be ignored
  - Data needs to be in compressed format for network transmission
- Solution: Distributed Processing or Hadoop MapReduce

High Performance Computing (HPS) and Grid Computing

- Large-scale data processing has been done for years
- Distribute work across cluster of machines with shared filesystem
- Example: DNA sequencing, stock market predictions, etc.

Grid Computing Challenges

- Works well for process-intensive jobs
  - (Monte-Carlo simulation, DNA sequencing, etc.)
- Problematic accessing large data volumes
- Network bandwidth can be a bottleneck, causing nodes become idle
- Coordinating processes and node failures is a challenge

Hadoop Main Components: HDFS and MapReduce

- Hadoop provides a reliable shared storage and analysis system for large-scale data processing
  - Storage provided by **HDFS**
  - Analysis provided by **MapReduce**

Hadoop MapReduce vs RDMS

	Traditional RDMS	Hadoop MapReduce
Data size	Gigabytes	Petabytes
Access	Interactive	Batch
Updates	Read and Write many times	Write once, Read many times
Structure	Static schema	Dynamic schema
Integrity	High	Low in simple setup, can be improved with additional servers
Scaling	Non-linear	Linear(up to 10,000 machines as of Dec 2012)

Hadoop MapReduce vs RDMS… cont.

- MapReduce fits well to analyze whole dataset in a batch fashion
- RDMS is for real-time data retrieval with low-latency and small data sets

Hadoop Origin

- Created by Doug Cutting, part of Apache project
- 2004: Google publishes GFS paper
- 2005: Nutch (open source web search) uses MapReduce
- 2008: Becomes Apache top-level project, was Lucene sub-project before.
- 2009: Yahoo used Hadoop to sort 1TB in 62 sec.
- 2013: Hadoop is used by hundreds of the companies

Hadoop Components

- **HDFS** – Distributed Filesystem
- **MapReduce** – Distributed Data Processing Model

Hadoop Components… cont.

- **Hive** – Distributed Data Warehouse, provides SQL-based query language
- **HBase** – Distributed column-based database
- **Pig** – Data Flow Language and execution environment

Summary

- In this chapter we have covered:
  - What Hadoop is and why it is important
  - Hadoop comparison with traditional systems
  - Hadoop origin
  - Hadoop main components
- Questions?

Chapter 2:
Hadoop Distributed File System (HDFS)

- HDFS Overview and Design
- HDFS Architecture
- HDFS File Storage
- Component Failures and Recoveries
- Block Placement
- Balancing the Cluster

HDFS Overview

- Based on Google's GFS (Google File System)
- Provides redundant storage of massive amounts of data
  - Using commodity hardware
- Data is distributed across all nodes at load time
  - Provides for efficient Map Reduce processing (discussed later)

