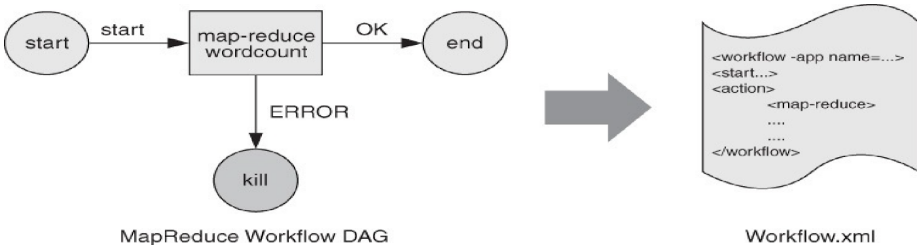
Internal Assessment Test 2 – Nov 2023

Sub:	Big Data Analytics					Sub Code:	18CS72	Branch:	CSE
Date:	1/12/2022	Duration:	90 mins	Max Marks:	50	Sem / Sec:	7 –A/B/C		OBE

	Scheme and solution	MARKS	CO	RBT
1	How will you consider MongoDB as a complete data store which imbibes highly functional secondary indices and provides a powerful framework for data analysis? Enlist business use cases for use of MongoDB. MongoDB as powerful framework: justification – 6 marks At least four use cases- 4 marks	[10]	CO3	L4
2	Write queries in Cassandra for CURD operations by taking the ProductInfo table. Consider suitable columns and decide the partition key. Table structure and partition key: 2 marks 4 queries- each 2.5 marks	[10]	CO3	L4
3	Explain ORC file format with features by taking an automatic chocolate vending machine (ACVM) example. Explain Parquet file format with suitable diagram. ORC file format and ACVM structure - 3+2 marks Parquet file format and diagram – 4+1 marks	[10]	CO1	L3
4	Explain MapReduce execution steps with suitable examples. MapReduce with all steps explanation – 6 marks Diagram – 2 marks Example -2 marks	[10]	CO 4	L3
5	Identify and explain the context. Explain the below diagram in detail.  MapReduce Workflow DAG Workflow.xml Identification and explanation of the Context- 4 mark Diagram node explanation - 6 marks	[10]	CO 2	L4
6	Explain working of Flume agent. Explain the importance of AVRO in pipelining Flume. Flume agent component – 4 marks AVRO pipeline with diagram- 6 marks	[10]	CO2	L2

Solution :

Q1.How will you consider MongoDB as a complete data store which imbibes highly functional secondary indices and provides a powerful framework for data analysis? Enlist business use cases for use of MongoDB.

Features of MongoDB:

1. MongoDB

data store is a physical container for collections. Each DB gets its own set of files on the file system . A number of DBs can run on a single MongoDB server. The database server of MongoDB is mongod and the client is mongo

2.
Collection stores a number of MongoDB documents. It is analogous to a table of RDBMS . A collection exists within a single DB to achieve a single purpose. Documents of the collection are schema less . Thus, it is possible to store documents of varying structures in a collection
3
. Document model is well defined . Document is the unit of storing data in a MongoDB database. Insert , update and delete operations can be performed on a collection . Documents have dynamic schema
4
. MongoDB is a document data store in which one collection holds different documents . Number of fields, content and size of the document can differ from one document to another.
5
. Storing of data is flexible, and data store consists of JSON like documents . This implies that the fields can vary from document to document and data structure can be changed over time ; JSON has a standard structure, and scalable way of describing hierarchical data
6. Storing of documents on disk is in BSON serialization format. BSON is a binary representation of JSON documents. The mongo JavaScript shell and MongoDB language drivers perform translation between BSON and language specific document representation.
7.
Querying , indexing, and real time aggregation allows accessing and analyzing the data efficiently
8. Deep
query ability Supports dynamic queries on documents using a document based query language that's nearly as powerful as SQL.
9.
No complex Joins
10. Distributed
DB makes availability high, and provides horizontal scalability
11.
Indexes on any field in a collection of documents : Users can create indexes on any field in a document. Indices support queries and operations . By default, MongoDB creates an index on the _id field of every collection.

