NAGARJUNA COLLEGE OF ENGINEERING AND TECHNOLOGY COMPUTER SCIENCE AND ENGINEERING

Big Data (16CSI732)

By,

Ms. Priyanka k
Asst.Professor,CSE,
NCET

MODULE 2
NOSQL DATAMANAGEMENT-1

LECTURE 9
INTRODUCTION TO NOSQL

NOSQL: DEFINITION

- NoSQL databases is an approach to data management that is useful for very large sets of distributed data
  - NoSQL should not be misleading: the approach does not prohibit Structured Query Language (SQL)
  - And indeed they are commonly referred to as “NotOnlySQL”

NOSQL: MAIN FEATURES

- Non relational/Schema Free: little or no pre-defined schema, in contrast to Relational Database Management Systems
- Distributed
- Horizontally scalable: able to manage large volume of data also with availability guarantees
    - Transparent failover and recovery using mirror copies

CLASSIFICATION OF NOSQL DB

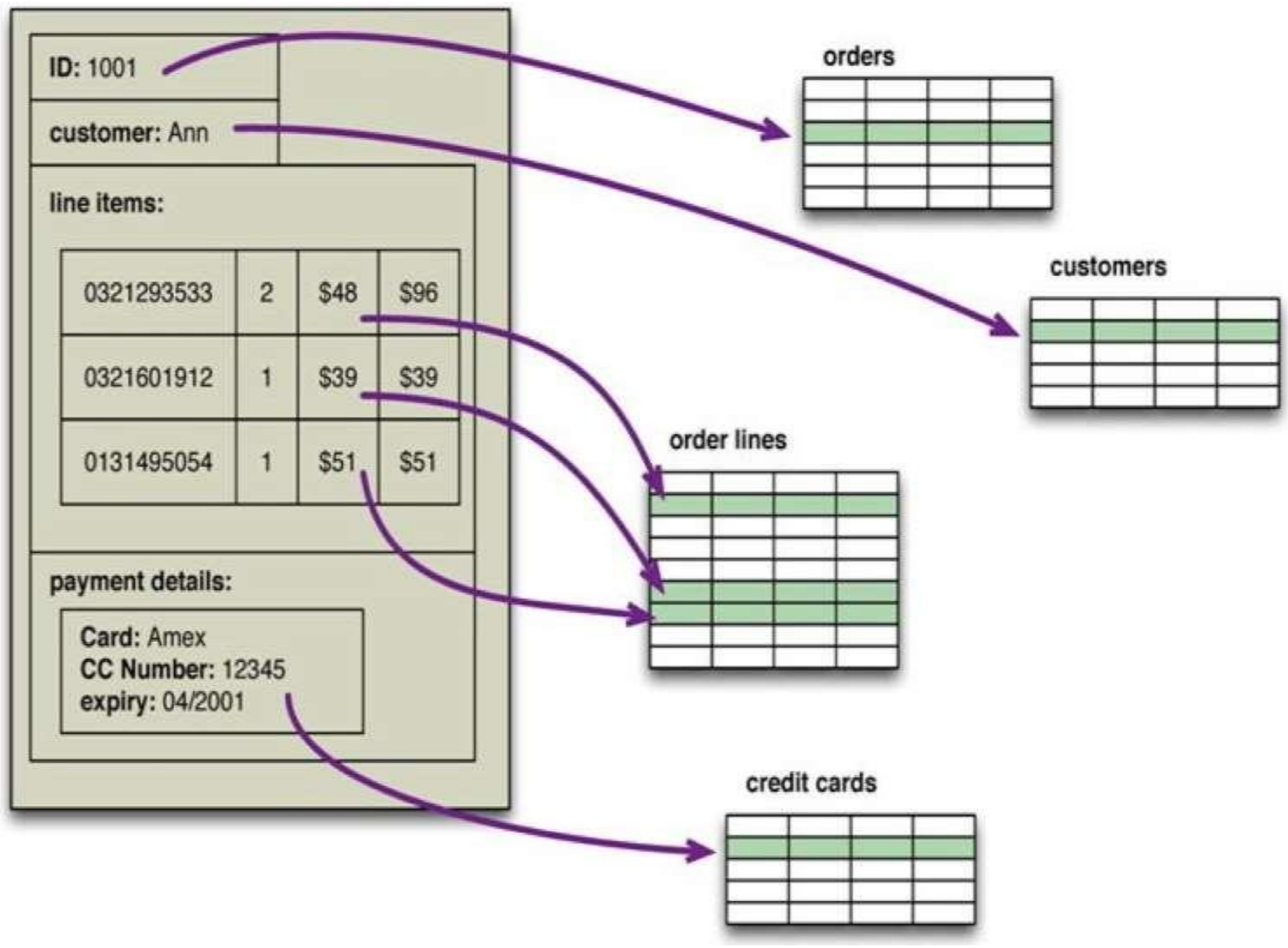
- Key-values
- Document store
- Column Oriented
- Graph database

