MapReduce: Simplified Data Processing on Large Clusters

Authors: Jeffery Dean and Sanjay Ghemawat

Google, Inc.

This paper explains how MapReduce works.

Agenda

- What is MapReduce (MR)?
- Why MR?
- Commodity Machine
- Hadoop Architecture
- Word count problem
- Job Execution
- Localization
- Commands
- Conclusion

What is MapReduce??

- MapReduce is a programming model and an associated implementation for processing and generating large data sets!
- There are two basic functions used in this model. Namely, “map” and “reduce”.
- map: processes a key/value pair to generate a set of intermediate key/value pairs. [map(String key, String value)]
- reduce: merges all intermediate values associated with the same intermediate key. [reduce(String key, Iterator values)]

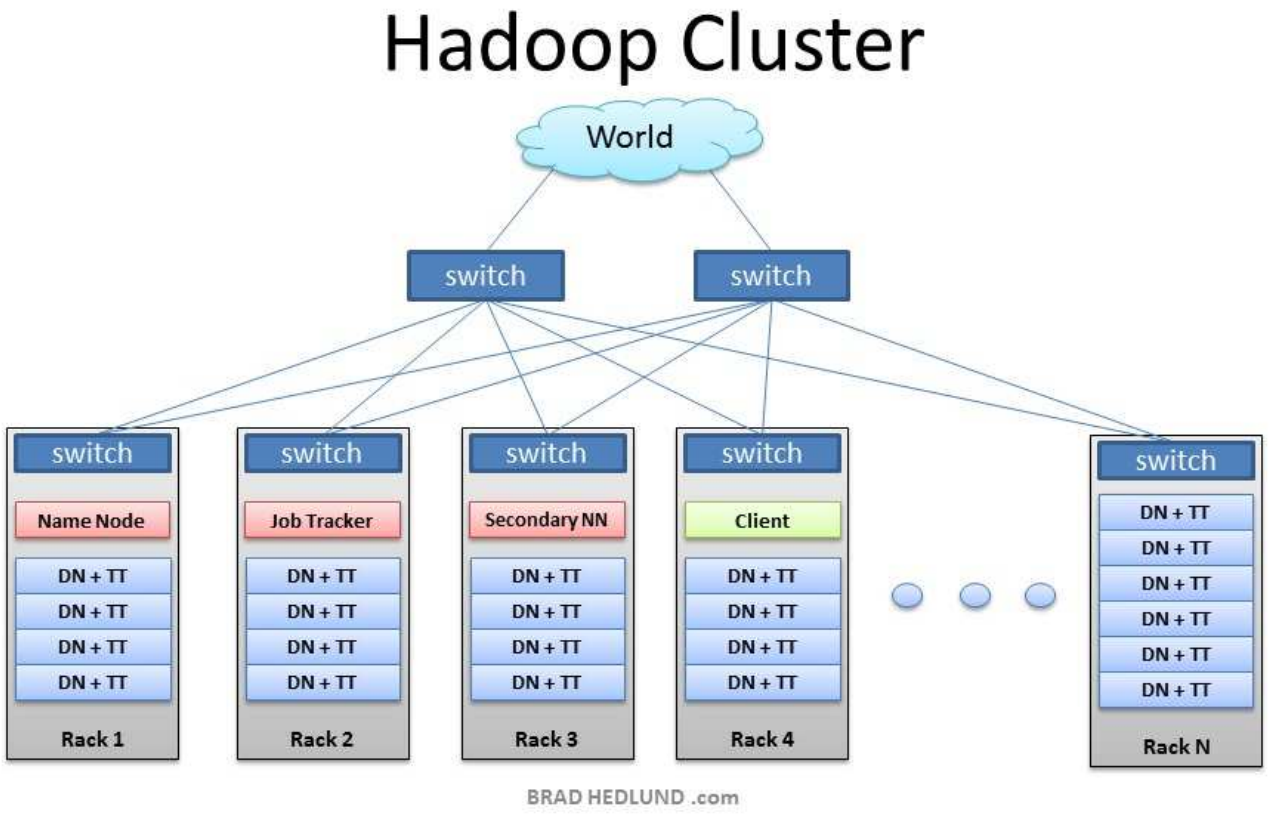
Why use MapReduce?

