Hadoop Map-Reduce NoSQL Database

By Akash Desai, Nihal Parchand, Viraj Chaudhari

Overview

- Challenges:
  - Input data is too large (PetaBytes)
  - Straightforward repetitive work

- Issues:
  - Recovering from machine failure
  - Debugging and Optimization
  - Communication and coordination
  - Locality
  - Scheduling

Limitations of RDBMS

- Only structured Data
- Average sized data (GBs)
- Scalability (Vertical)
- Licensed
- Static schema (Pre-defined)
- Eg: PostgreSQL, MySQL, Oracle, Microsoft SQL server etc.

Benefits of NoSQL

- Structured, semi-structured and unstructured data
- Large datasets (TBs and PBs)
- Scalability (Horizontal)
- Dynamic Schema
- Eg: Cassandra, MongoDB, BigTable, HBase, Neo4j, CouchDB, MapReduce etc.

Introduction of Hadoop

- Apache Open Source Framework
- Stores large heterogeneous dataset on distributed system
- Process data using MapReduce Framework
- Data storage using Hadoop Distributed File System
- Hive Query Language to access data
- Different components - Hive, Pig, MapReduce, Mahout, Latin, Oozie, HDFS etc.

HDFS ARCHITECTURE

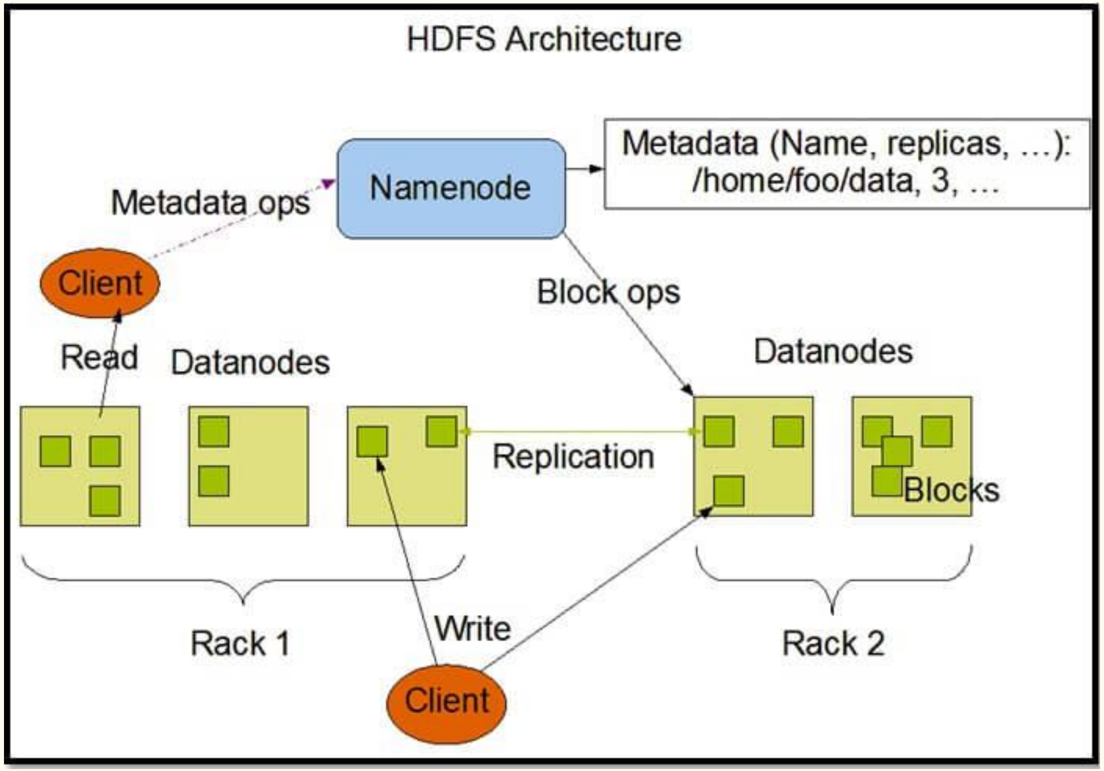


Image source: https://www.mssqltips.com/tipimages2/3180_HDFS_Architecture.jpg

Map-Reduce History

- Popularized as programming model by Jeffery Dean and Sanjay Ghemawat in 2004.
- Google used it to collect and analyze website data for search optimizations.
- Apache started using map-reduce in the subproject Apache Lucern. Provides text search capabilities across large databases.
- In 2007, Doug Cutting released Hadoop as open source Apache project.