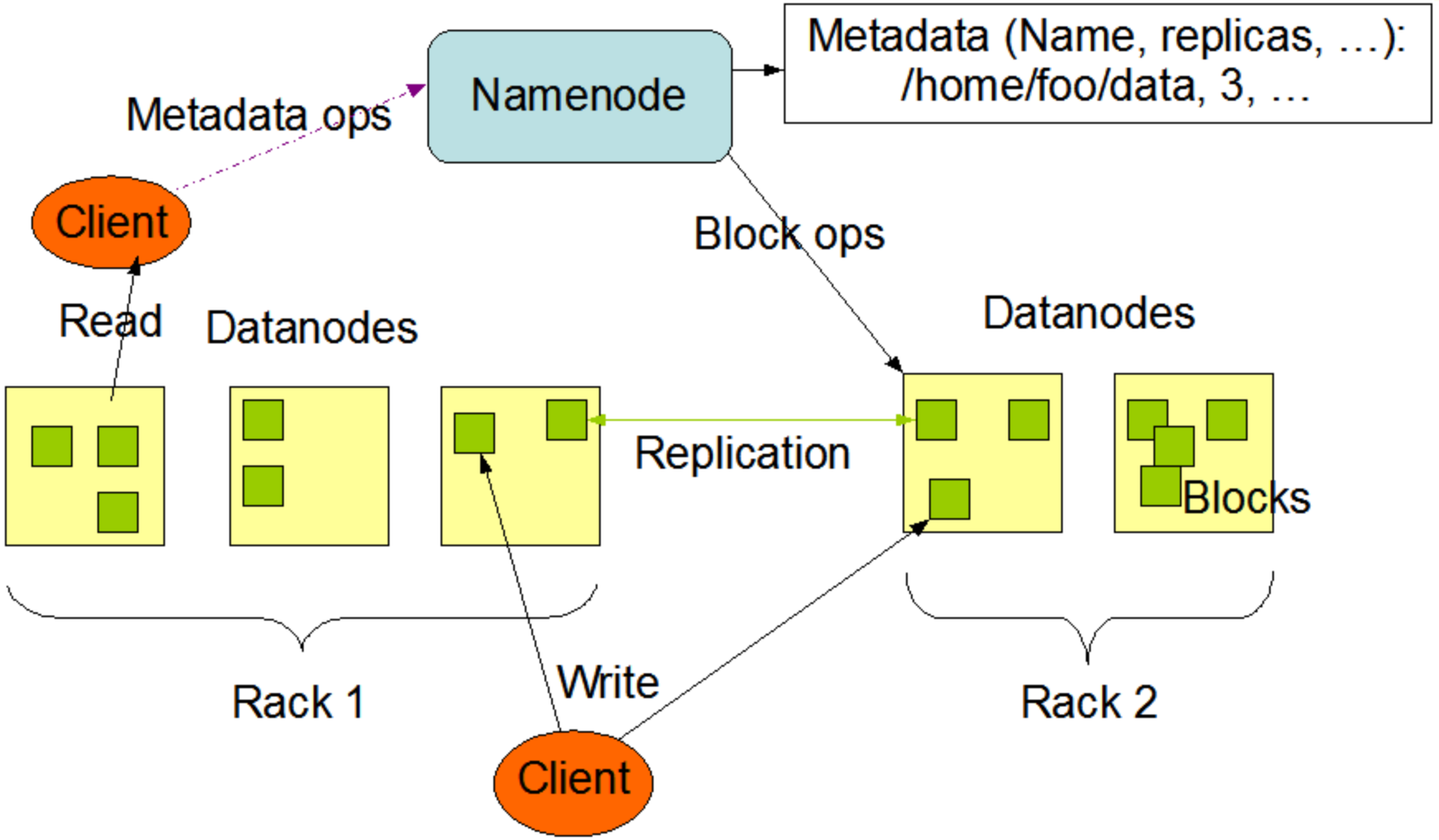
HDFS Design

- Runs on commodity hardware
  - Assumes high failure rates of the components
- Works well with lots of large files
  - Hundreds of Gigabytes or terabytes in size
- Built around the idea of "write-once, read-many-times"
- Large streaming reads
  - Not random access
- High throughput is more important than low latency

HDFS Architecture

- Operates on top of an existing filesystem
- Files are stored as ‘Blocks’
  - Block’s default size – 64MB
- Provides reliability through replication
  - Each Block is replicated across several Data Nodes
- NameNode stores metadata and manages access
- No data caching due to large datasets

HDFS Architecture Diagram



HDFS File Storage

- NameNode
  - Stores all metadata: filenames, locations of each block on DataNodes, file attributes, etc…
  - Keeps metadata in RAM for fast lookup
  - Filesystem metadata size is limited to the amount of available RAM on NameNode
- DataNode
  - Stores file contents as blocks
  - Different blocks of the same file are stored on different DataNodes
  - Same block is replicated across several DataNodes for redundancy
  - Periodically sends a report of all existing blocks to the NameNode

DataNode Failure and Recovery

- DataNodes exchange heartbeats with NameNode
- If no heartbeat received within a certain time period – DataNode is assumed to be lost
  - NameNode determines which blocks were on the lost node
  - NameNode finds other copies of these ‘lost’ blocks and replicates them to other nodes
  - Block replication is actively maintained

NameNode Failure

- Losing a NameNode is equivalent to losing all the files on the filesystem
- Hadoop provides two options:
  - Back up files that make up the persistent state of the filesystem (local or NFS mount)
  - Run a Secondary NameNode

Secondary NameNode

- Is not a failover NameNode
  - A real HA solution still needed
- Performs memory-intensive administrative functions
  - NameNode keeps metadata in memory and writes changes to an Editlog.
  - Secondary NameNode periodically combines a prior filesystem snapshot and Editlog into a new snapshot
  - New snapshot is sent back to the NameNode
- Recommended to run on a separate machine
  - It requires as much RAM as the primary NameNode

Block Placement

- Default strategy:
  - One replica on local node
  - Second replica on a node in the remote rack
  - Third replica on the some node in the remote rack
  - Additional replicas are random
- Clients always read from nearest node

Block Placement … cont.

- Client retrieves a list of DataNodes on which to place replicas of a block
- Client writes block to the first DataNode
- The first DataNode forwards the data to the next DataNode in the Pipeline
- When all replicas are written, the Client moves on to write the next block in file
- Please, note that multiple machines are involved in writing one files and they could be different machines

Balancing Hadoop Cluster

- Hadoop works best when blocks are evenly spread out
- Goal: Have % Disk Full on DataNodes at the same level
- Balancer – Hadoop daemon
  - ***% start-balancer.sh***
  - Re-distributes blocks from over-utilized to under-utilized DataNodes
  - Run it when new DataNodes are added
  - Runs in the background and can be throttled to avoid network congestion/negative cluster impact

Summary

- In this chapter we have covered:
  - HDFS Overview and Design
  - HDFS Architechture
  - HDFS File Storage
  - Component Failures and Recoveries
  - Block Placement
  - Balancing the Cluster
- Questions?

