Using Map-Reduce to Teach Parallel Programming Concepts

Dick Brown, St. Olaf College
Libby Shoop, Macalester College
Joel Adams, Calvin College

Macalester College

Calvin
Minds in the Making

St. Olaf College

Workshop site

CSinParallel.org -> Workshops -> WMR Workshop

See also workshop handout

Introductory comments

- Role of undergraduate researchers: There would be no workshop without them!
- Thanks to Amazon Web Services for providing credits to host our WMR instance

- Disclaimer: We are not proposing map-reduce as the only approach to introducing parallelism, concurrency
- A value: Honor thy neighbor's curricular approach

Goals

- Introduce map-reduce computing, using the *WebMapReduce (WMR)* simplified interface to Hadoop
  - Why use map-reduce in the curriculum?
- Hands-on exercises with WMR for foundation courses

Goals

Part 1 – Introduction

- Map-reduce computing, and the *WebMapReduce (WMR)* simplified interface to Hadoop
- Hands-on exercises with WMR for foundation courses

Part 2 – Teaching with WMR

- Why use map-reduce in the curriculum?
- Use of WMR for intermediate and advanced courses
- Hands-on exercises for more advanced use

Part 3 (optional) – What's under the hood?

Sneak Preview: Materials available

(In case you already know your map-reduce…)

- CSinParallel module: *Map-reduce Computing for Introductory Students using WebMapReduce*
  - See `csinparallel.org`

Part 1: Introduction to Map-Reduce Computing and WMR

Introduction to Map-Reduce Computing

History

- The computational model of using map and reduce operations was developed decades ago, for LISP
- Google developed MapReduce system for search engine, published (Dean and Ghemawat, 2004)
- Yahoo! created Hadoop, an open-source implementation (under Apache); Java mappers and reducers

Map-Reduce: The 2-minute overview

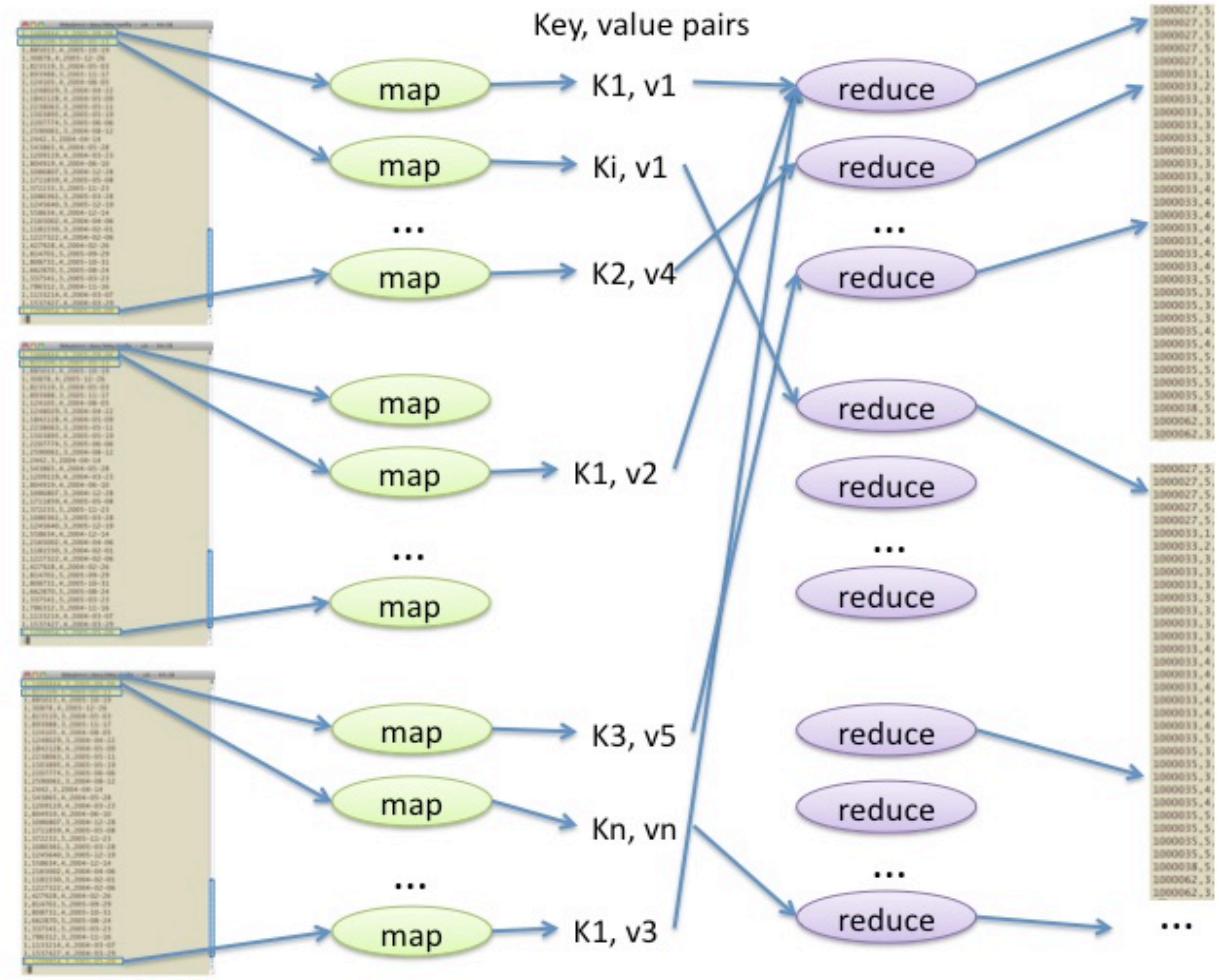
What if you wanted to
count the frequencies of all words
in 1,000,000 books?

