Take An Internal Look at Hadoop

Hairong Kuang
Grid Team, Yahoo! Inc
hairong@yahoo-inc.com

What's Hadoop

- **Framework for running applications on large clusters of commodity hardware**
  - Scale: petabytes of data on thousands of nodes
- **Include**
  - Storage: HDFS
  - Processing: MapReduce
    - Support the Map/Reduce programming model
- **Requirements**
  - Economy: use cluster of comodity computers
  - Easy to use
    - Users: no need to deal with the complexity of distributed computing
  - Reliable: can handle node failures automatically

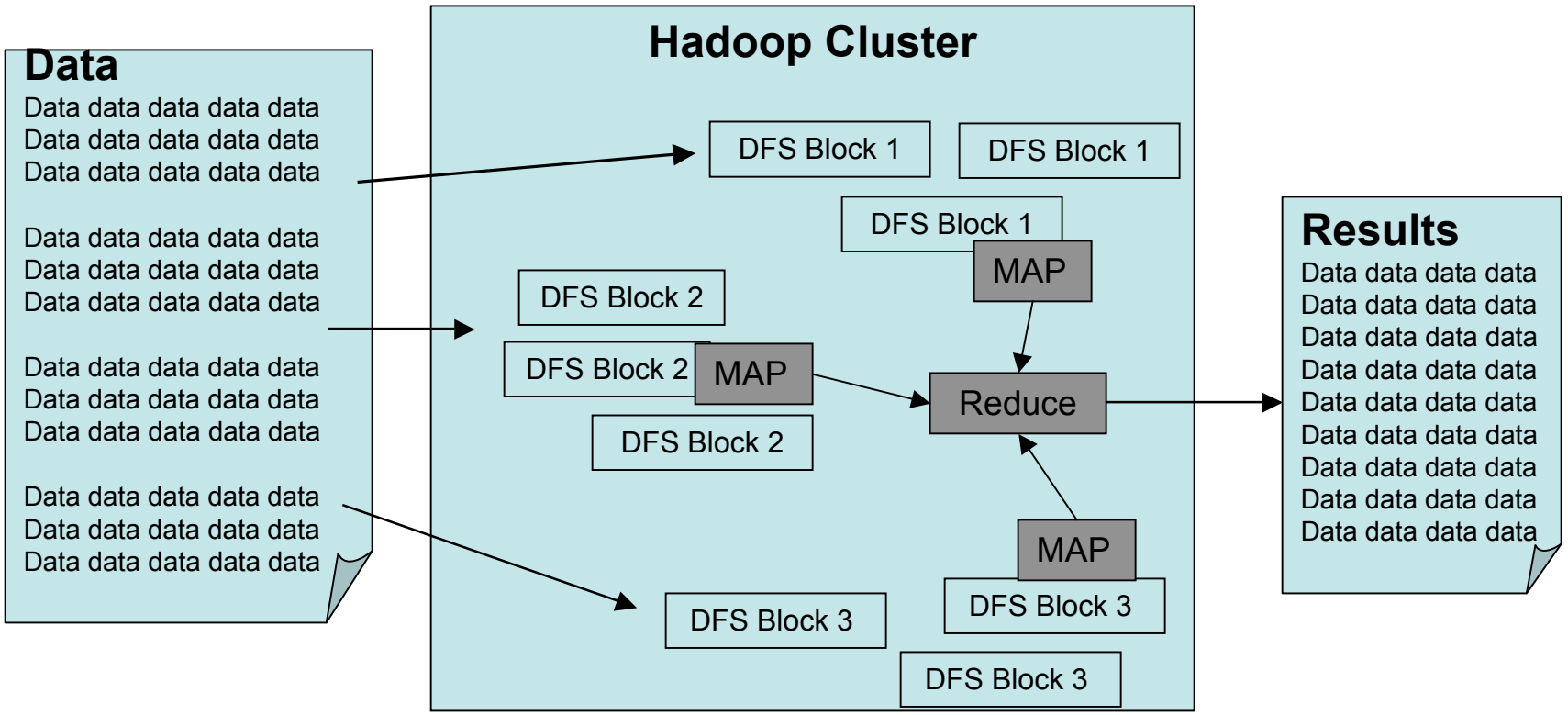
Open source Apache project

- **Implemented in Java**
- **Apache Top Level Project**
  - http://hadoop.apache.org/core/
  - Core (15 Committers)
    - HDFS
    - MapReduce
- **Community of contributors is growing**
  - Though mostly Yahoo for HDFS and MapReduce
  - You can contribute too!