Chapter 3: Working with HDFS

- Ways of accessing data in HDFS
- Common HDFS operations and commands
- Different HDFS commands
- Internals of a file read in HDFS
- Data copying with ‘distcp’

Ways of Accessing Data in HDFS

Data can be accessed with the following methods:

- Programmatically via Java API
- Via command line
- Via web-interface

Most Common HDFS Operations

- Creating directory
- Removing directory
- Copying files to/from HDFS
- List content of the directory
- Display file content
- Analyze space allocation
- Check permissions/write ability
- Set replication for specific directory

Commands

- hadoop dfs -mkdir <path>
  - Create a directory in specified location
- hadoop dfs -rmr <src>
  - Remove all directories which match the specified file pattern
- hadoop -put <localsrc> ... <dst>
  - copy files from the local file system into fs

Commands… cont.

- hadoop dfs -copyFromLocal <localsrc> ... <dst>
  - Identical to the -put command
- hadoop dfs -moveFromLocal <localsrc> ... <dst>
  - Same as -put, except that the source is deleted after it's copied.

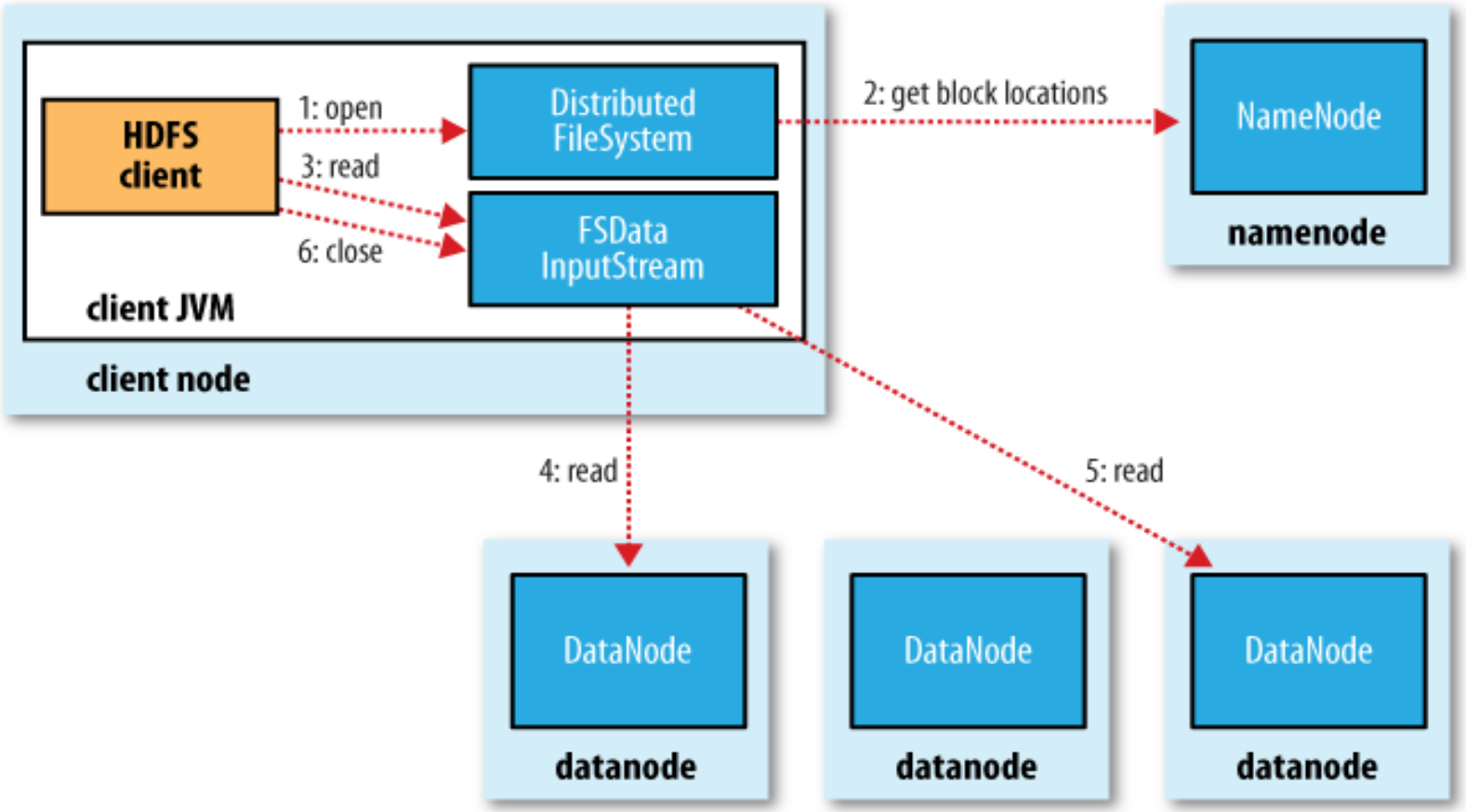
Commands… cont.

