UNIT-5
HDFS(Hadoop Distributed File System)

Prof. Vipul Gamit
PIET-MCA
Parul University

The Design of HDFS

- When a dataset outgrows the storage capacity of a single physical machine, it becomes necessary to partition it across a number of separate machines.
- Filesystems that manage the storage across a network of machines are called distributed filesystems.
- The Design of HDFS : HDFS is a filesystem designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware.
- Hadoop Distributed File System is the core component or you can say, the backbone of Hadoop Ecosystem.
- HDFS is the one, which makes it possible to store different types of large data sets (i.e. structured, unstructured and semi structured data).
- HDFS creates a level of abstraction over the resources, from where we can see the whole HDFS as a single unit.
- It helps us in storing our data across various nodes and maintaining the log file about the stored data (metadata).

HDFS Concepts

- Blocks:
  - HDFS is a block structured file system. Each HDFS file is broken into blocks of fixed size usually 128 MB which are stored across various data nodes on the cluster. Each of these blocks is stored as a separate file on local file system on data nodes (Commodity machines on cluster).
- Name Node:
  - Name Node is the single point of contact for accessing files in HDFS and it determines the block ids and locations for data access. So, Name Node plays a Master role in Master/Slaves Architecture where as Data Nodes acts as slaves. File System metadata is stored on Name Node.
  - File System Metadata contains majorly, File names, File Permissions and locations of each block of files. Thus, Metadata is relatively small in size and fits into Main Memory of a computer machine. So, it is stored in Main Memory of Name Node to allow fast access.
- Data Nodes:
  - Data Nodes are the slaves part of Master/Slaves Architecture and on which actual HDFS files are stored in the form of fixed size chunks of data which are called blocks.
  - Data Nodes serve read and write requests of clients on HDFS files and also perform block creation, replication and deletions.
- Blocks: HDFS has the concept of a block, but it is a much larger unit—64 MB by default. Files in HDFS are broken into block-sized chunks, which are stored as independent units.

Command Line Interface

- Some of the basic command-line commands are presented which are useful to interact with HDFS.
- Refer to practical

Data Ingest with Flume and Sqoop and Hadoop archives

- Sqoop and Flume both are meant to fulfill data ingestion needs but they serve different purposes. Apache Flume works well for streaming data sources that are generated continuously in Hadoop environment such as log files from multiple servers whereas where as Apache Sqoop works well with any RDBMS has JDBC connectivity.

What is Sqoop?

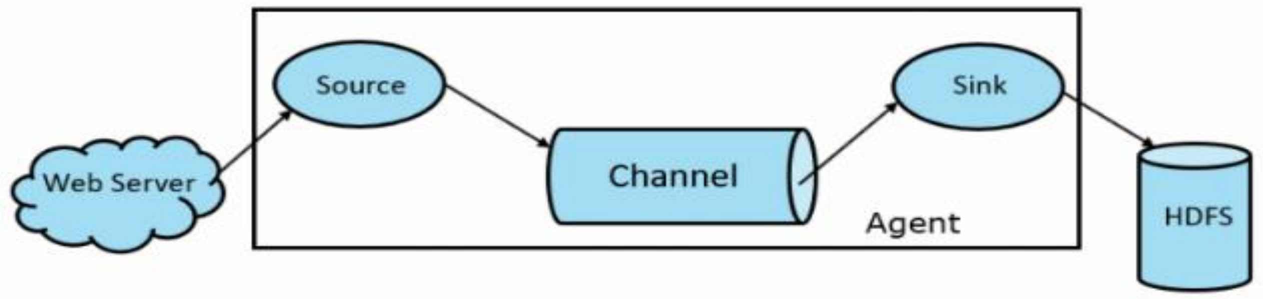
- This is how Sqoop got its name – “SQL to Hadoop & Hadoop to SQL”.
- Sqoop is used to import data from external datastores into Hadoop ecosystem’s tools like Hive & HBase.
- Sqoop is a command-line interface application for transferring data between relational databases and Hadoop.
- Why Sqoop?
  - For Hadoop developer, the actual game starts after the data is being loaded in HDFS. They play around this data in order to gain various insights hidden in the data stored in HDFS.

How does sqoop work?

- Steps:
- 1: Sqoop send the request to Relational DB to send the return the metadata information about the table(Metadata here is the data about the table in relational DB).
- From the received information it will generate the java classes (Reason why you should have Java configured before get it working-Sqoop internally uses JDBC API to generate data).
- Now Sqoop (As its written in java tries to package the compiled classes to be able to generate table structure) , post compiling creates jar file(Java packaging standard).

