Cloud Computing
Applications and Paradigms

Session 7

Acknowledgement

The slides in this presentation are taken from the book
"Cloud Computing: Theory and Practice"

Contents

- Challenges for cloud computing.
- Existing cloud applications and new opportunities.
- Architectural styles for cloud applications.
- Workflows - coordination of multiple activities.
- Coordination based on a state machine model.
- The MapReduce programming model.
- A case study: the GrepTheWeb application.
- Clouds for science and engineering.
- High performance computing on a cloud.
- Legacy applications on a cloud.
- Social computing, digital content, and cloud computing.

Cloud applications

- Cloud computing is very attractive to the users:
  - Economic reasons.
    - low infrastructure investment.
    - low cost - customers are only billed for resources used.
  - Convenience and performance.
    - application developers enjoy the advantages of a just-in-time infrastructure; they are free to design an application without being concerned with the system where the application will run.
    - the execution time of compute-intensive and data-intensive applications can, potentially, be reduced through parallelization. If an application can partition the workload in *n* segments and spawn *n* instances of itself, then the execution time could be reduced by a factor close to *n*.
- Cloud computing is also beneficial for the providers of computing cycles - it typically leads to a higher level of resource utilization.

Cloud applications (cont'd)

- Ideal applications for cloud computing:
  - Web services.
  - Database services.
  - Transaction-based service. The resource requirements of transaction-oriented services benefit from an elastic environment where resources are available when needed and where one pays only for the resources it consumes.
- Applications unlikely to perform well on a cloud:
  - Applications with a complex workflow and multiple dependencies, as is often the case in high-performance computing.
  - Applications which require intensive communication among concurrent instances.
  - When the workload cannot be arbitrarily partitioned.

Challenges for cloud application development

- Performance isolation - nearly impossible to reach in a real system, especially when the system is heavily loaded.
- Reliability - major concern; server failures expected when a large number of servers cooperate for the computations.
- Cloud infrastructure exhibits latency and bandwidth fluctuations which affect the application performance.
- Performance considerations limit the amount of *data logging*; the ability to identify the source of unexpected results and errors is helped by frequent logging.

Existing and new application opportunities

- Three broad categories of existing applications:
  - Processing pipelines.
  - Batch processing systems.
  - Web applications.
- Potentially new applications
  - Batch processing for decision support systems and business analytics.
  - Mobile interactive applications which process large volumes of data from different types of sensors.
  - Science and engineering could greatly benefit from cloud computing as many applications in these areas are compute-intensive and data-intensive.

Processing pipelines

- Indexing large datasets created by web crawler engines.
- Data mining - searching large collections of records to locate items of interests.
- Image processing .
  - Image conversion, e.g., enlarge an image or create thumbnails.
  - Compress or encrypt images.
- Video transcoding from one video format to another, e.g., from AVI to MPEG.
- Document processing.
  - Convert large collections of documents from one format to another, e.g., from Word to PDF.
  - Encrypt documents.
  - Use Optical Character Recognition to produce digital images of documents.

Batch processing applications

- Generation of daily, weekly, monthly, and annual activity reports for retail, manufacturing, other economical sectors.
- Processing, aggregation, and summaries of daily transactions for financial institutions, insurance companies, and healthcare organizations.
- Processing billing and payroll records.
- Management of the software development, e.g., nightly updates of software repositories.
- Automatic testing and verification of software and hardware systems.

Web access

- Sites for online commerce.
- Sites with a periodic or temporary presence.
  - Conferences or other events.
  - Active during a particular season (e.g., the Holidays Season) or income tax reporting.
- Sites for promotional activities.
- Sites that "sleep" during the night and auto-scale during the day.

Architectural styles for cloud applications

- Based on the client-server paradigm.
- Stateless servers - view a client request as an independent transaction and respond to it; the client is not required to first establish a connection to the server.
- Often clients and servers communicate using Remote Procedure Calls (RPCs).
- Simple Object Access Protocol (SOAP) - application protocol for web applications; message format based on the XML. Uses TCP or UDP transport protocols.
- Representational State Transfer (REST) - software architecture for distributed hypermedia systems. Supports client communication with stateless servers, it is platform independent, language independent, supports data caching, and can be used in the presence of firewalls.