- -ls <path>
  - List the contents that match the specified file pattern
- -lsr <path>
  - Recursively list the contents that match the specified file pattern
- -cat <src>:
  - Fetch all files that match the file pattern <src> and display their content on stdout

Commands… cont.

- -df [<path>]
  - Shows the capacity, free and used space of the filesystem
- -du <path>
  - Show the amount of space, in bytes, used by the files that match the specified file pattern
- -touchz <path>
  - Create empty file at <path>

Internals of a File Read in HDFS



Internals of a File Read… cont.

- Client is guided by the NameNode to the best DataNode for each block
- Client contacts DataNodes directly to retrieve data
- NameNode only services block locations requests
- This design allows HDFS to scale to large number of clients

Parallel Copying with distcp

- Hadoop comes with a useful program called distcp for copying large amounts of data to and from Hadoop filesystems in parallel
- hadoop distcp <src> <dst>
- distcp is implemented as a MapReduce job where of copying is done the maps that run in parallel across the cluster

Summary

- In this chapter we have covered:
  - Ways of accessing data in HDFS
  - Common HDFS operations and commands
  - Different HDFS commands
  - Internals of a file read in HDFS
  - Data copying with ‘distcp’
- Questions?

Chapter 4: Map Reduce Abstraction

- What MapReduce is and why it is popular
- The Big Picture of the MapReduce
- MapReduce process and terminology
- MapReduce components failures and recoveries

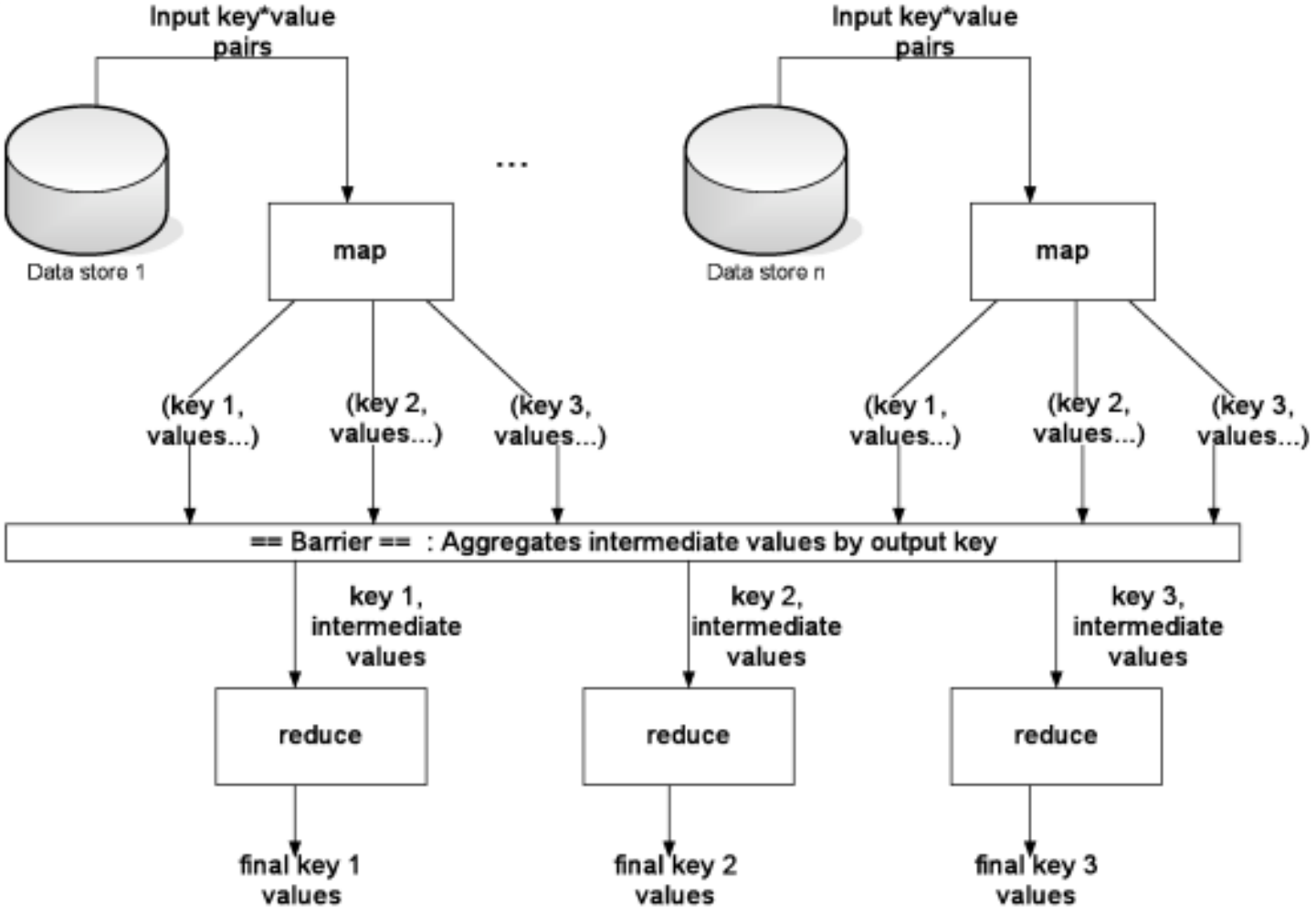
What Is MapReduce?