RDBMS	Map-Reduce
Use B-Tree for updates	Use Sort/Merge for updates
Many read/write operations continually	Write once, read many times
Structured data (XML documents/Tables)	Semi-structured data (Spreadsheet) and unstructured data (plain text/image data)
Schema-on-write	Schema-on-read

Map-Reduce Overview

- Simple programming model (Map + Reduce) that applies to many large-scale computing problems
- Distributed implementation hides messy details:
  - Fault Tolerance
  - Parallelization
  - I/O Scheduling
  - Network and Disk transfer optimization

Map-Reduce Overview

- Two types of nodes:
  - Cluster (Same local network and similar hardware)
  - Grid (Shared across geographically and heterogeneous hardware)
- Data stored in :
  - Filesystem (Unstructured)
  - Database (Structured)

Map-Reduce Operations

- Map - Process the I/P key-value pair and produce set of O/P intermediate key/value pairs.

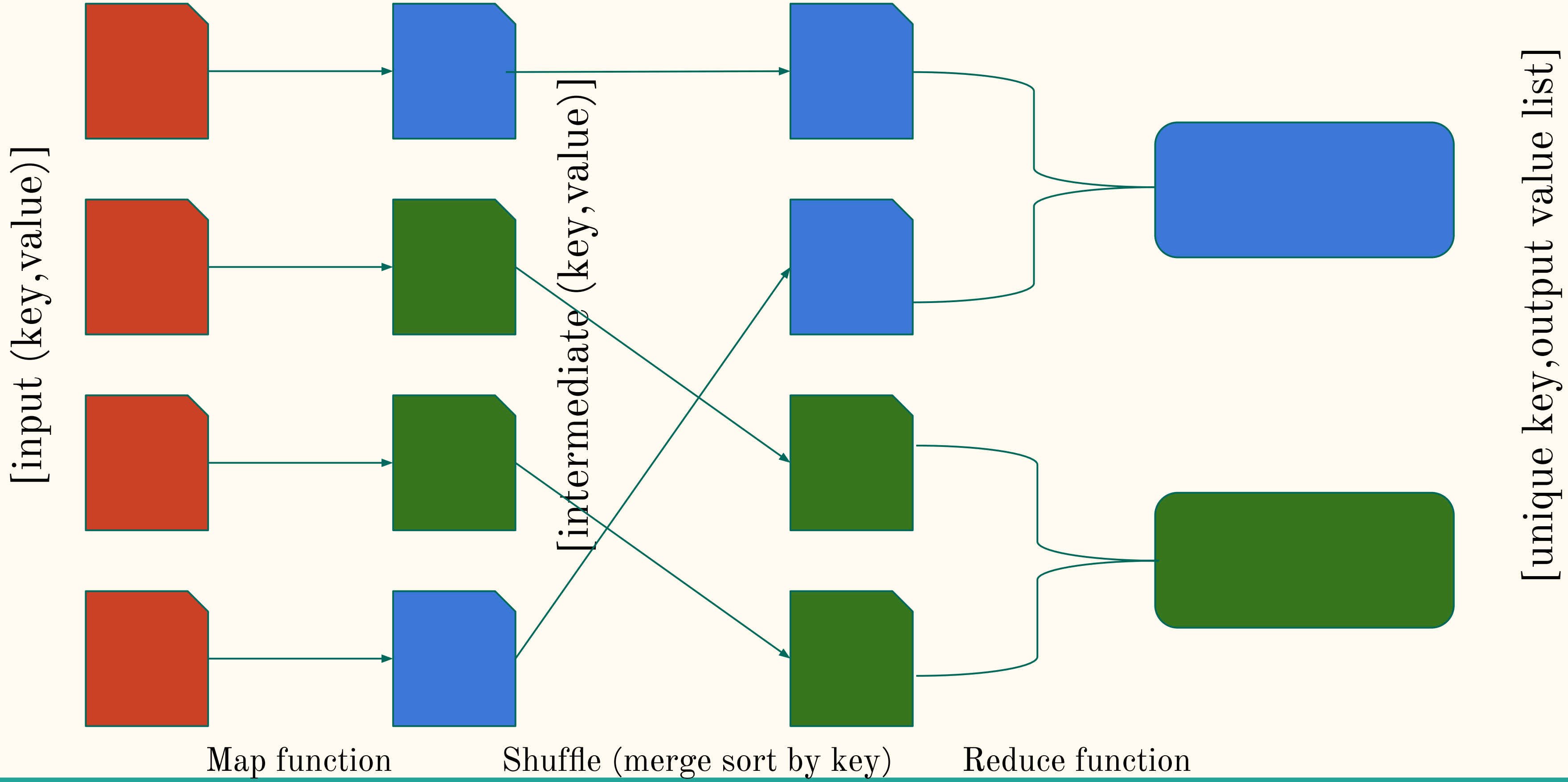
  map(in_key,in_value) -> list(out_key,interm_value)

- Shuffle - Use sort/merge to combine intermediate results.

