10.3 (00, 01, 02, 03)

00

Lecture email: [fren@uow.edu.au](mailto:fren@uow.edu.au)

Tutor email: [sjapit@uow.edu.au](mailto:sjapit@uow.edu.au)

Assign: 10% + 20% + 20% = 50%

Final : 50%

TF(technical fail) : 40%

01 Cluster Computing

Computer Cluster(计算机集群)

- **collection of computers(nodes)** connected through **high speed network**, that work together to simulate a single much more powerful computer system

- **each node** in a computer cluster is controlled by its **own operating system**

- each node in a computer cluster performs a **different version of the same task**

- difference between **computer cluster** and **computer grid** is the nodes in a computer grid **perform** **different tasks**

- an architecture of **computer cluster** ranges from a **simple two-node system** connecting two personal computers to a super computers to a supercomputer with a cluster architecture

- used to **speed up computing** through **shared nothing(sharding) partitioning** of data and **paralellization** of data processing on the nodes of clusters

- provide **high availability** through **automatic replacement** of a failed node with a replica node

- advantages of computer clusters: faster processing speed, larger storage capacity, better data integrity, greater reliability and wider availability of resources

- Linux cluster is a **collection of connected computers** that can be viewed and managed as a single system

Cluster computing(集群计算)

- is the progress of **sharing the computation tasks** among **multiple computers** included in a **computer cluster**

- advantage of cluster computing: cost efficiency, processing speed, expandability, high vailability of resources

- is an attractive paradigm for **processing large scale** science, engineering and commercial applications

- requires the **specialized algorithms** like load balancing, resource sharing and resource scheduling for **optimization of data processing**

-attractive alternative to **data processing** on **large parallel supercomputers**

**-** the simplest configuration of nodes for **cluster computing** consists of a **master node** and **slave nodes**

Big Data

- to big that it cannot be stored on the persistent storage devices attached to a single computer system

- an infinite amount of data

3V features :

- volume : billions of rows, millions of columns

- variety : complexity of data types and structures

- velocity : speed of new data creation and growth

Examples:

- clickstream data

- call centre data

...

Traditional Data Architectures(传统数据架构)

Data warehousing technologies:

- excel, DB/2, Oracle, MySQL, SQL server, files

The strength of traditional data architectures

- centralised governance of data repositories

- light-fast inquires performed regularly in daily business

- optimisation for OLTP and OLAP

- security and access control

- fault-tolerance and backup

The challenges for traditional data architectures:

- new types of data such as unstructured data and semi-structured data

- large amounts of data flowing into organisations

- new computational paradigms use non-traditional NoSQL databases to rapidly mine analyse very large data set

- increasing cost of storing and analysing the large amounts of data

- increasing use of data analytics, which requires significant storage and processing capabilities

Hadoop

- is a project that develops open-source software for reliable, scalable, distributed computing

Features of Hadoop

- capability to handle large data sets, e.g. simple scalability and coordination

- file size range from gigabytes to terabytes

- can store millions of those files

- high fault tolerance

- support data replication

- supports streaming access to data

- supports batch processing

- support interactive, iterative and stream processing

- implements a data consistency model of write-one- read-many access model

- run on commodity hardware, not high-performance computers

- inexpensive

- it can be deployed on premises or in the cloud

Core components of Hadoop

- different data-processing frameworks

- YARN: an operating system for Hadoop(Hadoop cluster resource management)

- HDFS (Hadoop Distributed File System)

A typical Hadoop cluster consist of

- a set of master node(servers) where the daemons supporting key Hadoop frame-works run

- a set of worker nodes that host the storage (HDFS) and computing (YARN) work

- one or more edge servers, which are used for accessing the Hadoop cluster to launch applications

- one or more relational databases such as MySQL for storing the metadata repositories

- dedicated servers for special frameworks such as Kafka

- support the pseudo-distributed mode

- all HDFS and YARN daemons running on a single node

- highly simulate the full cluster

- easy for beginner’s practice

- easy for testing and debug

Big data on Database Clusters

- a database cluster is a collection of databases that is managed by a single instance of a running database server

