Lecture Notes

Data Ingestion with Apache Sqoop and Apache Flume

In this module, you will learn data ingestion for structured and unstructured data. You will first be introduced to data ingestion. After this, you will learn about Apache Sqoop, which is a tool for ingesting structured data, and Apache Flume, which is a tool for ingesting unstructured data. You will first learn about the entire data ingestion process, the various challenges associated with designing such systems, the key steps of the data ingestion process, and the different tools used in the process. You will also learn about the different types of data and the file formats that you typically deal with during the process of data ingestion. Thereafter, you will learn more about Apache Sqoop and its architecture. You will then see some case studies to understand the various concepts and arguments of Sqoop. In the following session, you will learn more about Apache Flume, its components, characteristics, and some of its use cases. You will then be introduced to a log collection case study wherein you will use the concepts of Flume to solve the problems in it. Finally, you will learn about tuning Flume and understand the various differences between Flume and Sqoop.

Data Ingestion

Data ingestion is referred to as the process of absorbing data for immediate use or storage. It is a bridge to transfer data from the source to the destination, such as HDFS, where it can be used efficiently.

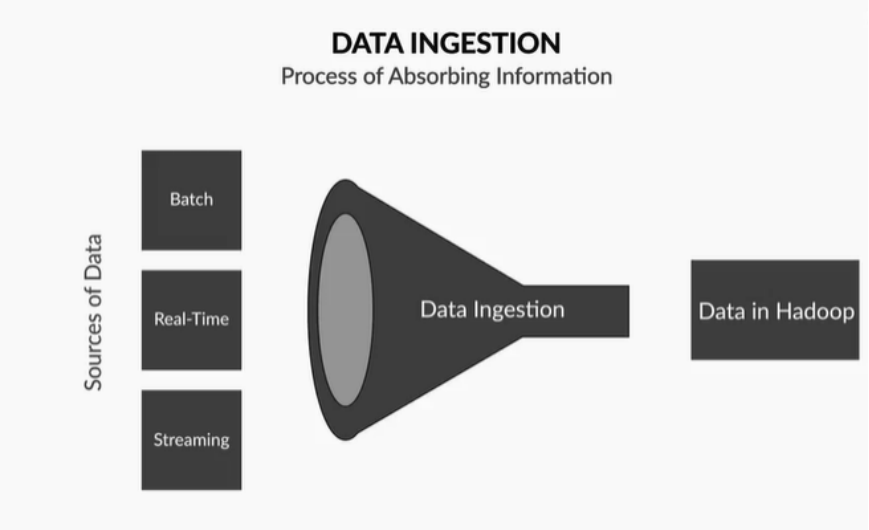

Figure 01: The Process of Data Ingestion

Data ingestion is referred to as the 'Extract' step in an ETL/ELT process. In a big data scenario, there are multiple sources of data, and each source can generate different types of data. For example, your daily transaction data derived from banking, online orders, etc., are structured and, hence, stored in a normalised relational database. In contrast, the data generated from random, daily activities, such as web surfing and social media activity, are usually stored as weblogs (unstructured data) in a web server. In an

organisation with multiple business verticals, the data must be collated into a single centralised data repository before it is analysed. This is exactly why you require specialised tools and technologies for data ingestion.

Data can be classified into three different types:

- Batch
- Real-time
- Streaming

Various specialised tools and technologies are used for the ingestion of the different types of data.

Challenges in Data Ingestion

Over the past decade, the amount of data generated has increased exponentially. In fact, the International Data Corporation (IDC) has projected that by 2025, the total data generated all over the world will be approximately 175 zettabytes (ZB). To provide some context, 1 zettabyte is equal to 1 billion terabytes or 1 trillion gigabytes.

There are several challenges in the process of data ingestion. Data can come from numerous sources and in a plethora of formats. So, it is a challenge for businesses across the globe to ingest data efficiently and process it at a reasonable speed in order to get business insights. Moreover, data nowadays is being generated at a rapid pace, resulting in humongous volumes of data. With such colossal volumes of data being generated at such a high speed , businesses find it challenging to ingest data and process it quickly so as to get the desired analysis at the required time.

Moreover, to generate any business insights, you have to prioritise the sources of data you want to analyse, filter out data that you do not want, and set up a system that allows you to draw conclusions from this data.

