BIG DATA

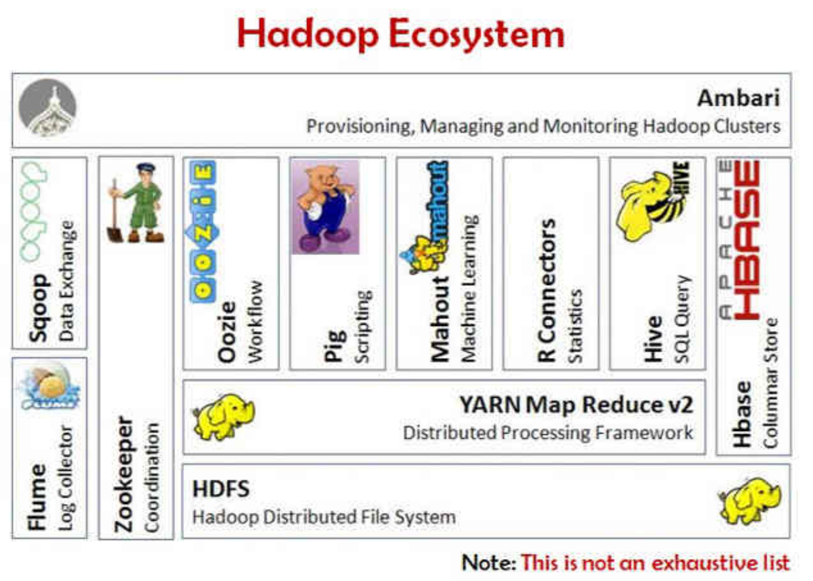
UNIT-2

Big Data Infrastructures (Compute/Storage)

feedback/corrections: vibha@pesu.pes.edu

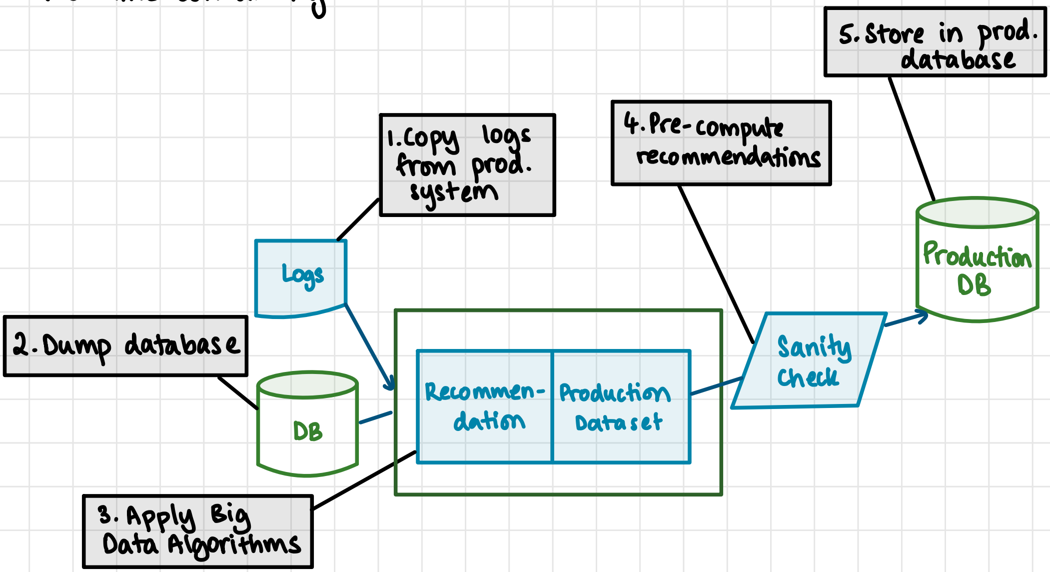
VIBHA MASTI

- https://www.facebook.com/hadoopers/photos/408924142550465



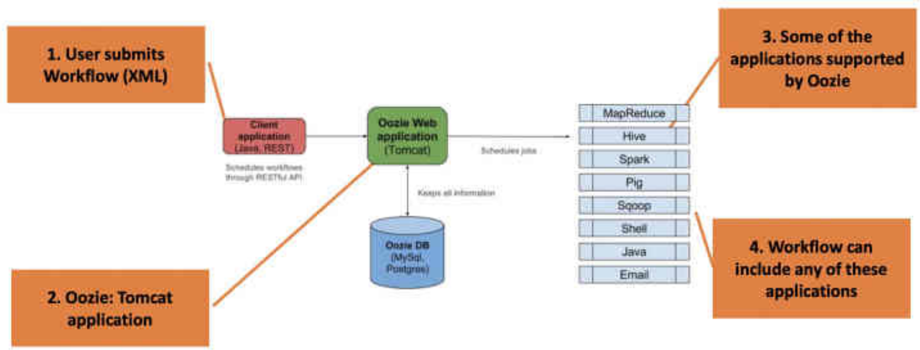
Oozie

- Recommendation system
- Workflow
- Pre-compute recommendation, update over weekend
- Too time-consuming



Apache Oozie

- User submits workflow as XML
- Tomcat web application



Oozie Workflow

- Workflow consists of action nodes and control flow nodes

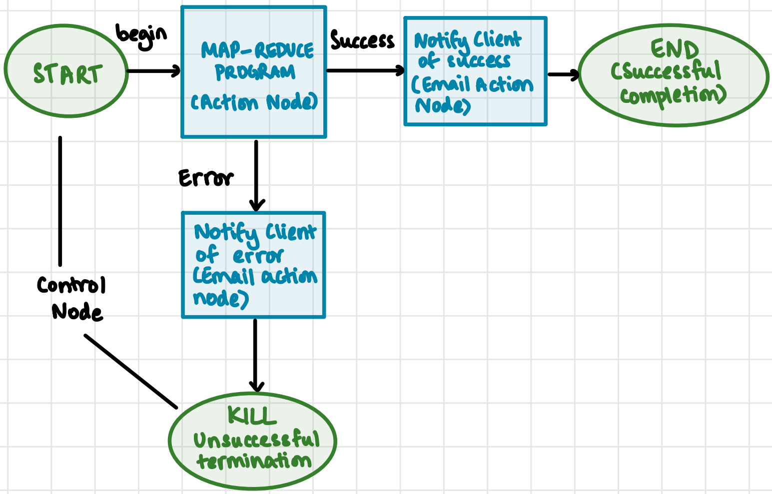
- **Action Nodes:** represents workflow tasks (eg: move files into HDFS, run MR, Pig or HIVE job, import data using Sqoop, run shell script of java program)

- **Control-flow Nodes:** controls workflow execution between actions by allowing conditional logic
  - **Start Node:** designates the start of the workflow job
  - **End Node:** signals end of job
  - **Error Node:** designates occurrence of an error and corresponding error message

- Other control nodes
  - **Fork** and **Join** (start parallel tasks — could be MR — and merge parallel tasks)
  - **Decision** (like C's switch statements)

- End of workflow, HTTP callback used by Oozie to update the client with the workflow status

- Callback may also be triggered during entry to or exit from an action node



Workflow

1. A workflow app consists of wf definition and all associated resources (MR jar files, pig scripts etc.) specified in a workflow.xml file placed in HDFS

```
$ hadoop fs -put hadoop-examples/target/<name_of_workflow dir>/<name_of_workflow.xml>
```

2. The oozie environment in terms of which server to use is specified

```
$ export OOZIE_URL="http://localhost:11000/oozie"
```

3. The wf is run

```
$ oozie job -config ch05/src/main/resources/max-temp-workflow-properties -run
```

4. The config files contains definitions of parameters in the workflow XML file

```
nameNode=hdfs://localhost:8020
jobTracker=localhost:8021
oozie.wf.application.path=${nameNode}/user/${user.name}/<name_of_workflow>
```

5. Results of successful wf execution can be viewed as

```
$ hadoop fs -cat <location_of_result>
```

Ambari

- Open source web-based management framework/tool for Hadoop clusters

- Functions supported

  1. Cluster provisioning
     - Simplified deployment across platforms
     - Wizard-driven cluster install
     - Cloud, virtual and physical environment

  2. Managing cluster
     - Consistent controls across the stack
     - Single point for cluster operations (start, pop etc.)
     - Advanced configurations and host controls

  3. Monitor
     - Visibility into key cluster metrics
     - Dashboard for cluster health and status
     - Pre-configured and customisable metrics, notifications and alerts

- Supports easy, efficient, repeatable creation of clusters

- Supports the following Hadoop components in 3 layers
  - Core Hadoop: HDFS, MR
  - Essential Hadoop: Pig, Hive, Hcatalog, HBase, Zookeeper
  - Hadoop Support: Oozie, Sqoop, Ganglia, Nagios

Components of Ambari

1. Ambari Web

- Runs on client side
- Calls Ambari REST API to access cluster information and perform cluster operations

2. Ambari Server

- Master process which communicates with Ambari agents
- Has DB used to maintain cluster-related metadata
- Provides communication with agents

3. Ambari Agent

- Installed on each node
- Periodically sends health status (+ different metrics) to master ?
- Actions driven by the master