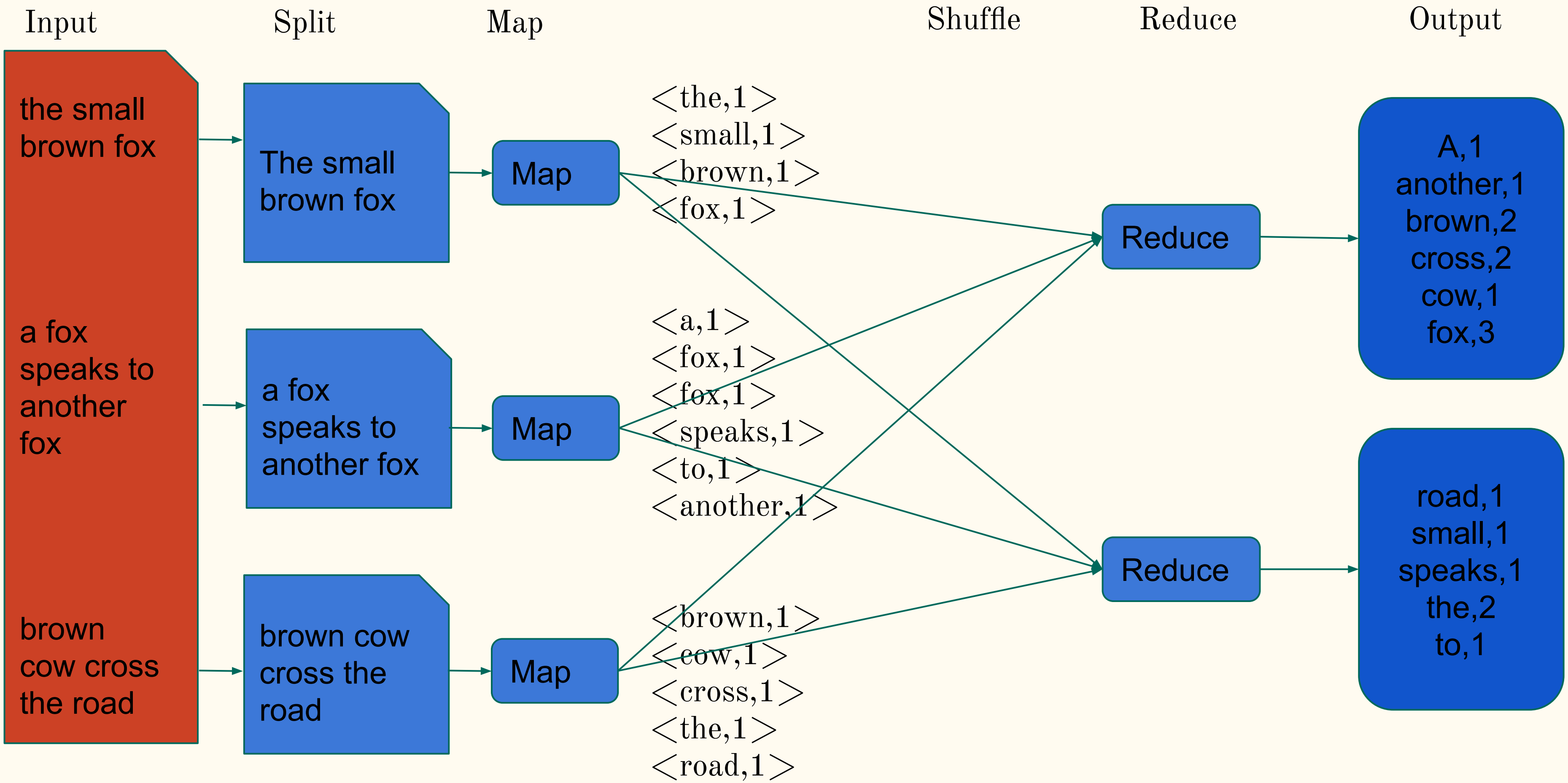
- Reduce - Process the intermediate key/value, combine intermediate values for each unique key and generate a set of merged output values

  reduce(out_key,interm_value) -> list(out_value)

Map-Reduce Programming Model


[input (key,value)]
[intermediate (key,value)]
[unique key,output value list]
Map function
Shuffle (merge sort by key)
Reduce function