Workflows

- *Process description* - structure describing the tasks to be executed and the order of their execution. Resembles a flowchart.
- *Case* - an instance of a process description.
- *State of a case at time t* - defined in terms of tasks already completed at that time.
- *Events* - cause transitions between states.
- The *life cycle of a workflow* - creation, definition, verification, and enactment; similar to the life cycle of a traditional program (creation, compilation, and execution).

Workflow (cont…)

- **Creation**: In this stage, the workflow is first conceived and created. This involves identifying the purpose of the workflow and the tasks that need to be performed to achieve that purpose. The workflow is typically created by a team of stakeholders, including **business analysts, process owners, and subject matter experts**.

- **Definition**: Once the workflow has been created, it needs to be defined in more detail. This involves specifying the inputs, outputs, and steps involved in the workflow. The workflow is typically documented using a workflow modeling language, such as **BPMN (Business Process Model and Notation**).

- **Verification**: Once the workflow has been defined, it needs to be verified to ensure that it is accurate and complete. This involves reviewing the workflow model to identify any errors or omissions. The verification process may also involve testing the workflow using simulation or prototyping tools.

- **Enactment**: Once the workflow has been verified, it can be enacted or put into operation. This involves implementing the workflow using a **workflow management system (WfMS)** or other automation tools. The WfMS manages the execution of the workflow, assigning tasks to appropriate users and tracking progress until the workflow is complete.

Safety and liveness

- There are two desirable properties of workflows.

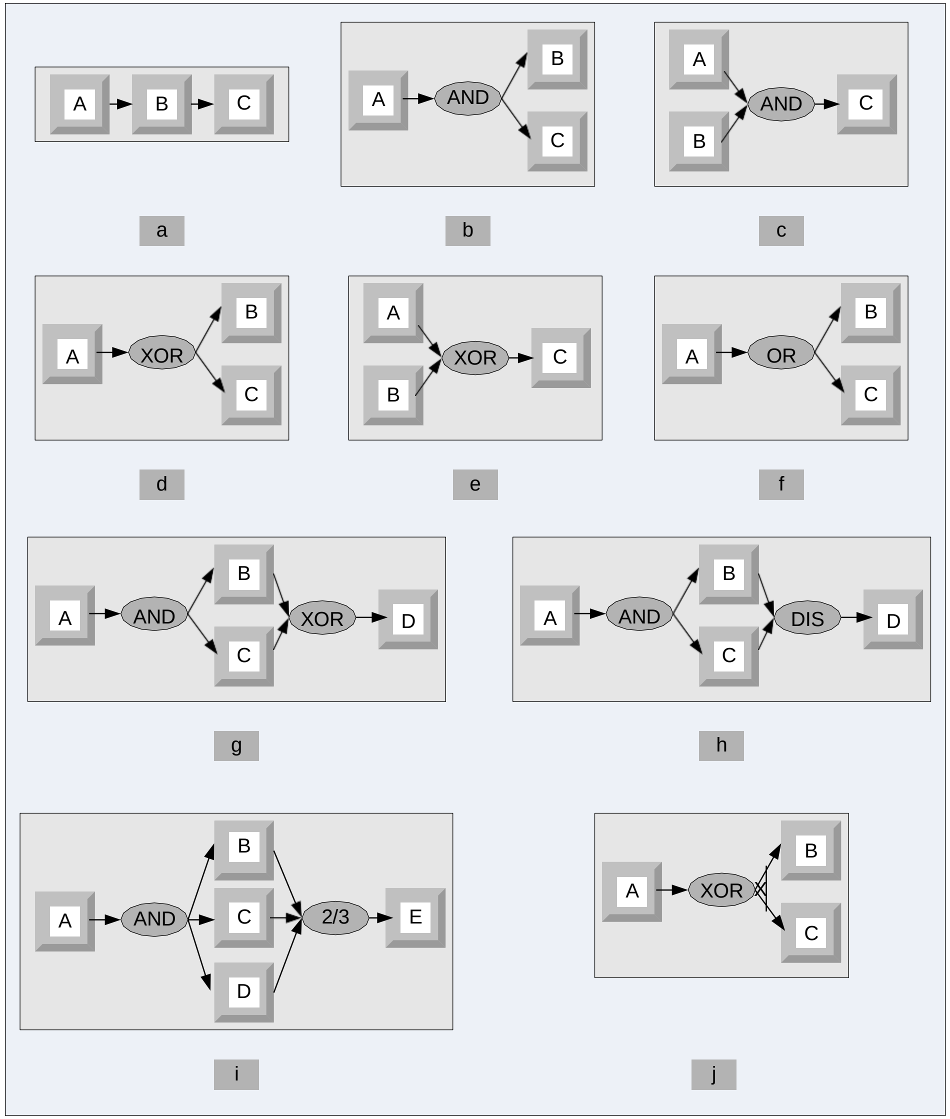
1. **Safety** → Safety refers to the property of a workflow where "bad" things never happen. A safe workflow is one that is free from errors, faults, or failures. In other words, it ensures that the system or workflow does not violate any constraints or invariants, and that it always behaves correctly. For example, **in a payment processing workflow, safety would mean that no payments are processed twice, and that all payments are processed correctly.**
2. **Liveness** → Liveness refers to the property of a workflow or system where "good" things eventually happen. A live workflow is one that is always making progress towards its goal. In other words, it ensures that the system or workflow continues to make progress and eventually completes its task. For example, **in a customer support workflow, liveness would mean that all customer inquiries are eventually resolved, and that customers are satisfied with the resolution.**

