Cloud Computing and Hadoop/MapReduce

Objectives

- Discuss the concept of Cloud Computing
- Explain the use of Distributed File Systems, in particular, Hadoop
- Explain the MapReduce framework for large scale, distributed computing

Cloud Computing

What is Cloud Computing?

NIST Definition:

A model for enabling

- *ubiquitous, convenient, on-demand network access*
- *to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services)*
- *that can be rapidly provisioned and released with minimal management effort or service provider interaction.*

Cloud Characteristics

- On-demand self-service (dynamic provisioning)
- Ubiquitous network access
- Location independent resource pooling
- Rapid elasticity (scalability)
- Pay per use (utility-like service)

Delivery Models

Software as a Service (SaaS)

- Use provider's applications over a network
- SalesForce.com

Platform as a Service (PaaS)

- Deploy customer-created applications to a cloud
- The networks, servers, storage and other services are provided
- Google App Engine

Infrastructure as a Service (IaaS)

- Rent processing, storage, network capacity, and other fundamental computing resources
- Amazon EC2, S3

Deployment Models

Private Cloud

- The cloud infrastructure has been deployed, and is maintained and operated for a specific organization.
- The operation may be in-house or with a third party on the premises.

Community Cloud

- The cloud infrastructure is shared among a number of organizations with similar interests and requirements.
- This may help limit the capital expenditure costs for its establishment as the costs are shared among the organizations.
- The operation may be in-house or with a third party on the premises.

Deployment Models (cont.)

Public Cloud

- The cloud infrastructure is available to the public on a commercial basis by a cloud service provider.
- This enables a consumer to develop and deploy a service in the cloud with very little financial outlay compared to the capital expenditure requirements normally associated with other deployment options.

Hybrid Cloud

- The cloud infrastructure consists of a number of clouds of any type, but the clouds have the ability through their interfaces to allow data and/or applications to be moved from one cloud to another.
- This can be a combination of private and public clouds that support the requirement to retain some data in an organization, and also the need to offer services in the cloud.

Why Cloud Computing?

Without a Cloud:

- Capital investment needed
  - Heavy fixed costs
- Redundant expenditures
- High energy cost, low CPU utilization
- Dealing with unreliable hardware
  - Maintenance / replacement costs
- High-levels of overcapacity (Technology and Labor)
  - Need to provide enough computing resources for most demanding applications at all times

Why Cloud Computing? (cont.)

Solution: supply computing resources like a utility

- Efficient: economy of scale
- All problems are outsourced to the cloud provider
- The provider will share the overhead costs among many customers, amortizing the costs
- You only pay for:
  - the amortized overhead
  - Your real CPU / Storage / Bandwidth usage

Cloud Computing Benefits

Cost Savings

- Companies can reduce their capital expenditures and use operational expenditures for increasing their computing capabilities. This is a lower barrier to entry and also requires fewer in-house IT resources to provide system support.

Scalability/Flexibility

- Companies can start with a small deployment and grow to a large deployment fairly rapidly, and then scale back if necessary. Also, the flexibility of cloud computing allows companies to use extra resources at peak times, enabling them to satisfy consumer demands.

Reliability

- Services using multiple redundant sites can support business continuity and disaster recovery.

Maintenance

- Cloud service providers do the system maintenance, and access is through APIs that do not require application installations onto PCs, thus further reducing maintenance requirements.

Mobile Accessible

- Mobile workers have increased productivity due to systems accessible in an infrastructure available from anywhere.

Hadoop

Distributed Computing

Data intensive applications:

- Data collection too large to transmit economically over Internet --- Petabyte data collections
- Computation produces small data output containing a high density of information

Solution: ***Hadoop/MapReduce***

- Compute "near" the data
- Aggregate the results and send back

What is Hadoop?