12.
Atomic operations on a single document can be performed even though support of multi document transactions is not present . The operations are alternate to ACID transaction requirement of a relational DB.
13
. Fast in place updates: The DB does not have to allocate new memory location and write a full new copy of the object in case of data updates . This results into high performance for frequent update use cases. For example , incrementing a counter operation does not fetch the document from the server. Here, the increment operation can simply be set
14
. No configurable cache: MongoDB uses all free memory on the system automatically by way of memory mapped files . The most recently used data is kept in RAM . If indexes are created for queries and the working dataset fits in RAM, MongoDB serves all queries from memory.
15.
Conversion/mapping of application objects to data store objects not

needed

MongoDB Use Case 1: Product Data Management
MongoDB is perfect for Product Data Management. It enables product data and related information to be managed and processed in a single, central system. This allows for Detailed Cost Analysis, Increased Productivity, and Improved Collaboration.

MongoDB Use Case 2: Operational Intelligence
Another real-world MongoDB use case is Operational Intelligence, as it aids in Real-time Decision-making. It allows companies to seamlessly gather various data feeds representing their ongoing business operations and information of related external factors. They can then analyze these feeds as the data arrives for developing profitable and functional business strategies.

MongoDB Use Case 3: Product Catalog
Product catalogs have been in existence for years in the ever-evolving digital space. However, with the rapid evolution in technology, product catalogs sometimes feel like a new digital experience. This is because the richness and volume of data feed product catalogs' interactions today are remarkable.

MongoDB is useful in such applications, as it provides an excellent tool for storing different types of objects. In addition, its Dynamic Schema Capability ensures that product documents only contain attributes relevant to that product.

MongoDB Use Case 4: Scaling And Application Mobility
For most Mobile Application Development, it is expected that the companies involved will have to deal with different data structures from several sources and potentially highly dynamic growth. Interestingly, MongoDB provides High Flexibility and Scalability that serves as an excellent database solution for such challenges. In addition, it allows developers to focus on developing a better customer experience, instead of spending time adjusting the database.

MongoDB Use Case 5: Customer Analytics

Q2. Write queries in Cassandra for CRUD operations by taking the ProductInfo table. Consider suitable columns and decide the partition key.

Cassandra CURD Operations: (CURD
Create, Update, Read and Delete data into tables)
(a)
Insert Command: INSERT command creates data in a table:
INSERT INTO <
tablename > columnl name>, <column2 name>....) VALUES (< valuel >, <value2> ....) USING <option
(
b) Update Command: UPDATE command updates data in a table. The following keywords are used while updating data in a table:
Where This clause is used to select the row to be updated. Set Set the value using this keyword.
Must
Includes all the columns composing the primary key. If a given row is unavailable, then UPDATE creates a new row.
UPDATE <
tablename > SET <column name>= <new value > column name>= <value>.... WHERE <condition>

[A WHERE clause can be used only on the columns that are a part of primary key or have a secondary index on them.]
(
c)Select Command : SELECT command reads the data from a table. The command can read a whole table, a single column, or a
particular cell:
SELECT <column name(s)> FROM <Table Name>
To select all
records: SELECT * FROM <Table
To select records that fulfils required condition:
SELECT <
columnl , column2,..> FROM <Table Name> where <
Example: Select Product Type, Product Id, Product Name, and Product Cost of Product whose Productid is 31047:
SELECT Product Type, Product Id, Product Name, and Product Cost
from
Productinfo where Productid

d) Delete Command : DELETE command deletes data from a table: 31047;
DELETE FROM <identifier> WHERE <condition>; Example: Delete row from a table where Product id is 31047: DELETE FROM
Productinfo WHERE Productid = 31047
(5) Creating a Table with List
CREATE Table command is used for creating a table with a list.
The following query creates a table with two columns, one is the primary key and the other has multiple items (List):
CREATE TABLE data (<column name>, <data type> PRIMARY KEY, <column name list<data type>);
Example : Create a sample table
Contactlnfo with three columns: Sno , name
and
Emailld . To store multiple Email Ids, use a
create table
Contactinfo Sno int Primary key, Name text, emailid list <text>);
(
6) Insert Command for inserting data into a list
INSERT Command also inserts data into a list. To insert data into the elements in a list, enter all the values separated by a comma within square braces []:
INSERT
INTO <table name> columnl column2,) VALUES (valuel , value2, [list valuel ,
list value2 , ...]
Example: Insert data of three persons into the
Contactlnfo Table
Insert
into ContactInfo values (Sno , EmailId) (1 , 'rahul@yahoo.
'Rahul', ['rahul@gmail.com',