- Imagine you have a large scale data file that has to be processed.
- Using a single machine to process such a file will take hours or probably days.
- While processing, you might experience troubles such as a system crash which would ultimately require you to start this process all over again. Other kinds of troubles include I/O Scheduling, managing distribution of resources etc.
- Using MapReduce will take care of such problems.
- MapReduce is usually executed on a cluster of commodity hardware.

What is a commodity machine?

- An off-the-shelf device that is readily available for purchase. Commodity PCs and servers often refer to x86 machines, which are the world's largest desktop, laptop and server platform.
- The network bandwidth used to connect these machines is typically 1 Gbps or 100 Mbps.
- Storage includes inexpensive IDE disks attached directly to individual machines. Each disk uses an in-house DFS to manage the data stored. This file system provides availability and reliability on top of unreliable machines.
- A cluster consists of 100s or 1000s of such systems.
- The tasks are submitted to the scheduling system and are then mapped to the available machines by the scheduler.

Cluster Architecture



Name Node: A master node.
Data Node: a worker node where the MapReduce job is executed.
Task Tracker: runs on every data node and manages individual MapReduce tasks.
Job Tracker: Responsible for handling jobs submissions.
Secondary NN: secondary name node is a backup master node.

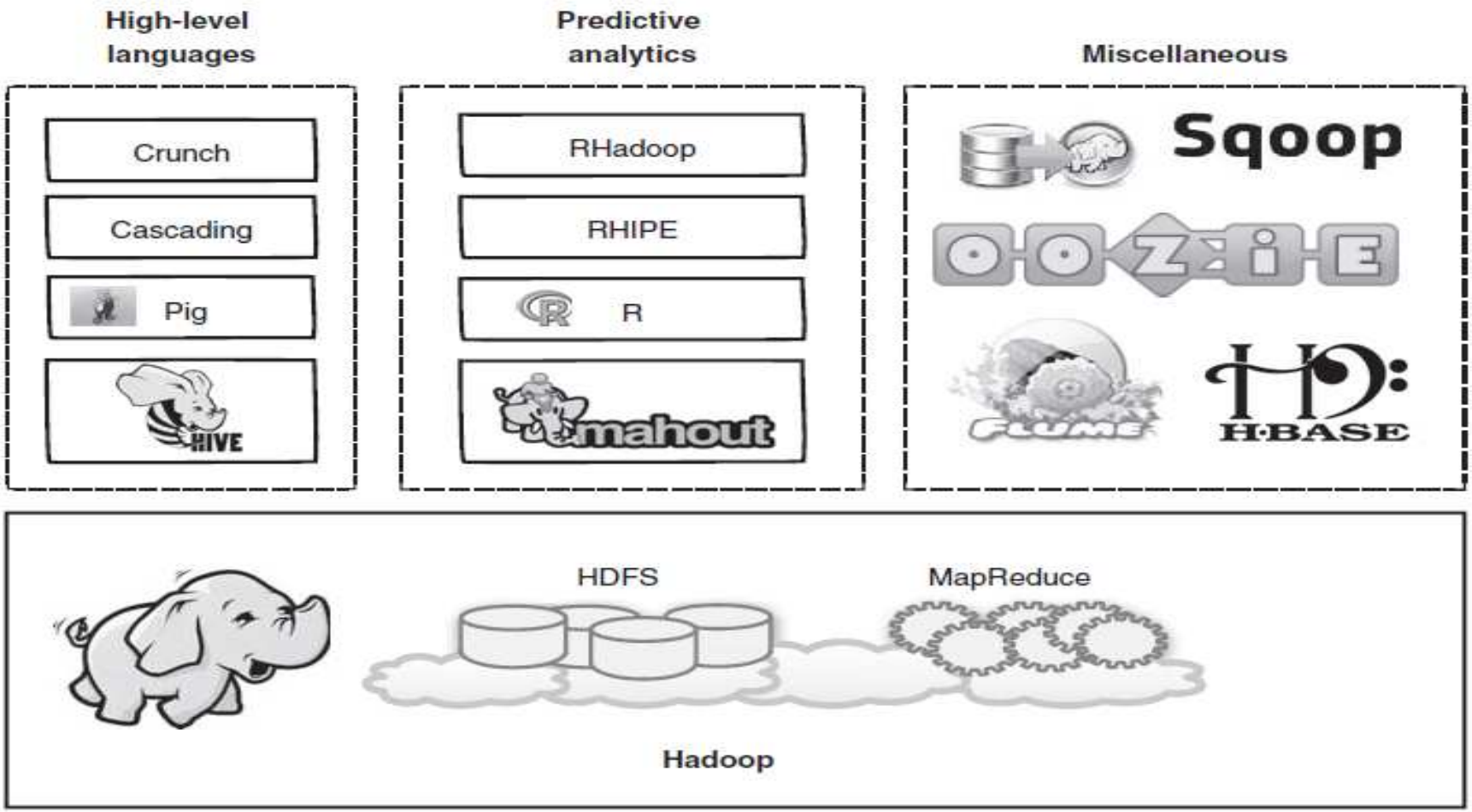


Figure 1.9 **Hadoop and related technologies**

Word Count Problem

Count the number of occurrences of a word

Basic Functions in MR

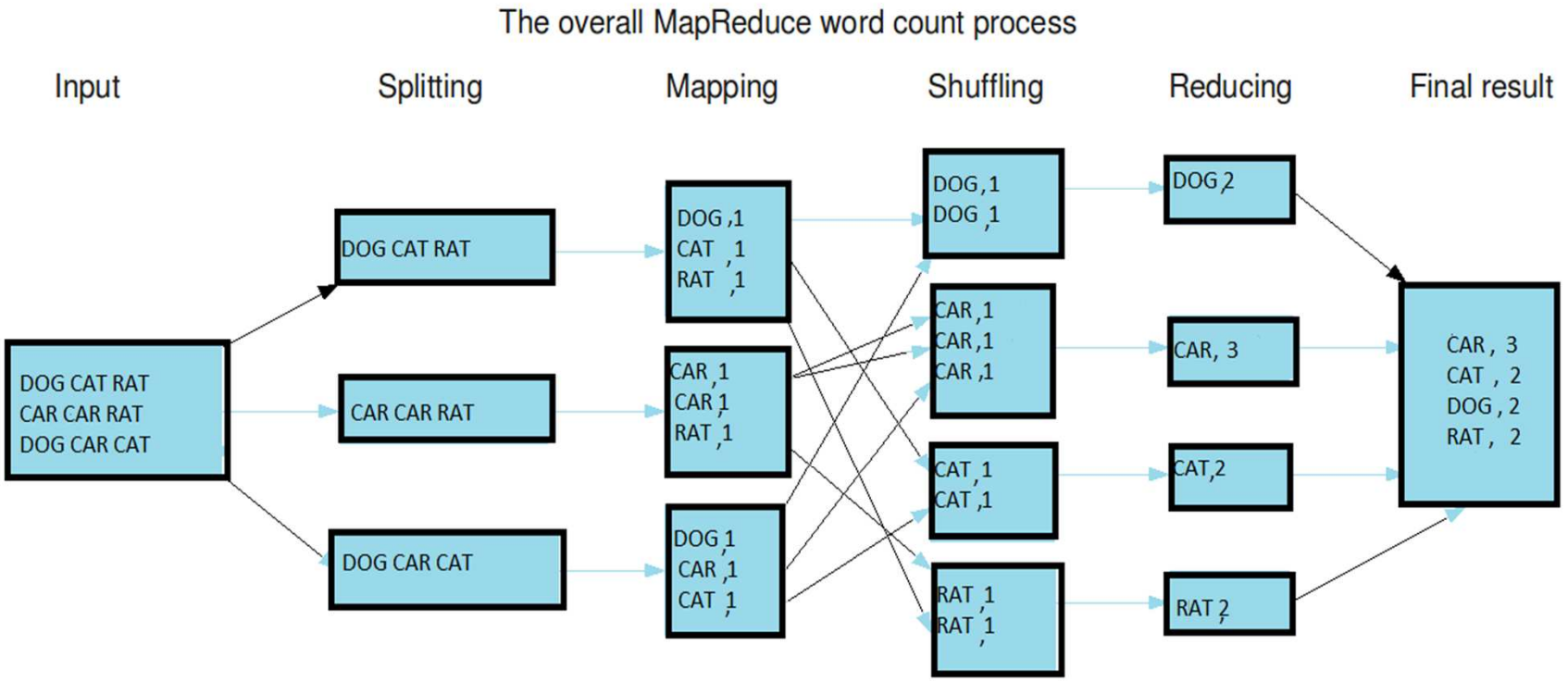
`implements Mapper`

- public void **map**(LongWritable **key**, Text **value**, OutputCollector<Text, IntWritable> output, Reporter reporter)