Architecture

(a) Ambari Stacks

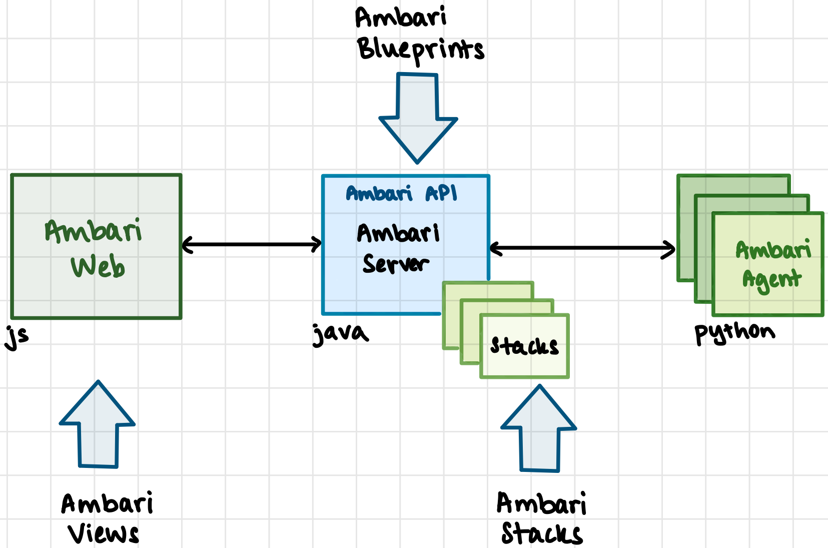
- Coordinated and tested set of Hadoop ecosystem components to be installed
- Eg: Hadoop, its components and its structure

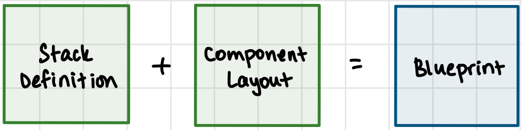
(b) Ambari Blueprints

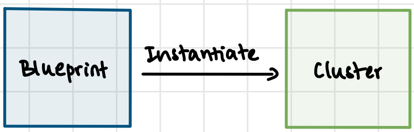
- Cluster definition files (2 JSON files), one generic template and one that sets specific properties to launch the deployment process
- Could include: stack name, stack version, security etc.

(c) Ambari Views

- UI







Disadvantages of MR

- Too low-level for data analytics (series of map and reduce stages)
- Writing code requires training
- Abstraction will be helpful (inputs in high-level scripting language converted to MR) — using Apache Pig and Pig Latin
- SQL-like abstraction also good — HIVE

PIG

- Abstraction over MR
- Supports all kinds of data (structured, unstructured) and stores in HDFS

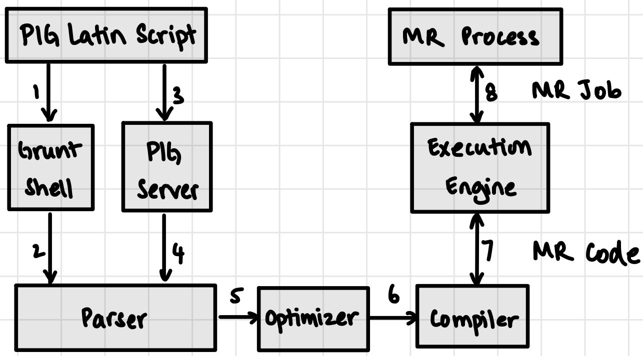
Architecture of PIG

1. PigLatin
   - SQL-like high-level language
   - Supports complex transformations
   - Operations: join, group, filter, limit etc.
   - Supports automatic optimisation
   - Supports running of functions written in other languages (eg: Java)

2. Pig Server
   - Runtime env for PigLatin

3. Grunt
   - Pig shell

- PigLatin scripts are converted to MR Jobs internally
- Created at Yahoo



Example Data Analytics Taks

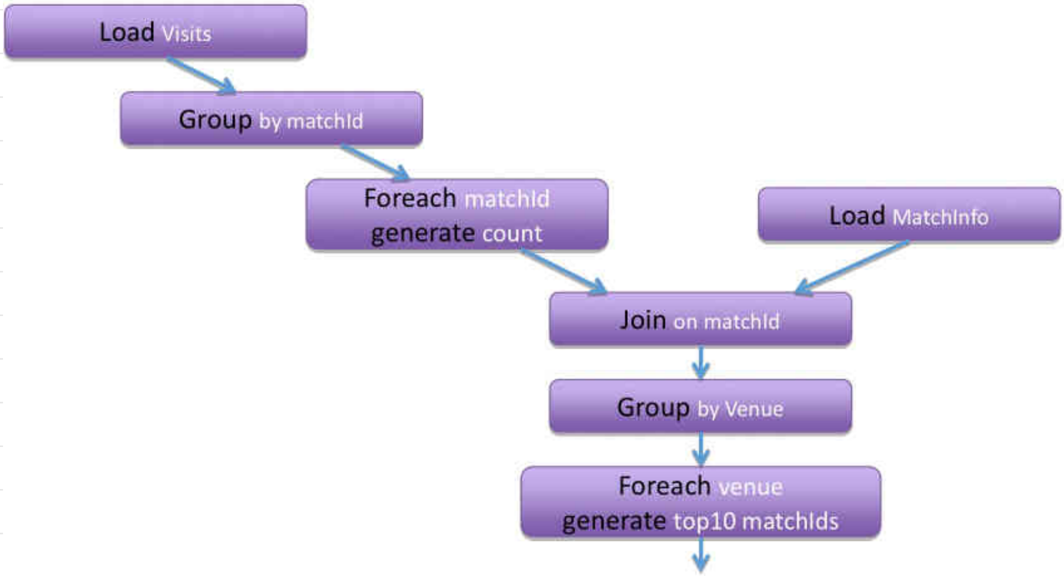
Find the top 10 most popular IPL matches in each venue.

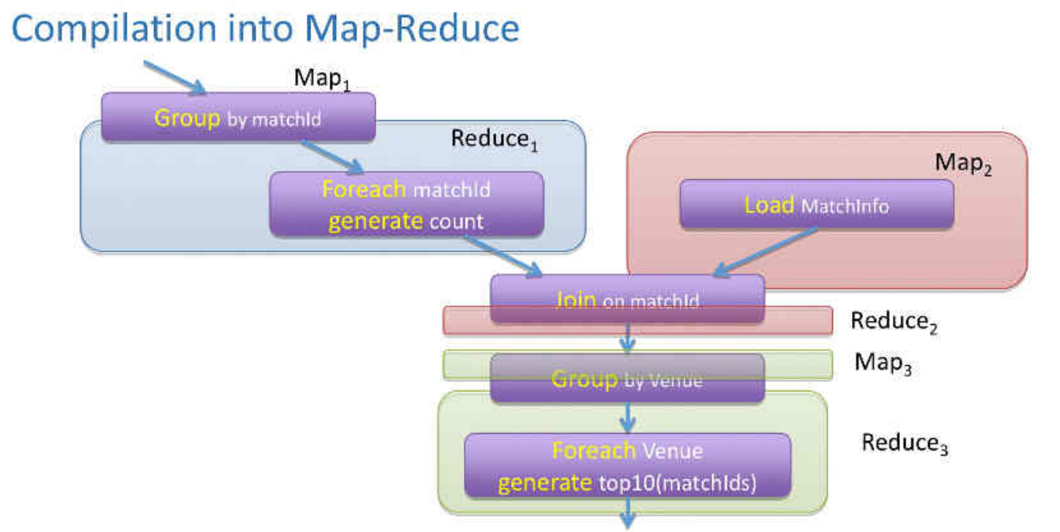
Visits

User	matchId	Time
RajniKan	Match 400	2:00
RajniKan	Match 201	5:00
Superman	Match 42	10:05
Spiderman	Match 108	13:03

MatchInfo

matchId	Venue	Winner
Match 108	Chennai	CSK
Match 201	Bengaluru	RCB
Match 42	Kolkata	DC
Match 400	Mumbai	RCB





In Pig Latin

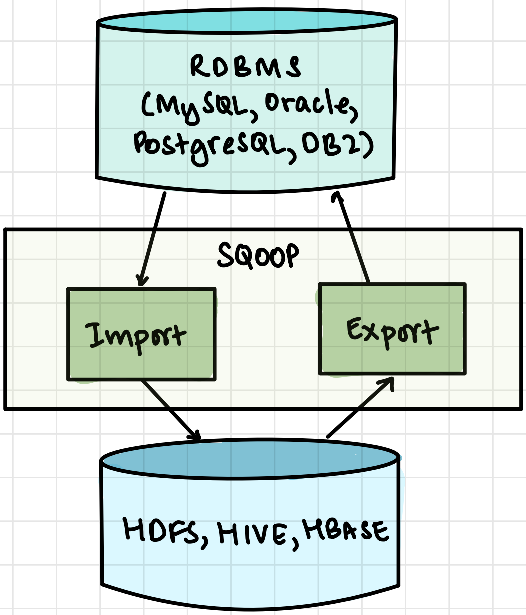
```
visits = load /ipldata/visits as (user,matchid, time);
gMatches = group visits by matchId;
matchPopularity = foreach gMatches generate matchId, count(visits);

matchInfo = load /ipldata/matchInfo as (url, venue, winner);
venueCounts = join gMatches by matchId, matchInfo by matchId;
gVenues = group venueCounts by venue;
topMatches = foreach gVenues generate top(matchPopularity,10);

store topMatches into /data/topMatches;
```

SQOOP

- SQL-to-Hadoop (Apache)
- Supports bulk import and export of data into and out of HDFS
- From structured DBs like RDBMS, NOSQL etc (defines schema for import)
- Data migration tool based on connector architecture
- Advantage of migrating to HDFS: streaming data access
- Supports plugins for data sources
- https://blogs.apache.org/sqoop/entry/apache_sqoop_overview



IMPORT : SQL to HDFS

```
$ sqoop import --connect jdbc:mysql://localhost/acmedb \
  --table ORDERS --username test --password ****
```