Map-Reduce: The 2-minute overview

What if you wanted to
count the frequencies of all words
in 1,000,000 books?

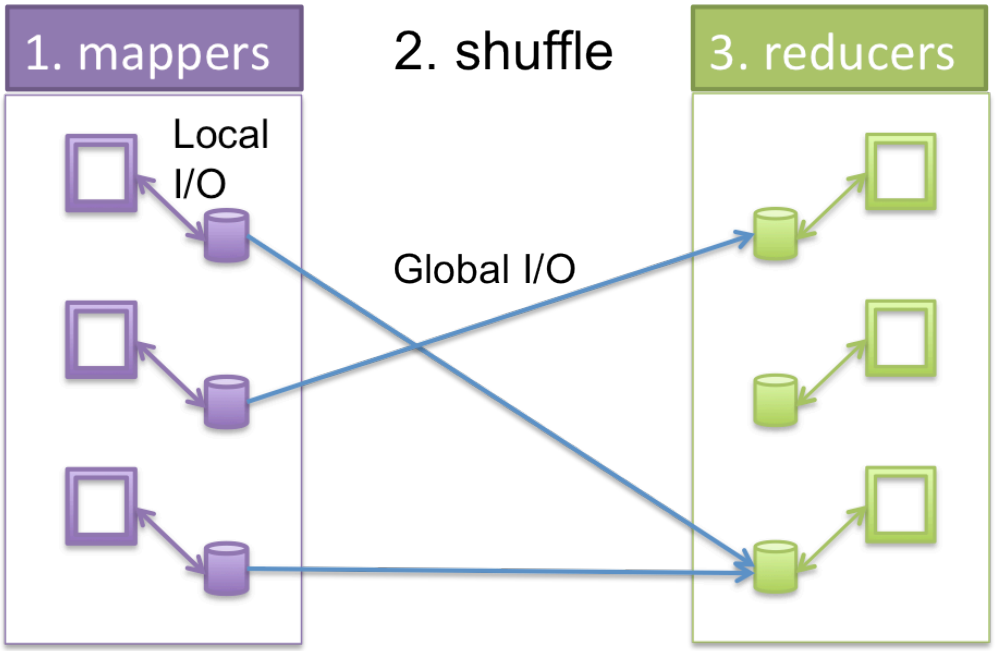
1. Break up the lines of text:
   generate one *labelled piece* per word
   - Use that word as label; value 1 for each piece
2. Group the pieces according to label (word)
3. Add up the 1's in each group

Map-Reduce Concept



The map-reduce computational model

- Map-reduce is a two-stage process with a "shuffle twist" between the stages.