- a very large database in a database cluster is partitioned over a number of smaller databases each located on a separate node of a computer cluster

- database clustering requires replication and sharding

- database clustering improve performance, availability, and scalability

Big data on Kubernetes

- Kubernetes(K8) is a container or microservice platform that orchestrates computing, networking, and storage infrastructure workloads

- Kubernetes is an orchestration platform to manage any containerized application

- a Kubernetes cluster consists of a single master node and potentially multiple corresponding worker nodes

The benefits of Kubernetes:

- horizontal scaling

- automated rollouts and rollbacks

- service discovery and load balancing

- storage orchestration

- self healing

- batch execution

- automatic binpacking

02 MapReduce(分布式计算系统) Framework

MapReduce

- the most important processing framework in Hadoop

- many high-level data processing languages are abstractions of MapReduce, e.g. Pig and Hive or are heavily influenced by MapReduce concepts e.g. Spark

- is a platform and language-independent programming model at the heart of most big data and NoSQL platforms

- programming model means a pattern/format in accordance to which we write our programs

- the logic of a MapReduce application consists of a Map phase and a Reduce phase

- limitations of early distributed computing and grid computing frameworks

(complexity in parallel programming, hardware failures, bottlenecks in data exchange, scalability problem)

- MapReduce model uses key-value pairs for processing data

MapReduce implementation in Hadoop

- Hadoop MapReduce frees the users from the low-level communication and coordination of nodes and processes

- focus on the MapReduce implementation and a few configuration parameters

- as the data file is usually too large to be stored in a single persistent storage device(of the commodity hardware), Hadoop handles the shipment of code to data fragments(aka, data locality) // reduce the overhead of network transmits

Why Hadoop is useful to Big data

- cost-effective fault-tolerant storage(HDFS)

- scalability

- data that is ingested may be interpreted at runtime

- low cost in storing unstructured and semi-structured data

- fast transfer of data into storage

- separation of programming logic and scheduling/management

- multiple levels of distributed system abstractions: Hive, Pig, Spark

- multi-language tooling: Java: MapReduce; SQL: Hive; data-flow: Pig; Scala, Python: Spark

03 Hadoop Architecture

HDFS : Hadoop Distributed File System

- HDFS is deigned for very large files, stream data access, commodity hardware

- but not for low-latency data access, lots of small files, multiple writers, arbitrary file modifications

- HDFS contains the following key components

NameNode:

- HDFS master node process

- manages the filesystem metadata

- does not store a file itself

SecondaryNameNode and Standby NameNode

- SecondaryNameNode expedites the filesystem metadata recovery

- Standby NameNode (optional) provides high availability

DataNode

- runs HDFS slave node process

- manages block storage and access for reading or writing of data, block replication

- HDFS is a virtual filesystem

- appears to a client as one file system, but the data is stored in multiple different locations

- deployed on the top of the native filesystems

- each file in HDFS consists of blocks

- the size of each block defaults to 128MB but is configurable

- the default number of replicates for blocks is 3,but it is also configurable

NameNode Metadata

- NameNode stores the metadata of the files in HDFS

NameNode functions:

- maintain the metadata pertaining to the file system(e.g. the file hierarchy and the clock locations for each file)

- manage user access to the data files

- Map the data blocks to the DataNodes in the cluster

- perform file system operations(e.g. opening and closing the files and directories)

- provide registration services and periodic heartbeats for DataNodes

DataNode and Secondary node

DataNode function:

- provide the block storage by storing blocks on the local file system

- fulfil the read/write requests

- replicating data across the cluster

- keeping in touch with the NameNode by sending periodic block reports and heartbeats

- a heartbeat confirms the DataNode is alive and healthy, and a block report shows the blocks being managed by DataNode

Secondary NameNode and Standby NameNode functions:

- without a NameNode, there is no way to know to which files the blocks stored on the DataNodes correspond to

- all files in HDFS are lost

- Secondary NameNode periodically backups the metadata in the (primary) NameNode, which is usually for recovery

- Standby NameNode is a hot node that running together with the(primary) NameNode in the cluster, facilitating high-availability