Along with the above-mentioned challenges, there are several network-related challenges in a typical production setup for data ingestion.

In the recent past, several data ingestion tools such as Sqoop, Flume, Kafka, Gobblin have been developed , and these can help address these challenges and generate business insights.

Key Steps of Data Ingestion

Data ingestion involves the following key functions:

1. **Data collection**: For conducting any analysis, you have to first collect data. This is where you analyse the sources from where the data has to be imported, based on the requirement, out of the many resources available.

2. **Data validation**: Once you prioritise the resources and collect the data, you have to validate it so that the unwanted data can be filtered out.

3. **Data routing**: Next, you have to route the validated data to its specified destination such as HBase, HDFS, Hive, or some other system, where it will be further analysed.

Once data is successfully imported, validated, and routed to its specified destination, data processing tools are run on the data to get the desired output. You may also use business intelligence (BI) and business analytics (BA) tools to get meaningful insights.

Tools for Data Ingestion

Data can come from multiple sources. We have structured data coming from RDBMS, multi-structured data coming from social media, streaming data and real-time data coming from back-end servers, weblogs, etc.

To ingest data from all such sources, the following commands and tools can be used:

1. **File transfer using commands**: It uses commands such as 'put' to copy files from a local file system to HDFS and 'get' to copy files from HDFS back to the local file system. For data transformation, however, you cannot depend on this method. There is another command 'Distcp' which can be used to copy large data sets between two Hadoop clusters.
2. **Apache Sqoop**: Sqoop is the short form for SQL to Hadoop. It is used for importing data from RDBMS to Big Data Ecosystem (Hive, HDFS, HBase, etc.) and exporting the data back to RDBMS after it is processed in the Big Data Ecosystem. It was created by Cloudera and then was open sourced.
3. **Apache Flume**: Flume is a distributed data collection service for collecting, aggregating, and transporting large amounts of real-time data from various sources to a centralised system, where the data can be processed. It was released as an open-source service.
4. **Apache Kafka**: Kafka is a fast, scalable distributed system that can handle high volumes of data. It enables programmers to pass messages from one point to another. Apache Kafka was developed by LinkedIn and later, it became an open-source service.
5. **Apache Gobblin**: Gobblin is an open-source data ingestion framework for extracting, transforming, and loading a large volume of data from different data sources. It supports both streaming and batch data ecosystems. Gobblin is LinkedIn's Data Ingestion Platform.

Apart from the tools you have seen till now, tools such as Apache Storm, Apache Chukwa, and Apache Spark are also used for ingesting data as per your requirement.

Types of Data and File Formats

Since data can be of any type and choosing a particular tool for data ingestion depends a lot on the type of data that you are going to ingest, let's have a quick refresher on the types of data.

Data can be categorised as follows:

1. **Structured**: It is organised and can be stored in databases, i.e. in tables with rows and columns. Therefore, it has the advantage of being entered, stored, queried, and analysed efficiently using Structured Query Language (SQL). In other words, structured data can be read easily by machines, e.g., Aadhaar data, financial data, metadata of files, etc.
2. **Unstructured**: In simple terms, you can describe unstructured data as a complement to structured data, i.e. it cannot be easily organised and stored in databases. However, this data may be stored and retrieved using NoSQL databases. Its examples include images, audio, video, chat messages,

etc., which are usually generated and consumed by humans. Unsurprisingly, the vast majority (~80%) of data being created in today's world is unstructured.

3. **Semi-structured**: There is no predefined schema for semi-structured data. In terms of readability, it sits between structured and unstructured data. XML and JSON files are examples of semi-structured data. Emails are also an example of semi-structured data since they contain fields such as 'From', 'To', 'Subject', and 'Body'. It maintains internal tags and markings that identify separate data elements, which enables information grouping and the creation of hierarchies among the elements. However, this schema does not constrain the data as in an RDBMS, e.g., the 'Subject' and 'Body' fields may contain text of any size, making it difficult for machines to read them.