- MapReduce is a method for distributing a task across multiple nodes
- Each node processes data stored on that node
- Consists of two phases:
  - Map
  - Reduce
- Map: (K1, V1) -> (K2, V2)
- Reduce: (K2, list(V2)) -> list(K3, V3)

Why Map Reduce is So Popular?

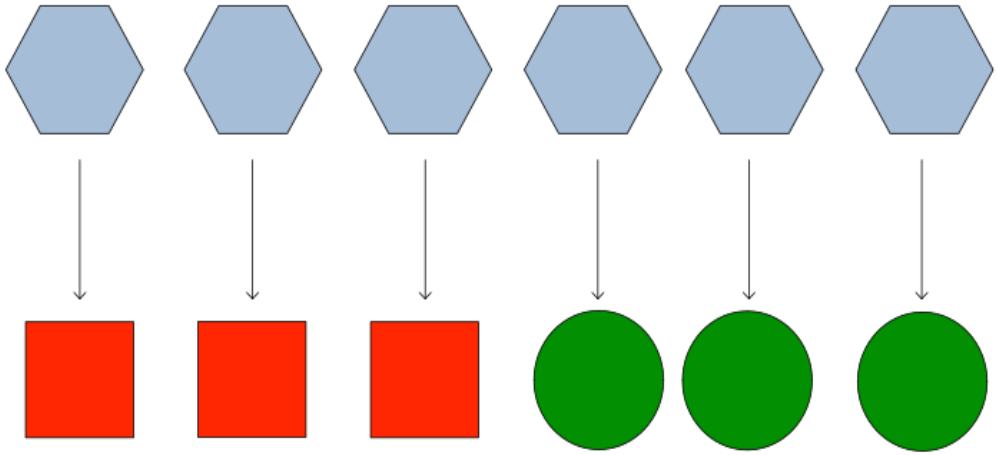
- Automatic parallelization and distribution (The biggest advantage!!!)
- Fault-tolerance (individual tasks can be retried)
- Hadoop comes with standard status and monitoring tools
- A clean abstraction for developers
- MapReduce programs are usually written in Java (possibly in other languages using streaming)