Yet Another Resource Negotiator(YARN)

- YARN, the core subsystem in Hadoop responsible for governing, allocating, and managing the finite distributed processing resources available on the Hadoop cluster

- YARN provides its core services via two types of long-running daemons:

- a resourceManager(one per cluster) to manage the use of resources across the cluster, and

- NodeManagers running on all the nodes in the cluster to lauch and monitor containers

Architecture of YARN

- a client is the program that submits jobs to the cluster

- a job, also called an application, contains one or more tasks

- a task in a MapReduce job can be either a mapper and a reducer task

- each mapper and reducer task runs within a container

- containers are logical constructs that represent a specific amount of memory and other resources, such as processing cores(CPU)

- for example, a container can represent 2GB memory and 2 processing cores

- containers may also refer to the running environment of an application

ResourceManager: YARN’s daemon running on a mater node

- ResourceManager is responsible for granting cluster computing resources to applications running

- resource are granted the items of containers

NodeManager: YARN’s daemon running on a slave node

- NodeManager manages containers on a slave node

- ApplicationMaster: the first container allocated by the ResourceManager to run on a NodeManager for each application

ResourceManager - The role of ResourceManager is pure management and scheduler

- There is one ResourceManager per cluster, which consists of two key components: Scheduler and ApplicationManager

Key functions of ResourceManager:

- Creates the first container for an application to run ApplicationMaster

for that application

- Tracks the heartbeats from NodeManagers to manage DataNodes

- Runs Scheduler to determine resource allocation among the clusters

- Manages cluster level security

- Manages the resource requests from ApplicationMasters

- Monitors the status of ApplicationMasters and restarts that container

upon its failure

- Deallocates the containers when the application completes or after they expire

It does not perform any actual data processing, for example the Map and Reduce functions in a MapReduce application

NodeManager

- each DataNode runs a NodeManager darmon for performing YARN functions

Main functions of a NodeManager daemon:

- Communicates with ResourceManager through health heartbeats and

container status notifications.

- Registers and starts the application processes

- Launches both ApplicationMaster and the rest of an application's resource containers (that is, the map and reduce tasks that run in the containers) on request from ApplicationMaster

- Oversees the lifecycle of the application containers

- Monitors, manages and provides information regarding the resource consumption (CPU/memory) by the containers

- Tracks the health of DataNode

- Provides auxiliary services to YARN applications, such as services used by the MapReduce framework for its shuffle and sort operations

ApplicationMaster

- for each YARN application, there is a dedicated ApplicationMaster

- functions of ApplicationMaster

- managing task scheduling and execution

- allocating resources locally for the application’s task

- ApplicationMaster is running within a container

- ApplicationMaster’s existence is associated with the running application

- when an application is completed, its ApplicationMaster no longer exists

- Once created, ApplicationMaster is in charge of reguesting resources with ResourceManager to run the application

- the resource request are very specific, for example:

- the file blocks needed to process the job

- the amount of the resource, in terms of the number of containers to create for the application

- the size of the containers

Summary

Terminologies

- for convenience, we use the names of HDFS and YARN processes to refer to both the hosts and the daemons running on the corresponding hosts

- for example, ResourceManager refers to both a master node and the ResourceManager daemon on that master node; DataNode refers to both a slave node and the DataNode daemon on that slave node

- Hadoop consists of a storage layer (HDFS), a coordination and management layer (YARN) and a processing layer (MapReduce)

- HDFS and YARN have key services(daemons)

- MapReduce is a fundamental computing model(batch processing) for big data

04 HDFS Interfaces

Hadoop Cluster vs. Pseudo-Distributed Hadoop (伪分布式Hadoop)

- A Hadoop cluster is deployed in a cluster of computer nodes

- as Hadoop is developed in Java, all Hadoop services sit on Java Virtual Machines running on the cluster nodes

Hadoop provides a pseudo-distributed mode on a single machine

- all Java Virtual Machines for necessary Hadoop services are running on a single machine

HDFS provides the following interfaces to read, write, interrogate, and manage the filesystem