- Stages are controlled by functions: mapper() , reducer()

The map-reduce computational model

- mapper() function:
  - Argument is one line of input from a file
  - Produces (key, value) pairs

- Example: word-count mapper()

"the cat in the hat"

--> [mapper for this line]

("the", "1"), ("cat", "1"), ("in", "1"),
("the", "1"), ("hat", "1")

The map-reduce computational model

- Shuffle stage:
  - group all mappers' (key, value) pairs together that have the same key, and feed each group to its own call of reduce()
  - Input: all (key,value) pairs from all mappers
  - Output: Those pairs rearranged, sent to calls of reduce() according to key

- Note: Shuffle also sorts (optimization)

The map-reduce computational model

- reducer() function:
  - Receives all key-value pairs for one key
  - Produces an aggregate result

- Example: word-count reducer()

("the", "1"), ("the", "1")
    -->
("the", "2")

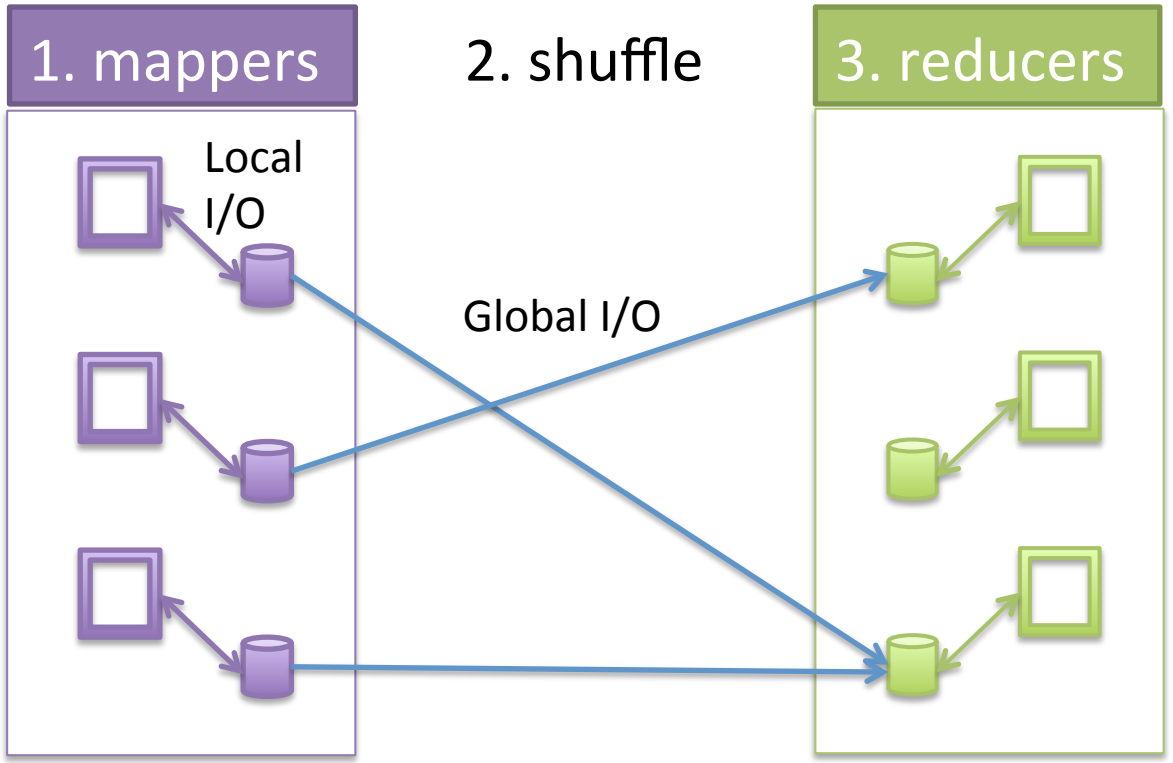
[reducer for "the"]