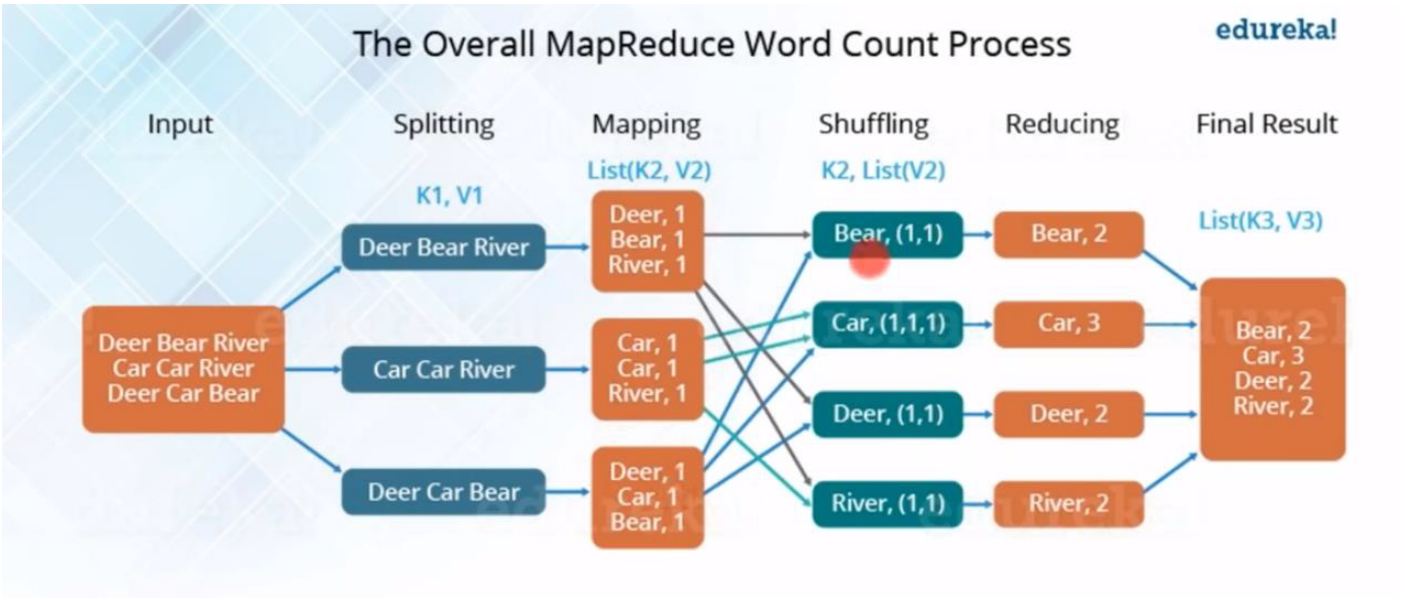
Q3 Explain ORC file format with features by taking automated chocolate vending machine (ACVM) example. Explain Parquet file format with suitable diagram.
ORC file format and ACVM structure - 3+2 marks