Hadoop is a software framework

Makes it easy to process vast amounts of data (>**TBs**), in-parallel on large clusters (**1000s**)

Hadoop includes:

- Hadoop File System (HDFS) - distributes data
- Map/Reduce - distributes application

Hadoop Software

Hadoop is open source software from Apache

Written in Java, but can interface with other languages

Runs on

- Linux, Mac OS/X, Windows, and Solaris
- Commodity hardware

Distributed File Systems

Hadoop is different from traditional ***distributed file systems (DFSs)***

Traditional DFSs:

- distribute files over a network.
- provide performance, scalability, reliability, availability

Differences for Hadoop:

- Assumes component failures are the norm (large number of commodity machines).
- Files may be large (e.g., 100MB).
- High, sustained bandwidth is more important than low latency.

Hadoop File System (HDFS)

HDFS is designed to store large files

- Stores files as large blocks (64 to 128 MB)
- Each block stored on multiple servers
- Data is automatically re-replicated on need

Hadoop File System (HDFS)

HDFS provides automatic handling of hardware failures:

- An HDFS instance may consist of hundreds or thousands of server machines, each storing part of the file system's data.

- Huge number of components
  - Each component has a non-trivial probability of failure
  - Some component of HDFS is always non-functional.

- Detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

HDFS Architecture

HDFS has a master/slave configuration

Master is a single ***NameNode***

- it manages the file system namespace
- also regulates access to files by clients

In addition, there are multiple ***DataNodes***

- usually one per node in the cluster
- these manage storage attached to the nodes that they run on

HDFS exposes a file system namespace and allows user data to be stored in files.

HDFS Architecture

Internally, a file is split into one or more ***blocks***

These blocks are stored in a set of **DataNodes**.

The **NameNode** executes file system namespace operations, e.g.:

opening, closing, and renaming files and directories.