Each type of data can be stored in a wide variety of file formats, and each format has its advantages and disadvantages. When it comes to data ingestion, file formats play a crucial role. A file format represents the way in which an information is stored or encoded in a computer. Choosing a particular file format is important if you want to have maximum efficiency in terms of factors such as processing power, network bandwidth, and available storage. A file format directly affects the processing power of the system ingesting the data, the capacity of the network carrying the data, and the available storage for storing the ingested data. Following are some of the widely used file formats:

1. **Text/CSV**: Comma-Separated Values, also called CSV, is the most commonly used file format for exchanging large data between Hadoop and external systems. A CSV file has very limited support for schema evolution and it does not support block compression. Also, CSV files are not compact.
2. **XML and JSON**: XML stands for Extensible Markup Language, which defines a set of rules using which documents can be encoded in a format that is both machine-readable and human-readable. JSON stands for JavaScript Object Notation and is an open-standard file format consisting of key-value pairs. Since both are text files, they do not support block compression and are not compact. Splitting is very tricky in these files as Hadoop does not provide a built-in InputFormat for either. Since splitting is tricky, these files cannot be split easily for processing in parallel in Hadoop.
3. **Sequence Files**: These are binary files and store data as binary key-value pairs. Binary files are files stored in the binary format and, hence, are more compact than text files. In a binary file, each byte has 256 possible values as opposed to pure unextended ASCII which only has only 128 possible values, making a binary file twice as compact. Sequence files support block compression and can be split and processed in parallel due to which they find extensive use in MapReduce jobs.
4. **Avro**: This is a language-neutral data serialisation system developed within Apache's Hadoop project. Serialisation is the process of turning data structures into a format that can be used for either storage or transmission over a network. Being language-neutral means Avro files can be easily read later, even from a language different from the one used to write the file. Avro files are self-describing, compressible, and splittable, making these suitable for MapReduce jobs as they can be split and processed in parallel. These are binary files and, hence, more compact than text files. Avro also supports schema evolution, which means that the schema used to read the file does not have to match the schema used to write the file.

Based on your requirement, you may choose the appropriate file format for storing data.

Schema Evolution: Let's use an example here to understand schema evolution. Suppose, you are working on a particular schema of a database in a software company. Now, your client requests you to update the schema based on some requirements. In such a situation, can you go ahead and directly update the schema? Well, No! This is because there will be other applications running and generating data based on the current schema. Now, if you just update the schema without considering other applications, then all the currently running applications will be affected. Hence, you need to evolve the schema in such a way that it caters to the new requirements as well as to the existing applications. This is schema evolution.

Block Compression: It is the process of compressing each individual block. For instance, as Hadoop stores large files by splitting them into blocks, it will be best to compress the individual blocks.

Introduction to Sqoop and its Architecture

Apache Sqoop, short for 'SQL to Hadoop', is used for ingesting relational data. Sqoop provides efficient, bidirectional data transfer between Hadoop and relational databases in parallel. The data can be imported directly in the HDFS or to HBase or Hive tables as per the use case.

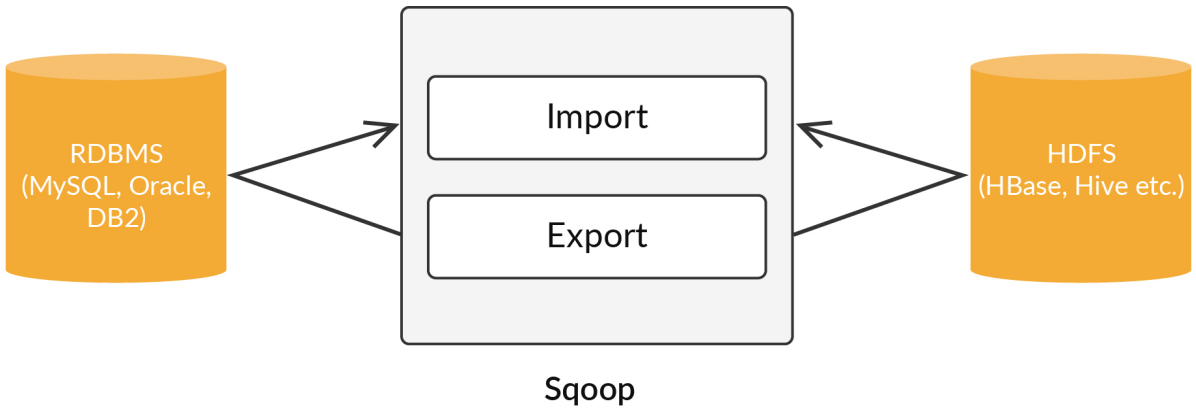

Figure 02: Bidirectional Data Transfer in Apache Sqoop

When you use Sqoop to transfer data, the dataset is split into multiple partitions and a map-only job is launched. Individual mappers are then responsible for the transfer of each slice/partition of the dataset. The metadata of the database is used to handle each data record in a type-safe manner.