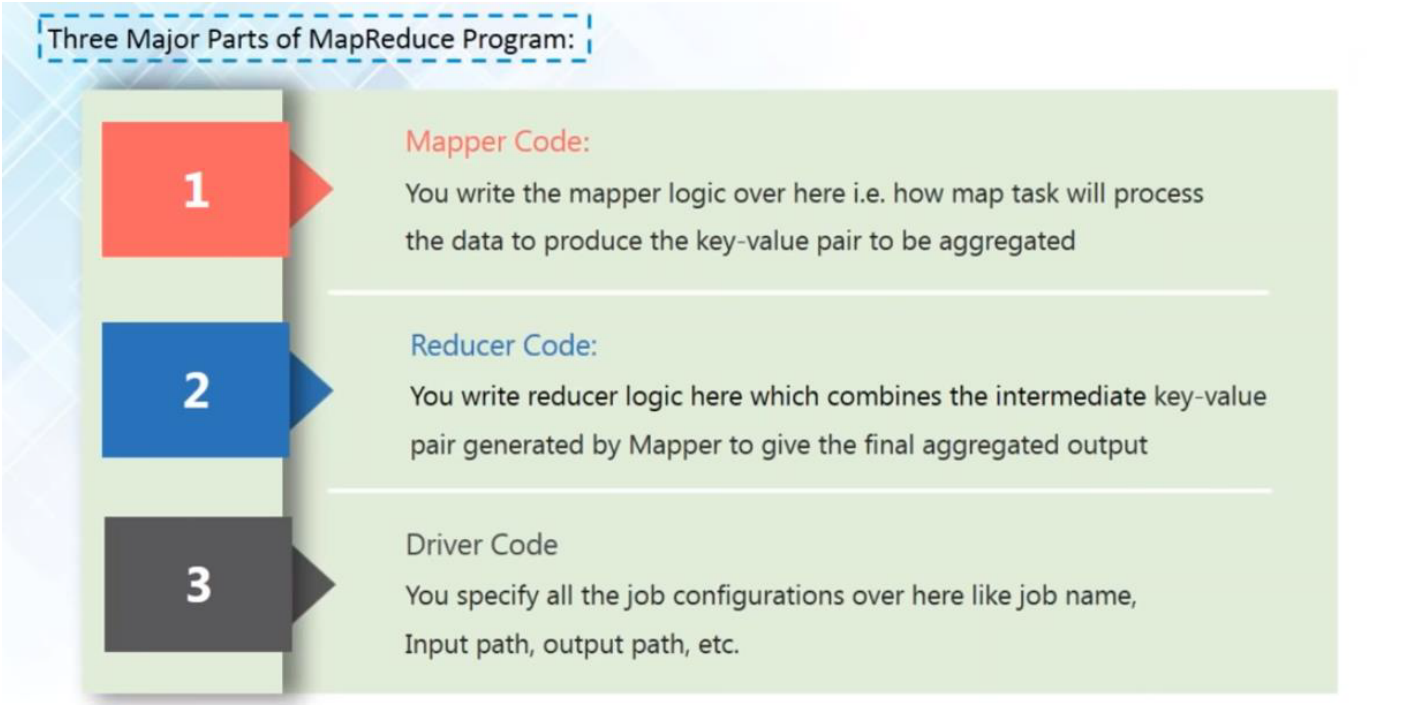
Parquet file format and diagram – 4+1 marks

- **An ORC (Optimized Row Columnar)** file consists of row-group data called stripes.
- ORC enables concurrent reads of the same file using separate RecordReaders. Metadata store uses Protocol Buffers for addition and removal of fields.
- ORC is an intelligent Big Data file format for HDFS and Hive.
- An ORC file stores a collections of rows as a row-group. Each row-group data store in columnar format. This enables parallel processing of multiple row-groups in an HDFS cluster.
- An ORC file consists of a stripe the size of the file is by default 256 MB.
- Stripe consists of indexing (mapping) data in 8 columns, row-group columns data (contents) and stripe footer (metadata).
- An ORC has two sets of columns data instead of one column data in RC. One column is for each map or list size and other values which enable a query to decide skipping or reading of the mapped columns.
- A mapped column has contents required by the query. The columnar layout in each ORC file thus, optimizes for compression and enables skipping of data in columns. This reduces read and decompression load.
- Lightweight indexing is an ORC feature.
- Each index includes the aggregated values of minimum, maximum, sum and count using aggregation functions on the content columns.
- Therefore, contents column key for accessing the contents from a column consists of combination of row-group key, column mapping key, min, max, count (number) of column fields of the contents column.
- Table 3.5 gives the keys used to access or skip a contents column during querying. The keys are Stripe_ID, Index-column key, and contents-column name, min, max and count.
- The throughput increases due to skipping and reading of the required fields at contents-column key. Reading less number of ORC file content-columns reduces the workload on the NameNode.