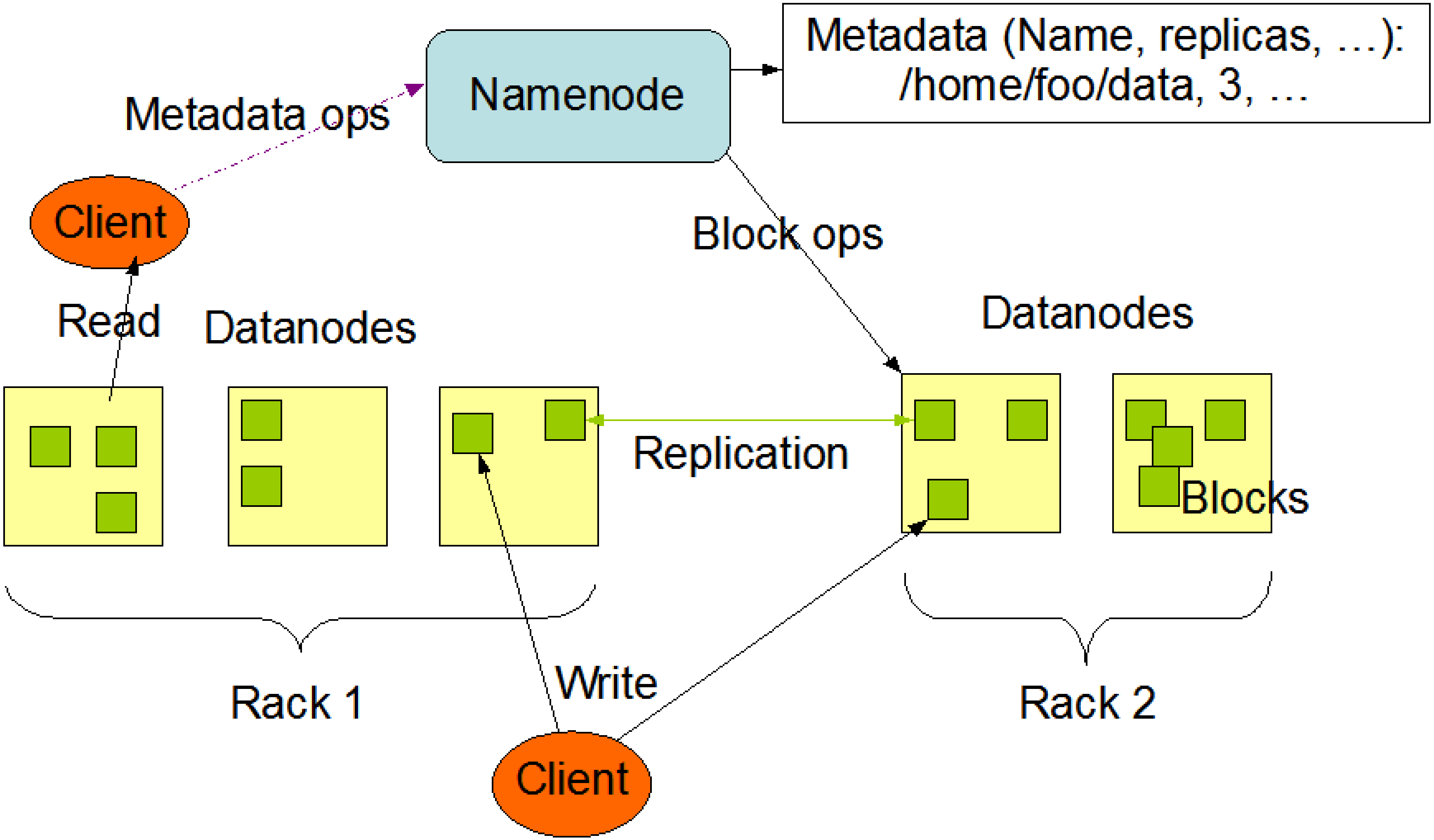
HDFS Architecture

NameNode also determines the mapping of blocks to DataNodes.

The DataNodes are responsible for serving read and write requests from the file system's clients.

The DataNodes also perform block creation, deletion, and replication upon instruction from the **NameNode**.

HDFS Architecture



HDFS Architecture

The **NameNode** and **DataNode** are pieces of software designed to run on commodity machines.

- typically running the Linux OS

HDFS is built using the *Java* language

- any machine that supports Java can run the NameNode or the DataNode software
- Using Java makes the framework highly portable to a wide range of machines

HDFS - Typical Deployment

A dedicated machine runs only the **NameNode** software.

Each of the other machines in the cluster runs one instance of the DataNode software.

The architecture does not preclude running multiple **DataNodes** on the same machine but in a real deployment that is rarely the case.

NameNode

The **NameNode** maintains the file system namespace.

- Any change to the file system namespace or its properties is recorded by the **NameNode**.

An application can specify the number of replicas of a file that should be maintained by HDFS.

- The number of copies of a file is called the ***replication factor*** of that file.
- This information is stored by the **NameNode**.
- Having more replicas increases system reliability

NameNode EditLog

The NameNode uses a transaction log called the ***EditLog*** to persistently record every change that occurs to file system metadata

- Creating a new file in HDFS causes the NameNode to insert a record into the EditLog indicating this

- Changing the replication factor of a file causes a new record to be inserted into the EditLog

NameNode EditLog

The NameNode uses a file in its local host OS file system to store the EditLog.

The entire file system namespace, including the mapping of blocks to files and file system properties, is stored in a file called the FsImage

The ***FsImage*** is stored as a file in the NameNode’s local file system too

DataNode

The DataNode stores HDFS data in files in its local file system.

It has no knowledge about HDFS files.

It stores each *block of HDFS data* in a separate file in its local file system.

DataNode

The DataNode does not create all files in the same directory.

Uses a *heuristic* to determine the optimal number of files per directory and creates subdirectories appropriately.

It is not optimal to create all local files in the same directory because the local file system might not be able to efficiently support a huge number of files in a single directory.

DataNode

DataNode start up:

- Scans through its local file system
- Generates a list of all HDFS data blocks that correspond to each of the local files
- Sends this report to the NameNode

– this is called the ***Blockreport***

Fault-Tolerance

Each DataNode sends a ***Heartbeat message*** to the NameNode periodically

A **network partition** can cause a subset of DataNodes to lose connectivity with the NameNode.

