UNIT-4

* Features of Cloud and Grid Platforms:

→ Cloud Capabilities and Platform features:

* offer cost effective utility computing with the elasticity to scale up & down in power.

* Commercial clouds offers additional capabilities commonly known as PaaS.

→ for Azure: Azure Table, queues, blobs, DB SQL, worker roles.

→ for Amazon: IaaS, SimpleDB, monitoring, MapReduce (Hadoop)

→ Google App Engine (GAE): powerful web application

→ Traditional Features: (Common to Grids & Clouds)

* Workflow: links multiple cloud & non-cloud services in real applications on demand.

Ex: Pipeline Pilot€, AVS(dated) & LIMS environment, Trident(Microsoft)

AVS: Address Verification Service.

LIMS: Laboratory Information Management System.

* Data Transport: Cost (time & money)
  → high bandwidth links

* Security, Privacy & Availability:
  - → Virtual clustering
  - → Stable and persistent data storage with fast queries
  - → Special APIs
  - → Fine-grained access control
  - → Cloud resources are accessed with security protocols such as HTTPS & SSL
  - → Shared data sets are protected from malicious alteration, deletion.

* Data Features and databases:

→ Program Library: design a VM image library to manage images used in academic & commercial clouds.

→ Blobs and drives:
  - → Blobs: Azure
  - → S3 : Amazon

→ Distributed Parallel File System (DPFS)

* precisely designed for efficient execution of data intensive applications.

* supports file systems such as Google File System (MapReduce), HDFS (Hadoop), Cosmos (Dryad.

→ SQL and Relational Databases:

* Amazon & Azure clouds offers RDs.

* FutureGrid: a new private cloud computing model for the Observational Medical Outcomes Partnership (OMOP)

→ Table and NoSQL Non-relational databases:

* typically emphasizing distribution & scalability

* BigTable: Google
  SimpleDB = Amazon
  Azure Table = Azure.

→ Queuing Services:

* Both Amazon & Azure offer similar scalable, robust queuing services – to communicate b/w components of an application.

* REST service interface with deliver atleast once semantics and messages are short.

* Programming and Runtime Support:

→ Worker and Web Roles:

* Worker roles are basic schedulable processes and are automatically launched.
* Web Roles provide an interesting approach to portals.
* GAE is largely aimed at web applications, whereas science gateways are successful in TeraGrid.

→ MapReduce:

* There has been substantial interest in "data parallel" languages - largely aimed at loosely coupled combinations - which execute over different data samples.

*→ Major open source/commercial MapReduce implementations - Hadoop & Dryad - execution possible with (or) without VMs.

*→ Hadoop is currently offered by Amazon.

*→ Cloud Programming Models:

→ Iterative MapReduce: an interesting programming model that offers portability between cloud, HPC & cluster environments.

*→ SaaS:

→ Services are used in a similar fashion in commercial clouds & most modern distributed systems.

→ provides many useful tools to develop cloud applications over large data sets

* Hadoop Library from Apache:

→ Hadoop = open source implementation of MapReduce

→ Hadoop core is divided into 2 layers.

↳ MapReduce: computation engine running on top of HDFS as its data storage manager.

(Hadoop Distributed FS)

→ HDFS: distributed file system: organizes files & stores their data on a distributed computing system.

*→ HDFS architecture: has master/slave architecture containing a single Name node (master) and a no. of data nodes (slaves).

* HDFS splits the file into fixed size blocks & stores them on workers (data nodes).

→ Each datanode is responsible for storing & retrieving its file blocks.

*→ HDFS features:

→ Security has never been supported.

→ 2 main characteristics

① HDFS fault tolerance

following are considered to fulfill reliability:

→ Block replication:

↳ file blocks are replicated

↳ replication factor is set by user, default is 3.

→ Replica replacement:

↳ the storage of replicas is another factor.

↳ cost of communication b/w nodes in diff. racks is relatively high in comparison; hence reliability is compromised to reduce cost.

→ Heartbeat and Block Report: messages

↳ periodic messages sent to NameNode by each DataNode.

↳ describes about proper functionality of DataNode

↳ Block report consists of list of all blocks

② HDFS high throughput Access to large datasets:

→ HDFS is primarily designed for batch processing rather than interactive processing.

→ throughput is more important than latency

→ Applications running on HDFS have large data; files are broken into blocks to allow HDFS to use the amount of metadata storage required per file.

→ 2 advantages:

① List of blocks per file will shrink.

② provides fast streaming reads of data.