Hadoop Characteristics

- **Commodity HW**
  - Add inexpensive servers
  - Storage servers and their disks are ***not*** assumed to be highly reliable and available
- **Use replication across servers to deal with unreliable storage/servers**
- **Metadata-data separation - simple design**
  - Namenode maintains metadata
  - Datanodes manage storage
- **Slightly Restricted file semantics**
  - Focus is mostly sequential access
  - Single writers
  - No file locking features
- **Support for moving computation close to data**
  - Servers have 2 purposes: data storage and computation
  - Single ‘storage + compute’ cluster vs. Separate clusters

Hadoop Architecture



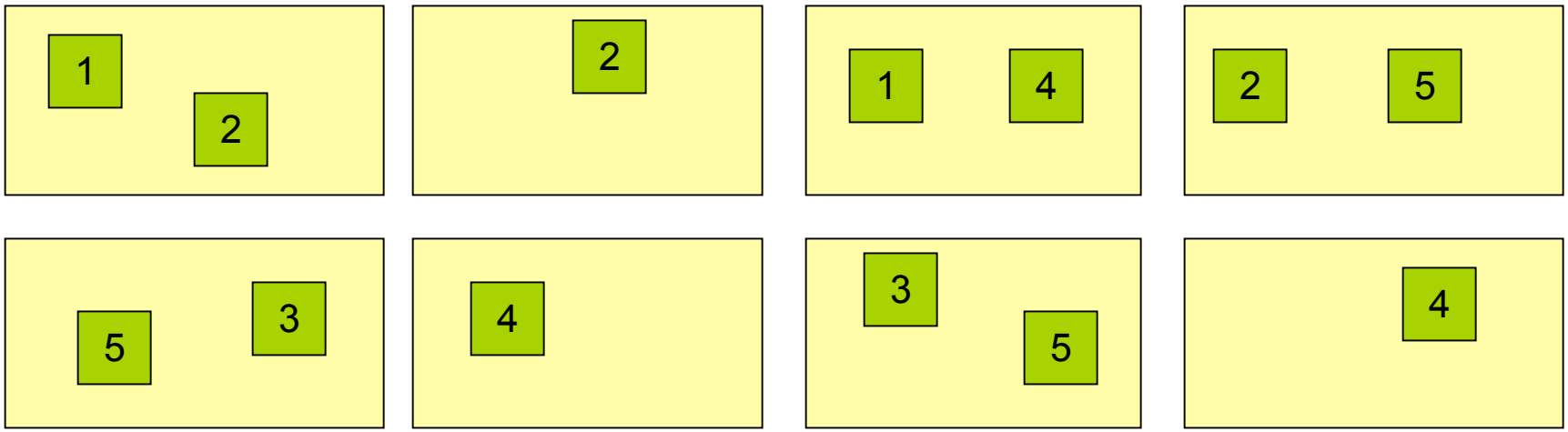
HDFS Data Model

- 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
- Filesystem keeps checksums of data for corruption detection and recovery
- HDFS exposes block placement so that computation can be migrated to data

HDFS Data Model

NameNode(Filename, replicationFactor, block-ids, …)
/users/user1/data/part-0, r:2, {1,3}, …
/users/user1/data/part-1, r:3, {2,4,5}, …