Map-Reduce WordCount Example



Input

the small brown fox

a fox speaks to another fox

brown cow cross the road

Split

The small brown fox

a fox speaks to another fox

brown cow cross the road

Map

Map

Map

Map

<the,1>
<small,1>
<brown,1>
<fox,1>

<a,1>
<fox,1>
<fox,1>
<speaks,1>
<to,1>
<another,1>

<brown,1>
<cow,1>
<cross,1>
<the,1>
<road,1>

Shuffle

Reduce

Reduce

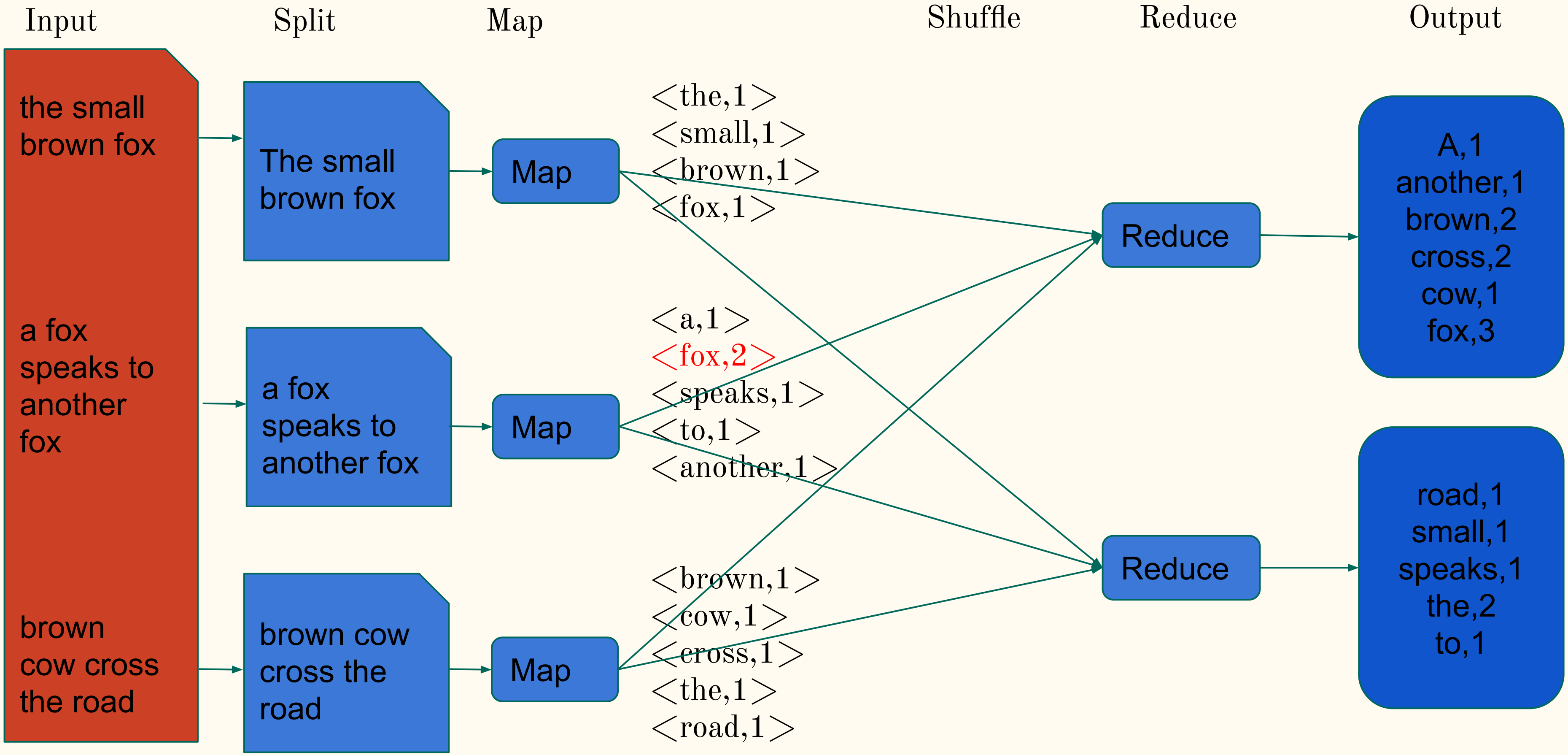
Reduce

Output

A,1
another,1
brown,2
cross,2
cow,1
fox,3

road,1
small,1
speaks,1
the,2
to,1

Map-Reduce WordCount Example (Combiner)



Map-Reduce Indexing

- Index based on File URI
  - Indexed query will be equivalent to full scan query
- Index based on InputSplit
  - Indexed query better than full scan.
  - Performance is optimized