The first three share a common characteristic of their data models which we will call aggregate orientation. An aggregate is a collection of related objects that we wish to treat as a unit

LECTURE 10
AGGREGATE DATA MODELS, AGGREGATES

AGGREGATE DATAMODELS

- **The relational model divides the information** that we want to store **into tuples** (rows): this is a very simple structure for data
- **Aggregate orientation** takes a different approach. It recognizes that often you want to operate on data in units that have a more complex structure
- It can be handy to think in terms of a complex record that allows lists and other record structures to be nested inside it
- Aswellse, key-value, document, andcolumn -family databasesalmakeuseofthismorecomplexrecord
- However, thereisnocommonterm forthiscomplexrecord; weuseherethetermn **ag regate**


ID: 1001
customer: Ann
line items:
0321293533 2 $48 $96
0321601912 1 $39 $39
0131495054 1 $51 $51
payment details:
Card: Amex
CC Number: 12345
expiry: 04/2001
orders
customers
order lines
credit cards

WHY AGGREGATE?

Two main aspects however should be emphasized:

- Dealing with aggregates makes it much easier for these databases to handle operating on a cluster, since the aggregate makes a natural unit for replication and sharding
- Also, it may help solving the impedance mismatch problem, i.e., the difference between the relational model and the in-memory data structures

Also Aggregates have specific consistency behaviour

LECTURE 11
KEY VALUE AND DOCUMENT DATAMODELS

KEY-VALUEAND DOCUMENTDATA MODELS

- Earlier on that key-value and document databases were strongly aggregate oriented.
- Both of these types of databases consist of lots of aggregates with each aggregate having a key or ID that's used to get at the data.
- The two models differ in that in a key-value database, the aggregate is opaque
- to the database—just some big blob of mostly meaningless bits.
- In contrast, a document database is able to see a structure in the aggregate.
- The advantage of opacity is that we can store whatever we like in the aggregate.
- The database may impose some general size limit, but other than that we have complete freedom.

- A document database imposes limits on what we can place in it, defining allowable structures and types.
- In return, however, we get more flexibility in access.
- With a key-value store, we can only access an aggregate by lookup based on its key.
- With a document database, we can submit queries to the database based on the fields in the aggregate.
- We can retrieve part of the aggregate rather than the whole thing, and database can create indexes based on the contents of the aggregate .
- With key-value databases, we expect to mostly look up aggregates using a key.
- With document databases, we mostly expect to submit some form of query based on the internal structure of the document; this might be a key, but it’s more likely to be something else.

COLUMN-FAMILYSTORES

- Acolumnfamilyisadatabaseobjectthatcontainscolumns ofrelateddata.
- Itisatulethatconsistsofakey-valuepair,wherethekey ismap edtoavaluethatisasetofcolumns.
- Acolumnfamilyisa“table”,eachkey-valuepairbeinga “row”.
- Eachcolumnisatupleconsistingofacolumnname,a valueandatimestamp.
- Inarelationaldatabasetable,thisdatawouldbegrouped togetherwithinatablewithothernon-relateddata.

Twotypesofcolumnfamiliesexist:

- Standardcolumnfamily: containsonlycolumns
- Supercolumnfamily: containamapofsupercolumns

LECTURE 12
RELATIONSHIPS, GRAPH DATABASES, SCHEMA LESS DATABASES

RELATIONSHIPS

- Aggregates are useful to put together data that is commonly accessed jointly
- But we still have cases when we need todeal with relationships.
- Also in the previous examples we modeled some relation between orders and customers.
- In this cases we want to separate order and customer aggregates but with some kind of relationship.
- The simplest way to provide such link is to embed the ID of the customer within the order’s aggregate data.