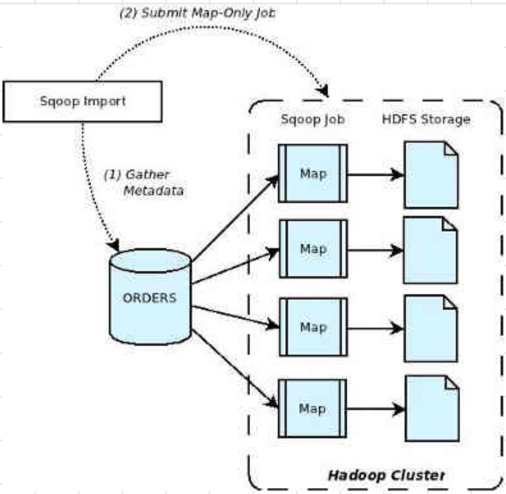
- connect argument is used to connect to the database
- table argument specifies table name

STEP 1

- Inspect DB to gather required metadata on data being imported

STEP 2

- Transfers data
  - Map-only Hadoop job Sqoop submits to cluster
  - Imported data stored in HDFS directory
  - CSV by default



https://blogs.apache.org/sqoop/entry/apache_sqoop_overview

EXPORT : HDFS to SQL

```
$ sqoop export --connect jdbc:mysql://localhost/acmedb \
  --table ORDERS --username test --password **** \
  --export-dir /user/<name>/ORDERS
```

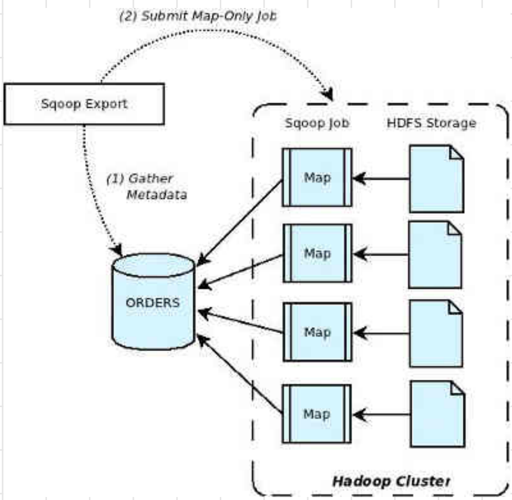
- connect argument is used to connect to the database
- table argument specifies table name to be populated
- export-dir argument is dir from which data is exported

STEP 1

- Inspect DB to gather required metadata on data being exported

STEP 2

- Each map task performs this transfer over many transaction
  - Minimal resource, max throughput



https://blogs.apache.org/sqoop/entry/apache_sqoop_overview

FLUME

- BD from various data sources (app servers, social net sites, cloud servers, enterprise servers) using Hadoop will be producing log files and events
- Cannot put into HDFS file-by-file
- Apache Flume: ingestion mechanism for effectively collecting, aggregating and moving large amounts of data
- Streaming data flows for collecting large amounts of streaming data to a centralized store
  - from events
  - from logs

Architecture

1. Agents
   - receive Flume events from data generators and store in centralized store (HDFS, HBase)
   - individual daemon process (JVM)
2. Source
   - part of an agent
   - receive Flume events from data generators and transfers it to one or more channels in the form of Flume events

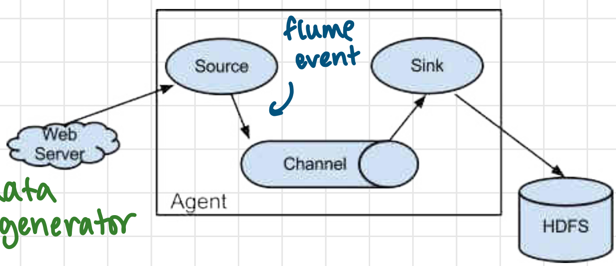
3. Flume event

- basic unit of data transfer inside Flume
- contains a payload of byte array to be transported from source to dest with the structure

Header	Byte Payload

4. Sink

- receives data and stores into centralized stores (HDFS, HBase)
- consumes data from channels and delivers to the destination



5. Channels

- connect sources to sink
- transient store that receives events from source and buffers them till they are consumed by sinks
- bridge between source & sink
- JDBC channel, File system channel

MAP-REDUCE ALGORITHMS – MATRIX MULTIPLICATION

Vectors

- Ordered list of numbers
- Size & direction
- Operations: addition, scalar multiplication

$$\begin{pmatrix} 3 \\ 4 \end{pmatrix} + \begin{pmatrix} 7 \\ 2 \end{pmatrix} = \begin{pmatrix} 10 \\ 6 \end{pmatrix}$$

Matrices

- Rectangular array of elements
- $n \times m$: n rows and m columns
- Vectors maybe considered as row $(1 \times n)$ or column $(n \times 1)$ matrices
- Represented in memory as row-major or column-major form as multi-dimensional array
  - Row major: C/C++, Python, Java etc.
  - Column major: Fortran
- Represented on disk

Matrix Multiplication

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}_{n \times m} \times \begin{bmatrix} b_{11} & \cdots & b_{1p} \\ b_{21} & \cdots & b_{2p} \\ \vdots & \vdots & \vdots \\ b_{m1} & \cdots & b_{mp} \end{bmatrix}_{m \times p} = \begin{bmatrix} c_{11} & \cdots & c_{1p} \\ c_{21} & \cdots & c_{2p} \\ \vdots & \vdots & \vdots \\ c_{n1} & \cdots & c_{np} \end{bmatrix}_{n \times p}$$

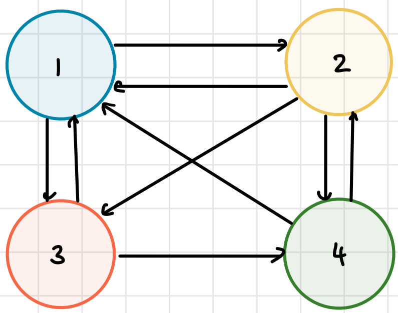
$$c_{ij} = \sum_{k=1}^{m} a_{ik} \cdot b_{kj}$$

Matrix Representation of WWW

- Model WWW as a directed graph

Page 1	Page 2
Links to 2 3	Links to 1 3 4
Page 3	**Page 4**
Links to 4 1	Links to 1 2

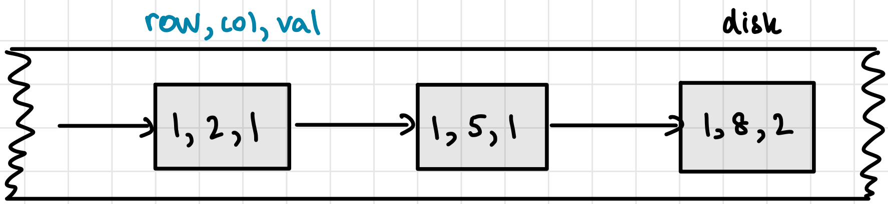
- Directed graph

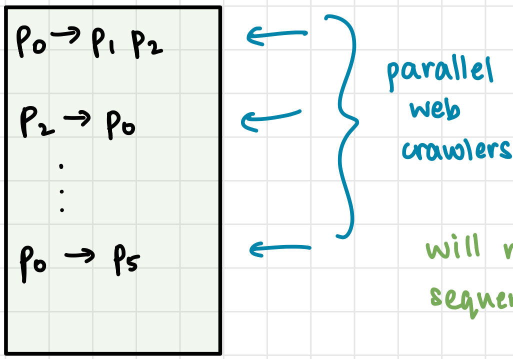


- As adjacency matrix

$$
\begin{array}{cc}
 & \text{Source} \\
\text{Dest} & \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}
\end{array}
$$

Representing as Sparse Matrix





will not be stored sequentially on file

- Store non-zero values in csv file on HDFS
  <row number, column number, value>

- As many entries as links

Source

$$\text{Dest}\quad \begin{array}{c} \\ 1\\ 2\\ 3\\ 4\end{array}\begin{array}{c}\begin{matrix}1&2&3&4\end{matrix}\\ \begin{bmatrix}0&1&1&1\\1&0&0&1\\1&1&0&0\\0&1&1&0\end{bmatrix}\end{array}$$

1, 2, 1
1, 3, 1
1, 4, 1
2, 1, 1
2, 4, 1
3, 1, 1
3, 2, 1
4, 2, 1
4, 3, 1

Q: Exercise: try saving it in a file and loading it onto HDFS

Matrix-Vector Multiplication with MR

- $A_{m \times n} \times V_n$

$$x_i = \sum_{j=1}^{n} a_{ij} \times v_j$$

- A is sparse
- A is a large matrix stored as an HDFS file

Case 1: Assumptions

- Assume vector v can fit in memory
- Assume vector v shared by all machines
- Assume M_{ij} is stored as a CSV file on HDFS and is distributed across multiple nodes

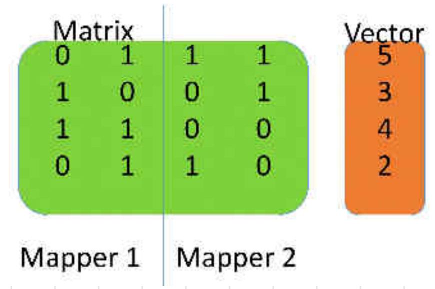
Map

- Computes partial product
- Use key as $i \rightarrow$ index in target vector
- Output $(i, m_{ij} v_j)$

Reducer

- Sum all partial products

Example



Map:

Mapper 1

Key	Value
1	1*3 = 3
2	1*5 = 5
3	1*5= 5
3	1*3=3
4	1*3=3

Mapper 2

Key	Value
1	1*4=4
1	1*2=2
2	1*2=2
4	1*4=4

Reduce:

Reducer Input

Key	Intermediate Value List
1	2,3,4
2	2, 5
3	3, 5
4	3, 4

Reducer output

Key	Value
1	9
1	7
2	8
4	7

Q: Assume Row1, Row 3 in DN1 and R2, R4 in DN2

Show input/output of M & R

Matrix $\quad$ Vector

$$\begin{bmatrix} 0 & 0 & 3 & 8 \\ 0 & 9 & 5 & 0 \\ 0 & 10 & 0 & 0 \\ 5 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