Basic workflow patterns

- Workflow patterns - the temporal relationship among the tasks of a process
  - **Sequence** - several tasks have to be scheduled one after the completion of the other.
  - **AND split** - both tasks B and C are activated when task A terminates.
  - **Synchronization** - task C can only start after tasks A and B terminate.
  - **XOR split** - after completion of task A, either B or C can be activated.
  - **XOR merge** - task C is enabled when either A or B terminate.
  - **OR split** - after completion of task A one could activate either B, C, or both.
  - **Multiple Merge** - once task A terminates, B and C execute concurrently; when the first of them, say B, terminates, then D is activated; then, when C terminates, D is activated again.
  - **Discriminator** – wait for a number of incoming branches to complete before activating the subsequent activity; then wait for the remaining branches to finish without taking any action until all of them have terminated. Next, resets itself.

Basic workflow patterns (cont'd)

- **N out of M join** - barrier synchronization. Assuming that M tasks run concurrently, N (N<M) of them have to reach the barrier before the next task is enabled. In our example, any two out of the three tasks A, B, and C have to finish before E is enabled.
- **Deferred Choice** - similar to the XOR split but the choice is not made explicitly; the run-time environment decides what branch to take.


A B C
a
A AND B C
b
A B AND C
c
A XOR B C
d
A B XOR C
e
A OR B C
f
A AND B C XOR D
g
A AND B C DIS D
h
A AND B C D 2/3 E
i
A XOR B C
j

Zookeeper

ZooKeeper is a distributed co-ordination service to manage large set of hosts. Apache ZooKeeper is a service used by a cluster (group of nodes) to coordinate between themselves and maintain shared data with robust synchronization techniques.