MapReduce: The Big Picture



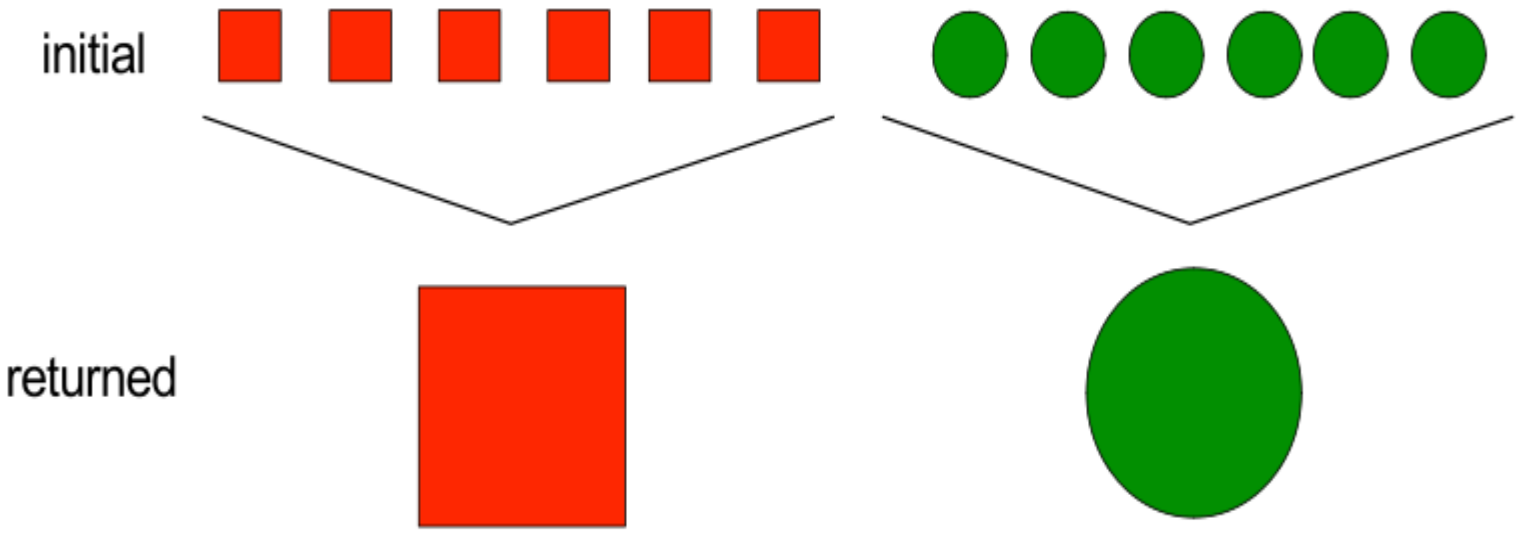
Map Process

- **map (in_key, in_value) -> (out_key, out_value)**



Reduce Process

- **reduce (out_key, out_value list) -> (final_key, final value) list**



MapReduce: Word Count Example

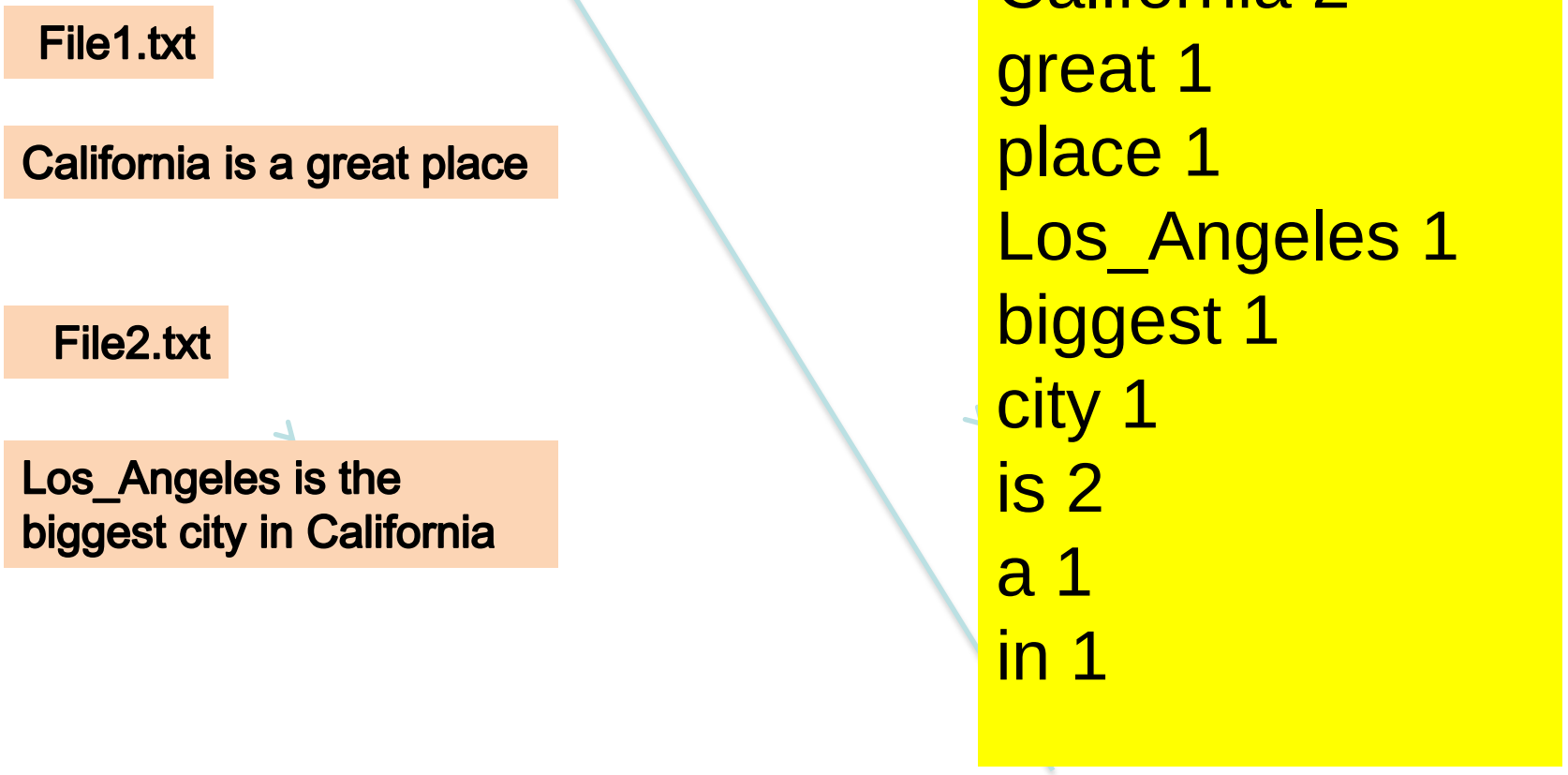
Map

```
// assume input is a
// set of text files
// k is a line offset
// v is the line for that offset

let map(k, v) =
foreach word in v:
emit(word, 1)
```