Node 1:

(1,3,3) $\longrightarrow$ (1,9)
(1,4,8) $\longrightarrow$ (1,32)
(3,2,10) $\longrightarrow$ (3,20)

Node 2:

(2,2,9) $\longrightarrow$ (2,18)
(2,3,5) $\longrightarrow$ (2,15)
(4,1,5) $\longrightarrow$ (4,5)
(4,4,1) $\longrightarrow$ (4,4)

Reducer:

(1,35)
(2,33)
(3,20)
(4,9)

Case 2: v does not fit in memory

- Partition M into stripes

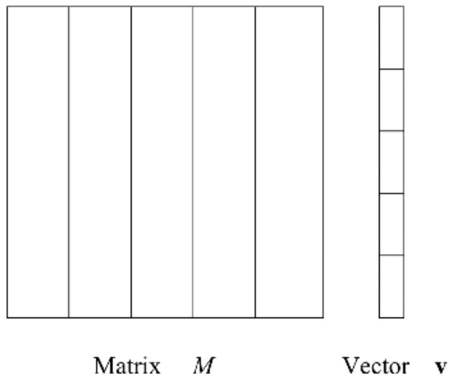


Figure 2.4: Division of a matrix and vector into five stripes

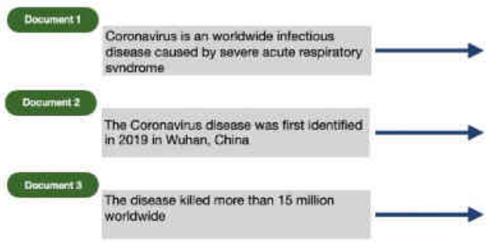
R1

- Multiple stages of MR

Search Engine Working

Search for a term
Eg: "Big Data" → Lookup an inverted index: maps every term to documents that contain the term → Sort pages based on importance

Inverted Index



ID	Term	Document
1	Coronavirus	1, 2
2	worldwide	1, 3
3	infectious	1
4	disease	1,2,3
5	severe	1
6	acute	1
7	respiratory	1
8	syndrome	1
9	first	2
10	2019	2
11	identified	2
12	Wuhan	2
13	China	2
14	killed	3
15	million	3

source: Medium

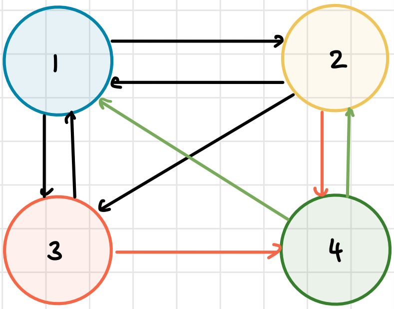
Inverted Index vs Forward Index

Inverted Index	Forward Index
Maps words or texts to documents	Maps documents to words
Indexing is slower since a check needs to validate if the word exists	Indexing is faster since the addition of texts/ words can be append only
Searching or information retrieval is faster	Searching or information retrieval is slower

PAGE RANK

- Cannot use term frequency
- Cannot simply use no. of incoming links
- Technique: start from a random page and start traversing ; probability of reaching a page from a random page using links (random surfer)

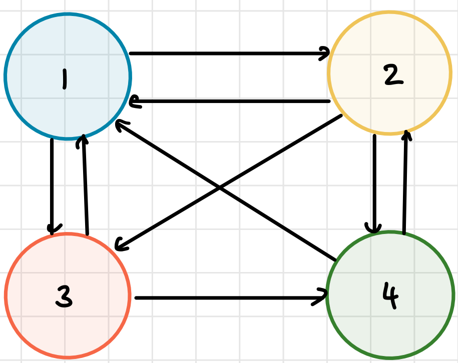
Converting Adjacency Matrix to Transition Matrix



$$\begin{array}{cc} & \text{Source} \\ \text{Dest} & \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

- If random surfer starts at 4 and follows links out
  - can go to 1 or 2
- Assume equal probability of transition to either node
  - P(transition at node 4) = 1/(no. of outlinks)
- P(transition at 4) = 1/2
- Column vector of probabilities of directly transitioning to every other page

$$\begin{bmatrix} 0 \\ 1/2 \\ 1/2 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} 1/3 \\ 0 \\ 1/3 \\ 1/3 \end{bmatrix}$$



$$\begin{bmatrix} 1/2 \\ 0 \\ 0 \\ 1/2 \end{bmatrix} \qquad \begin{bmatrix} 1/2 \\ 1/2 \\ 0 \\ 0 \end{bmatrix}$$

Transition Matrix

$$M = \quad \text{Dest} \quad \begin{array}{c} \\ \bullet \\ \bullet \\ \bullet \\ \bullet \end{array} \begin{array}{c} \text{Source} \\ \begin{array}{cccc} \bullet & \bullet & \bullet & \bullet \end{array} \\ \begin{bmatrix} 0 & 1/3 & 1/2 & 1/2 \\ 1/2 & 0 & 0 & 1/2 \\ 1/2 & 1/3 & 0 & 0 \\ 0 & 1/3 & 1/2 & 0 \end{bmatrix} \end{array}$$

Random Surfer – Initialisation

- Random surfer randomly chooses a starting node (each node has equal probability of being a starting node)

- Vector v called the importance vector ; represents relative importance of each node

- Vector v initialised to v_0 (for the graph above)

$$v_0 = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}$$

- For each node, compute probability of ending up on that node based on previous node

- Multiply M and v (v tells importance of node and M tells probability of transition)

$$\underbrace{\begin{array}{c} \\ \bullet \\ \bullet \\ \bullet \\ \bullet \end{array} \begin{array}{c} \begin{array}{cccc} \bullet & \bullet & \bullet & \bullet \end{array} \\ \begin{bmatrix} 0 & 1/3 & 1/2 & 1/2 \\ 1/2 & 0 & 0 & 1/2 \\ 1/2 & 1/3 & 0 & 0 \\ 0 & 1/3 & 1/2 & 0 \end{bmatrix} \end{array}}_{M} \times \underbrace{\begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}}_{v_0} = \underbrace{\begin{bmatrix} 1/3 \\ 1/4 \\ 5/24 \\ 5/24 \end{bmatrix}}_{v_1}$$

- Repeat again

$$\begin{bmatrix} 0 & 1/3 & 1/2 & 1/2 \\ 1/2 & 0 & 0 & 1/2 \\ 1/2 & 1/3 & 0 & 0 \\ 0 & 1/3 & 1/2 & 0 \end{bmatrix} \times \begin{bmatrix} 1/3 \\ 1/4 \\ 5/24 \\ 5/24 \end{bmatrix} = \begin{bmatrix} 7/24 \\ 13/48 \\ 1/4 \\ 3/16 \end{bmatrix}$$

$$\qquad M \qquad\qquad\qquad v_1 \qquad\qquad v_2$$

- Stop when $Mv \sim v$ or $MV = V$

- v is the eigenvector of M

Page Rank – MR Implementation

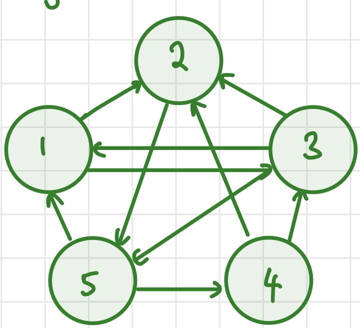
- Mapper stage outputs (k,v) pairs where key is name of the page (index in vector v) and value is the transition probability x the initial value in v

- Performed with a damping factor β

$$v_i = \beta M v_{i-1} + \frac{(1-\beta)e}{n}$$

- Combiner can be used to reduce network traffic

Q: Compute Page Rank



Adj matrix

	1	2	3	4	5
1	0	0	1	0	1
2	1	0	1	1	0
3	1	0	0	1	0
4	0	0	0	0	1
5	0	1	1	0	0

Transition Matrix

	1	2	3	4	5
1	0	0	1/3	0	1/2
2	1/2	0	1/3	1/2	0
3	1/2	0	0	1/2	0
4	0	0	0	0	1/2
5	0	1	1/3	0	0

V_0
1/5
1/5
1/5
1/5
1/5

Mapper Output

(1, 1/15)
(1, 1/10)
(2, 1/10)
(2, 1/15)
(2, 1/10)
(3, 1/10)
(3, 1/10)
(4, 1/10)
(5, 1/5)
(5, 1/15)

Reducer Output

(1, 1/6)
(2, 4/15)
(3, 1/5)
(4, 1/10)
(5, 4/15)

Where is the Comparison Done?

- v_i and v_{i-1} cannot be compared in the mapper or the reducer
- Where the code is written

```
do {
    v_{i+1} = MR(v_i)
} while (v_{i+1} != v_i)
```

RELATIONAL OPERATIONS

Relations

- Tables
- Set of rows/ tuples
- Columns are attributes

Relational Operators

- Selection : $\sigma_c(R)$ c: condition
- Projection: $\pi_s(R)$ s: subset of attributes
- Union: $\cup$
- Intersection: $\cap$
- Difference : $-$
- Join : $\bowtie$
- Group
- Aggregation

Q: Table PLAYERS

id	Name	Role	Team
1	Virat Kohli	Captain	RCB
2	Gautham Gambhir	Captain	KKR
3	Anil Kumble	Coach	MI
4	Virender Sehwag	Coach	KXIP

1. Write SQL queries to list
   (a) All details for coaches
   (b) Only the names of the coaches
   (c) Total no. of coaches

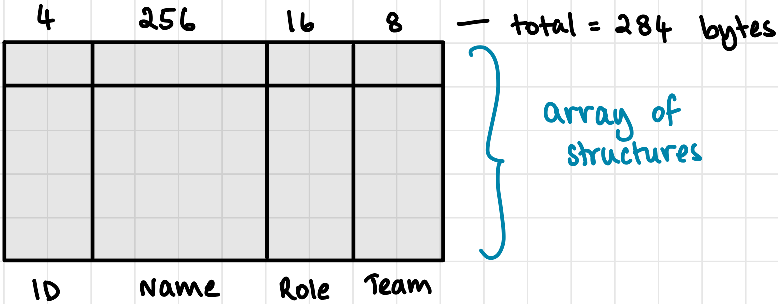
2. What type of relational operation is being used?