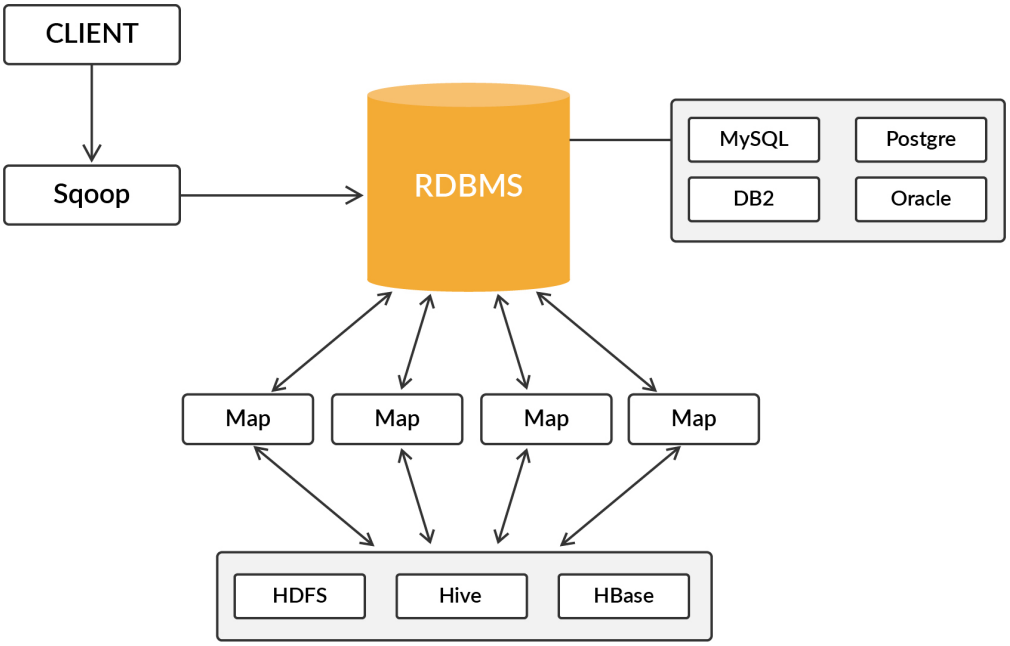

Figure 03: Architecture of Apache Sqoop

Once Sqoop is connected to the database, it uses JDBC to examine the table to be imported by retrieving a list of all the columns and their SQL data types. The SQL data types (integer, varchar, etc.) can be mapped to Java data types (Integer, String, etc.). Sqoop has a code generator which creates a table-specific Java class to hold the extracted records from the table by using the information given by the JDBC about the data types. Sqoop then connects to the cluster to submit a MapReduce job using the Java class generated. The dataset being transferred is split into multiple partitions and a map-only job is launched. The output of this is a set of files containing the imported data. Since the import process is performed in parallel, the output is achieved in multiple files.

Apache Sqoop offers several advantages:

- It provides stable and reliable interactions.
- It moves bulk data between Hadoop and RDBMSes.
- It is cost-efficient.

Tuning Sqoop

While working with Sqoop in the IT industry, especially in the production environment, password is not passed as a parameter along with the Sqoop command. Instead, the password is stored in a text file with read-only permissions to the ID through which the Sqoop command is running in production.

If the data to be imported is of large size, then Sqoop takes a long time to transfer data between RDBMS and Hadoop. In such a scenario, you can tune Sqoop using boundary queries and mappers to fetch data quickly.

Moreover, there are some network latency issues when data transfer takes place between two networks in different zones. Here, since the RDBMS system is on-premise and Hadoop is on cloud, network latency will also be a factor that slows down the transfer of data.