The map-reduce computational model

- In map-reduce, a programmer codes only two functions (plus config information)
  - A model for future parallel-programming frameworks
- Underlying map-reduce system reuses code for
  - Partitioning the data into chunks and lines,
  - Runs mappers/reducers where the chunks are *local*
  - Moving data between mappers and reducers
  - Auto-recovering from any crashes that may occur
  - …
- Optimized, Distributed, Fault-tolerant, Scalable

The map-reduce computational model

- Optimized, Distributed, Fault-tolerant, Scalable



Demo of WMR

cumulus.cs.stolaf.edu/wmr

Intro module

Materials available

- CSinParallel module: *Map-reduce Computing for Introductory Students using WebMapReduce*
  - See `csinparallel.org`

Overview of suggested exercises

Available on the csinparallel.org site

- Run word count (provided), with small and large data
- Modify, run variations on word count: strip punctuation; case insensitive; etc.

- Alternative exercises

Additional exercises

Beyond your first simple exercises, consider exploring the following:

- Various data sets
  - ***Note:*** *Please avoid large Gutenberg "groups" for this workshop*
- Extended set of exercises for CS1 (text analysis)

Hands-on exploration of WMR

Part 2: Teaching with WMR

Why map-reduce?

Why WMR?

Teaching WMR in CS1; in other courses

Why teach map-reduce?

Why map-reduce for teaching notions of parallelism/concurrency?

– Concepts:
  - data parallelism;
  - task parallelism;
  - locality;
  - effects of scale;
  - example effective parallel programming model;
  - distributed data with redundancy for fault tolerance; ...

Why map-reduce for teaching notions of parallelism/concurrency?

- Real-World: Hadoop widely used
- Exciting: the appeal of Google, Facebook, etc.
- Useful: for appropriate applications
- Powerful: scalability to large clusters, large data

Why WMR?

- Introduce concepts of parallelism
- Low bar for entry, feasible for CS1 (and beyond)
- Capture the imaginations of students

- Supports rapid introduction of concepts of parallelism for every CS student
  - Intro module designed for 1-3 days of class

WebMapReduce (WMR)

- Simplified web interface for Hadoop computations
- Goals:
  - Strategically Simple
    suitable for CS1, but not a toy
  - Configurable
    write mappers/reducers in any language
  - Accessible web application
  - Multi-platform, front-end and back-end

WMR Features (Briefly)

- Testing interface
  - Error feedback
  - Bypasses Hadoop -- small data only!
- Students enter the following information:
  - **choice of language**
  - data to process
  - definition of mapper in that language
  - definition of reducer in that language

WMR system information

– Languages currently supported:
  Java, C++, Python, Scheme, C, C#, Javascript
  - R coming soon

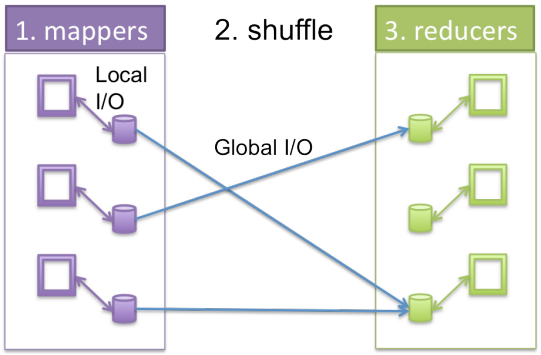
– Back ends to date:
  local cluster, Amazon EC2 cloud images
  - Version limits and more back ends coming soon