Datanodes

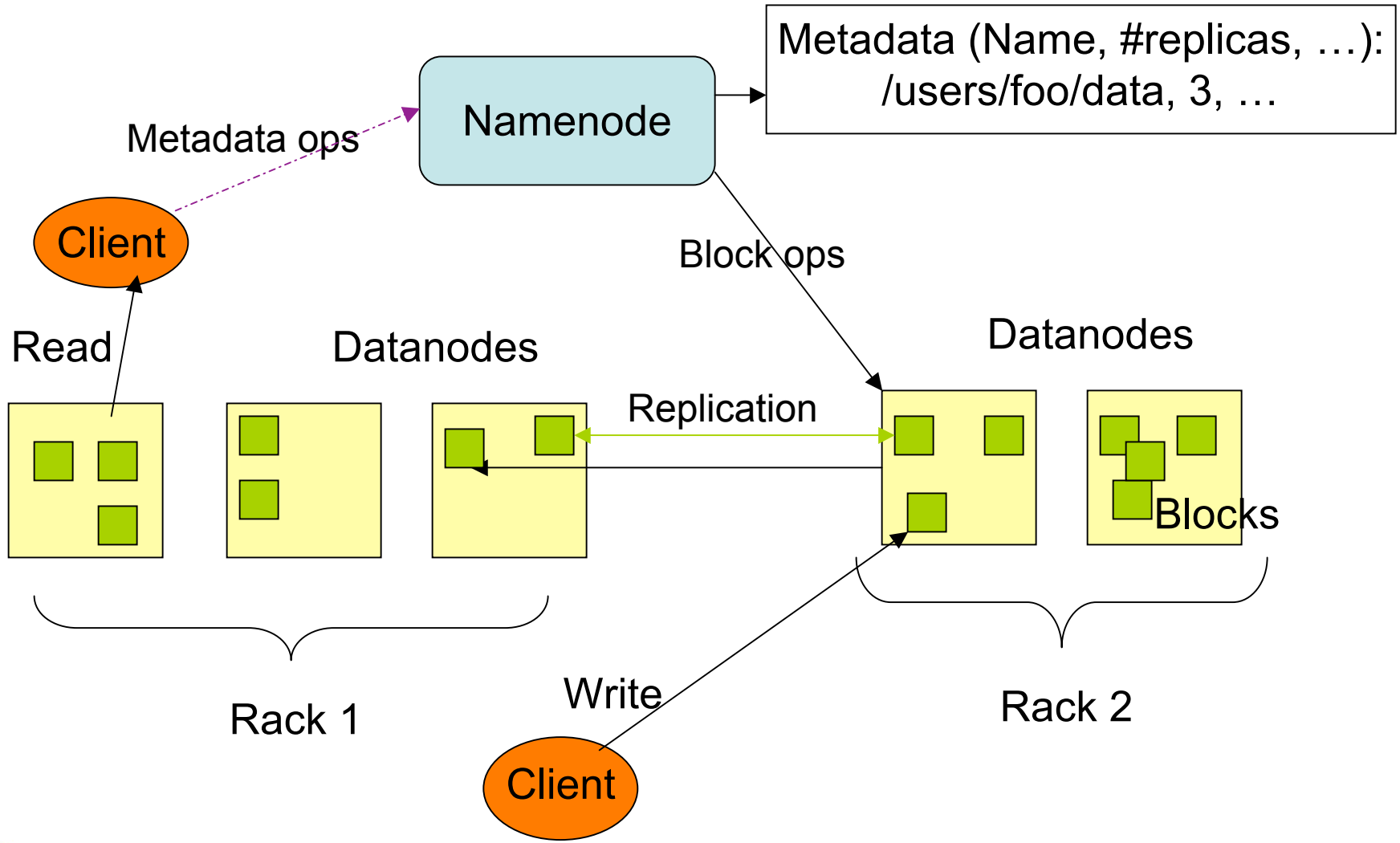


hadoop

HDFS Architecture

- **Master-Slave architecture**
- **DFS Master "Namenode"**
  - Manages the filesystem namespace
  - Maintain file name to list blocks + location mapping
  - Manages block allocation/replication
  - Checkpoints namespace and journals namespace changes for reliability
  - Control access to namespace
- **DFS Slaves "Datanodes" handle block storage**
  - Stores blocks using the underlying OS's files
  - Clients access the blocks directly from datanodes
  - Periodically sends block reports to Namenode
  - Periodically check block integrity

HDFS Architecture



Block Placement And Replication

- **A file's replication factor can be set per file (default 3)**
- **Block placement is rack aware**
  - Guarantee placement on two racks
  - 1st replica is on local node, 2rd/3rd replicas are on remote rack
  - Avoid hot spots: balance I/O traffic
- **Writes are pipelined to block replicas**
  - Minimize bandwidth usage
  - Overlapping disk writes and network writes
- **Reads are from the nearest replica**
- **Block under-replication & over-replication is detected by Namenode**
- **Balancer application rebalances blocks to balance DN utilization**