- the file system shell (command-line interface): hadoop fs or hdfs dfs

- Hadoop Filesystem Java API

- Hadoop simple Web User interface

- Other interfaces, such as RESTful proxy interfaces(e.g. HttpFS)

Shell interface to HDFS

Some commands

-put upload a file from the local filesystem to HDFS

-mkdir create a directory in HDFS

-ls List the files in a directory in HDFS

-cat Read the content of a file (or files) in HDFS

-copyFromLocal copy a file from from the local filesystem to HDFS

-copyToLocal copy a file from HDFS to the local filesystem

-rm delete a file in HDFS

-rm -r delete a directory in HDFS

Web interface of HDFS

Java interface to HDFS

- a file in Hadoop filesystem is represented by a Hadoop Path object

- its syntax is URL

for example,

hdfs://localhost:8020/user/bigdata/input/README.txt

- to get an instance of FileSystem, use the following factory methods:

public static FileSystem get(Configuration conf) throws IOException

public static FileSystem get(URL uri, Configuration conf) throws IOException

public static FileSystem get(URL uri, Configuration conf, String user) throws IOException

- to get a local filesystem instance

public static FileSystem getLocal(Comfiguration conf) throw IOException

- a configuration object is determined by the Hadoop configuration files or user-provided parameters

- Using the default configuration, one can simply set

Configuration conf = new Configuration()

- with a filesystem instance, we invoke an open() method to get the input stream for a file

public FSDataInputStream open(Path f) throw IOException

public abstract FSDataInputStream open(path f, int bufferSize) throws IOException

- a path object can be created by using a designated URI

Path f = new Path(uri)

Internals of HDFS

05MapReduce Data Processing Model

Key-value pairs - MapReduce basic data model

- input, output, and intermediate records in MapReduce are represented as key-value pairs

- a key is an identifier,for example, a name of attribute

- in MapReduce, a key is not required to be unique

- a value is a data associated with a key

- it may be simple value or a complex object

MapReduce Model

- MapReduce data processing model is a sequence of Map, Partition, Shuffle and Sort, and Reduce stages

Map phase

- Map phase uses input format and record reader functions to derive records in the form of key-value pairs for the input data

- Map phase applies a function or functions to each key-value pair over a portion of the dataset

- in the case of a dataset hosted in HDFS, this portion is usally called as a block

- if there are n blocks of data in the input dataset, there will be at least n Map tasks(also referred to as Mappers)

- Each Map task operates against one filesystem(HDFS) block

- Each call of the map() function accepts one key-value pair and emits zero or more key-value pairs

// map(in\_key, in\_values) -> list(intermediate\_key, intermediate\_value)

- the emitted data from Mapper, also in the form of lists of key-value pairs, will be subsequently processed in the Reduce phase

- different Mappers do not communicate or share data with each other

- common Map() functions include filtering of specific keys, such as filtering log messages if you only wanted to count or analyse ERROR log messages

// Map(k,v) = if (ERROR in v) then emit(k, v)

- another example of Map() function would be to manipulate values, such as a function that converts a text value to lowercase

// Map(k, v) = emit(k, v.toLowercase())

- Partition function, or Partitioner, ensures each key and its list of values is passed to one and only one Reduce task or Reducer

- The number of partitions is determined by the(default or user-defined) number of Reducers

- Custom Partitioners are developed for various practical purposes

Reduce Phase

- Input of the Reduce phase is output of the Map phase(via shuffle-and sort)

- Each Reduce task(or Reducer) executes a reduce() function for each intermediate key and its list of associated intermediate

- the output from each reduce() function is zero or more key-values

// reduce (intermediate\_key, list (intermediate\_value)) -> (out\_key, out\_value)

- note that, in the reality, an output from Reducer may be an input to another Map Phase in a complex multistage computational workflow

Shuffle and Sort - the heart of MapReduce

- is the process where data are transferred from Mapper to Reducer

- the most important purpose of Shuffle-and-sort is to minimise data transmission through a network

- in general, in Shuffle-and-Sort, the Mapper output is sent to the target Reduce task according to the partitioning function