- In this way if we need data from a customerrecord,
  - We read the customer ID and
  - We make another call to the database to read the customer data
- This can be applied several times butthe database does not know about such relationship.
- However, provide a way to make a relationship visible to the database can be useful.
- For example,
  - Document store make the content of an aggregate available to the database to form indexes.
  - Some key-value store allows us to put link information in metadata, supporting partial lookup (Riak).

GRAPH DATABASE

- A graph database is a database that uses graph structures with nodes, edges, and properties to represent and store data
- A management systems for graph databases offers Create, Read, Update, and Delete (CRUD) methods to access and manipulate data
- Graph databases can be used for both OLAP (since are naturally multidimensional structures) and OLTP
- Systems tailored to OLTP (e.g., Neo4j) are generally optimized for transactional performance, and tend to guarantee ACID properties

SCHEMALESSDATABASES

NoSQL databases are schemaless:

- A key-value store allows you to store any data you like under a key
- A document database effectively does the same thing, since it makes no restrictions on the structure of the documents you store
- Column-family databases allow you to store any data under any column you like
- Graph databases allow you to freely add new edges and freely add properties to nodes and edges as you wish

SCHEMALESS DATABASES

This has various advantages:

- Without a schema binding you, you can easily store whatever you need, and change your data storage as you learn more about your project
- You can easily add new things as you discover them
- A schemaless store also makes it easier to deal with nonuniform data: data where each record has a different set of fields (limiting sparse data storage)

SCHEMALESS DATABASES

And also some problems

- Indeed, whenever we write a program that accesses data, that program almost always relies on some form of implicit schema: it will assume that certain field names are present and carry data with a certain meaning, and assume something about the type of data stored within that field
- Having the implicit schema in the application means that in order to understand what data is present you have to dig into the application code
- Furthermore, the database remains ignorant of the schema: it cannot use the schema to support the decision on how to store and retrieve data efficiently.
- Also, it cannot impose integrity constaints to maintain information coherent

LECTURE 13
MATERIALIZED VIEWS, DISTRIBUTION MODELS

WHAT IS AMATERIALIZED VIEW?

- A database objectthat stores the results of a query
  - Marries the query rewrite features found in Oracle Discoverer with the data refresh capabilities of snapshots

- Features/Capabilities
  - Can be partitioned and indexed
  - Can be queried directly
  - Can have DML applied against it
  - Several refresh options are available
  - Best in read-intensive environments

MATERIALIZED VIEWS

- Wi aggregate-models we stressed their advantage of aggregation
  - – If you want to access orders, it is useful to have orders stored in a single unit
- We know that aggregated orientation has the disadvantage over analytics queries.
- A first solution to reduce this problem is by building an index but we are still working against aggregates

MATERIALIZED VIEWS: RDBMS

- Relational databases offer another mechanism to deal with analytics queries, that is **views**.
- A view is like A relational table but defined by computation over the base tables
- When a views is accessed the databases execute the computation required.
- Or with materialized views are computed in advance and cached on disk

MATERIALIZED VIEWS: NOSQL

- NoSQL database do not have views, but they have pre-computed and cached queries.
- This is a central aspect for aggregate-oriented databases sincesome queries may not fit with the aggregate structure.
- Often, materialized views are created using map-reduce computation.

MATERIALIZED VIEWS: APPROACHES

- There are two approaches: Eager and Lazy. Eager
  - the materialized view is updated when the base tables are updated.
  - This approach is good when we have more frequent reads than writes
- Lazy:
  - The updates are run via batch jobs at regular interval
  - It is good when the data updates are not business critical

Moreover, we can create views outside the database.