`implements Reducer`

- public void **reduce**(Text **key**, Iterator<IntWritable> **values**, OutputCollector<Text, IntWritable> output, Reporter reporter)

Programming Model



Input and Output types

```
public abstract interface Writable {
    public abstract void write(DataOutput paramDataOutput) throws IOException;

    public abstract void readFields(DataInput paramDataInput) throws IOException;
}
```

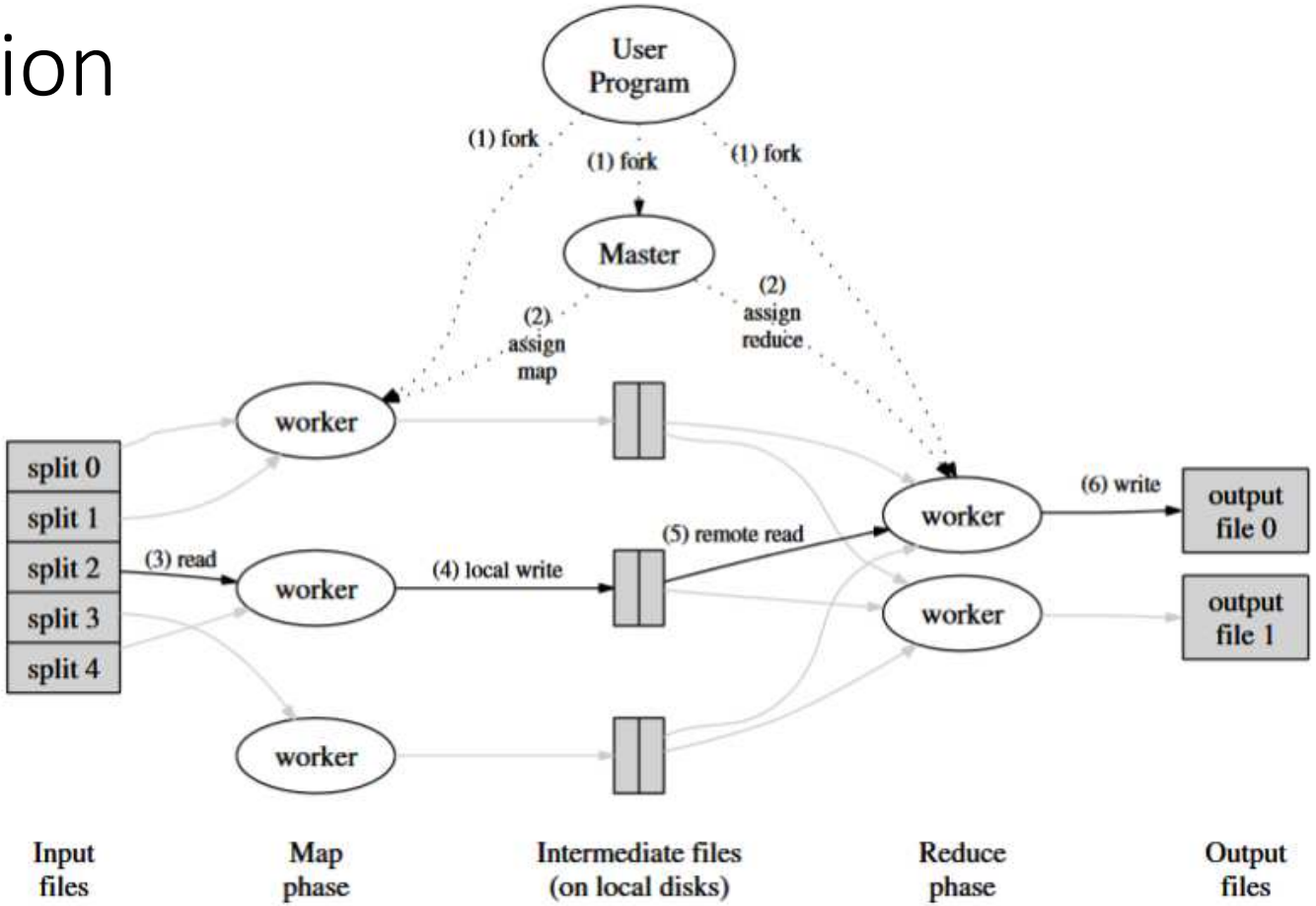
A serializable object which implements a simple, efficient, serialization protocol, based on `DataInput` and `DataOutput`.