The NameNode detects this condition by the absence of a Heartbeat message.

The NameNode marks DataNodes without recent Heartbeats as dead and does not forward any new IO requests to them.

Fault-Tolerance

Any data that was registered to a dead DataNode is not available to HDFS any more.

DataNode death may cause the replication factor of some blocks to fall below their specified value.

Fault-Tolerance

The NameNode constantly tracks which blocks need to be replicated and **initiates replication whenever necessary**.

The necessity for re-replication may arise due to many reasons:

- A DataNode may become unavailable
- A replica may become corrupted
- A hard disk on a DataNode may fail
- The replication factor of a file may be increased

Accessing the File Space

Access to the file system can be done using:

- Java API
- Web interface
- Unix command tools

Examples using Unix commands:

Create a directory named /foodir	`bin/hadoop dfs -mkdir /foodir`
Remove a directory named /foodir	`bin/hadoop dfs -rmr /foodir`
View the contents of a file named /foodir/myfile.txt	`bin/hadoop dfs -cat /foodir/myfile.txt`

MapReduce

Map/Reduce Job

A ***Map/Reduce Job*** consists of files stored on the distributed file system (HDFS)

Data is split into independent datasets

Processed by ***map tasks*** (in parallel)

Outputs of maps are sorted and then input to the ***reduce tasks***

Map/Reduce Framework Responsibilities

The framework takes care of:

- Scheduling Tasks
- Monitoring Tasks
- Re-Executing Failed Tasks

Advantages:

- Schedule tasks where the data exists
- Easy to write programs, fast turn around

HDFS + Map/Reduce Configuration

Typically the compute nodes and the storage nodes are the same

- i.e., the Map/Reduce framework and the Hadoop Distributed File System are running on the same set of nodes.
- Allows the framework to effectively schedule tasks on the nodes where data is already present
- Results in very high aggregate bandwidth across the cluster