- We can read the data, computing the view, and saving it back to the database (MapReduce).
- Often the databases support building materialized views themselves (Incremental MapReduce).
  - We provide the need computation and
  - The databases execute computation when needed

ADVANTAGES AND DISADVANTAGES OF MATERIALIZED VIEWS

- **Advantages**
  - Usefulforsummarizing, pre -computing, replicatingand distrib utin gdata
  - Fasteracces forexpensiveandco mplexjoins
  - Transparenttoend-users
    - MVscanbead ed/drop edwithoutinvalidatin gcod edSQL

- **Disadvantages**
  - Performanceco stso fmaintainingtheviews
  - Sto rageco stso fma in tai ni ngt he vie ws

DISTRIBUTION MODELS

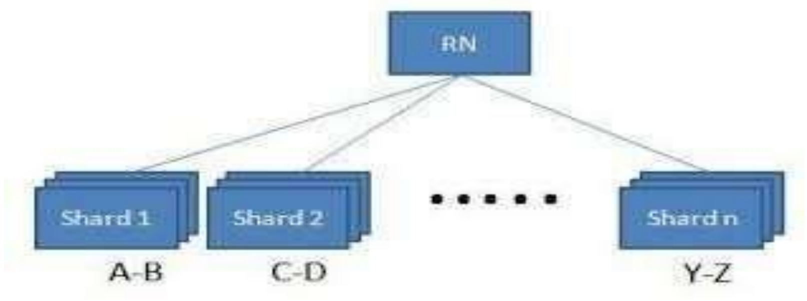
- We already discussed the advantages of scale up vs. scale out.
- Scale out is more appealing since we can run databases on a cluster of servers.
- Depending on the distribution model the data store can give us the ability:
  - To handle large quantity of data,
  - To process a greater read or write traffic
  - To have more availability in the case of network slowdowns of breakages
- These are important benefit, but they come at a cost. Running over a cluster introduces complexity.
- There are two path for distribution:
  - Replication and
  - Sharding

DISTRIBUTION MODEL: SINGLE SERVER

- It is the first and simplest distributionoption.
- Also if NoSQL database are designed to run on a cluster they can be used in a single server application.
- This make sense if a NoSQL database is more suited for the application data model.
- Graph database are the more obvious
- If data usage is most about processing aggregates, than a key or a document store may be useful.

SHARDING

- Often, a data store is busy because different people are accessing different part of the dataset.
- In this cases we can support **horizontal scalability** by putting different part of the data onto different servers (**Sharding**)
- The concept of sharding is not new as a part of application logic.
- It consists in put all the customer with surname A-D on one shard and E-G to another
- This complicates the programming model as the application code needs to distributed the load across the shards
- In the ideal setting we have each user to talk one server and the load is balanced.
- Of course the ideal case israre



SHARDING: APPROACHES

- In order to get the ideal case we have to guarantee that data accessed together are stored in the same node.
  - This is very simple using aggregates.
- When considering data distribution acrossnodes.
  - If access is based on physical location, we can place data close to where are accessed.
- Another factor is trying to keep data balanced.
- We should arrange aggregates so they are evenly distributed inorder that each node receive the same amount of the load.
- Another approach is to put aggregate together if we think they may be read in sequence (BigTable).
- In BigTable as examples data on web addresses are storedin reverse domain names.

SHARDING AND NOSQL

- In general, many NoSQL databases offers **auto- sharding.**
- This can make much easier to use sharding in an application.
- Sharding is especially valuablefor performance because it improves read and write performances.
- It scales read and writes on the different nodes of the same cluster.

MASTER-SLAVE REPLICATION

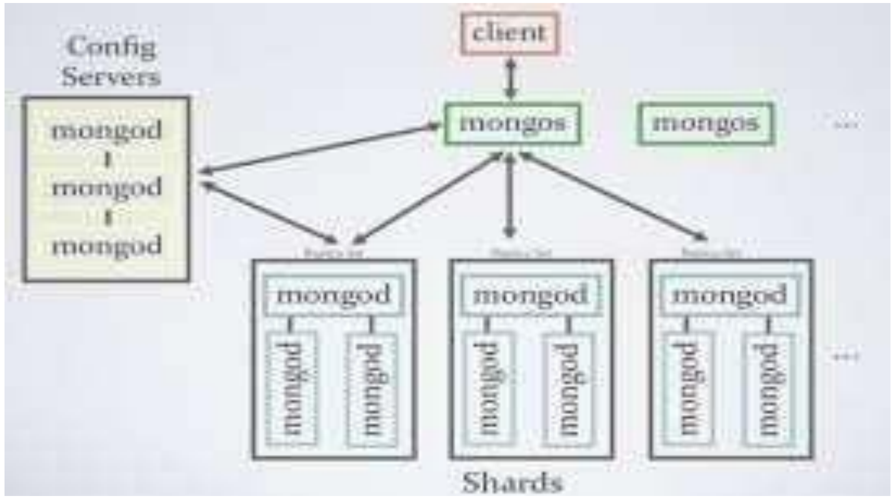
- In this setting one node is designated as the master, or primary ad the other as slaves.
- The master is the authoritative source for the date and designed to process updates and send them to slaves.
- The slaves are used for read operations.
- This allows us to scale in data intensive dataset
- We can scale horizontally by adding more slaves

- But, we are limited by the ability of the master in processing incoming data.