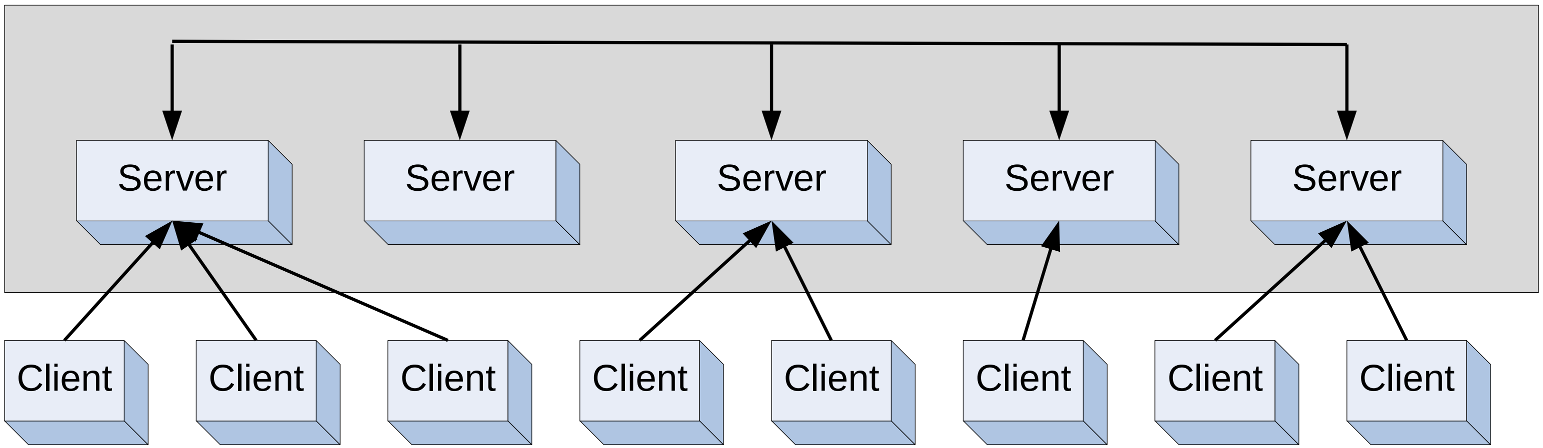
Co-ordinating and managing a service in a distributed/cloud environment is a complicated process. ZooKeeper solves this issue with its simple architecture and API.

ZooKeeper allows developers to focus on core application logic without worrying about the distributed nature of the application.

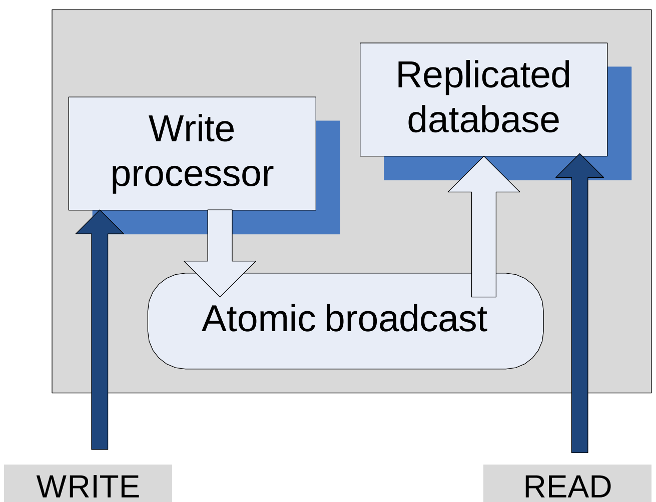
The ZooKeeper framework was originally built at “**Yahoo**!” for accessing their applications in an easy and robust manner. Later, Apache ZooKeeper became a standard for organized service.

Coordination - ZooKeeper

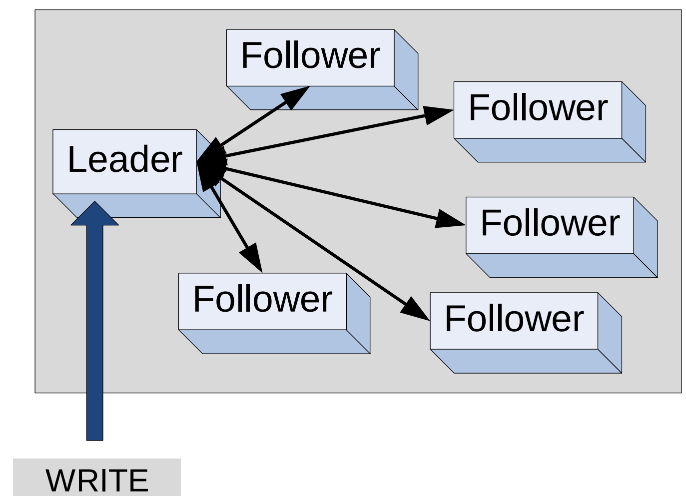
- Cloud elasticity → distribute computations and data across multiple systems; coordination among these systems is a critical function in a distributed environment.
- ZooKeeper
  - Distributed coordination service for large-scale distributed systems.
  - High throughput and low latency service.
  - Implements a version of the Paxos consensus algorithm.
  - Open-source software written in Java with bindings for Java and C.
  - The servers in the pack communicate and elect a leader.
  - A database is replicated on each server; consistency of the replicas is maintained.
  - A client connect to a single server, synchronizes its clock with the server, and sends requests, receives responses and watch events through a TCP connection.