Map/Reduce Framework

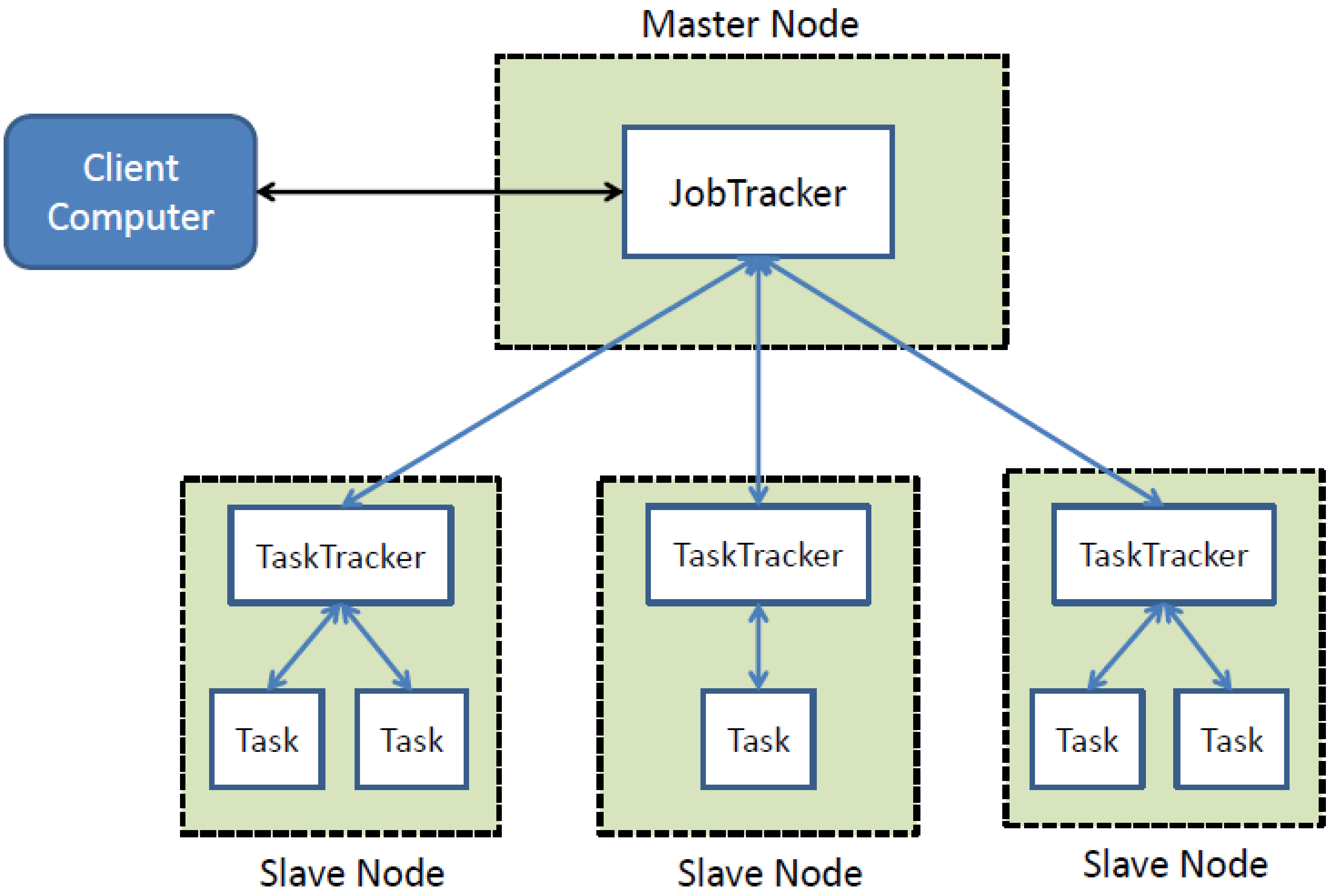
Consists of a single master ***JobTracker*** and one slave ***TaskTracker*** per cluster-node.

The master is responsible for:

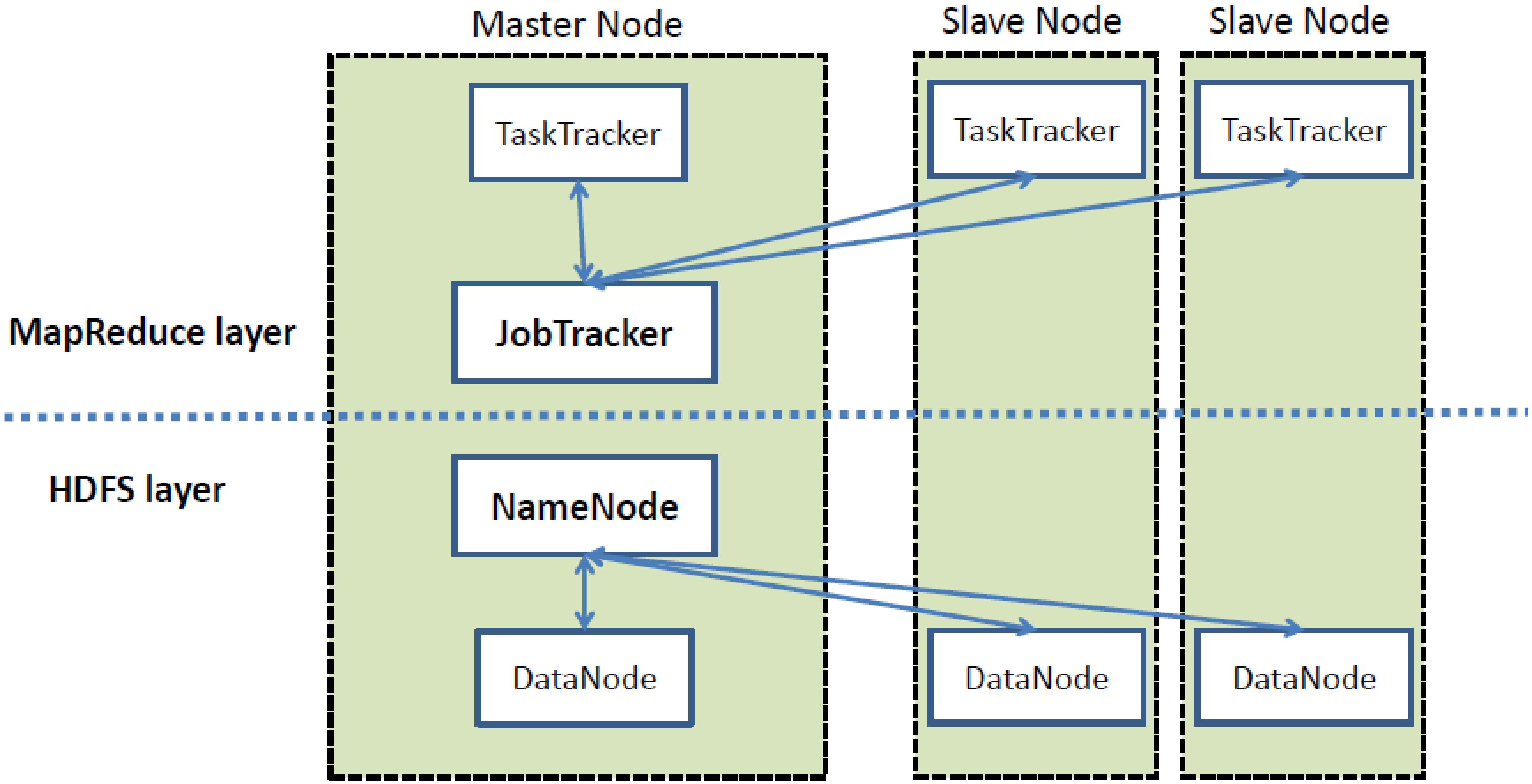
- Scheduling the jobs' component tasks on the slaves
- Monitoring tasks on the slaves
- Re-executing failed tasks

The slaves execute the tasks as directed by the master.

Map/Reduce Framework



Map/Reduce w/ HDFS



Map/Reduce Applications

A Hadoop application consists of a ***Job client*** that submits a job by passing its job configuration

A ***job configuration*** consists of:

- The input/output locations
- Specification of *map* and *reduce* functions

Hadoop Applications

The Hadoop *job client* then submits the job (**jar/executable** etc.) and **configuration** to the JobTracker

The JobTrack then assumes the responsibility of:

- distributing the software/configuration to the slaves
- scheduling tasks and monitoring them
- providing status and diagnostic information to the job-client.

Mapping and Reducing

MapReduce works using **`<key, value>`** pairs

The input to the job is a set of <key, value> pairs

The output produces a set of <key, value> pairs, possible of different types.