There is a concept of availability zones which can be used to reduce network latency.

An Introduction to Apache Flume

Apache Flume is a tool used for ingesting unstructured data. According to the Flume user guide, Apache Flume is a distributed, reliable, and available system for efficiently collecting, aggregating, and moving large amounts of log data from many different sources to a centralised data store. Flume is also used for ingesting massive quantities of event data such as network traffic data, data generated from social media, data from email messages, and other sources. All of this is unstructured data and ingesting these has its own challenges.

The first challenge is an overwhelmed network. Unstructured data is getting generated at a very rapid rate; therefore, ingesting it at the same rate is difficult since the number of machines available for ingesting data is generally less than the number of machines generating data. Hence, when many servers generating data try to write data to a small cluster of machines ingesting data, it leads to overwhelming of the network. Here, Flume acts as an intermediate system (buffer) between the data production source and the target Hadoop system.

The second challenge is latency issues. Servers are scattered across multiple geographic locations, but they all try to write data to a centralised Hadoop system. The servers which are at a far geographic location will not be able to write data quickly due to a network lag which is called latency. Hence, you need a system that is extendable to such far-off locations. Flume is extendable in such scenarios, which means that you can configure Flume once and extend it to anywhere with the same configuration. It will work in the same way at all locations.

The third challenge is the loss of data in the network due to network-related issues. So, you need to ensure that you have a fault-tolerant system (keeps account of the data sent and received), which Flume is.

A Flume agent is a Java application that generates or receives data and buffers it until it is written to the next Flume agent or a storage system. A chain of Flume agents can be used to move data from some data sources to the target HDFS or HBase in a scalable and durable manner.

Components of Flume

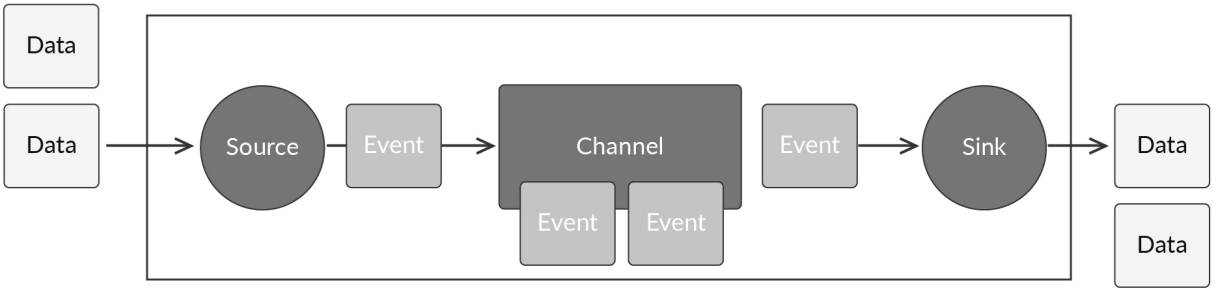

Figure 04: Components of Apache Flume

A Flume agent has various components:

- Event:
  - An event can be considered equivalent to a data structure that holds data. It consists of the following elements:
    - Header: A header contains the key-value pairs that are used for showing routing information or other structured information.
    - Body: The body contains the actual data, which is an array of bytes.
  - Data flows in the form of these events only.

- Source:
  - Sources are those components of a Flume agent that receive data from any application that produces the data.
  - They receive the data by listening to a port or the file system.
  - All sources are connected to at least one channel.
  - Sources write data into channels in the form of events.
  - Flume supports different types of sources such as an Avro source and an HTTP source.

- Sinks:
  - In a Flume agent, sinks deliver the data to the final destination.
  - A sink continuously polls the connected channel to retrieve the events that were written by the source in the channel.
  - A sink writes events into the next hop (Flume agent) or the final destination.
  - Once it has successfully written the events into the next destination, it informs the channel to remove the written events.
  - Flume supports different types of sinks such as an Avro sink or an HDFS sink.

- Channel:
  - Channels act as a conduit between the source and the sink.
  - It is a buffer (in-memory queue) that keeps events till the sink writes them into the next hop or the target destination.
  - Multiple sources can write events into the same channel and multiple sinks can read events from the same channel. However, a sink can be connected to only one channel.
  - Flume supports different types of channels such as the JDBC or Kafka channel.