Map-Reduce Performance

- Optimized shuffle operation and writing only Map and Reduce function.
- Combiner - reduce data written to disk
- Complexity of mapping, shuffle, sorting, and reducing
- Fault Tolerance can be handled by re-execution
  - Worker failure
  - Master failure

Map-Reduce Refinement

- Redundant execution
- Skipping bad records
- Backup tasks
- Locality optimization
- Optional secondary keys for ordering

Transaction Management

ACID

- Atomicity
- Consistency
- Isolation
- Durability

BASE

- Basically available
- Soft state
- Eventual consistency

2 PHASE COMMIT PROTOCOL

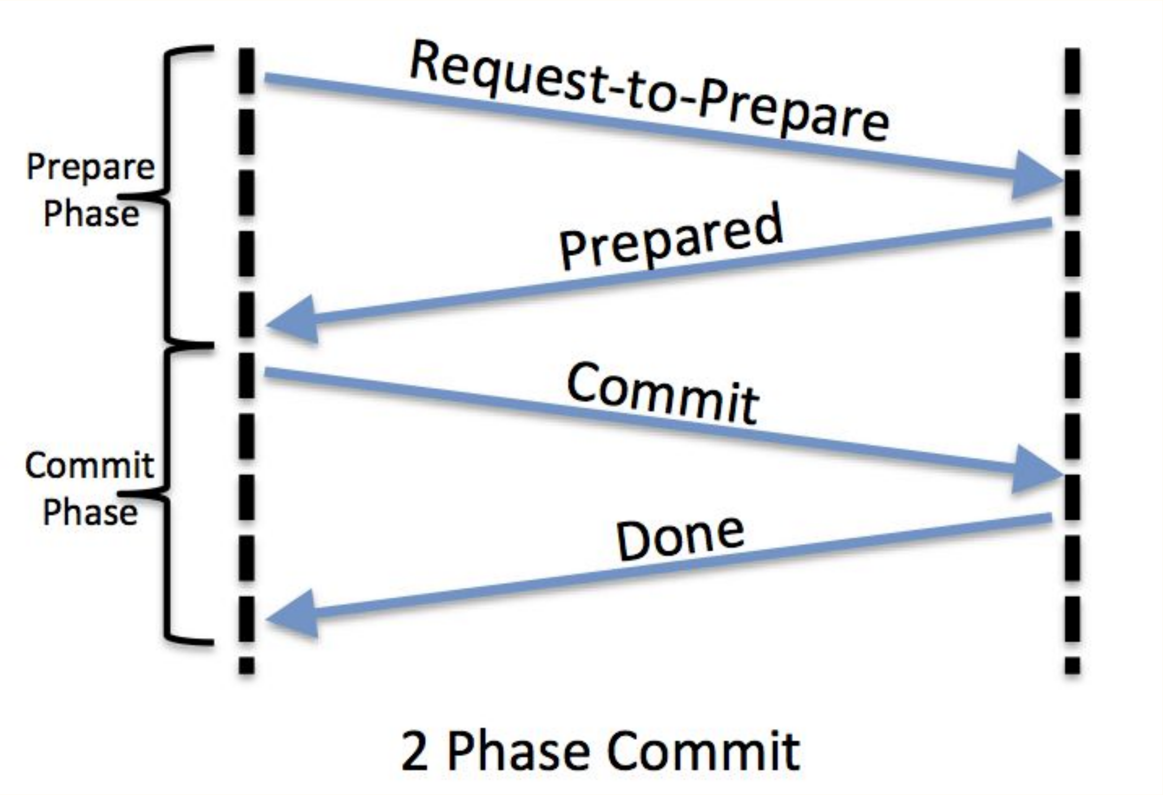


Image source: https://www.tigerteam.dk/wp-content/uploads/2014/03/2-phase-commit-protocol-flow.png

Map-Reduce Uses

- Distributed pattern-based searching
- Distributed sorting
- Web link-graph reversal
- Inverted index construction
- Document clustering

Application

- Cloudera VM-Oracle Virtual Box virtualization software provides the HDFS instances as well as all the services in the Hadoop framework, which can be installed separately. The components include the HDFS file system, Hive, Impala, MapReduce, PigLatin.