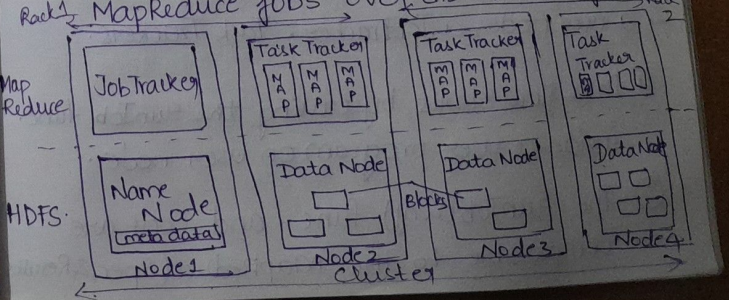
* HDFS operation:

① Reading a file: "open request"

② Writing a file: "create request"

*Architecture of MapReduce in Hadoop:

→ Top most layer of Hadoop engine that manages the data flow and control flow of MapReduce jobs over distributed systems



→ MapReduce engine has a master/slave architecture.
  - → Job Tracker: Master
  - → No. of Task Trackers: Slave.

* Job Trackers manages the MapReduce job over a cluster & is responsible for monitoring and assigning tasks to TaskTrackers.
* TaskTracker manages execution of map or reduce tasks on a single computation node in the cluster.
  - → Each task tracker has a no. of simultaneous execution slots.
  - → Slots are no. of simultaneous threads supported by CPUs.

Ex: N CPUs; each supporting M threads;
Simultaneous execution slots: M * N.

* Running a task in Hadoop:

→ a user node, a JobTracker, Task Trackers..

① Data flow starts by calling the runJob func. inside a user program on user node.

② The runJob(conf) func. and conf are comparable to the MapReduce(Spec, &Results) func and spec in the first implementation of MapReduce by Google.

* Parallel and distributed Programming Paradigms:

→ parallel program running on a set of computing engines or distributed computing system.

→ a distributed computing system is a set of computational engines connected by a network to achieve a common goal.

→ parallel computing is the simultaneous use of more than one engine.

* Running a parallel program on distributed system:
  - → ↓ses application response time.
  - → ↑ses throughput & resource utilization.

* Issues in running parallel program on distributed systems:

① Partitioning:
  - → Computation: splits jobs into smaller tasks.
  - → Data: splits I/P (or) intermediate data into smaller pieces.

(2) Mapping: assigning smaller tasks (or) smaller pieces of data to resources.

(3) Synchronization: coordination & synchronization among workers is necessary to avoid race conditions.

(4) Communication: triggered when intermediate data is sent.

(5) Scheduling: selects a sequence of tasks (or) data pieces to be assigned to the workers.

* MapReduce, Twister and Iterative MapReduce:

→ MapReduce:
  - → software framework
  - → which supports parallel & distributed computing on large data sets.

→ Abstracts data flow of running a parallel program on a distributed computing system:
  - → by providing users with 2 interfaces.
  - → 2 functions: Map and Reduce.

→ Users can override these 2 functions to interact with & manipulate the data flow of running their programs.

→ In this framework;
  value part of the data; is actual data
  key part is only used by the MapReduce controller to control the data flow.

* Formal Definition of MapReduce:

→ MapReduce: software framework
  - → provides an abstraction layer
  - → with data flow and flow of control to users.
  - → hides implementation of all data flow steps.
  - → such as data partitioning, mapping, synchronization, communication, scheduling.

→ abstraction layer provides 2 well-defined interfaces in the form of 2 functions: Map and Reduce.

→ User overrides the Map and the Reduce functions first & then invokes the provided MapReduce (Spec, &Results) func; from the library to start the flow of data.

* MapReduce Logical Data flow:

→ The I/P data to both the Map and the Reduce functions has a particular structure.

→ The I/P data to the Map function is in the form of a (key, value) pair.

→ The O/P data from the Map function is structured as (key, value) pairs called intermediate (key, value) pairs.

→ The goal is to process all input (key, value) pairs to the Map function in parallel.

→ The Reduce function receives the intermediate (key, value) pairs in the form of group of intermediate values associated with one intermediate key (key, [set of values])

→ MapReduce forms these groups by first sorting the intermediate (key, value) pairs and then grouping values with same key.

* Most known application: Count Word.

* Example:

(1) most people ignore most poetry
(2) most poetry ignores most people.

→ Map function simultaneously produces a no. of intermediate (key, value) pairs for each line of content so that each word is the intermediate key with 1 as its value

Ex: (ignore, 1)

→ MapReduce collects all and sorts them

Ex: (people, [1, 1])

→ Groups are sent to Reduce function; so that the arrays are added up.