Map/Reduce Process

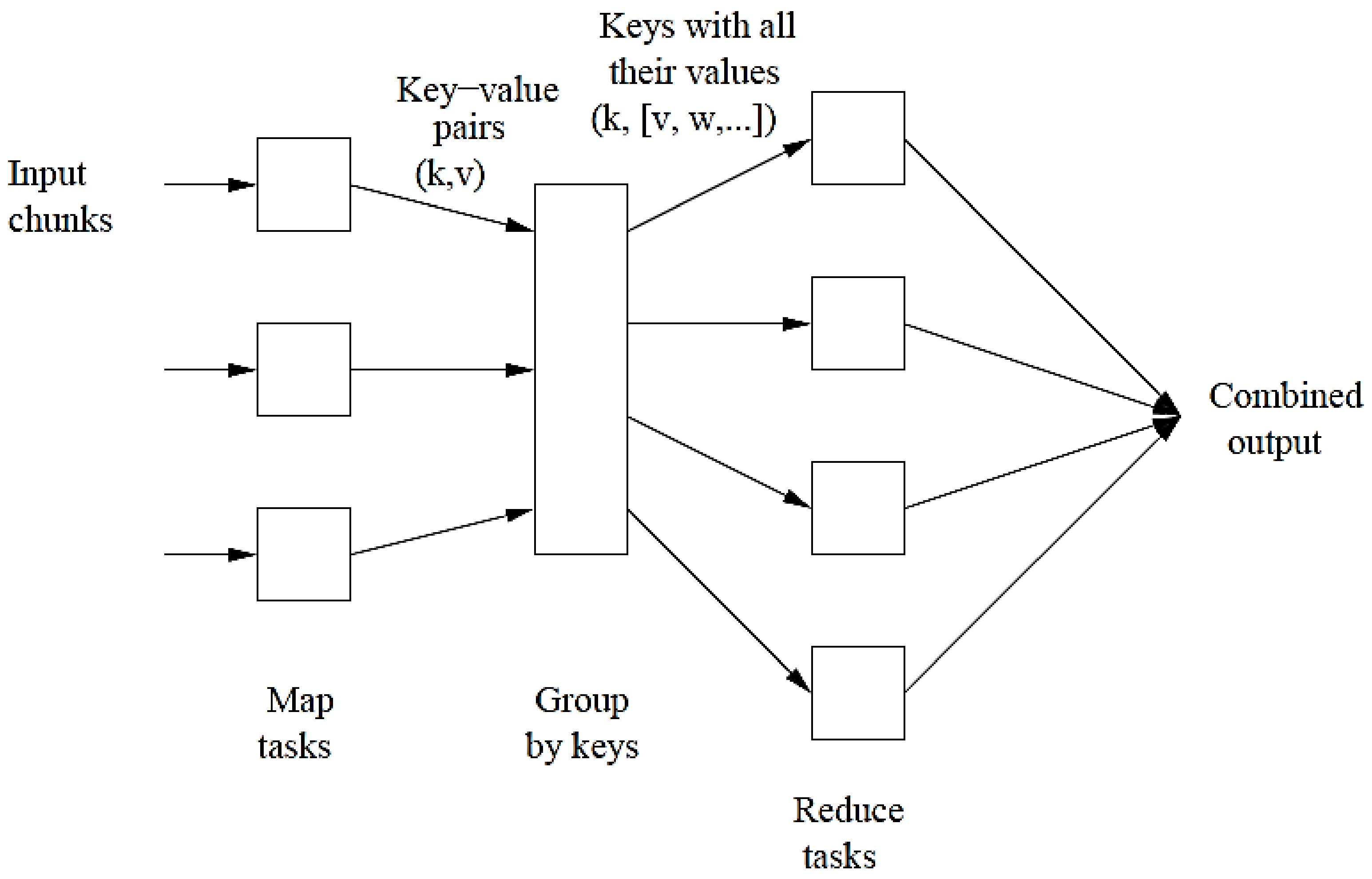
(input) <k_i, v_i>

-> **map** -> <k, v>

-> Group by k -> <k, [v_1, v_2, …]>

-> **reduce** -> <k_o, v_o> (output)

Map/Reduce Process



WordCount Example

Application: count the number of unique words in a set of documents

Input: `<lineKey, lineText>`

Mapper:

- Splits the line into tokens (i.e. words)
- Outputs < `<word> , 1>` for each word

Grouping:

- Takes the < `<word>, 1>` pairs and outputs < `<word>, [1,1,…]>` pairs for each word

Reducer:

- Takes the < `<word>, [1,1,…]>` pairs and outputs < `<word>, count>`

Example Mapper Outputs

Sample input:

File 1: “Hello World Bye World” (for mapper 1)
File 2: “Hello Hadoop Goodbye Hadoop” (for mapper 2)