An advantage is **read resilience**.

- Also if the master fails the slaves can still handle read requests.
- Anyway writes are not allowed until the master is not restored.

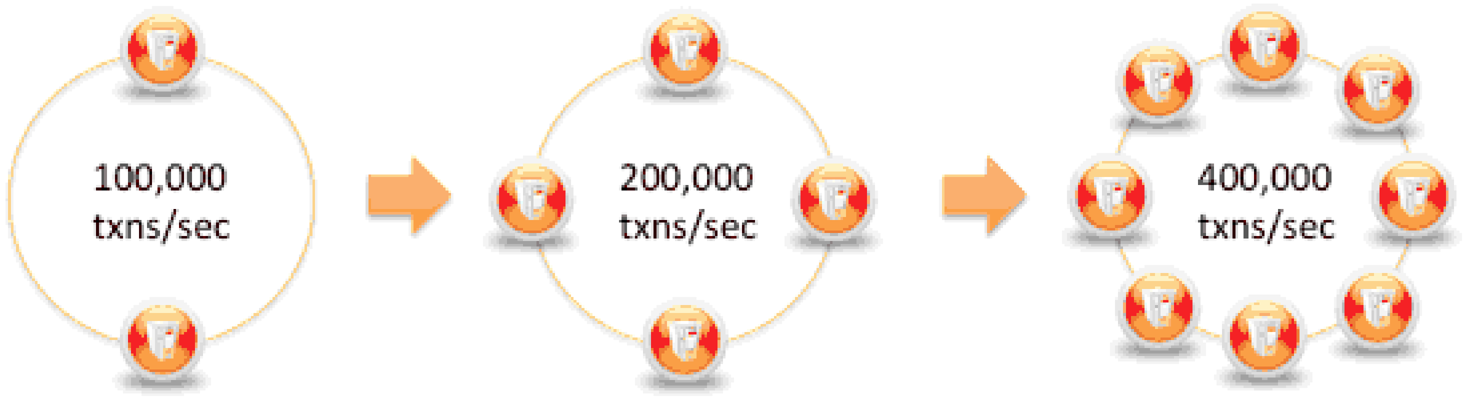
- Another characteristic is that a slave can be appointed as master.
- Masters can be appointed manually orautomatically.
- In order to achieve resilience we need that read and write paths are different.
- This is normally done using separate database connections.
- Replication in master-slave have the analyzed advantages but it come with the problem of **inconsistency**.
- The readers reading from the slaves can read data notupdated.



PEER-TO-PEER REPLICATION

- Master-Slave replication helps with read **scalability** but has problems on scalability of writes.
- It provides **resilience** on read but not on writes.
- The master is still a single point of failure.
- Peer-to-Peer attacks these problems by not having a master.
- All the replica are equal (accept writes and reads)
- With a Peer-to-Peer we can have node failures without lose write capability and losing data.

- Furthermore we can easily add nodes for performances.
- The bigger compliance here is**consistency**.
- When we can write on different nodes, we increase the probability to have inconsistency on writes.
- However there is a way to deal with thisproblem.



COMBINING SHARING WITH REPLICATION

- Master-slave and sharding: we have multiple masters, but each data has a single master.
  - Depending on the configuration we can decide the master for each group of data.
- Peer-to-Peer and sharding is a common strategy for column-family databases.
  - This is commonly composed using replication of the shards

VERSIONS

- Version stamps help you detect concurrency conflicts. When you read data, then update it, you can check the version stamp to ensure nobody updated the data between your read and write.
- Version stamps can be implemented using counters, GUIDs, content hashes, timestamps, or a combination of these.
- With distributed systems, a vector of version stamps allows you to detect when different nodes have conflicting updates.