Any `key` or `value` type in the Hadoop Map-Reduce framework implements this interface.

Other functions

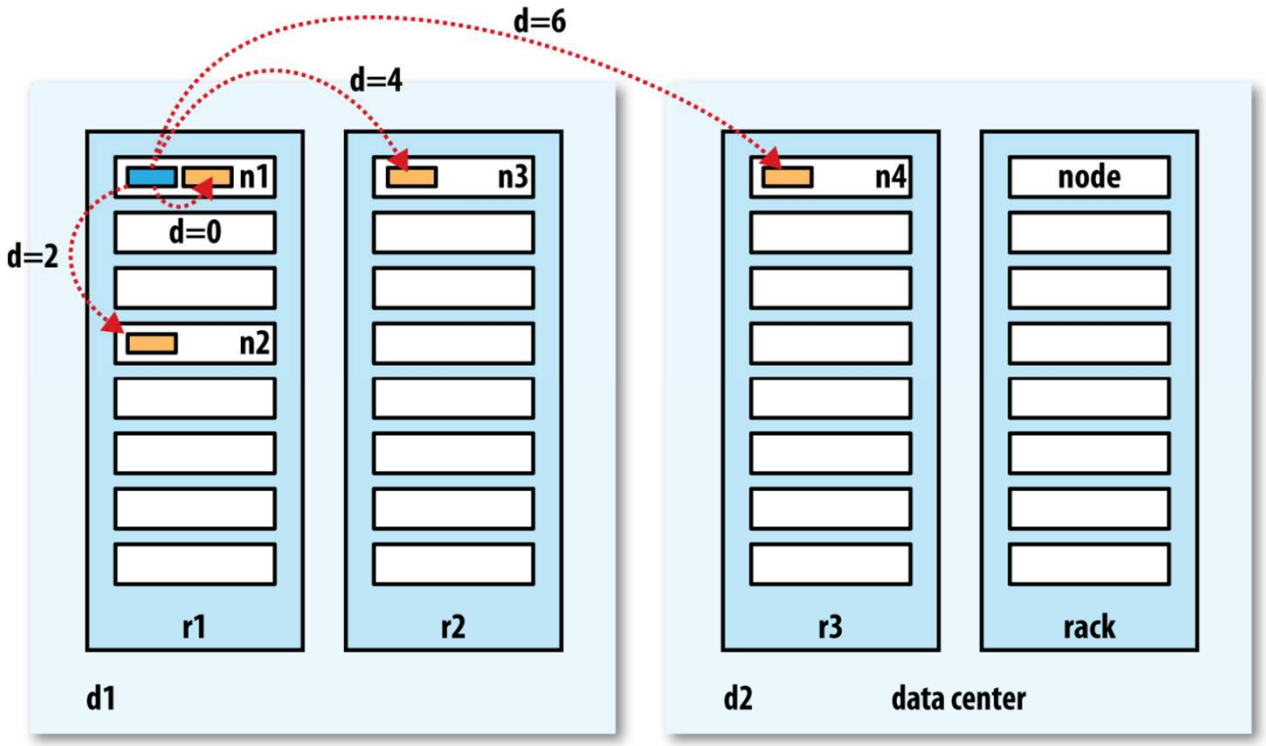
- Combiner (local reducer)
- Practitioner (condition)
- Splits (control split)