More details about the system in (optional) Part 3 of the workshop

Teaching map-reduce with WMR in the introductory sequence

Kinesthetic student activity

- Visualizations of map-reduce computations are enough for some students, but not all



- An **in-class activity** to act out the map/shuffle/reduce process helps others
- Also helpful: images of clusters; sequential versions; context of well-known web services

WMR in advanced courses

Example: PDC elective

- CS1 module
- Map-reduce programming techniques
  - Features of WMR
  - *Context forwarding*
  - *Structured values; structured keys*
  - *Multi-case mappers; multi-case reducers*
  - *Broadcasting data values*

Examples for Text Processing Techniques

- Combining data within a mapper
  - Mapper: Tally counts of words before sending to reducer
- Computational linguistics:
  - words that are co-located
    - Find and count pairs

      Example In: the cat in the cat hat

      Emits:

      1 cat|in
      1 in|the
      1 cat|hat
      2 the|cat
    - Use combining procedure to find 'stripes'

      Example In: the cat and the dog fought over the dog bone

      Emits: (the, {cat:1, dog:2}

Thanks to:

Data Intensive Text Processing, by Jimmy Lin and Chris Dyer

Application ideas

- Examples in the introductory module
- Big data sets people care about

- Especially for unstructured data
- Convenient for certain kinds of **projects**
  - E.g., most common medical terminology