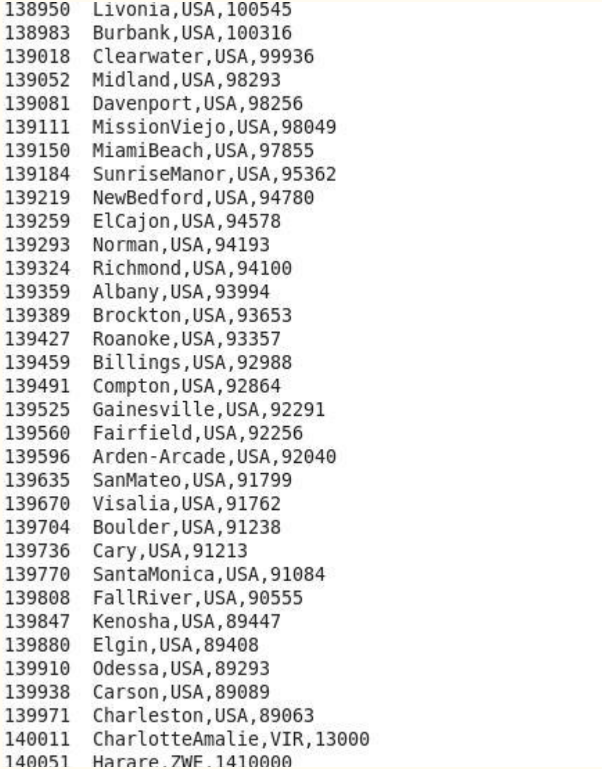
- CentOs operating system provides terminal support.

- City database as an input with columns - ID, Name, CountryCode, District, Population. The database is stored as a CSV input in the HDFS folder.

- We implemented java code, which takes SQL queries as an input and generates output using MapReduce Framework.

RESULT

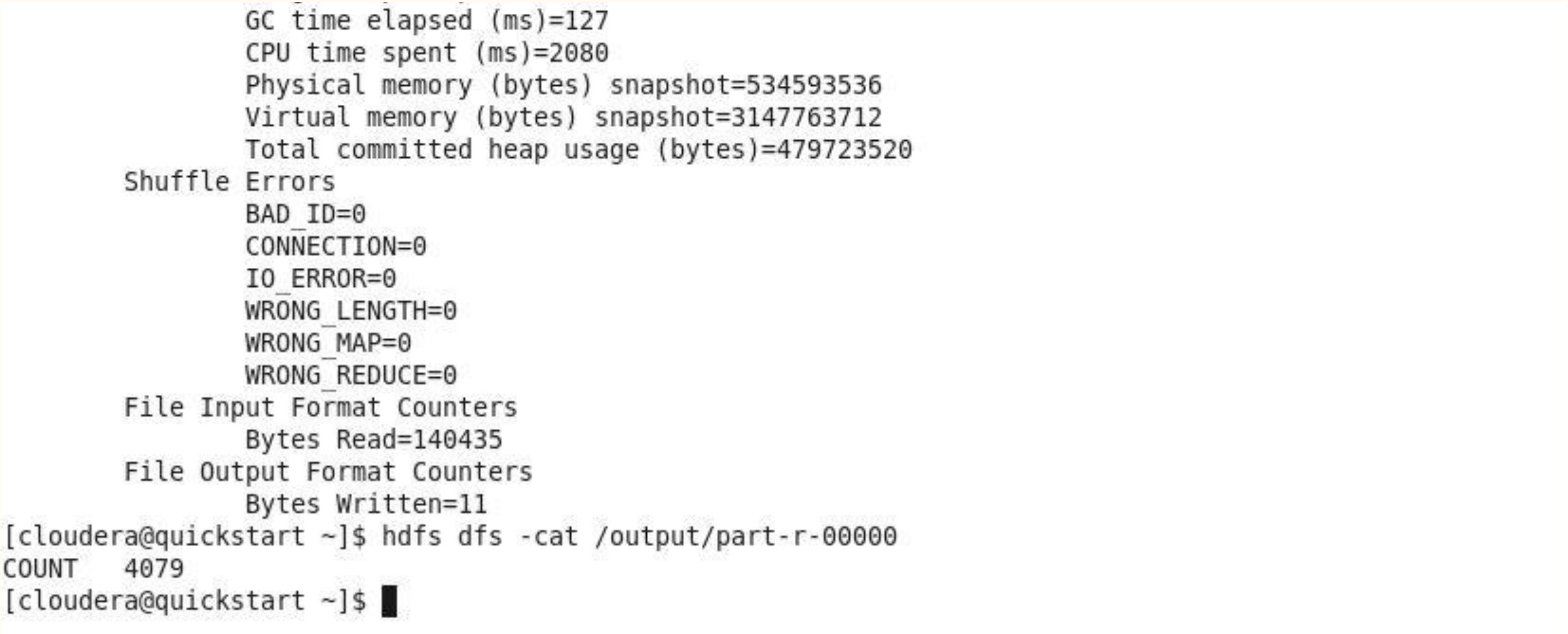
select name, countrycode, population from city



```
138950  Livonia,USA,100545
138983  Burbank,USA,100316
139018  Clearwater,USA,99936
139052  Midland,USA,98293
139081  Davenport,USA,98256
139111  MissionViejo,USA,98049
139150  MiamiBeach,USA,97855
139184  SunriseManor,USA,95362
139219  NewBedford,USA,94780
139259  ElCajon,USA,94578
139293  Norman,USA,94193
139324  Richmond,USA,94100
139359  Albany,USA,93994
139389  Brockton,USA,93653
139427  Roanoke,USA,93357
139459  Billings,USA,92988
139491  Compton,USA,92864
139525  Gainesville,USA,92291
139560  Fairfield,USA,92256
139596  Arden-Arcade,USA,92040
139635  SanMateo,USA,91799
139670  Visalia,USA,91762
139704  Boulder,USA,91238
139736  Cary,USA,91213
139770  SantaMonica,USA,91084
139808  FallRiver,USA,90555
139847  Kenosha,USA,89447
139880  Elgin,USA,89408
139910  Odessa,USA,89293
139938  Carson,USA,89089
139971  Charleston,USA,89063
140011  CharlotteAmalie,VIR,13000
140051  Harare,ZWE,1410000
```