What is Flume?

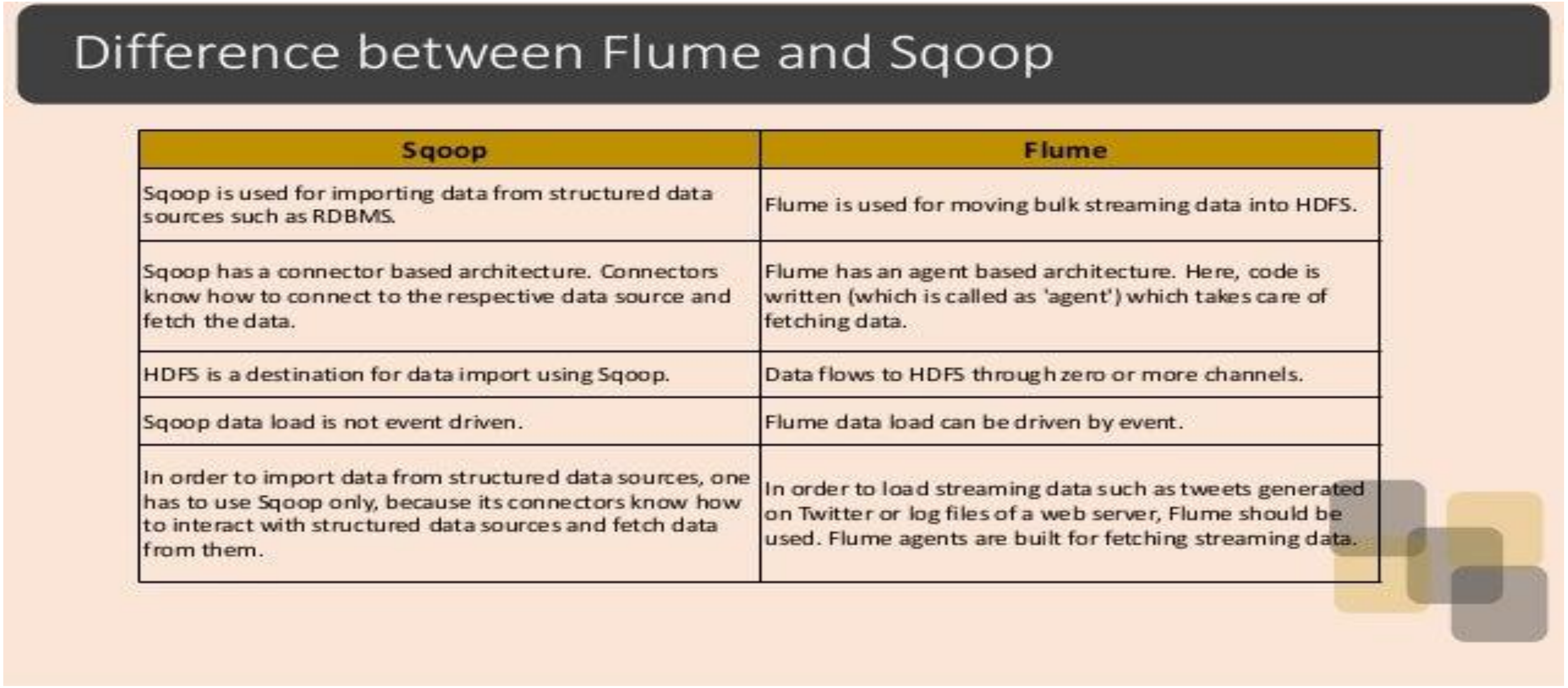
- Apache Flume is a tool/service/data ingestion mechanism for collecting aggregating and transporting large amounts of streaming data such as log data, events (etc...) from various webserves to a centralized data store.
- Apache Flume architecture is composed of the following components:
  - Flume Source: A Flume Source is present on Data generators like Face book or Twitter. Source collects data from the generator and transfers that data to Flume Channel in the form of Flume Events.
  - Flume Channel: An Intermediate Store that buffers the Events sent by Flume Source until they are consumed by Sink is called Flume Channel. Channel acts as an intermediate bridge between Source and Sink.
  - Flume Sink: A Flume Sink is present on Data repositories like HDFS, HBase. Flume sink consumes events from Channel and stores them to Destination stores like HDFS.
  - Flume Agent: A Flume agent is a long-running Java process that runs on Source – Channel – Sink Combination. Flume can have more than one agent.
  - Flume Event: An Event is the unit of data transported in Flume. The general representation of the Data Object in Flume is called Event.