A Flume agent can have more than one source, channel, or sink. Apart from these, you need to take into account other components of a Flume agent.

Now, if the available sources, sinks, and/or channels are not appropriate for your particular use case, then you can customise them as per requirement and build a custom source, sink, and/or channel.

Flume has a plethora of characteristics which makes it stand apart and of immense use in the industry. Flume is fault tolerant, scales by itself, and extensible. The following points will help you understand it better. The data might get lost in the network due to network-related issues. So, you must ensure you have a fault-tolerant system (to keep an account of the data sent and received). Flume serves this purpose well.

There will be situations where the rate of data generation will increase manifold. To handle such sudden spikes in data volumes, Flume has the ability to scale horizontally as the machine is equipped with the necessary hardware. Flume also uses the Producer-Consumer model to handle such transient spikes in data generation.

Servers are scattered across multiple geographic locations, but they all try to write data to a centralised Hadoop system. The servers at a far geographic location will not be able to write data quickly due to a network lag which is called latency. Hence, you need a system that is extendable to such far-off locations. Flume is extendable in such scenarios, which means that you can configure Flume once and extend it to anywhere you want with the same configuration, and it will work in the same way at all locations.

Some industry use cases of Flume include recommendation systems and sentiment analysis using Twitter.

Today, Flume is mostly used for log collection, and its use case is limited to data streaming. Other tools such as Apache Spark, Apache Kafka, and Apache Flink are preferred over Flume for the overall ingestion of unstructured data.

Flume Configuration Files

A Flume configuration file contains information about the Flume agent to be created. It also contains information regarding the sources, sinks, and channels and how they are bound together. Essentially, it is used for defining a Flume agent.

The generic template of a Flume configuration file is given below.

```
#list sources, sinks and channels in the agent
<Agent>.sources = <Source>
<Agent>.sinks = <Sink>
<Agent>.channels = <Channel1> <Channel2>

# define the flow
<Agent>.sources.<Source>.channels = <Channel1> <Channel2>
<Agent>.sinks.<Sink>.channel = <Channel1>

# source properties
<Agent>.sources.<Source>.<someProperty> = <someValue>

# sink properties
<Agent>.sinks.<Sink>.<someProperty> = <someValue>
```

```
# channel properties
<Agent>.channels.<Channel>.<someProperty> = <someValue>
```

Flume Flows

A Flume flow represents the path taken by the data from its source to reach the target destination using Flume agents.

Different types of Flume flows can be used for different types of data ingestion tasks. Flume flows can also be used independently or used in combination with other Flume flows for solving complex data ingestion problems. Following are some of the various types of Flume flows:

1. **Multi-agent flow**: In these flows, multiple Flume agents are used. These flows are preferred when the rate of data generated is high. Multiple agents can be connected in a series-like configuration, wherein the sink of one agent is connected to the source of another agent. In order to set up a multi-agent flow, the avro/thrift sink of the first hop should be pointing towards the avro/thrift source of the next hop. These flows will enable the first Flume agent to forward events to the next Flume agent.
2. **Fan-out flow**: In these flows, multiple channels are connected to the same source. These can be either multiplexing or replicating in nature. The channel selector property can be used for configuring these flows. In the replicating flows, a particular event is sent to all the configured channels, whereas in the multiplexing flows, the event is sent to only one subset of qualifying channels. To fan out the flow, you need to specify a list of channels for a source and the policy for fanning it out. You can do this by adding a channel 'selector' that could be either replicating or multiplexing.
3. **Tiered data collection flow**: These flows can be used for configuring multiple Flume agents such that they can receive data from the initial sources and consolidate the data into fewer agents that can finally dump the data into the final sink. This basically sets up multiple levels/tiers of Flume agents, thereby reducing the number of direct connections to the final source. These flows are useful in production setups. For example, these flows can be used for log collection, wherein logs are collected from multiple sources and then consolidated at a single destination. You also used the tiered data collection flow in the following case study.