RESULT

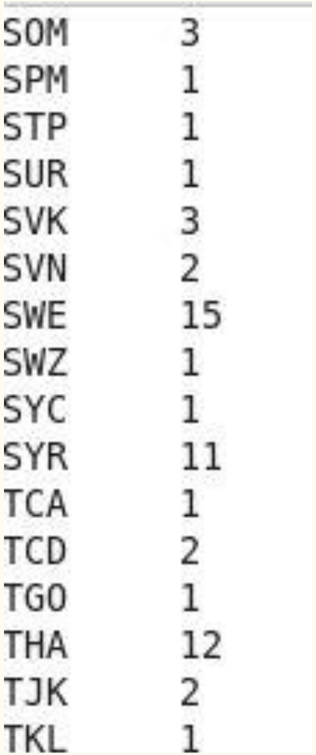
select count(*) from city



```
                GC time elapsed (ms)=127
                CPU time spent (ms)=2080
                Physical memory (bytes) snapshot=534593536
                Virtual memory (bytes) snapshot=3147763712
                Total committed heap usage (bytes)=479723520
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=140435
        File Output Format Counters
                Bytes Written=11
[cloudera@quickstart ~]$ hdfs dfs -cat /output/part-r-00000
COUNT   4079
[cloudera@quickstart ~]$
```

RESULT

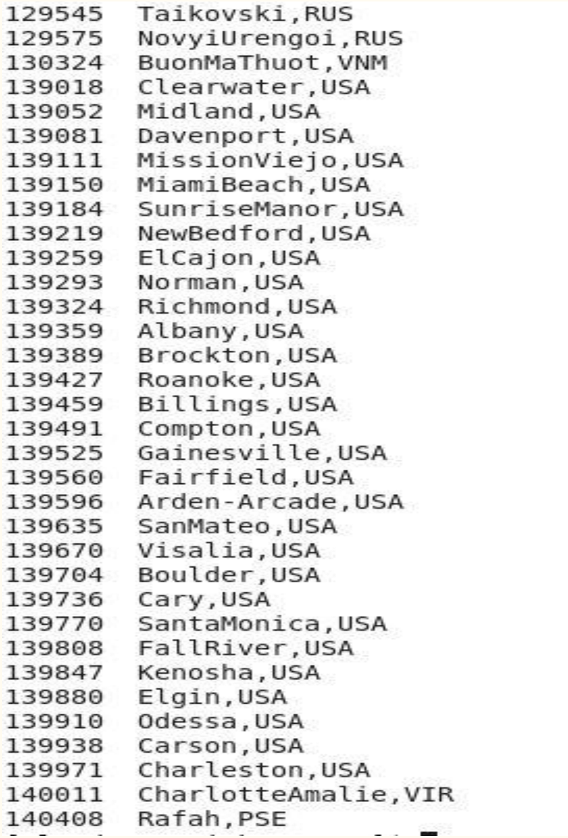
select count(*) from city group by countrycode



```
SOM        3
SPM        1
STP        1
SUR        1
SVK        3
SVN        2
SWE        15
SWZ        1
SYC        1
SYR        11
TCA        1
TCD        2
TGO        1
THA        12
TJK        2
TKL        1
```