1. (a) SELECT * FROM PLAYERS WHERE Role='Coach';
   (b) SELECT NAME FROM PLAYERS WHERE Role='Coach';
   (c) SELECT COUNT(*) FROM PLAYERS WHERE Role='Coach';

2. (a) Selection
   (b) Projection & selection
   (c) Aggregation

BIG DATA PERSPECTIVE

- In memory



- Offsets :
  - ID : 0
  - name : 4
  - Role : 260
  - Team : 276

- If stored in structured DB on disk, can convert array of bytes to array of structures easily

- Assume data in csv on HDFS

- Records stored sequentially

- Cannot read in the same way

1, Virat Kohli, Captain, RCB
2, Gautham Gambhir, Captain, KKR
3, Anil Kumble, Coach, MI
4, Virender Sehwag, Coach, KXIP

SELECT IN MAP-REDUCE

Query: SELECT * FROM PLAYERS WHERE Role = 'Coach';

Map

<"3, Anil Kumble, Coach, MI","3, Anil Kumble, Coach, MI">
<"4, Virender Sehwag, Coach,KXIP","4, Virender Sehwag, Coach,KXIP">

Reduce

3, Anil Kumble, Coach, MI
4, Virender Sehwag, Coach,KXIP

Map

- Read each row t of table
- Check if it satisfies condition c
- If so, output (t,t)

can also be (PK,t) or (t, null) or (null, t) etc

Reducer

- Identity

PROJECT IN MAP-REDUCE

Query: SELECT NAME FROM PLAYERS WHERE Role = 'Coach';

Map

<"Anil Kumble"," Anil Kumble">
<"Virender Sehwag","Virender Sehwag">

Reduce

Anil Kumble
Virender Sehwag

Map

- Read each row t of table
- Calculate subset of attributes t'
- Output (t',t')

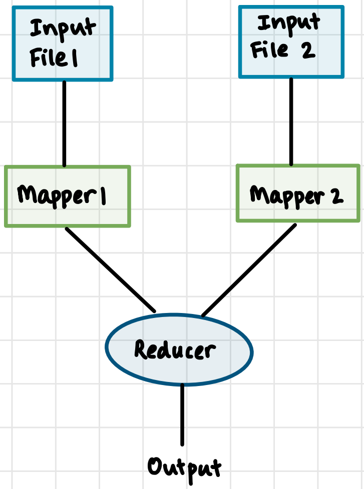
Reduce

- Eliminate duplicates (multiple rows with same field)

  (t', [t', t', t']) → (t', t')

UNION IN MAP-REDUCE

- R ∪ S : R and S have the same schema
- 2 input files ⟶ 1 output file
- 2 mappers for 2 input files



Mapper 1

- Read each row t of table R
- Output (t, t)

Mapper 2

- Read each row t of table S
- Output (t, t)

Reducer

- Eliminate duplicates
- (t, [t, t]) → (t, t)

Q: Given the following input for two files

File 1
- A
- B
- C
- D

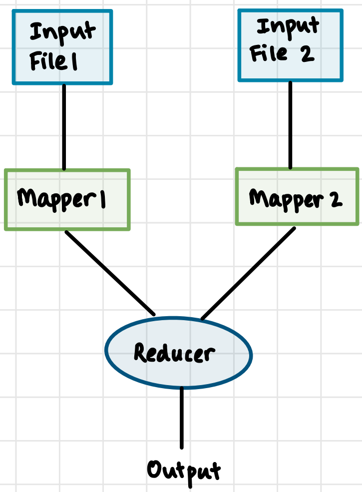
File 2
- A
- E
- F
- C

Show (for the Union algorithm)
- Input and output of mapper 1
- Input and output of mapper 2
- Input and output of reducer

Mapper 1	Mapper 2	Reducer
A → (A, A)	A → (A, A)	(A, [A, A]) → A
B → (B, B)	E → (E, E)	(B, B) → B
C → (C, C)	F → (F, F)	(C, [C, C]) → C
D → (D, D)	C → (C, C)	(D, D) → D
		(E, E) → E
		(F, F) → F

INTERSECTION IN MAP-REDUCE

- $R \cap S$: R and S have same schema



Mapper1

- Read each row t of table R
- Output (t,t)

Mapper 2

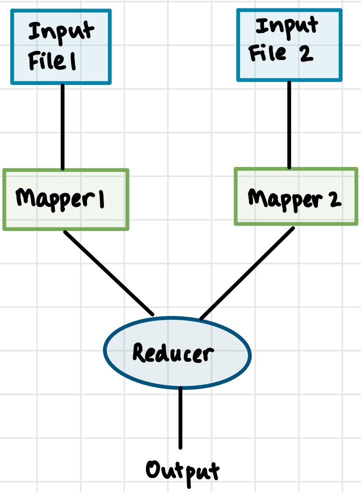
- Read each row t of table S
- Output (t,t)

Reducer

- Output only duplicates
- (t, [t, t]) → (t, t)

DIFFERENCE IN MAP-REDUCE

- R - S



Mapper1

- Read each row t of table R
- Output (t, R)

Mapper 2

- Read each row t of table S
- Output (t, S)

Reducer

- Output only those in R
- $(t, [R]) \rightarrow (t, t)$

Q: Perform set difference using MR. Show i/p and o/p at map & reduce ends

IPL 2010	IPL 2021
RCB	DC
KKR	RCB
KXIP	CSK
RR	RR
DD	
M_1	M_2

M_1

RCB → (RCB, 2010)
KKR → (KKR, 2010)
KXIP → (KXIP, 2010)
RR → (RR, 2010)
DD → (DD, 2010)

M_2

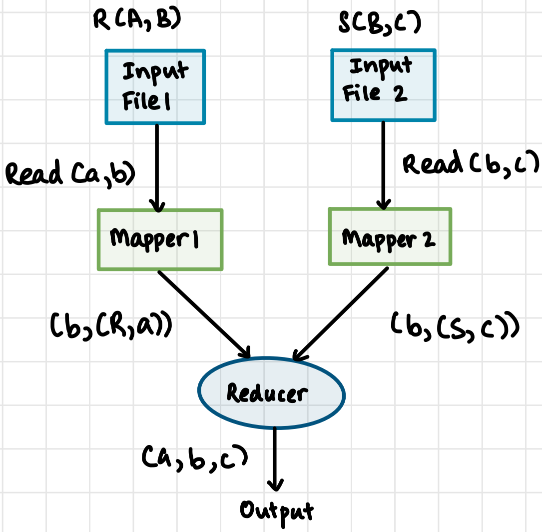
DC → (DC, 2021)
RCB → (RCB, 2021)
CSK → (CSK, 2021)
RR → (RR, 2021)

Reducer

(CSK, 2021) →
(DC, 2021) →
(DD, 2010) → DD
(KKR, 2010) → KKR
(KXIP, 2010) → KXIP
(RCB, [2010, 2021]) →
(RR, [2010, 2021]) →

NATURAL JOIN IN MAP-REDUCE

- Join R and S on attribute B
- A, C are the other attributes in R,S



Mapper 1
- Read (a,b) of R, output (b,(R,a))

Mapper 2
- Read (b,c) of S, output (b,(S,c))

Reducer
- For each pair (b,(R,a)) and (b,(S,c)), output (a,b,c)

Q: Given the following input for two files

Table Employee E(Name, age)

Gabbar 35
Viru 37
Jai 33
Baldev 44
Basanti 31

Table Dept D(Name, Dept)

Gabbar Bandit
Viru Hero
Jai Hero
Baldev Police
Basanti Heroine

Show (for the Natural Join algorithm)

1. Input and output of mapper 1
2. Input and output of mapper 2
3. Input and output of reducer

M1

(Gabbar, (E,35)
(Viru, (E,37)
(Jai, (E,33))
(Baldev, (E,44)
(Basanti, (E,31)

M2

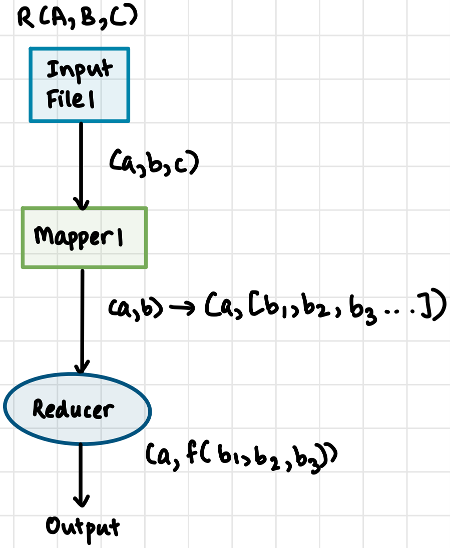
(Gabbar, (D, Bandit))
(Viru, (D, Hero))
(Jai, (D, Hero))
(Baldev (D, Police))
(Basanti, (D, Heroine))

R

(Baldev, [(E,44),(D,Police)]) → (Baldev, 44, Police)
(Basanti, [(E,31),(D, Heroine)]) → (Basanti, 31, Heroine)
(Gabbar, [(E,35), (D, Bandit)]) → (Gabbar, 35, Bandit)
(Jai, [(E,33), (D, Hero)]) → (Jai, 33, Hero)
(Viru, [(E,37), (D, Hero)]) → (Viru, 37, Hero)