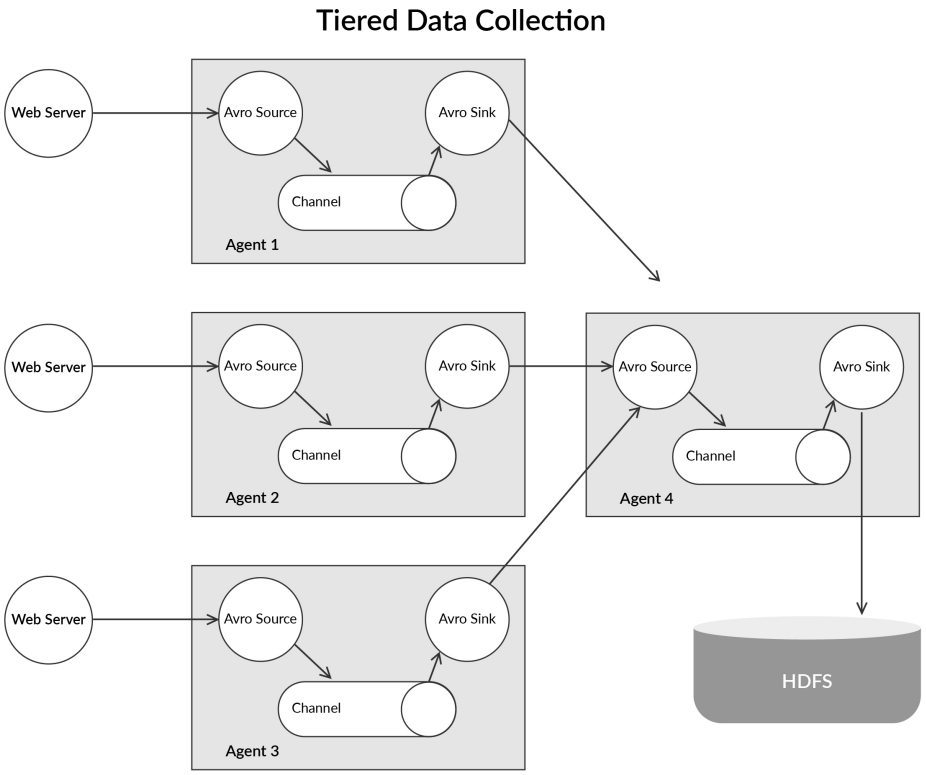

Figure 05: Tiered Data Collection Flow

For this case study, you can use a tiered data collection flow. These flows are used for the purpose of consolidation of data. In these flows:

1. Data is received from multiple web servers simultaneously;
2. The data is first received by the Avro source-sink pair (Flume agents) corresponding to each web server;
3. Next, the data is consolidated into another agent that can write the data into the final store, which, in our case, is the HDFS; and
4. The objective of applying a multi-agent approach is to reduce the number of direct connections to the final source so that it is not overburdened.

Tuning Flume

Flume can be tuned for better performance. Let's now look at some techniques to tune Flume's performance.

There are many different types of channels and choosing the right channel while designing the Flume flow is important. The two most commonly used channels are the file and memory channels.

The file channel is more durable because it continues to transfer all the events to disk. Thus, even if the JVM is killed or the operating system crashes or reboots, events that were not successfully committed will still be there in the file channel and will resume transferring once the Flume agent is restarted. On the

other hand, the memory channel is volatile as it buffers events in RAM. This means if the Java process is killed, then the events are lost locally and must be retransmitted from the previous stages in the pipeline, thus leading to increased latencies.

Since the memory channel maintains events in RAM, it has lower latencies compared to the file channels. However, the number of events that can be stored in memory is limited and memory channels have lower buffering capability compared to the file channel which utilises the hard disk space. You must keep all these factors in mind while choosing a channel type for a use case. A combination of both the channels across a Flume flow is also possible.

The next important factor for better performance is tuning of the batch size. Batch size refers to the maximum number of events that will be consumed from a channel in a single transaction. Batch size directly affects throughput, latency, and duplication under failure. Throughput is the rate of messages delivered successfully over the channel. Latency is the time interval between the generation of an event and the acknowledgement that the event is successfully transferred. Duplication refers to resending of the events in case of a failure. In an ideal scenario, you would want high throughput, low latency, and low duplication.