Server
Server
Server
Server
Server
Client
Client
Client
Client
Client
Client
Client
Client

(a)


Replicated database
Write processor
Atomic broadcast
WRITE
READ

(b)


Follower
Follower
Leader
Follower
Follower
Follower
WRITE

(c)

Common services provided by ZooKeeper

Naming service − Identifying the nodes in a cluster by name. It is similar to DNS, but for nodes.

Configuration management − Latest and up-to-date configuration information of the system for a joining node.

Cluster management − Joining / leaving of a node in a cluster and node status at real time.

Leader election − Electing a node as leader for coordination purpose.

Locking and synchronization service − Locking the data while modifying it. This mechanism helps you in automatic fail recovery while connecting other distributed applications like Apache HBase.

Highly reliable data registry − Availability of data even when one or a few nodes are down.

Zookeeper communication

- Messaging layer → responsible for the election of a new leader when the current leader fails.

- Messaging layer guarantees:

  - Reliable delivery: if a message **m** is delivered to one server, it will be eventually delivered to all servers.

  - Total order: if message **m** is delivered before message **n** to one server, it will be delivered before **n** to all servers.

  - Causal order: if message **n** is sent after **m** has been delivered by the sender of **n**, then **m** must be ordered before **n**.

ZooKeeper service guarantees

- Atomicity - a transaction either completes or fails.
- Sequential consistency of updates - updates are applied strictly in the order they are received.
- Single system image for the clients - a client receives the same response regardless of the server it connects to.
- Persistence of updates - once applied, an update persists until it is overwritten by a client.
- Reliability - the system is guaranteed to function correctly as long as the majority of servers function correctly.

Zookeeper API

- The API is simple - consists of following operations:
  - Create - add a node at a given location on the tree.
  - Delete - delete a node.
  - Get data - read data from a node.
  - Set data - write data to a node.
  - Get children - retrieve a list of the children of the node.
  - Synch - wait for the data to propagate.

Elasticity and load distribution

- Elasticity → ability to use as many servers as necessary to optimally respond to cost and timing constraints of an application.
- How to divide the load
  - Transaction processing systems → a front-end distributes the incoming transactions to a number of back-end systems. As the workload increases new back-end systems are added to the pool.
  - For data-intensive batch applications two types of divisible workloads are possible:
    - modularly divisible → the workload partitioning is defined a priori.
    - arbitrarily divisible → the workload can be partitioned into an arbitrarily large number of smaller workloads of equal, or very close size.
- Many applications in physics, biology, and other areas of computational science and engineering obey the arbitrarily divisible load sharing model.