GROUPING AND AGGREGATION

- For relation $R(A,B,C)$ group by A and aggregate by function $f(B)$



Mapper

- For each line (a,b,c) output (a,b)

Reducer

- Aggregate $(a, [b_1, b_2, b_3, \ldots, b_n])$ into $(a, f(b_1, b_2, b_3, \ldots, b_n))$

Q: Given the following input file

Table Dept D(Name, Dept)

Gabbar Bandit
Viru Hero
Jai Hero
Baldev Police
Basanti Heroine

Determine number of employees in each department.
1. Show mapper output
2. Show reducer input/output

Mapper

(Bandit, Gabbar)
(Hero, Viru)
(Hero, Jai)
(Police, Baldev)
(Heroine, Basanti)

Reducer

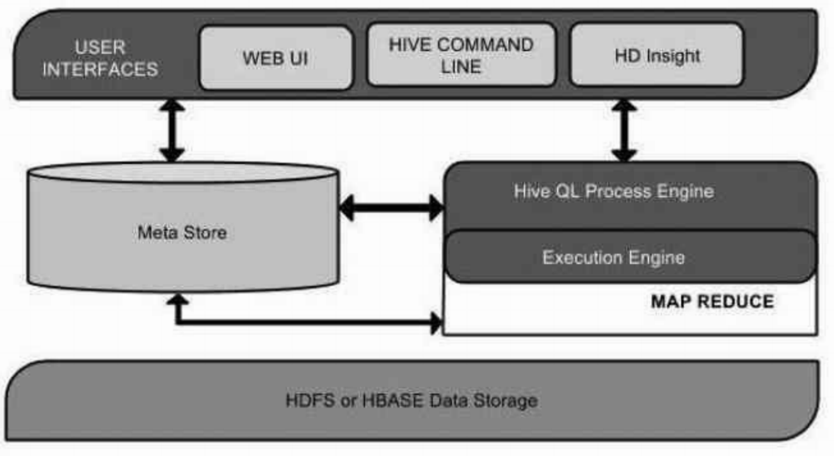
(Bandit, Gabbar) → (Bandit, 1)
(Hero, [Viru, Jai]) → (Hero, 2)
(Heroine, Basanti) → (Heroine, 1)
(Police, Baldev) → (Police, 1)

HIVE

- SQL → HDFS
- Converted to MR jobs
- SQL-like queries: HQL
- Storage: txt, RCFile, HBase

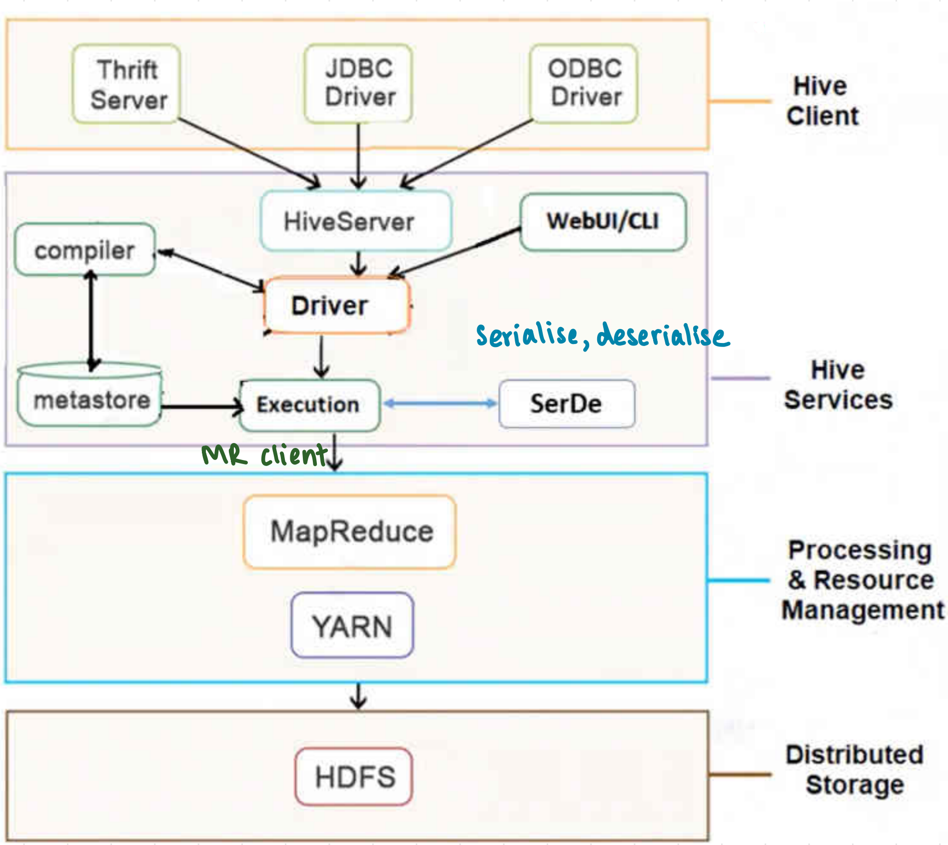
- Cannot handle real-time data
- Cannot handle OLTP — transactions should be atomically executed
- Hive queries contain latency

Architecture



- Meta Store : stores schema — maps columns to csv
- https://ieeexplore.ieee.org/document/5447738 - paper

HIVE Components

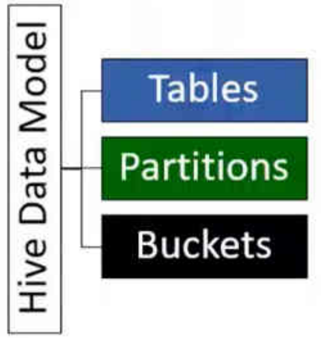


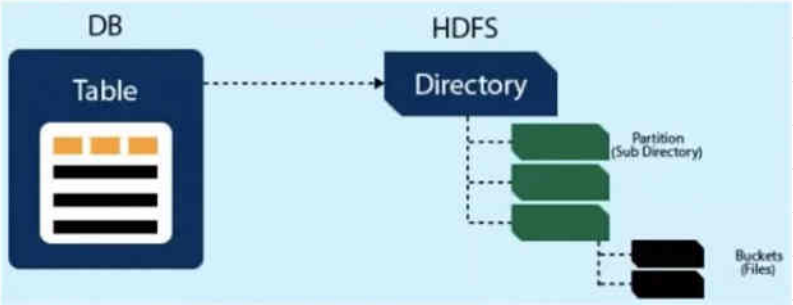
- On HDFS: can be stored in CSV, XML, JSON, Binary

- SerDe: Serialization / Deserialization
  - drivers for each format
  - for storing/ accessing files in HDFS

Hive Data Model

- Stored as HDFS files
- Categorized into three granular levels





Table

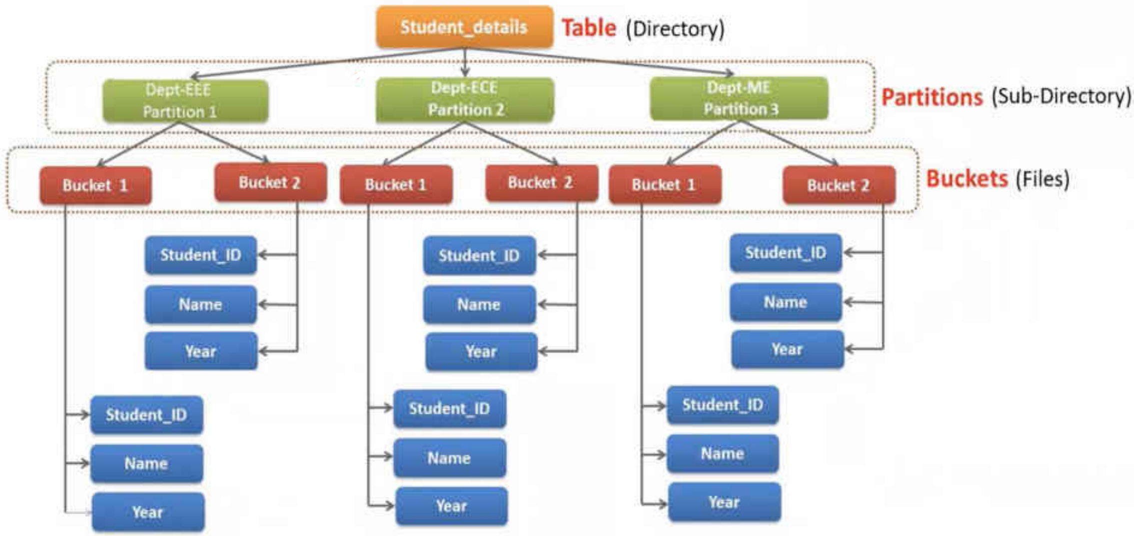
- Table mapped to HDFS directory
- Like table in RDBMS
- Operations: filter, project, join, union
- Tables divided into multiple files and directories

Partition

- HDFS subdirectory
- Tables organised into partitions based on column / key

Bucket

- Subdivision of partition
- Divided based on hash function of specified column
- Enhance query efficiencies



HIVE Compilation Example

http://www.vldb.org/pvldb/vol2/vldb09-938.pdf

```
FROM (SELECT a.status, b.school, b.gender
      FROM status_updates a JOIN profiles b
           ON (a.userid = b.userid and
               a.ds='2009-03-20' )
      ) subq1
INSERT OVERWRITE TABLE gender_summary
                       PARTITION(ds='2009-03-20')
SELECT subq1.gender, COUNT(1) GROUP BY subq1.gender
INSERT OVERWRITE TABLE school_summary
                       PARTITION(ds='2009-03-20')
SELECT subq1.school, COUNT(1) GROUP BY subq1.school
```