Submitted Job Execution



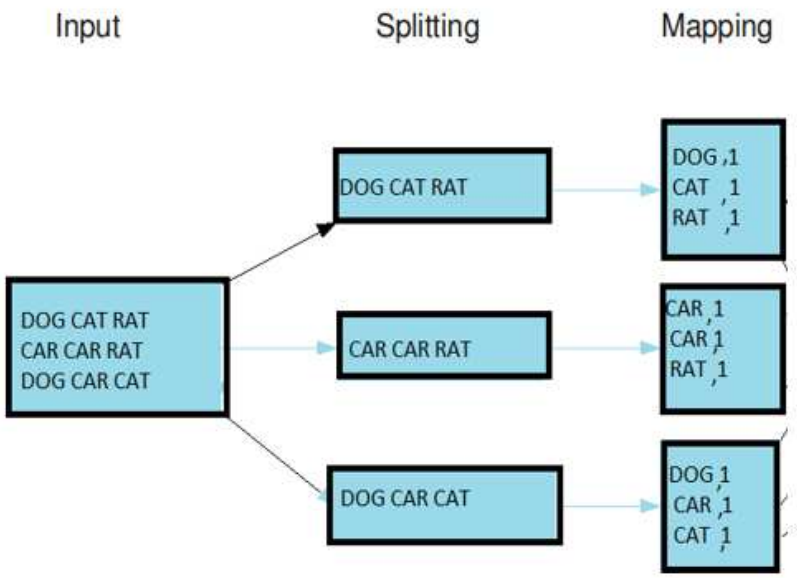
Localization

- Data Local
- Rack Local
- Different Rack



Side-effects

- Intermediate files



Skipping Bad Records

- Handle error in mapper/ reducer code and ignore a record gracefully
- Use Class SkipBadRecords "Utility class for skip bad records functionality. It contains various settings related to skipping of bad records."

org.apache.hadoop.mapred

Class SkipBadRecords

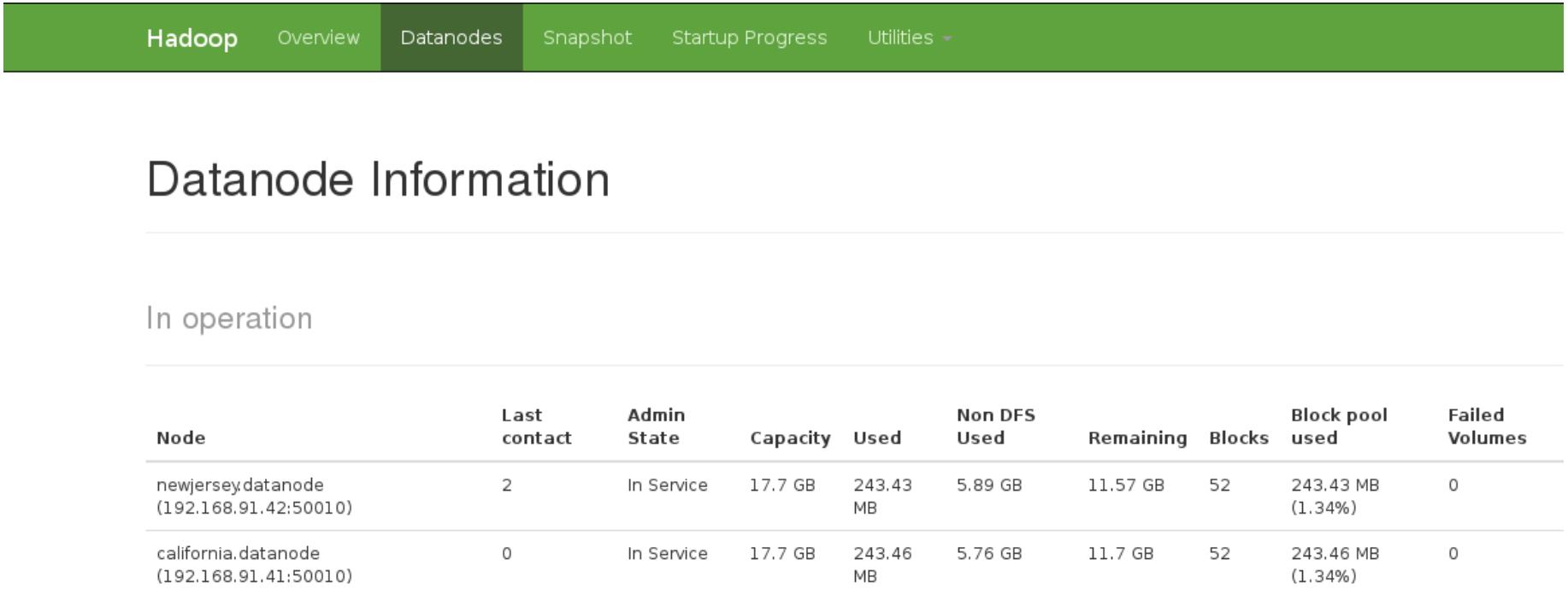
java.lang.Object
  └ **org.apache.hadoop.mapred.SkipBadRecords**