MapReduce

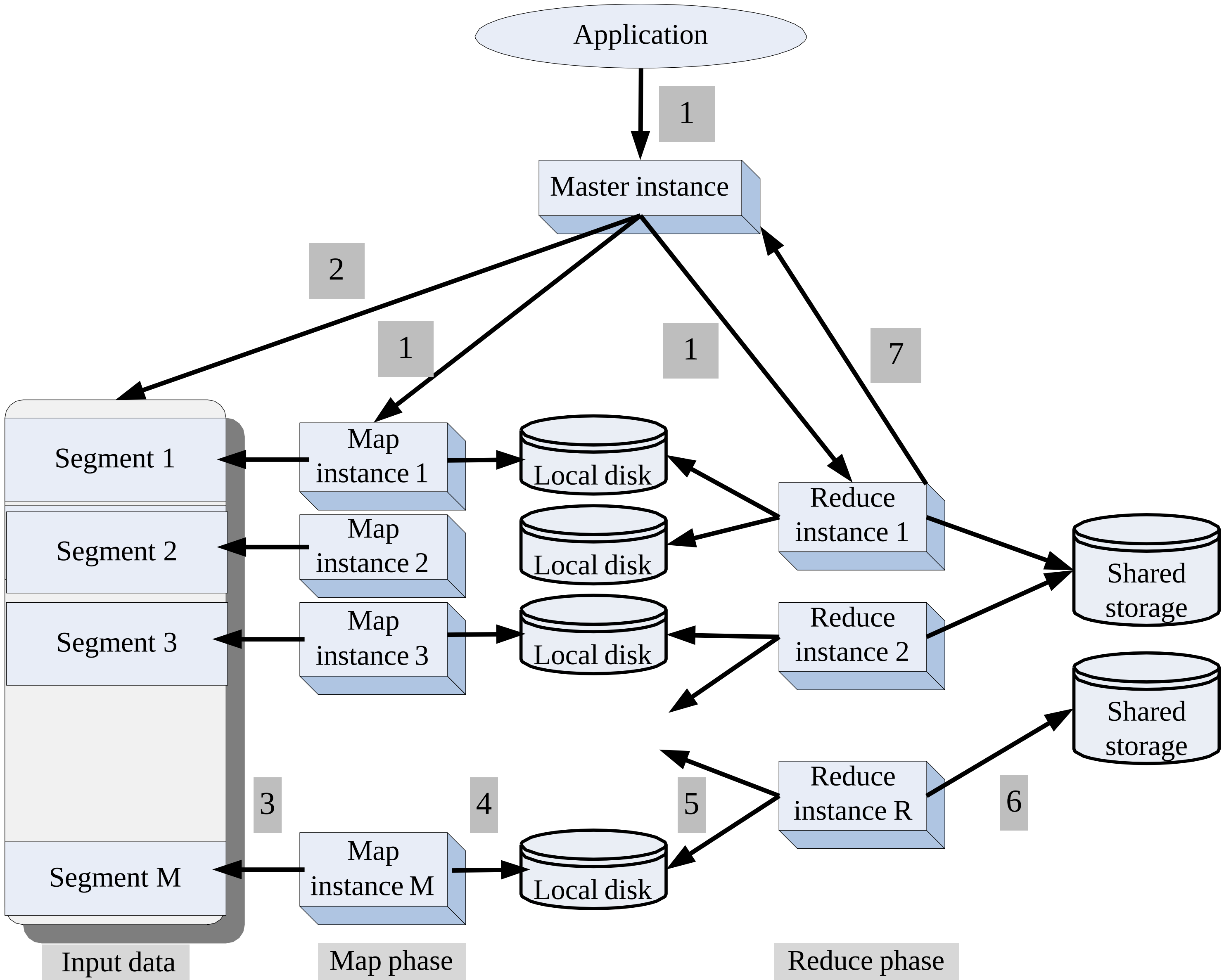
- MapReduce is a programming model and framework for processing large datasets in a cloud/distributed computing environment. It was originally developed by Google and is widely used in big data processing and analysis.
- The MapReduce model consists of two main functions:
- **Map**: The Map function takes an input dataset and transforms it into a set of key-value pairs. This transformation is typically performed in parallel across multiple nodes in a distributed system.
- **Reduce**: The Reduce function takes the output of the Map function (the key-value pairs) and combines them to produce the final output. The Reduce function is also executed in parallel across multiple nodes.

Cont….

- The MapReduce workflow typically involves the following steps:
- **Input**: The input dataset is divided into smaller chunks and distributed across multiple nodes in the cluster.
- **Map**: The Map function is applied to each input chunk in parallel, generating a set of key-value pairs.
- **Shuffle and Sort**: The key-value pairs produced by the Map function are shuffled and sorted based on the keys.
- **Reduce**: The Reduce function is applied to the sorted key-value pairs, combining the values associated with each key to produce the final output.
- **Output**: The output of the Reduce function is collected and stored as the final result.

MapReduce philosophy

1. An application starts a master instance, M worker instances for the *Map phase* and later R worker instances for the *Reduce phase*.
2. The master instance partitions the input data in M *segments*.
3. Each *map instance* reads its input data segment and processes the data.
4. The results of the processing are stored on the local disks of the servers where the map instances run.
5. When all map instances have finished processing their data, the R reduce instances read the results of the first phase and merge the partial results.
6. The final results are written by the reduce instances to a shared storage server.
7. The master instance monitors the reduce instances and when all of them report task completion the application is terminated.