LECURE 14
MAP REDUCE

MAP REDUCE

- Map-reduce is a pattern to allow computations to be parallelized over a cluster.
- The map task reads data from an aggregate and boils it down to relevant key-value pairs. Maps only read a single record at a time and can thus be parallelized and run on the node that stores the record.
- Reduce tasks take many values for a single key output from map tasks and summarize them into a single output. Each reducer operates on the result of a single key, so it can be parallelized by key.
- Reducers that have the same form for input and output can be combined into pipelines. This improves parallelism and reduces the amount of data to be transferred.

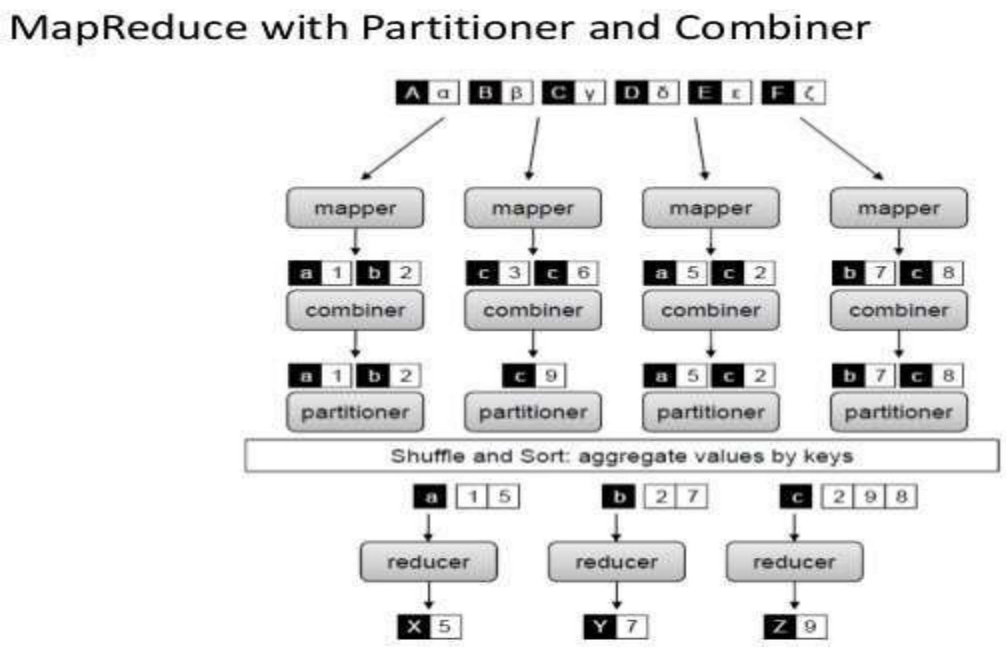
- Map-reduce operations can be composed into pipelines where the output of one reduce is the input to another operation's map.
- If the result of a map-reduce computation is widely used, it canbe stored as a materialized view.
- Materialized views can be updated through incrementalmap-reduce operations that only compute changes to the view instead of recomputing everything from scratch.

LECTURE 15
PARTITIONING AND COMBINING

PARTITIONING AND COMBINING

- **Partitioning** of the keys of the intermediate map output iscontrolled by the Partitioner.
- By hash function, key (or a subset of the key) is used to derivethe partition.
- According to the **key-value** each mapper output is partitioned and records having the same key value go into the same partition (within each mapper), and then each partition is sent to a reducer.
- Partition class determines which partition a given (key, value) pair will go.
- Partition phase takes place after map phase and before reducephase.
- The combiner in MapReduce is also known as'Mini-reducer'.
- The primary job of Combiner is to process the output data from the Mapper, before passing it to Reducer.
- It runs after the mapper and before the Reducer and its use is optional