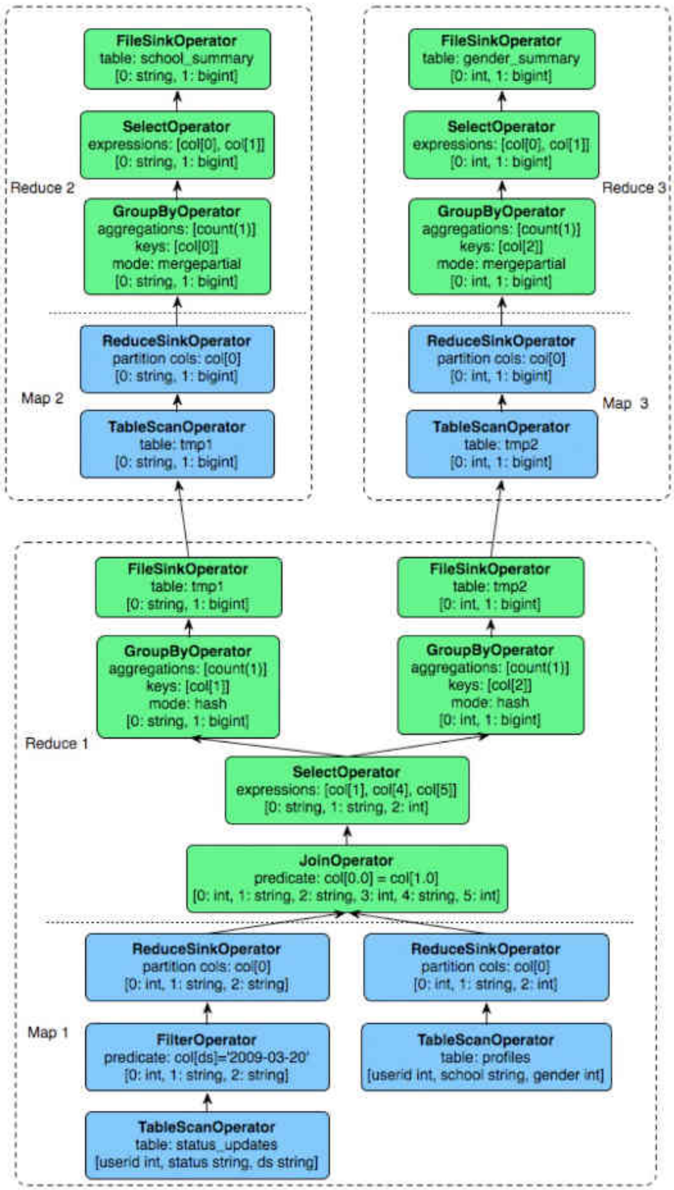
a

status (active, away)

b

gender
school

source: https://ieeexplore.ieee.org/document/5447738



COLUMNAR DATABASES - MOTIVATION

- HDFS - good for batch processing

- HDFS - Not good for
  - record lookup
  - incremental addition of small batches
  - updates

- HIVE not good for
  - record lookup
  - incremental addition of small batches
  - updates
  - unstructured / semi-structured data

- Hbase and Cassandra

  https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf

- Built on Bigtable model

- Good for
  - fast record lookup
  - record level insertion
  - updates (Hbase - creates new versions)
  - unstructured / semi-structured data

Summary

- HDFS
  - Unstructured data
  - Writes: no updates, only appends
  - Read entire file and analyse
- Hive
  - Structured data
  - Analytics via SQL
- HBase/Cassandra
  - Unstructured data
  - Arbitrary writes
  - Analytics

Q: Which of these could be stored in HDFS, Hive or Hbase?

1. Parsed transaction logs of user activity in a website where relevant fields from the log have been extracted

2. Unparsed transaction logs of user activity

3. Database of users and friends at a social website, which is periodically analyzed for social networking analysis

1. Hive – structured (relevant fields)
2. HDFS – unstructured (unparsed)

no updates required

3. Hbase – constant updates

Columnar Storage

Row Key	Info:height	Info:age	School:House	School:sports
HarryPotter	4.5ft	11	Gryffindor	Quidditch
Voldemort	7ft	50	Slytherin	

- Row storage: DB single file, one row per line (transactions)
- Column storage: each column stored as separate file, one value per line (analytics)
- Each load requires I/O to be performed

Q: Which method does less I/O for

1. Analyzing the relationship between age and earnings

   column

2. Adding a new row or read a row

   row

Structured vs Unstructured Data

- How to handle unstructured data in RDB?

Customer id	Visit id	Date
		2-Oct 2017

Customer id	Visit id	Symptom id	Symptom
		1	Chest pain
		2	Headache

- How to handle unstructured data in unstructured DB?

Customer id	Visit id	Info
		Date: {2-Oct 2017} Symptoms: {Chest pain, Headache}

— Hbase

- Distributed column oriented DB on top of HDFS
- Data logically: rows/ columns of a table

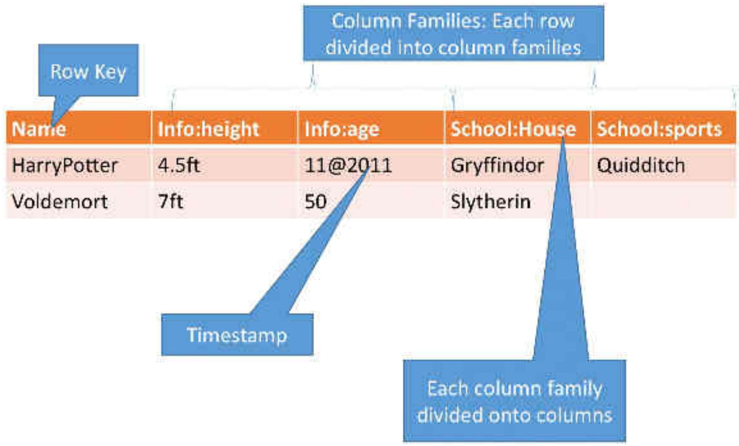
— Cassandra

- Distributed DB - P2P (Facebook)
- Inspired by DynamoDB

Hbase / Cassandra

- Both use same data model inspired by Bigtable

DATA MODEL : TERMINOLOGIES & CONCEPTS



Name	Info:height	Info:age	School:House	School:sports
HarryPotter	4.5ft	11@2011	Gryffindor	Quidditch
Voldemort	7ft	50	Slytherin	

Column Families

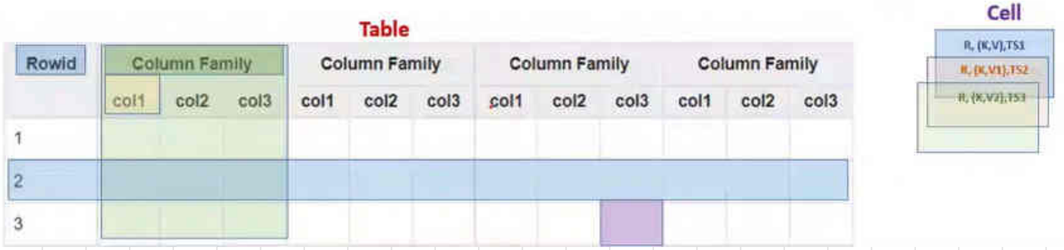
- Hbase schema: several tables
- Each table: set of column families
- Columns not part of schema
- Hbase: dynamic columns
- Column names encoded inside cells

Name	Data
HarryPotter	Info:{height:"4.5ft", age: "11@2011"} School:{House:"Gryffindor", Sports:"Quidditch"}
Voldemort	Info:{height:"7ft", age: "50"} School:{House:"Syltherin", Role:"Prefect"}

HBase Data Model

- Semi-structured

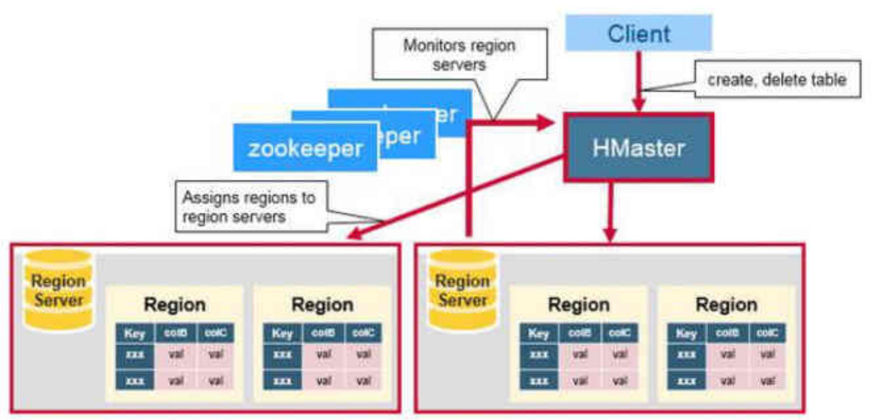
- Data partitioned into simpler components and spread across cluster



- **Table**: Data represented as a collection of rows sorted on RowID
- **Row:** Collection of column families identified by **RowID** (Row Key), a byte array, serving as the primary key for the table and is indexed for fast lookup
- **Column:** Collection of key-value pairs – represented by ColumnFamilyName:ColumnName
- **Column family**: Collection of variable number columns
- **Cell**: Stores data and is a combination of {row key, column, timestamp/version} tuple as a byte array
- Timestamp (System timestamp) or any other unique version number within a RowId, for the cell