Stripe_ID	Index Column 1				Index Column 2
	Index column 1 key 1				Index column 2 key 1
	Contents-Column name	Contents Minimum value	Contents Maximum value	Count (number) of content-column fields	
	...	...	...	...	
	...	...	...	...	
	Index column 1 key 2				Index column 2 key 2
	Column-name	Minimum value	Maximum value	Count of number of column fields	
	...	...	...	...	
	...	...	...	...	

Q4 Explain MapReduce execution steps by considering wordcount examples.
MapReduce with all steps explanation – 6 marks
Diagram – 2 marks
Example -2 marks





Partitioning

- The Partitioner does the partitioning The partitions are the semi mappers in MapReduce Partitioner is an optional class
- MapReduce driver class can specify the Partitioner A partition processes the output of map tasks before submitting it to Reducer tasks
- Partitioner function executes on each machine that performs a map task
- Partitioner is an optimization in MapReduce that allows local partitioning

before reduce task phase Typically, the same codes implement the Partitioner Combiner as well as reduce() functions

•

Functions for Partitioner and sorting functions are at the mapping node
The main function of a Partitioner is to split the map output records with the same key

Combiners

Combiners are semi reducers in MapReduce . Combiner is an optional class. MapReduce driver class can specify the combiner. The combiner() executes on each machine that performs a map task.

Combiners optimize MapReduce task that locally aggregates before the shuffle and sort phase. Typically, the same codes implement both the combiner and the reduce functions

The main function of a Combiner is to consolidate the map output records with the same key. The output (key value collection) of the combiner transfers over the network to the Reducer task as input

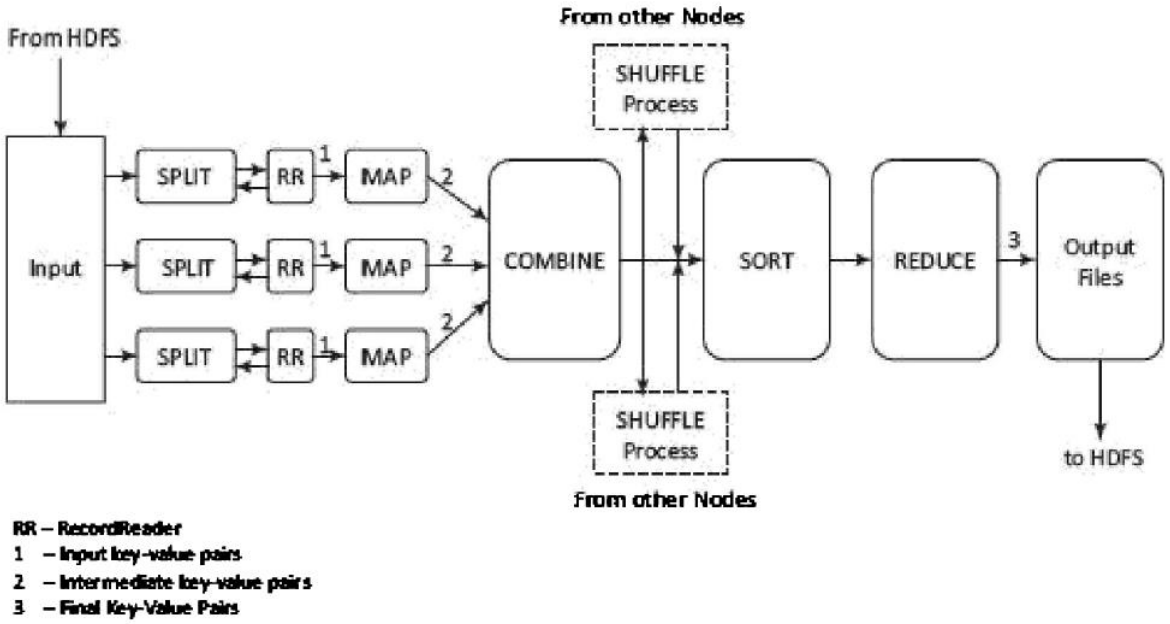
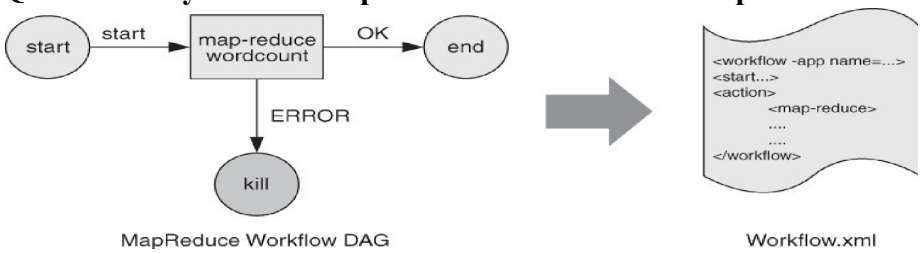