How does Flume work?

- A Flume agent is a java process that consists of Source – Channel – Sink in its simplest form.
- Source collects data from data generator in the form of Events and delivers it to Channel. A Source can deliver to multiple Channels as per requirement.
- Fan out is the process where a single source will write to multiple channels so that they can deliver to multiple sinks.
- An Event is the basic unit of data being transmitted in Flume.
- Channel buffers the data until it is ingested by Sink.
- Sink collects the data from Channel and delivers it to Centralized data storage like HDFS or Sink can forward that events to another Flume agent as per requirement.
- Flume supports Transactions.
- In order to achieve Reliability, Flume uses Separate Transactions from Source to Channel and from Channel to Sink.
- If events are not delivered, then the transaction is rolled back and later redelivered.

Difference between Sqoop and Flume



Difference between Flume and Sqoop

Sqoop	Flume
Sqoop is used for importing data from structured data sources such as RDBMS.	Flume is used for moving bulk streaming data into HDFS.
Sqoop has a connector based architecture. Connectors know how to connect to the respective data source and fetch the data.	Flume has an agent based architecture. Here, code is written (which is called as 'agent') which takes care of fetching data.
HDFS is a destination for data import using Sqoop.	Data flows to HDFS through zero or more channels.
Sqoop data load is not event driven.	Flume data load can be driven by event.
In order to import data from structured data sources, one has to use Sqoop only, because its connectors know how to interact with structured data sources and fetch data from them.	In order to load streaming data such as tweets generated on Twitter or log files of a web server, Flume should be used. Flume agents are built for fetching streaming data.

Hadoop I/O: Compression

- Keeping in mind the volume of data Hadoop deals with, compression is not a luxury but a requirement. There are many obvious benefits of file compression rightly used by Hadoop. It economizes storage requirements and is a must-have capability to speed up data transmission over the network and disks. There are many tools, techniques, and algorithms commonly used by Hadoop. Many of them are quite popular and have been used in file compression over the ages. For example, gzip, bzip2, LZO, zip, and so forth are often used.