Master-Slave Architecture

- Note: ppt links to mapr do not work as HPE has taken over



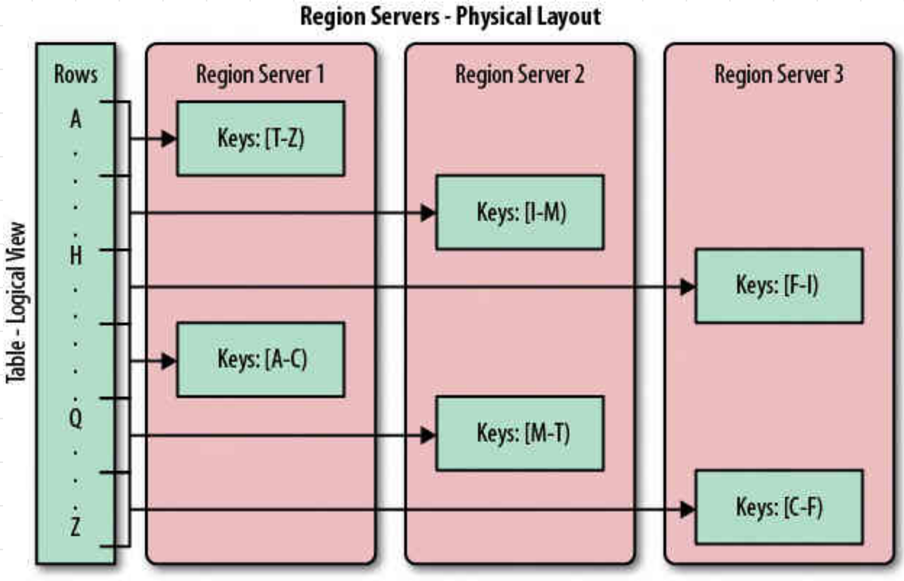
1. Master Server
   - Assigns region to region server
   - Detects failure using Zookeeper
   - Monitors RegionServers and load balances regions
   - Supports admin functions — schema changes, creation of tables, column families
   - Similar to NameNode of HDFS

2. Region Servers
   - contain regions
   - Communicates with client and manages data-related operations
   - Serves data for read/write for regions under it (using log)
   - Decides on region size
   - Similar to DataNodes of HDFS

3. Regions
   - Split parts of tables spread across region server
   - Subset of a table's rows
   - Automatically done

Region Servers

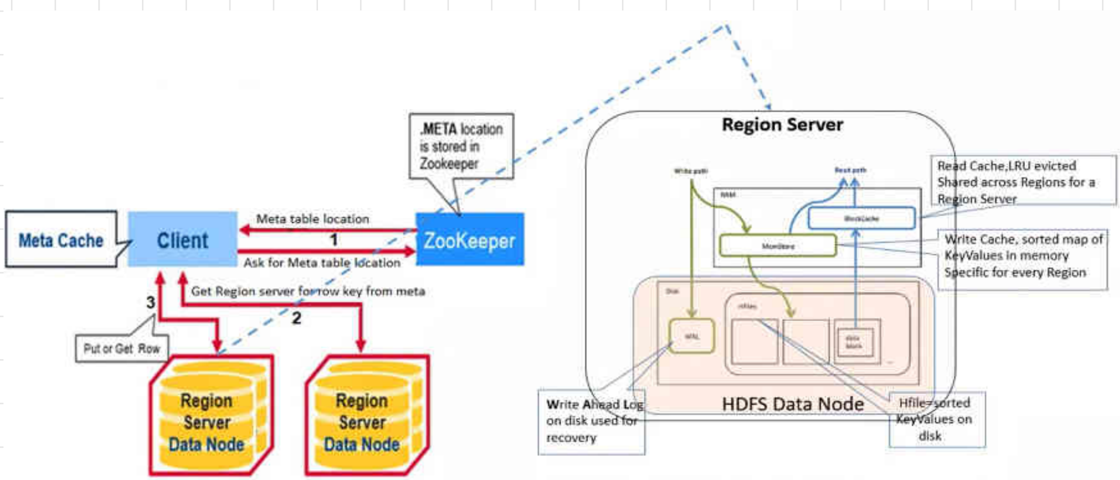
meta table: keeps track of regions



https://www.oreilly.com/library/view/hbase-the-definitive/9781449314682/ch01.html

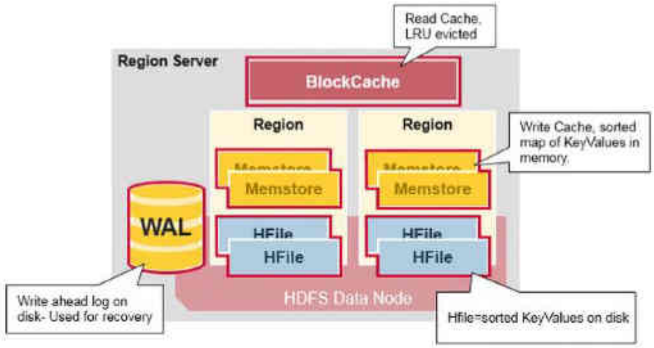
- Start with single region & modify dynamically

Read/Write Operations



Write-Ahead Log

- Before writing to DB
- Can restart from WAL if problem occurs
- Blockcache and Memstore in RAM
- HFiles and WAL in HDFS



Compments of Hbase

R3

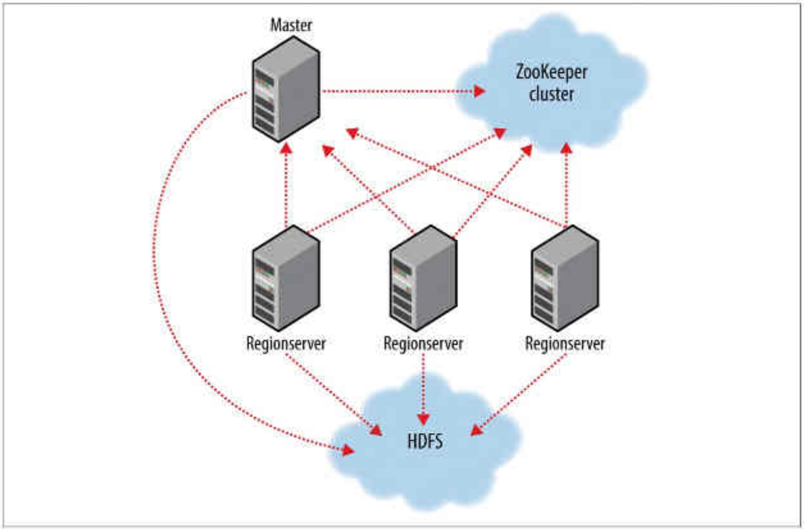
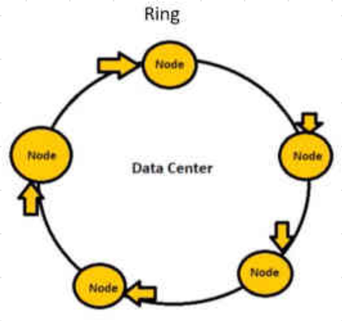


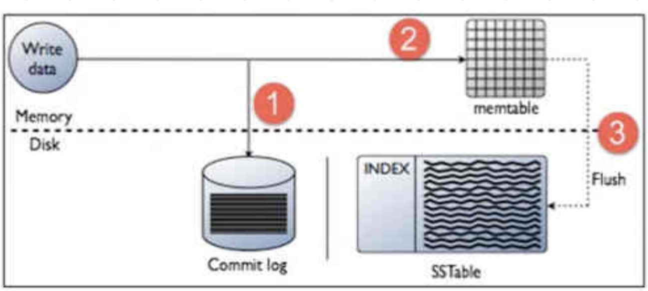
Figure 20-2. HBase cluster members

Cassandra

- Originates from Bigtable and Amazon's Dynamo DB
- Open Source columnar NOSQL DB, similar to Hbase
- P2P (not master-slave)
- Supports elastic scalability (allows no. of nodes in cluster to increase)
- Prevents failure with replication
- Can dynamically accomodate changes



- Uses consistent hashing - which machine contains what



- Data organised into partitions
- Partition: key, column names

Hbase Usage

1. Create table

table name

```
hbase(main):001:0> create 'test', 'data'
0 row(s) in 0.9810 seconds
```

column family name

2. Insert values

```
hbase(main):003:0> put 'test', 'row1', 'data:1', 'value1'
hbase(main):004:0> put 'test', 'row2', 'data:2', 'value2'
hbase(main):005:0> put 'test', 'row3', 'data:3', 'value3'
```

row key   column name   value

3. Retrieve values

specific row

```
hbase(main):006:0> get 'test', 'row1'
COLUMN                        CELL
 data:1                       timestamp=1414927084811, value=value1
1 row(s) in 0.0240 seconds
hbase(main):007:0> scan 'test'
ROW                           COLUMN+CELL
 row1                         column=data:1, timestamp=1414927084811, value=value1
```

all rows

QUESTIONS

Q: Combiners in MR execute on

(a) Map node (b) Reduce
(b) Network (d) Node manager

Q: Secondary NNs

(a) are used as backup for Active NN
(b) are used by ZK to help switch on a fault
(c) used to create FStree and relieve ANN of this task
(d) used to load balance requests to NN

Q: SRN NAME ADDRESS AADHAAR
students.csv on HDFS

AADHAAR VACCINATION STATUS
vaccstatus.csv on HDFS

Find no. of students vaccinated

(a) Write MR pseudocode
(b) keys at o/p of mapper?

ANSWERS

Q: Combiners in MR execute on

(a) Map node (b) Reduce
(b) Network (d) Node manager

Q: Secondary NNs

(a) are used as backup for Active NN
(b) are used by ZK to help switch on a fault
(c) used to create FStree and relieve ANN of this task
(d) used to load balance requests to NN

Q: SRN NAME ADDRESS AADHAAR
students.csv on HDFS

AADHAAR VACCINATION STATUS
vaccstatus.csv on HDFS

(a) Mapper1
- Read (AADHAAR, SRN) of students, o/p (AADHAAR, (stu, SRN))

Mapper 2
- Read (AADHAAR, SRN) of vaccine, o/p (AADHAAR, status) only if True

Reducer
- For each pair, increment count