Since a small batch size consumes a lesser number of events per transaction, it results in low throughput. However, this also means low latency and low risk of duplication of events. Small chunks of events are consumed faster and, in case of a failure, smaller batches lead to smaller resends. With a larger batch size, you do get high throughput, but you see increased latency and increased risk of duplication. Here are some suggestions to follow while choosing a batch size.

Start with a sink batch size equal to the sum of the batch sizes of the input sources. For example, if there are 10 input streams with a batch size of 100 each, then set the batch size of the current sink to 1,000. If the batch size of the current sink is too high, then consider adding another sink to balance the load. For instance, if the batch size of the current sink is 20,000, you can use two sinks with a batch size of 10,000 each.

You should opt for the lowest batch size that gives an acceptable throughput and monitor the channel state to get more insights.

Other important tuning parameters are channel capacity and a channel's transaction capacity. The channel capacity refers to the maximum number of events that the channel can hold at a given time. The transaction capacity refers to the maximum number of events that a channel accepts or sends in one transaction. Typically, it should be set to the value of the largest batch size that can be used to store or remove events from the channel. A channel's transaction capacity must always be lower than the channel capacity.

Sqoop vs Flume

Now that you are familiar with both Sqoop and Flume, let's compare and summarise the features of both tools. This will help you determine which tool to be applied in which use cases.

Sqoop is used to import data from RDBMS to HDFS, HBase, and Hive and also to export data from HDFS to RDBMS. It is a tool for ingesting structured data. On the other hand, Flume is a distributed data collection service for collecting, aggregating, and transporting large amounts of real-time data from various sources to a centralised location where the data can be processed. It is a tool for ingesting unstructured data.

Sqoop is not event-driven, which means that its functioning is dependent on events and, thus, suitable for moving data from and to RDBMS, such as Oracle, MySQL, among others. On the other hand, Flume has an event-based architecture, which means it is dependent on events. The events include tweets generated on Twitter, log files of a server, etc.

Sqoop has a connector-based architecture, which means the JDBC connector is primarily responsible for connecting with the data sources and fetching data in a corresponding manner. In contrast, Flume has an agent-based architecture, which means the Flume agent is responsible for the data transfer.

Sqoop can transfer data in parallel for better performance, while Flume scales horizontally, and multiple Flume agents can be configured to collect high volumes of data. Flume also has several recovery and failover mechanisms due to which it is highly reliable.

You have now learnt to determine when to use which tool. You have also learnt about use cases in which Sqoop and Flume are typically used.

Sqoop in a Typical Production Architecture

Several factors are involved in building a connection between RDBMS and Hadoop for transferring data via Sqoop.

A number of network challenges are faced in the production setup, and resolving these challenges is important in the successful transfer of the data. Once those network challenges are addressed, Sqoop can transfer data from the RDBMS system to Hadoop.

If the data to be imported is very large in size, then Sqoop takes a long time to transfer data between RDBMS and Hadoop. In such a scenario, you can tune Sqoop using boundary queries and mappers to fetch data quickly.

Moreover, there are network latency issues when data transfer takes place between two networks in different zones or geographic locations. Here, since the RDBMS system is on-premise and Hadoop is on cloud, network latency is also a factor that slows down the transfer of data.

There is a concept of availability zones using which you can use to reduce network latency.

Disclaimer: *All content and material on the upGrad website is copyrighted material, belonging to either upGrad or its bona fide contributors, and is purely for the dissemination of education. You are permitted to access, print, and download extracts from this site purely for your own education only and on the following basis:*

- *You can download this document from the website for self-use only.*
- *Any copies of this document, in part or full, saved to disc or to any other storage medium, may be used for subsequent, self-viewing purposes or to print an individual extract or copy for non-commercial personal use only.*
- *Any further dissemination, distribution, reproduction, and copying of the content of the document herein, or the uploading thereof on other websites, or use of the content for any other commercial/unauthorized purposes in any way that could infringe the intellectual property rights of upGrad or its contributors is strictly prohibited.*
- *No graphics, images, or photographs from any accompanying text in this document will be used separately for unauthorized purposes.*
- *No material in this document will be modified, adapted, or altered in any way.*
- *No part of this document or upGrad content may be reproduced or stored on any other website or included in any public or private electronic retrieval system or service without upGrad's prior written permission.*
- *Any rights not expressly granted in these terms are reserved.*