Figure 4.6 MapReduce execution steps

Q5. Identify and explain the context. Explain the below diagram in detail.



MapReduce Workflow DAG

Workflow.xml

Identification and explanation of the Context- 4 mark
Diagram node explanation - 6 marks

- Oozie is a workflow director system designed to run and manage multiple related Apache Hadoop jobs.
- Oozie is designed to construct and manage these workflows. Oozie is not a substitute for the YARN scheduler.
- YARN manages resources for individual Hadoop jobs, and Oozie provides a way to connect and control Hadoop jobs on the cluster.
- Oozie workflow jobs are represented as directed acyclic graphs (DAGs) of actions. Three types of Oozie jobs are permitted:

Workflow—a specified sequence of Hadoop jobs with outcome-based decision points and control dependency. Progress from one action to another cannot happen until the first action is complete.

Coordinator—a scheduled workflow job that can run at various time intervals or when data become available.

Bundle—a higher-level Oozie abstraction that will batch a set of coordinator jobs.

Oozie is integrated with the rest of the Hadoop stack, supporting several types of Hadoop jobs out of the box

Figure 7.6 depicts a simple Oozie workflow. In this case, Oozie runs a basic MapReduce operation. If the application was successful, the job ends; if an error occurred, the job is killed. Such workflows contain several types of nodes:

Control flow nodes define the beginning and the end of a workflow. They include start, end, and optional fail nodes.

Action nodes are where the actual processing tasks are defined. When an action node finishes, the remote systems notify Oozie and the next node in the workflow is executed. Action nodes can also include HDFS commands.

Fork/join nodes enable parallel execution of tasks in the workflow. The fork node enables two or more tasks to run at the same time. A join node represents a rendezvous point that must wait until all forked tasks complete.

Control flow nodes enable decisions to be made about the previous task. Control decisions are based on the results of the previous action (e.g., file size or file existence). Decision nodes are essentially switch-case statements that use JSP EL (Java Server Pages—Expression Language) that evaluate to either true or false.

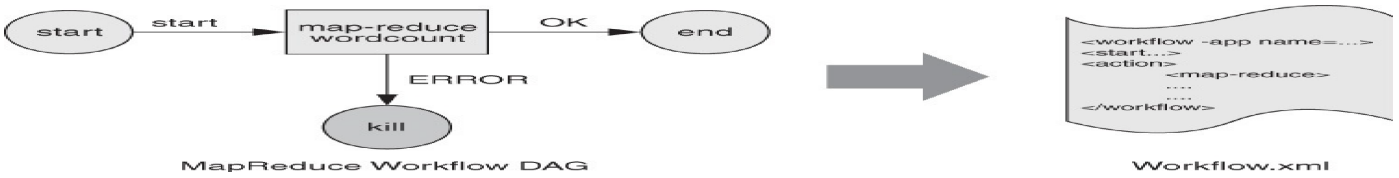
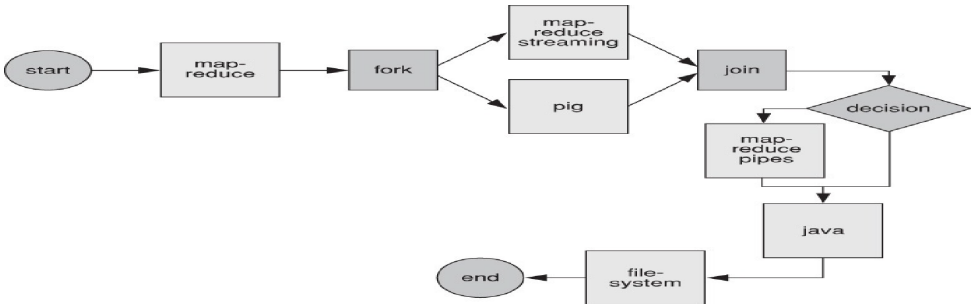