Combine phase

- if the Reduce function is commutative and associative then it can be performed before the Shuffle-and-Sort phase

- in this case, the Reduce function is called a Combiner function

- for example, sum and count is commutative and associative, but average is not

- the use of a Combiner can minimise the amount of data transferred to Reduce phase and in such way reduce the nerwork transmit overhead

- a MapReduce application may contain zero Reduce tasks

- in this case, it is a Map-Only application

06 Java MapReduce Application

Building blocks of MapReduce program

- Mapper, Reducer, Combiner, and Partitioner classes correspond to their counterparts in the MapReduce model, these classes implement the MapReduce logic

- the Driver or ToolRunner in a MapReduce program represents the client program

- an elementary MapReduce program consists only of a Mapper class, a Reducer and a Driver

- As the main method is contained in the Driver, sometimes(but not always) it is convenient to make Mapper and Reducer as inner classes in Driver, which contains routine codes

Building blocks of MapReduce program: Driver

- Driver is the program which sets up and starts a MapReduce application

- Driver code is executed on the client; this code submits the application to the ResourceManager along with the application’s configuration

- Driver can also configure and submit more than one application; for instance, running a workflow consisting of multiple MapReduce application

Building blocks of MapReduce program: Mapper

- Mapper Java class contains a map() method

- its object instance iterates through the input to execute a map() method, using the InputFormat and its associated RecordReader

- The number of HDFS blocks for the file determines the number of input splites, which, in turn, determines the number of Mapper objects(or Map tasks) in a MapReduce application

- Mapper do most of the heavy lifting in data processing in MapReduce as they read he entire input file for the application

- Mappers can also include setup and cleanup code to run in any given object lifespan

- setup is called before the map() method; cleanup is called after it

Building blocks of MapReduce program: Reducer

- Reducer runs against a partition and each key and its associated values are passed to a reduce() method inside Reducer class

- Reducer’s InputFormat matches Mapper’s OutputFormat

- Mapper usually do the data preparation, for example, filtering and extracting; Reducer usually contains the main application logic, for example, summation, counting, and averaging operations are implement in Reducer

- the runtime of Reducer instances is usually faster(and much faster in some cases) than the runtime of Mapper instances

Hadoop datatype objects

- in most programming languages, when defining most data elements, we usually use simple, or primitive, datatypes such as int, long, or char

- in Hadoop a key or a value is an object that is an instantiation of a class, wieh attributes and defined methods

- a key or value contains(or encapsulates) the data with methods defined for reading and writing data from and to the object

Writable interface

- Hadoop serialisation format is writable interface

- for example, a class that implements writable is IntWritable, which a wrapper for a java int

- one can create such a class and set its value in the following way

// IntWritable writable = new IntWritable();

// writable.set(163);

WritableComparable interface

- IntWritable implements WritableComparable interface

- it is interface of Writable and java.lang.Comparable interfaces

- Comparison is crucial for MapReduce, because MapReduce contains a sorting phase during which keys are compared with one another

- WritableComparable permits to compare records read from a stream without deserialising them into objects, thereby avoiding any overhead of object creation

Input and output formats

- FileInputFormat(the base class of InputFormat) reads data(keys and values) from a given path, using the default or user-defined format

- The default input format is LongWritable for the keys and Text for the values

- FileOutputFormat (the base class of OuputFormat) writes data into a file in a given path

- the output format is usually defined by a programmer

- for example, the output format is Text for the keys and IntWritable for the value

Java code of Driver

- Job object and configuration

- Driver class instantiates a Job object

- a Job object creates and stores the configuration options for a Job, including the classes to be used as Mapper and Reducer, input and output directories

- the configuration options are specified in the following places

- Hadoop defaults(\*-default.xml)

- A default configuration is documented in the Apache Hadoop documentation

- the \*-site.xml files on the client node where Driver code is processed

- the \*-site.xml files on the save node where Mapper runs on

- Configuration properties set at the command line as arguments to a MapReduce application(in a ToolRunner object)

- Configuration properties set explicitly in code and compiled through a Job object