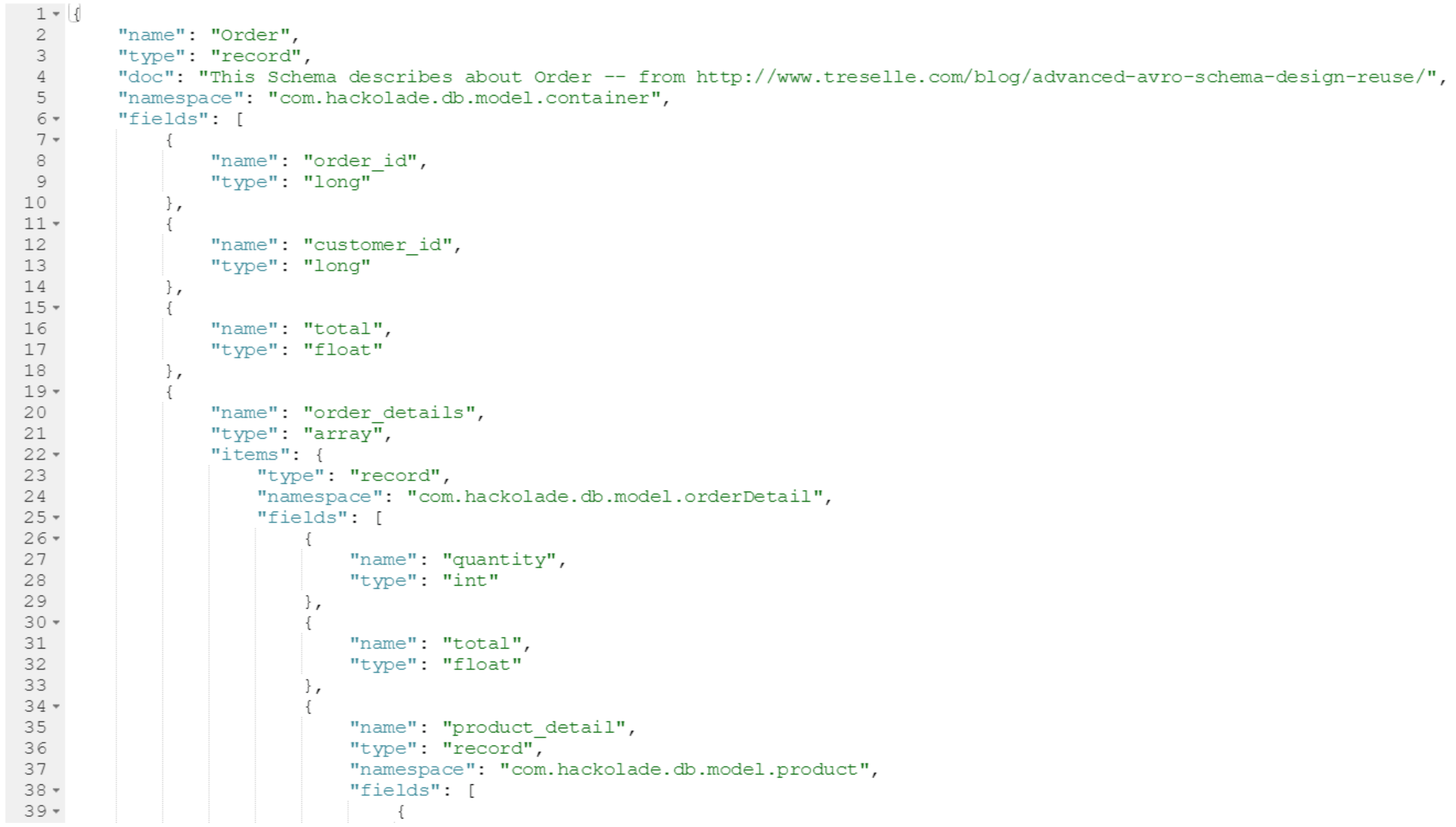
Serialization

- Data serialization is a mechanism to convert data (class objects, data structures) into a stream of bytes (binary form) in order to send it across network or store it persistently in a file or DB.
- The process that turns structured objects to stream of bytes is called serialization.
- This is specifically required for data transmission over the network or persisting raw data in disks.
- Deserialization is just the reverse process, where a stream of bytes is transformed into a structured object.
- This is particularly required for object implementation of the raw bytes.
- Therefore, it is not surprising that distributed computing uses this in a couple of distinct areas: inter-process communication and data persistence.
- Hadoop uses RPC (Remote Procedure Call) to enact inter-process communication between nodes. Therefore, the RPC protocol uses the process of serialization and deserialization to render a message to the stream of bytes and vice versa and sends it across the network. However, the process must be compact enough to best use the network bandwidth, as well as fast, interoperable, and flexible to accommodate protocol updates over time.
- Hadoop has its own compact and fast serialization format, Writables, that MapReduce programs use to generate keys and value types.

Avro and File-Based Data structures

- Apache Avro is a data serialization system native to Hadoop which is also language independent.
- Main features:
  - Avro is language independent
  - It is schema based
- Avro schemas are defined using JSON that helps in data interoperability.
- There are a couple of high-level containers that elaborate the specialized data structure in Hadoop to hold special types of data. For example, to maintain a binary log, the Sequence File container provides the data structure to persist binary key-value pairs. We then can use the key, such as a timestamp represented by LongWritable and value by Writable, which refers to logged quantity.
- There is another container, a sorted derivation of SequenceFile, called MapFile. It provides an index for convenient lookups by key.
- These two containers are interoperable and can be converted to and from each other.

- An Avro schema is defined in the JSON format and is necessary for both serialization and deserialization, enabling compatibility and evolution over time. It can be a: A JSON string, which contains the type name, like “int.” A JSON array, which represents a union of multiple data types.



```json
{
    "name": "Order",
    "type": "record",
    "doc": "This Schema describes about Order -- from http://www.treselle.com/blog/advanced-avro-schema-design-reuse/",
    "namespace": "com.hackolade.db.model.container",
    "fields": [
        {
            "name": "order_id",
            "type": "long"
        },
        {
            "name": "customer_id",
            "type": "long"
        },
        {
            "name": "total",
            "type": "float"
        },
        {
            "name": "order_details",
            "type": "array",
            "items": {
                "type": "record",
                "namespace": "com.hackolade.db.model.orderDetail",
                "fields": [
                    {
                        "name": "quantity",
                        "type": "int"
                    },
                    {
                        "name": "total",
                        "type": "float"
                    },
                    {
                        "name": "product_detail",
                        "type": "record",
                        "namespace": "com.hackolade.db.model.product",
                        "fields": [
                            {
```

What is the difference between Avro and JSON schema?

- Avro is a binary format that is more compact and efficient than JSON, and is more suitable for use in distributed systems because it supports schema evolution and is language independent. JSON is a text-based format that is more human-readable than Avro and is supported by many programming languages and frameworks.