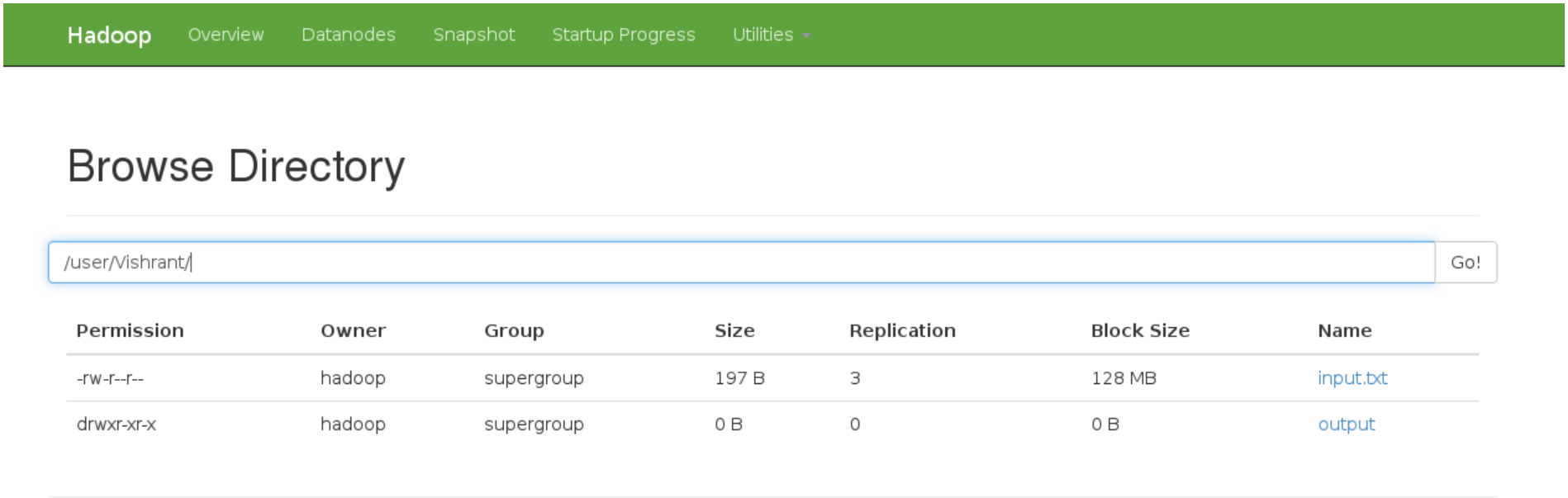
```
@InterfaceAudience.Public
@InterfaceStability.Stable
public class SkipBadRecords
extends Object
```

Utility class for skip bad records functionality. It contains va

Hadoop provides an optional mode of execution in which th

Status Information




Hadoop
Overview
Datanodes
Snapshot
Startup Progress
Utilities
Browse Directory
/user/Vishrant/
Go!
Permission
Owner
Group
Size
Replication
Block Size
Name
-rw-r--r--
hadoop
supergroup
197 B
3
128 MB
input.txt
drwxr-xr-x
hadoop
supergroup
0 B
0
0 B
output

Local execution

- MR is hard to debug, local execution can help facilitate debugging, profiling, and small-scale testing

Counters

- The MapReduce library provides a counter facility to count occurrences of various events. For example, user code may want to count total number of words processed or the number of German documents indexed, etc.
- There can be custom counters for user specific tasks.
- Users have found the counter facility useful for sanity checking the behavior of MapReduce operations.

Backup Tasks

- Hadoop Speculative execution
- Default enabled

Large-Scale Indexing

- Google used MR to completely rewrite the production indexing system that produces the data structures used for the Google web search service

Related Work

- Bulk Synchronous Programming
- Charlotte System
- BAD-FS
- TACC

Conclusions

- MR model has been successfully used at Google and it have many advantages like parallelization, fault-tolerance.