WMR Hands-on, continued

Module exercises

Extended exercise set

Data sets available

/shared/MovieLens2/movieRatings

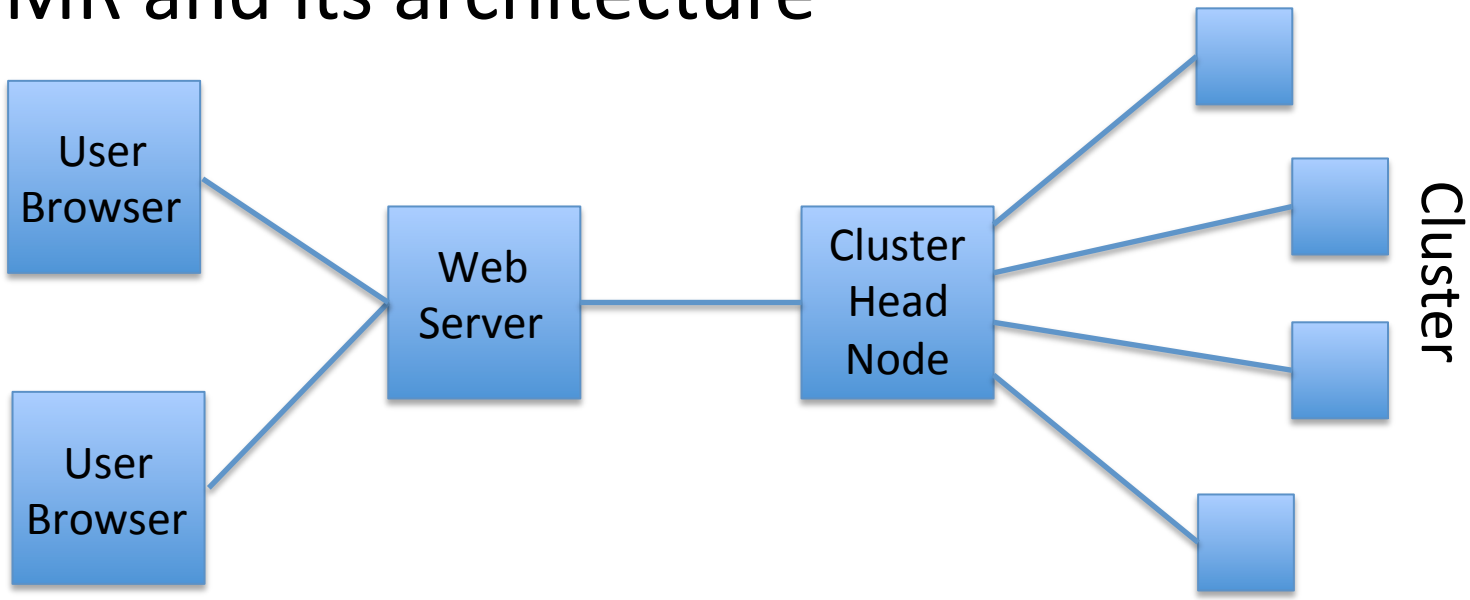
/shared/gutenberg/WarAndPeace.txt

/shared/gutenberg/CompleteShakespeare.txt

Part 3: What's under the hood

About WMR

- WMR and its architecture



- Obtaining and installing WMR
  - WebMapReduce.sf.com

Basic Hadoop components

- Internals:
  - Job management (per cluster)
  - Task management (per computation node)
- Some components visible to the user:
  - Hadoop API – Java, or arbitrary executables ("Streaming")
  - Hadoop Distributed File System (HDFS)
  - Support tools, including `hadoop` command
  - Limited job monitoring…

Goals

- Introduce map-reduce computing, using the *WebMapReduce (WMR)* simplified interface to Hadoop
  - Why use map-reduce in the curriculum?
- Hands-on exercises with WMR for foundation courses
- Use of WMR for intermediate and advanced courses
  - What's under the hood with WMR
  - A peek at Hadoop…
- Hands-on exercises for more advanced use

WMR in advanced courses

Inverting

"Chapter 1: Call me Ishmael. Some …"
"Chapter 2: I stuffed a shirt or two ..."
"Chapter 3: Entering that gable-ended ..."

--> [mapper]

("call", "1"), ("me", "1"), ..., ("i", "2"),
("stuffed", "2"), ..., ("entering", "3"), ...

--> [reducer]

"a" "1,1,1,1,...,2,2,2,..."
"aback" "3,7,7,8,..."
...

When is map-reduce appropriate?

- Massive, unstructured or irregularly structured "big data" (*Terascale* and upward)
  - Raw text
  - Web pages
  - XML
  - Unstructured streams of data
- Other approaches may fit structured "big data"
  - Scalable databases
  - Large-scale statistical approaches

Using Hadoop directly (Java)

WordCount.java example

Direct Hadoop Examples

- Word count
  - Java

Quick questions/comments so far?

Hands-on

Overview of suggested exercises

- Computations with MovieLens2 data; multiple map-reduce cycles
- Traffic data analysis
- Network analysis using Flixter data
- The Million Song dataset

Discussion

Evaluations!

Links at:

CSinParallel.org -> Workshops -> WMR Workshop

(end of the page)

Some considerations with Hadoop

- Numbers of mappers and reducers
- DFS
- Fault tolerance
- I/O formats

- Note: we have further slides with additional information about these aspects, for you to look at on your own.

Direct Hadoop exercise setup

- Edit your own files, locally
- scp to cluster's admin node (on cloud)
- ssh to compile, launch job
- Percentage progress output is provided
- Multiple cycle support via DFS
- (Cleanup)

Additional Details about Hadoop

The hadoop project documentation

- http://hadoop.apache.org/common/docs/current/index.html

How many mappers?

- The Hadoop Map/Reduce framework spawns one mapper task for each InputSplit generated by the InputFormat for the job.

- The number of mappers is usually driven by the total size of the inputs, that is, the total number of blocks of the input files.

How many reducers?

- The number of reducers for the job is set by the user via JobConf.setNumReduceTasks(int)

- The size of your eventual output may dictate how many reducers you choose.