Reduce

```
// k is a word
// vals is a list of 1s

let reduce(k, vals) =
emit(k, vals.length())
```

Input/Output for Map Reduce Job



Word Count Mapper

Mapper

California is a great place

	Key	Value
California	California	1
is	is	1
a	a	1
great	great	1
place	place	1

Word Count Mapper 2

SNIA Education

Mapper

Key	Value
California	1
is	1
a	1
great	1
place	1

Key	Value
Los_Angeles	1
is	1
the	1
biggest	1
city	1
in	1
California	1

Word Count Mapper Reducer Transition

Mapper to Reducer

Key	Value
California	1
is	1
a	1
great	1
place	1

Key	Value
Los_Angeles	1
is	1
the	1
biggest	1
city	1
in	1
California	1

Word Count Mapper Reducer Transition

Mapper to Reducer

Key	Value
California	1
is	1
a	1
great	1
place	1

Key	Value
Los_Angeles	1
is	1
the	1
biggest	1
city	1
in	1
California	1

Word Count Input to Reducer

Key	Values
California	{1,1}
Los_Angeles	1

Key	Values
a	1
is	{1,1}
the	1
biggest	1
city	1
in	1
great	1
place	1

Java Map Method

```
public static class Map extends MapReduceBase implements
   Mapper<LongWritable, Text, Text, IntWritable> {
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();

   public void map(LongWritable key, Text value,
               OutputCollector<Text,IntWritable> output,
               Reporter reporter) throws IOException {
       String line = value.toString();
       StringTokenizer tokenizer = new
   StringTokenizer(line);
       while (tokenizer.hasMoreTokens()) {
               word.set(tokenizer.nextToken());
               output.collect(word, one);
       }
   }
}
```

Word Count Mapper Code

Let's take a look how we can put together code to drive this data flow:

```
 public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
```

Key – bytes offset from the beginning of the data file
Value – line of the data file
Context – is an object that collects output and has other reporting capabilities