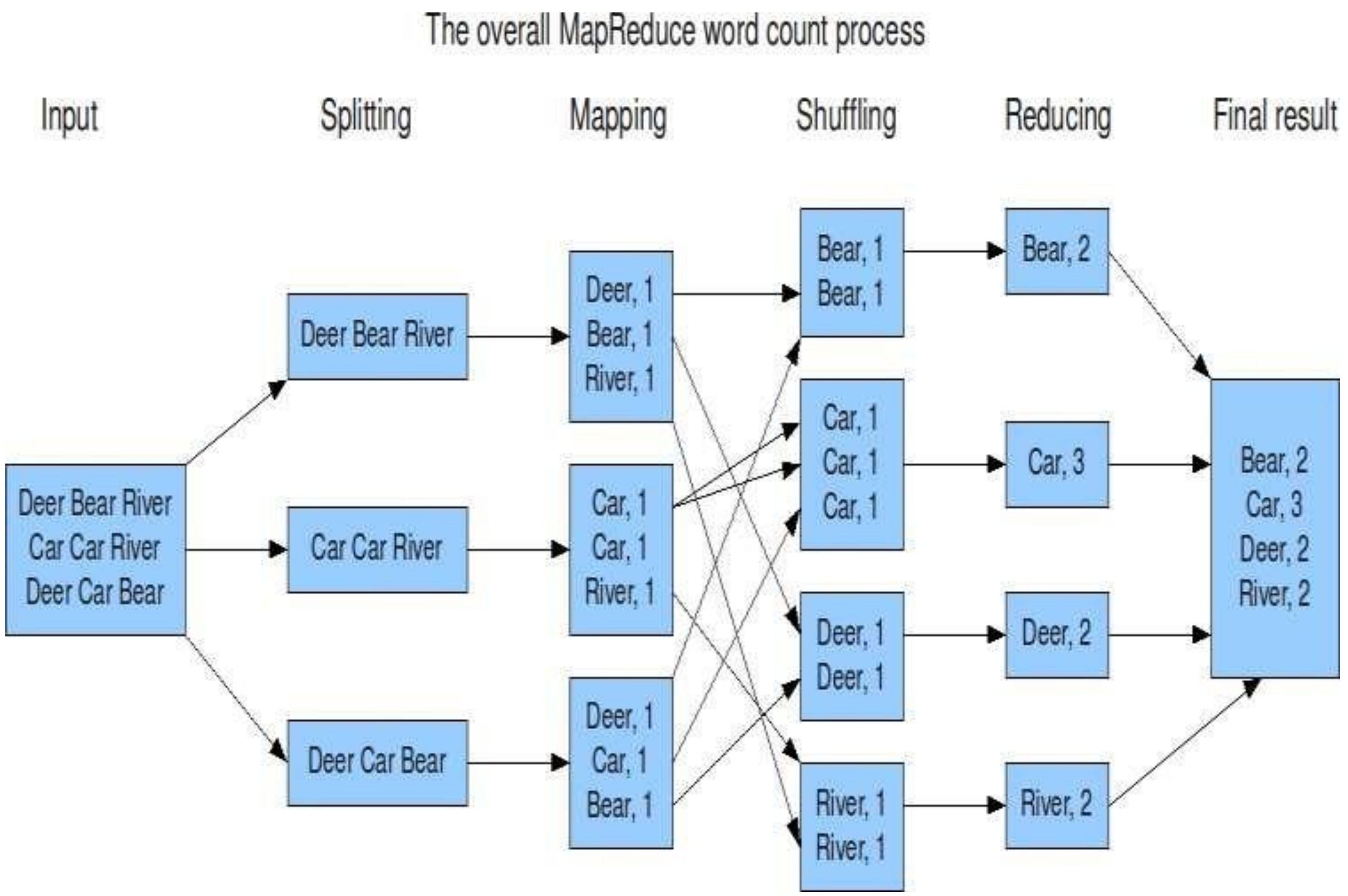
- On a large dataset when we run MapReduce job, large chunks of intermediate data is generated by the Mapper and this intermediate data is passed on the Reducer for further processing, which leads to enormous network congestion.
- MapReduce framework provides a function known as **Hadoop Combiner** that plays a key role in reducing network congestion.



LECTURE 16
COMPOSING MAP-REDUCE CALCULATIONS

COMPOSING MAP-REDUCE CALCULATIONS

- The map-reduce approach is a way of thinking about concurrent processing that trades off flexibility in how you structure your computation for a relatively straightforward model for parallelizing the computation over a cluster.
- Since it’s a tradeoff, there are constraints on what you can doin your calculations.
- Within a map task, you can only operate on a singleaggregate.
- Within a reduce task, you can only operate on a single key.
- This means you have to think differently about structuring your programs so they work well within these constraints.


The overall MapReduce word count process
Input
Splitting
Mapping
Shuffling
Reducing
Final result
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, 1
Bear, 1
Car, 1
Car, 1
Car, 1
Deer, 1
Deer, 1
River, 1
River, 1
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2