RESULT

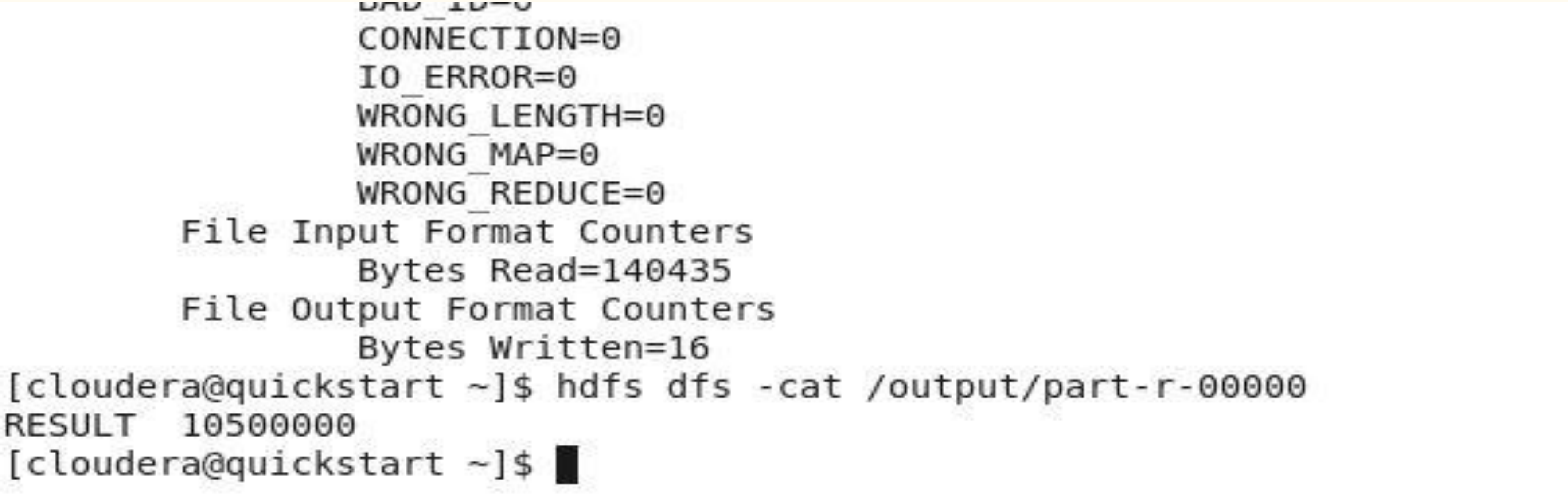
. select name, countrycode from city where population < 100000



```
129545   Taikovski,RUS
129575   NovyiUrengoi,RUS
130324   BuonMaThuot,VNM
139018   Clearwater,USA
139052   Midland,USA
139081   Davenport,USA
139111   MissionViejo,USA
139150   MiamiBeach,USA
139184   SunriseManor,USA
139219   NewBedford,USA
139259   ElCajon,USA
139293   Norman,USA
139324   Richmond,USA
139359   Albany,USA
139389   Brockton,USA
139427   Roanoke,USA
139459   Billings,USA
139491   Compton,USA
139525   Gainesville,USA
139560   Fairfield,USA
139596   Arden-Arcade,USA
139635   SanMateo,USA
139670   Visalia,USA
139704   Boulder,USA
139736   Cary,USA
139770   SantaMonica,USA
139808   FallRiver,USA
139847   Kenosha,USA
139880   Elgin,USA
139910   Odessa,USA
139938   Carson,USA
139971   Charleston,USA
140011   CharlotteAmalie,VIR
140408   Rafah,PSE
```

RESULT

select MAX(population) from city



```
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=140435
        File Output Format Counters
                Bytes Written=16
[cloudera@quickstart ~]$ hdfs dfs -cat /output/part-r-00000
RESULT  10500000
[cloudera@quickstart ~]$ █
```

Conclusion

- Inexpensive (Free to use)
- Hides all the messy details- Implement only Map and Reduce functions
- Provides easy model to implement parallel programs
- Performance of MapReduce is better when the volume of data is large i.e. in PetaBytes.

References

- MapReduce: Simplified Data Processing on Large Clusters

  https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf
- HADOOP vs RDBMS|Know The 12 Useful Differences

  https://www.educba.com/hadoop-vs-rdbms/
- Hadoop: The Definitive Guide
- https://en.wikipedia.org/wiki/MapReduce
- https://hadoopi.wordpress.com/2013/05/24/indexing-on-mapreduce-2/
- https://www.slideshare.net/rantav/introduction-to-map-reduce

Thank You