Driver routines

- parse the command line for positional arguments - input file/directly and output directory

- creates a new Job object instance, using gtConf() method to obtain configuration from the various sources(\*-default.xml and \*-site.xml)

- Gives a Job a friendly name, you will see in the ResourceManager UI

- sets the InputFormat and OutputFormat for a Job and determines the input splits for a Job

- defines Mapper and Reducer classes to be used for a job(They must be available in the Java classpath where Driver is run - typically these classes are packaged alongside the Driver)

- sets the final output key and value classes, which will be written out files in the output directory

- submits a Job object through job.waitForCompletion (true)

Java code of Mapper

- MyMapper class

- A class MyMapper extends a base Mapper class included within the Hadoop libraries

- in the example, the four generics in Mapper<Object, Text, Text, IntWritable> represent <map\_input\_key, map\_input\_value, map\_output\_key, map\_output\_value>

- these generics must correspond to:

- key-value pair types as defined by InputFormat in Driver

- job.setMapOutputKeyClass and job.setMapOutputValueClass defined in Driver

- input and output to the map() method

- in map() method, before performing any functions against a key or a value(such as split()), we need to get the value contained in the serialised Writable or WritableComparable object, by using the vaue.toString() method

- after performing operations against the input data(key-value pairs), the output data(intermediate data, also key-value pairs) are WritableComparable and Writable objects, both of which are emitted using a Context object

- in case of a Map-only job, the output from Map phase, namely the set of key-value pairs emitted from all map() methods in all map tasks, is the final output, without intermediate data or Shuffle-and-Sort phase

Context object

- a context object is used to pass information between processes in Hadoop

- invoke its write() method to write the output data from Mapper and Reducer

Java code of Reducer

- MyReducer class

- a class MyReducer extends the based Reducer class included with the Hadoop libraries

- a reducer() method accepts a key and an Iterable list of values as input

- the values have type Iterable<T> e.g.Iterable<IntWritable>

Java code of ToolRunner

- optional, Driver can leverage a class called ToolRunner, which is used to parse command-line options

- ToolRunner enables flexibility in supplying configuration parameters at the command line when submitting a MapReduce job

07 Hive

- is a software system that provides tabular view of data stored in HDFS and SQL-like methods for manipulating data in HDFS

- HQL provides a tabular view of data an it can be used to access data located in HDFS

Deployment and Configuration

Metastore

- Metastore contains the mappings of tables to the directory location in HDFS

- Metastore is a relational database read and written by Hive client

- Metastore includes the input and output formats for the files represented by the table objects

- input and output formats for the files and functions are used by Hive to extract records and fields from the files

Interfaces

- Hive provides Command Line Interface(CLI) that accepts and parses HQL commands

- Hive provides JDBC/ODBC connector with other tools

- Hive provides a storage handler mechanism to integrate with HBase

- HUE(Hadoop User Experience) provides a unified web interface to HDFS and Hive in an interactive environment

- HCatalog provides metadata management system for Hadoop, Pig, Hive and MapReduce

HQL

- HQL consists of Data Definition Language, Data Selection and Scope Language, Data Manipulation Language, and Data Aggregation and Sampling Language

- Data Definition Language is used for creating, deleting, and altering schema objects like database tables, views, partitions, and buckets

- Data Selection and Scope Language is used for querying data, linking data, and limiting the data ranges or scopes

- Data Manipulation Language is used for exchanging, moving, sorting, and transforming data

- Data Aggregation and Sampling Language is used for exchanging, moving, sorting, and transforming data

Hive vs relational DBMS

Similarities

- tabular view of data objects in HDFS

- directories and files viewed as tables

- types of columns in table

- access to tables through HQL very similar to SQL

- API interface the same as JDBC programming interface

Differences

- load and read-only data management system based on implementation of HDFS

- it is still possible to access data visible in tabular format in Hive directly through HDFS

- UPDATE supported as coarse-grained transformation instead of fine-grained transformation in relational DBMSs

- No transaction processing system

- No verification of consistency constraints e.g.primary keys, forreign keys domains constraints

08 Hive Data Structures