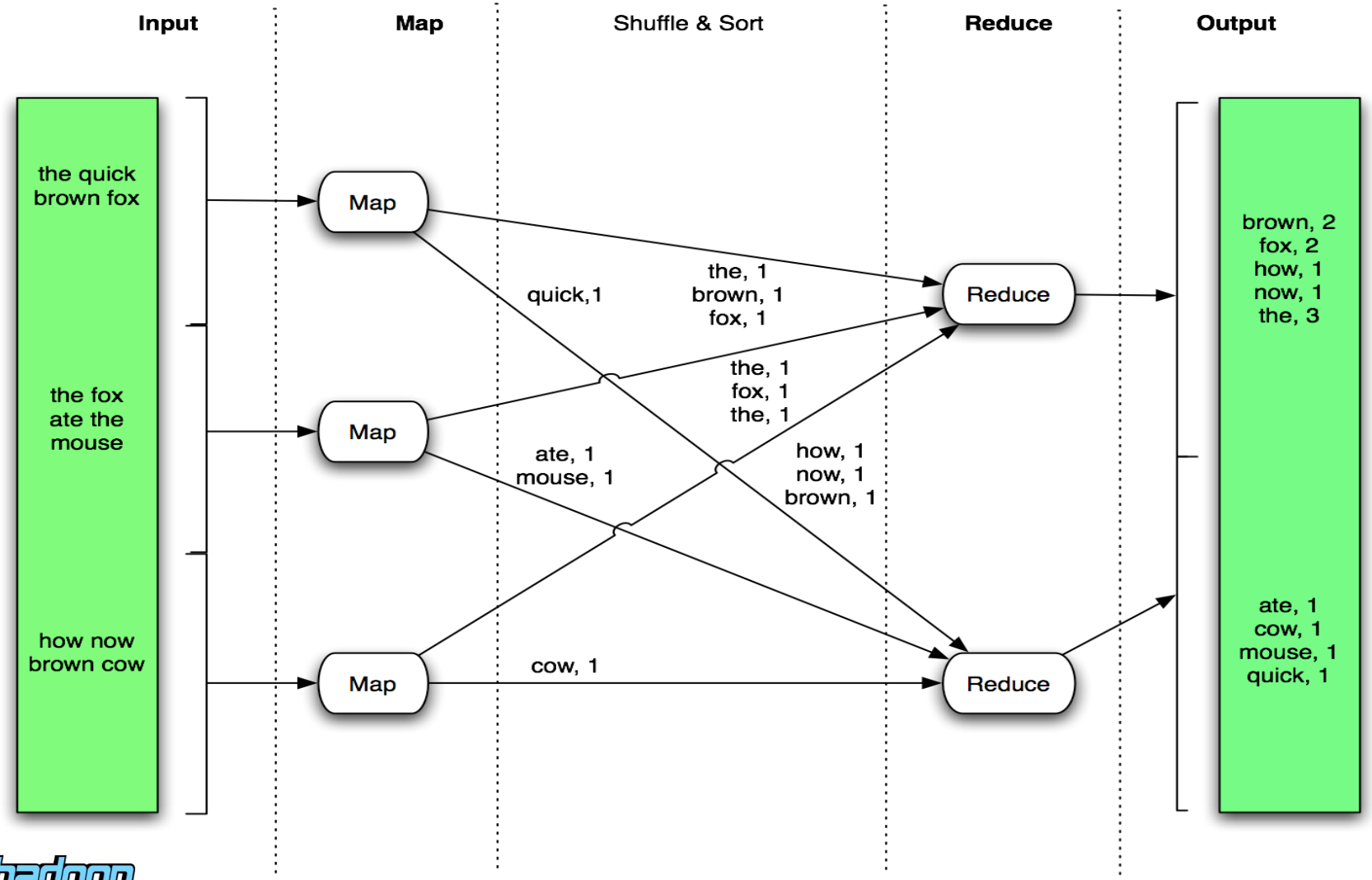
HDFS Future Work: Scalability

- **Scale cluster size**
- **Scale number of clients**
- **Scale namespace size (total number of files, amount of data)**
- **Possible solutions**
  - Multiple namenodes
    - Read-only secondary namenode
    - Separate cluster management and namespace management
    - Dynamic Partition namespace
    - Mounting

Map/Reduce

- **Map/Reduce is a programming model for efficient distributed computing**
- **It works like a Unix pipeline:**
  - cat input | grep | sort | uniq -c | cat > output
  - **Input** | **Map** | Shuffle & Sort | **Reduce** | **Output**
- **A simple model but good for a lot of applications**
  - Log processing
  - Web index building

Word Count Dataflow



Word Count Example

- **Mapper**
  - Input: value: lines of text of input
  - Output: key: word, value: 1
- **Reducer**
  - Input: key: word, value: set of counts
  - Output: key: word, value: sum
- **Launching program**
  - Defines the job
  - Submits job to cluster

Map/Reduce features

- **Fine grained Map and Reduce tasks**
  - Improved load balancing
  - Faster recovery from failed tasks
- **Automatic re-execution on failure**
  - In a large cluster, some nodes are always slow or flaky
  - Framework re-executes failed tasks
- **Locality optimizations**
  - With large data, bandwidth to data is a problem
  - Map-Reduce + HDFS is a very effective solution
  - Map-Reduce queries HDFS for locations of input data
  - Map tasks are scheduled close to the inputs when possible

Documentation

- **Hadoop Wiki**
  - Introduction
    - http://hadoop.apache.org/core/
  - Getting Started
    - http://wiki.apache.org/hadoop/GettingStartedWithHadoop
  - Map/Reduce Overview
    - http://wiki.apache.org/hadoop/HadoopMapReduce
  - DFS
    - http://hadoop.apache.org/core/docs/current/hdfs_design.html
- **Javadoc**
  - http://hadoop.apache.org/core/docs/current/api/index.html

Questions?

Thank you!