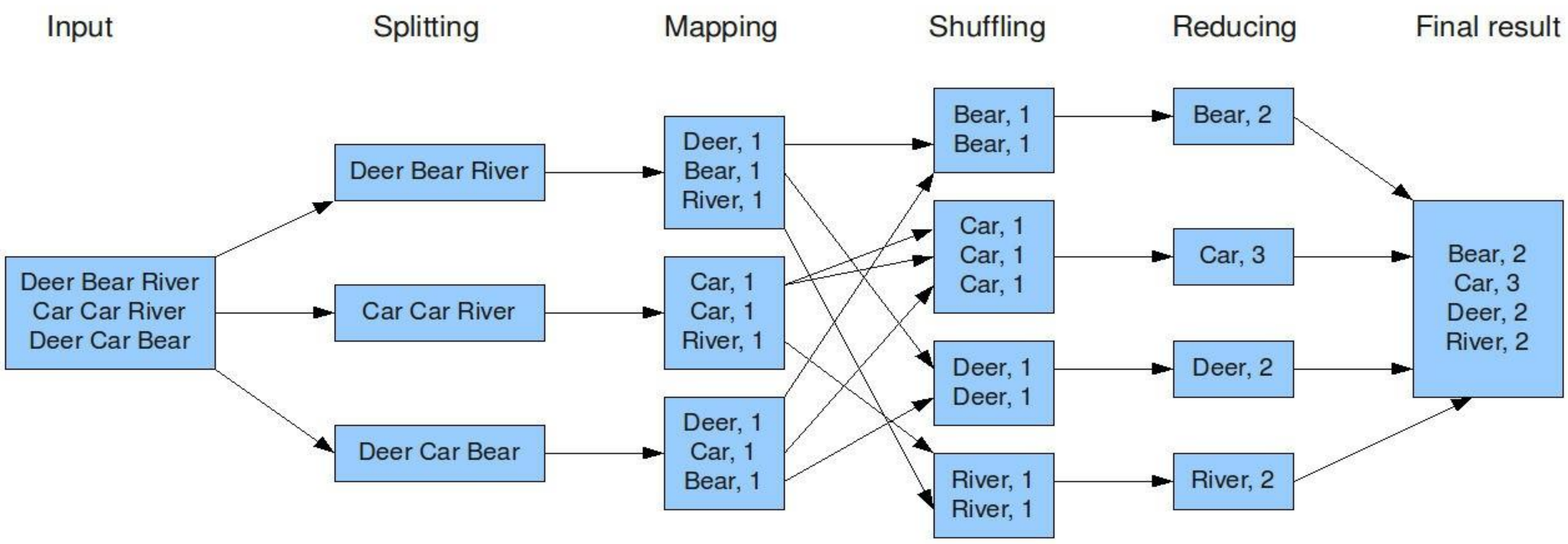

Application
1
Master instance
2
1
1
7
Segment 1
Segment 2
Segment 3
Segment M
Map instance 1
Map instance 2
Map instance 3
Map instance M
Local disk
Local disk
Local disk
Local disk
Reduce instance 1
Reduce instance 2
Reduce instance R
Shared storage
Shared storage
3
4
5
6
Input data
Map phase
Reduce phase

Word Count

Map: Input lines of text to breaks them into words gives outputs for each word <key=word, value=1>

Reduce: Input <word, 1> output <word, + value>

The overall MapReduce word count process



Clouds for science and engineering

- The generic problems in virtually all areas of science are:
  - Collection of experimental data.
  - Management of very large volumes of data.
  - Building and execution of models.
  - Integration of data and literature.
  - Documentation of the experiments.
  - Sharing the data with others; data preservation for a long periods of time.

- All these activities require “big” data storage and systems capable to deliver abundant computing cycles.
- Cloud Computing is able to provide such resources and support collaborative environments.

High performance computing on a cloud

- Comparative benchmark of *EC2* and three supercomputers at the **National Energy Research Scientific Computing Center (NERSC)** at Lawrence Berkeley National Laboratory. NERSC has some 3,000 researchers and involves 400 projects based on some 600 codes.
- Conclusion – communication-intensive applications are affected by the increased latency and lower bandwidth of the cloud. The low latency and high bandwidth of the interconnection network of a supercomputer cannot be matched by a cloud.