First map emits:

```
< Hello, 1>
< World, 1>
< Bye, 1>
< World, 1>
```

Second map emits:

```
< Hello, 1>
< Hadoop, 1>
< Goodbye, 1>
< Hadoop, 1>
```

Example Reducer Outputs

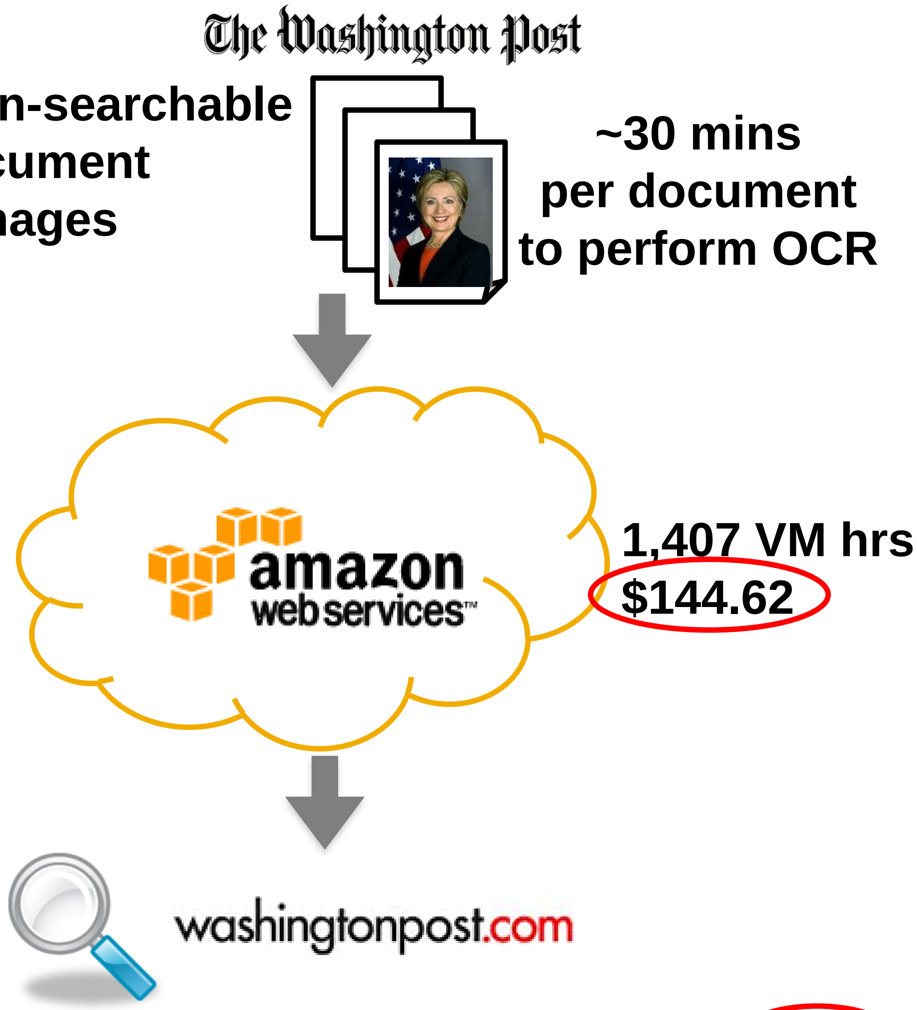
Reducer input:

```
< Hello, [1 1]>
< World, [1 1]>
< Bye, [1]>
< Hadoop, [1 1]>
< Goodbye, [1]>
```

Reducer output:

```
< Hello, 2>
< World, 2>
< Bye, 1>
< Hadoop, 2>
< Goodbye, 1>
```

Real-Life Success Story



17,481 non-searchable document images

~30 mins per document to perform OCR

1,407 VM hrs
$144.62

Search Tool Internal: 9 hrs, Web: 26 hrs

Summary

- Cloud computing is a cost-effective and efficient means of distributed computation
- Hadoop/MapReduce is a distributed computing solution used for very large data sets
- Hadoop provides a distributed file system (HDFS) and the MapReduce framework for computing