Word Count Reducer Output

Key	Values
California	2
Los_Angeles	1

Key	Values
a	1
is	2
the	1
biggest	1
city	1
in	1
great	1
place	1

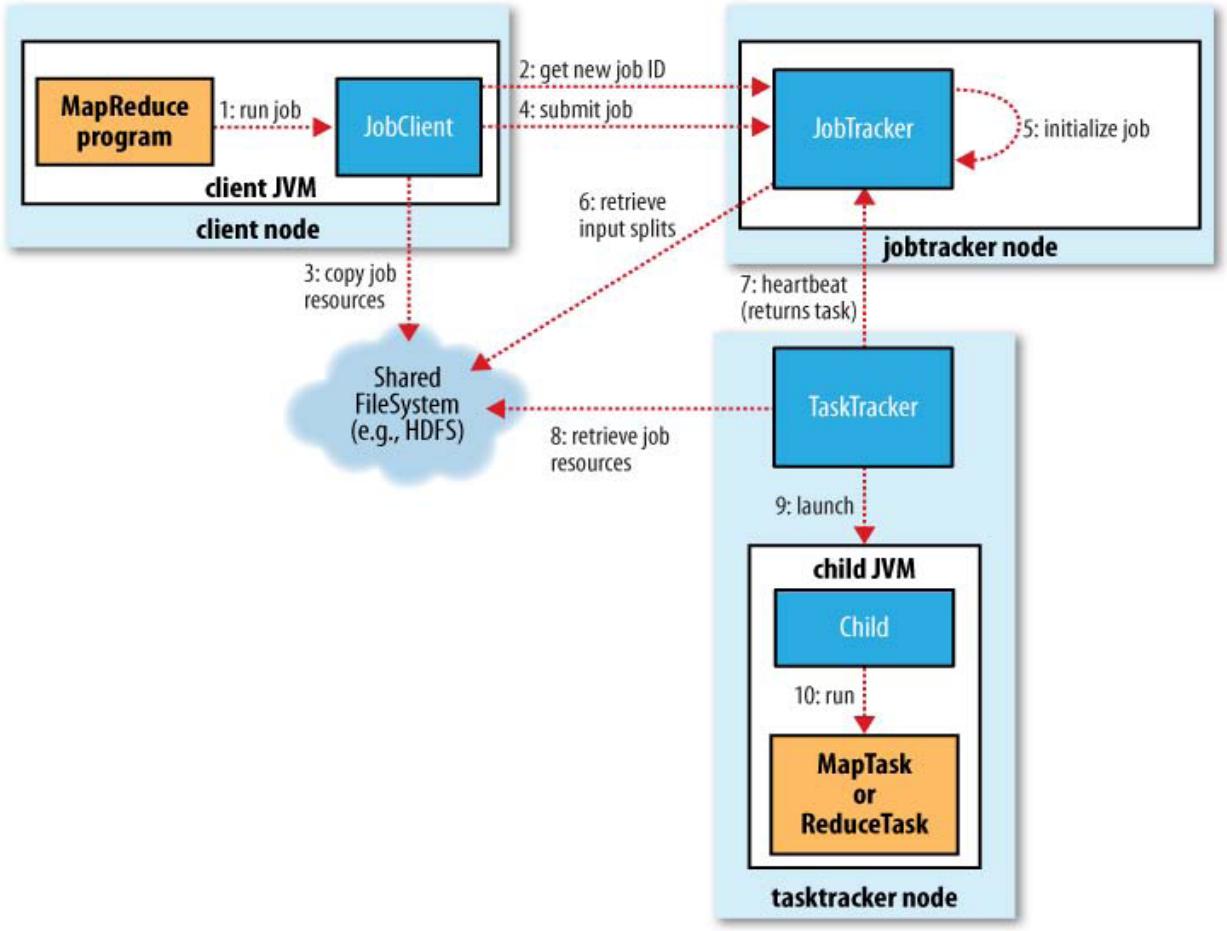
Word Count Reduce Code

```
public static class Reduce extends MapReduceBase implements
   Reducer<Text, IntWritable, Text, IntWritable> {

   public void reduce(Text key, Iterator<IntWritable> it,
       OutputCollector<Text, IntWritable> output,
       Reporter reporter) throws IOException {
       int sum = 0;
       while (it.hasNext()) {
              sum += it.next().get();
       }
       output.collect(key, new IntWritable(sum));
   }
}
```

Terminology

- The client program submits a job to Hadoop
  - The job consists of a mapper, a reducer, and a list of inputs
- The job is sent to the JobTracker process on the Master Node
- Each Slave Node runs a process called the TaskTracker
- The JobTracker instructs TaskTrackers to run and monitor tasks
- A Map or Reduce over a piece of data is a single task
- A task attempt is an instance of a task running on a slave node

MapReduce: High Level



MapReduce Failure Recovery

- Task processes send heartbeats to the TaskTracker!
- TaskTrackers send heartbeats to the JobTracker!
- Any task that fails to report in 10 minutes is assumed to have failed – its JVM is killed by the TaskTracker!
- Any task that throws an exception is said to have failed!
- Failed tasks are reported to the JobTracker by the TaskTracker!
- The JobTracker reschedules any failed tasks – it tries to avoid rescheduling the task on the same TaskTracker where it previously failed!
- If a task fails more than 4 times, the whole job fails

Task Tracker Recovery

- Any TaskTracker that fails to report in 10 minutes is assumed to have crashed
  - All tasks on the node are restarted elsewhere
  - Any TaskTracker reporting a high number of failed tasks is blacklisted, to prevent the node from blocking the entire job
  - There is also a "global blacklist", for TaskTrackers which fail on multiple jobs!
- The JobTracker manages the state of each job – partial results of failed tasks are ignored

Summary

- In this chapter we have covered:
  - What MapReduce is and why it is popular
  - The Big Picture of the MapReduce
  - MapReduce process and terminology
  - MapReduce components failures and recoveries
- Questions?

Attribution & Feedback

The SNIA Education Committee thanks the following individuals for their contributions to this Tutorial.

Authorship History

Serge Blazhievsky, September 2013

Updates:

Serge Blazhievsky
October 2013

Additional Contributors

Please send any questions or comments regarding this SNIA Tutorial to ***tracktutorials@snia.org***