Figure 7.6 A simple Oozie DAG workflow

Figure 7.7 depicts a more complex workflow that uses all of these node types



Q6. Explain working of Flume agent. Explain the importance of AVRO in pipelining Flume.
Flume agent component – 4 marks
AVRO pipeline with diagram- 6 marks

- Apache Flume is an independent agent designed to collect, transport, and store data into HDFS.
- Often data transport involves a number of Flume agents that may traverse a series of machines and locations. Flume is often used for log files, social media-generated data, email messages, and just about any continuous data source.
- Flume agent is composed of three components.

✔ ***Source***. The source component receives data and sends it to a channel. It can send the data to more than one channel. The input data can be from a real-time source (e.g., weblog) or another Flume agent.

✔ ***Channel.*** A channel is a data queue that forwards the source data to the sink destination. It can be thought of as a buffer that manages input (source) and output (sink) flow rates.

✔ ***Sink.*** The sink delivers data to destination such as HDFS, a local file, or another Flume agent.

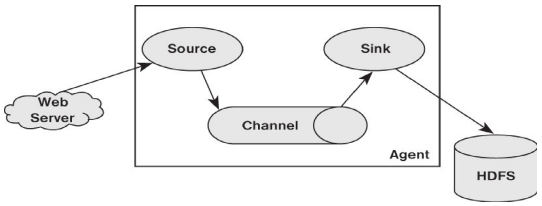


Figure 7.3 Flume agent with source, channel, and sink

A Flume agent can have several sources, channels, and sinks. Sources can write to multiple channels, but a sink can take data from only a single channel. Data written to a channel remain in the channel until a sink removes the data. By default, the data in a channel are kept in memory but may be optionally stored on disk to prevent data loss in the event of a network failure.

As shown in Figure 7.4, Sqoop agents may be placed in a pipeline, possibly to traverse several machines or domains. This configuration is normally used when data are collected on one machine (e.g., a web server) and sent to another machine that has access to HDFS.

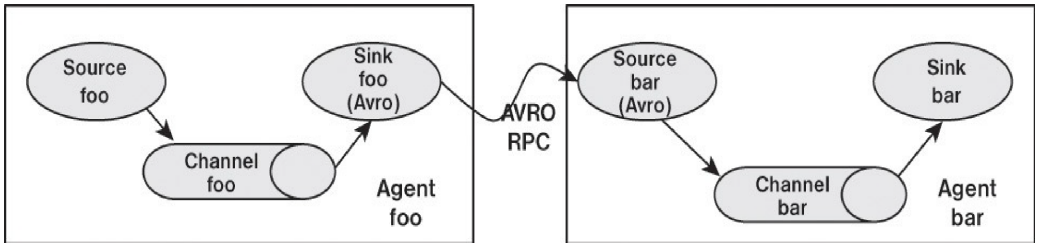


Figure 7.4 Pipeline created by connecting Flume agents

In a Flume pipeline, the sink from one agent is connected to the source of another. The data transfer format normally used by Flume, which is called Apache Avro, provides several useful features.

1) Avro is a data serialization/deserialization system that uses a compact binary format.

2) The schema is sent as part of the data exchange and is defined using JSON

3) Avro also uses remote procedure calls (RPCs) to

send data. That is, an Avro sink will contact an Avro source to send data.

Another useful Flume configuration is shown in Figure 7.5. In this configuration, Flume is used to consolidate several data sources before committing them to HDFS.