HDFS

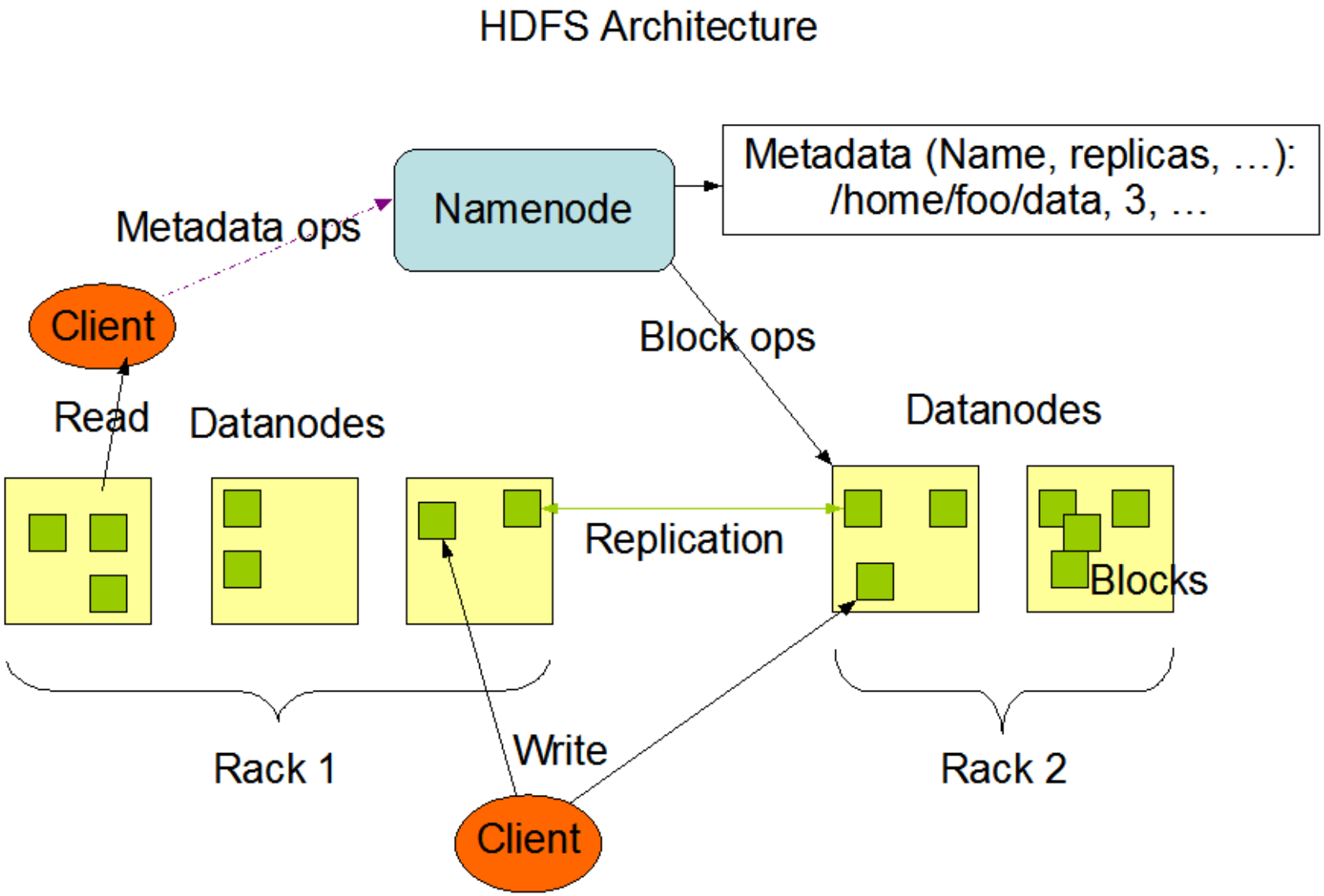
- Fault-tolerant distributed file system modeled after the Google File System
  - we've had students read the original GFS paper in an advanced course
- http://hadoop.apache.org/hdfs/docs/current/index.html
- Note the section about the file system commands you can run from the command line:

  hadoop fs -ls

  Hadoop fs -get or -put

HDFS Assumptions and Goals

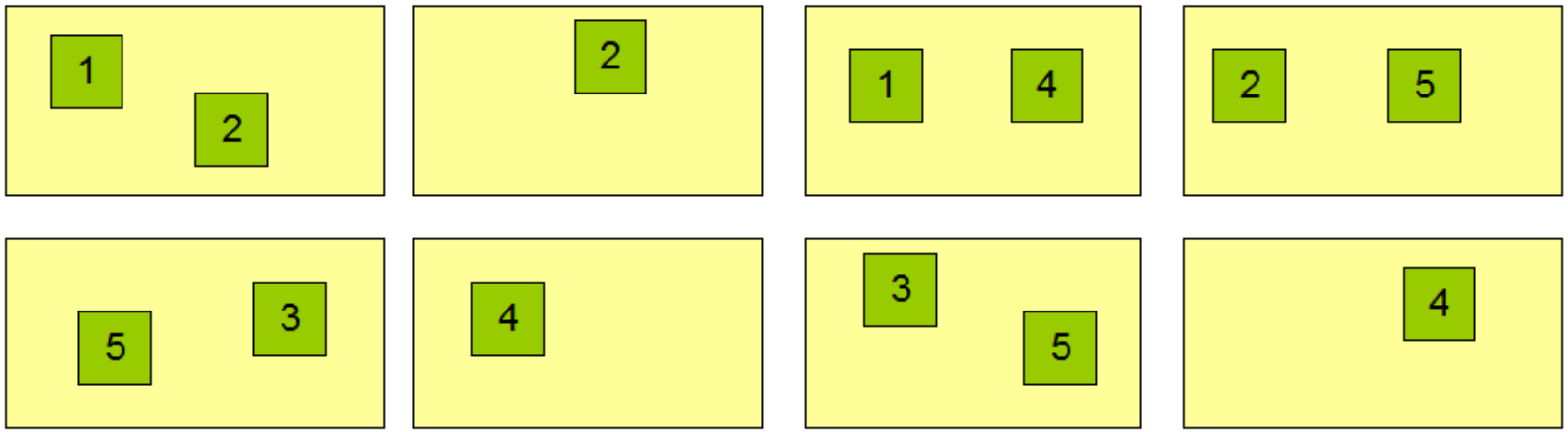
- Hardware failure
  - Hardware failure is the norm rather than the exception.
- Streaming data access
  - Applications that run on HDFS need streaming access to their data sets. They are not general purpose applications that typically run on general purpose file systems.
- Large Data Sets
- Simple coherency model
  - Read many, write once
- Moving computations is simpler than moving data
- Portability across various hardware/software


HDFS Architecture
Metadata ops
Namenode
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

Block Replication

Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …

Datanodes



Input/Output formats

- Input into mappers are interpreted using classes implementing the interface InputFormat, and out put from reducers are implemented using classes implementing the interface OutputFormat.
- In WMR, the mapper input and reducer output is performed with key-value pairs. This corresponds to using the classes KeyValueTextInputFormat and TextOutputFormat.
- In direct Hadoop, the default input format is TextInputFormat, in which values are lines of the file and keys are positions within that file.

Some further features of Hadoop

- Combiner, an optimization: perform some "reduction" during the map phase, after mapper() and before shuffle
- Sorting control
  - Note: hard to sort on secondary key
- Three programming interfaces: Java; pipes (C++); streaming (executables)

Page rank algorithm ideas

- Original data: one web page per line
  - mapper produces ("dest", "1/k Pn") for each link in page Pn
    where k links appear within that page Pn

    reducer produces ("dest", "weight_0 P1 P2 P2 P3 P4 ...")
    where weight is sum of the weights from key value pairs emitted by P1, P2, ...
  - Subsequent mappers and reducers produce refined weights that take into account deeper chains of pages pointing to pages
  - Final reducer delivers ("dest", "weight_k") [drop Pns]