Ex: (people, 2)

* Map function:

$(key_1, val_1) \xrightarrow{\text{Map func}} \text{List}(key_2, val_2)$

* Reduce function:

$(key_2, \text{list}(val_2)) \xrightarrow[\text{func}]{\text{Reduce}} \text{List}(val_2)$

→ find unique keys to solve MapReduce problem.

Ex: Chapter 6:2 : Slide 17 **

* Steps for MapReduce Actual data & control flow

(1) Data partitioning: I/P data ——→ M pieces
(2) Computation partitioning : Map & Reduce
(3) Determining the master & workers
(4) Reading the I/P data
(5) Map function
(6) Combiner function
(7) Partitioning function
} Map workers
(8) Synchronization
(9) Communication
} Networking steps
(10) Sorting and Grouping
(11) Reduce function
} Reduce worker

* Data Affinity:

→ GFS (Google file system) is a distributed file system where files are divided into fixed-size blocks & blocks are distributed & stored on cluster nodes.

Default GFS block size = 64 MB

→ MPI : Iterative MapReduce

→ 2 major sources of parallel overhead: load imbalance & communication

→ Communication overhead in MapReduce can be quite high due to:

(1) MapReduce reads & writes via files; whereas MPI transfers info directly b/w nodes over the network.

(2) MPI does not transfer all data from node to node ; but just the amount needed to update info.

* Programming Support of Google App Engine:

→ Google File System (GFS)

* fundamental storage service
* Assumptions concerning GFS:
  - high component failure rate
  - huge files.

→ hence ; GFS has 64 MB

* I/O Pattern:

→ Files - written once & write operations are often appending the data blocks to the end of files.

→ Highly sustained throughput is much more

more important than low latency.

→ Reliability is achieved by using replication

→ File system namespace & locking facilities - managed by master.

→ Potential weakness: single GFS master could be the performance bottleneck & single point of failure.

* Write and append operations in GFS:

① client-master: which chunk server holds the current lease for the chunk and the locations of other replicas.

② master replies with the identity of primary & locations of other replicas.

③ client pushes data to all replicas. Each chunk server will store the data in an internal LRU buffer until the data is used or aged out.

④ Once all replicas acknowledges data, client sends a write request. The request identifies the data pushed earlier to all replicas.

⑤ The primary forwards the write request to all secondary replicas.

⑥ The secondaries all reply to the primary indicating that they have completed the operation

⑦ The primary replies to the client

* BigTable: Google's NOSQL system:

→ designed to provide a service for storing & retrieving structured & semi-structured data.

→ storage of web pages, per-user data & geographic locations.

** Goals of BigTable:

→ Applications want asynchronous processes to be continuously updating diff. pieces of data.

→ DB needs to support very high read/write rates, scale might be millions of operations per second.

→ DB needs to support efficient scans over all (or) interesting subsets of data.

→ Examine data changes over time.

** Viewed as distributed multilevel

- → provides a fault tolerant & persistent database as in a storage service.
- → Scalable
- → Self-managing system.

* BigTable uses: GFS, Scheduler, Lock Service, MapReduce.

* Amazon AWS:

→ Offers Simple Queue Service (SQS) & Simple Notifications (SNS)

→ Both auto-scaling and elastic load balancing are enabled by CloudWatch.

→ CloudWatch is a web service that provides monitoring for AWS cloud resources, starting with Amazon EC2.

*→ Workflow to create VM:

Create an AMI → Create Key Pair → Configure Firewall → Launch

* AMI: Amazon Machine Images.

AMIs are formed from virtualized compute, storage & server resources.

* Classes of instances:

→ Standard instances
→ Micro instances
→ High memory instances
→ High CPU instances
→ Cluster compute instances

* Amazon Simple Storage Service (S3)

→ a simple web services interface that can be used to store and retrieve any amount of data.

→ SQS is responsible for ensuring a reliable message service btw 2 processes, even if the receiver processes are not running.

→ fundamental operation unit in S3 = object.

*→ bucket is the container of the object

→ Key features:

* Provides durability & availability
* Per-object URLs & ACLs.
* Default download protocol of HTTP.
* Redundant through geographic dispersion.

* Amazon Ebastic Block Store (EBS) & Simple DB

→ EBS provides volume block interface for saving & restoring the virtual images of EC2 instances.

→ EBS is analogous to a distributed file system accessed by traditional OS disk access mechanisms.

→ Volume range: 1GB to 1TB.