Big Data Technologies

Agenda

- Apache Hadoop
  - YARN
  - Uber Job
  - Hadoop Streaming
- Apache HBase

Apache Hadoop

Chained/Cascaded Map Reduce Jobs

- Complex processing may not be completed in single MR job.
- Such processing can be done by feeding output of first job as input to second job. This way multiple jobs can be chained to each other.
- Need to update driver code. Run second job, only if first job is successful.
- Example: Find the year with maximum avgerage temperature from NCDC dataset.
  - NCDC data --> AvgTemperatureJob --> (Year,AvgTemp) * 20 --> MaxTemperatureJob --> (Year,MaxTemp) * 1
  - AvgTemperatureJob
    - NCDC data (20 yrs) --> AvgTempMapper --> AvgTempReducer --> (Year,AvgTemp) * 20
      - Input: /user/nilesh/ncdc/input
      - AvgTempMapper
        - Input: offset, line (record)
        - Output: year, temperature
      - AvgTempReducer
        - Input: year, temperatures
        - Output: year, avgtemp
      - Output: /user/nilesh/ncdc/auxoutput
  - MaxTemperatureJob

        - (Year,AvgTemp)*20 --> MaxTempMapper --> MaxTempReducer --> (Year,MaxTemp)
            - Input: /user/nilesh/ncdc/auxoutput
            - MaxTempMapper
                - Input: offset, line (year \t avgtemp)
                - Output: null, "year,avgtemp" (Text)
            - MaxTempReducer
                - Input: null, "year,avgtemp" * 20 (Text) -- Single group
                - Output: year, maxtemp * 1
            - Output: /user/nilesh/ncdc/output
    - NcdcDriver -- run()
        - Create job1 and submit.

```
Job job1 = Job.getInstance(conf, "AvgTempJob");
job1.setMapperClass(AvgTempMapper.class);
// ...
job1.setReducerClass(AvgTempReducer.class);
// ...
TextInputFormat.addInputPaths(job1, "/user/nilesh/ncdc/input");
TextOutputFormat.setOutputPath(job1, new Path("/user/nilesh/ncdc/auxoutput"));

job1.submit();
boolean success = job1.waitForCompletition(true);
if(!success)
    return 1;
```

        - Create job2 and submit.

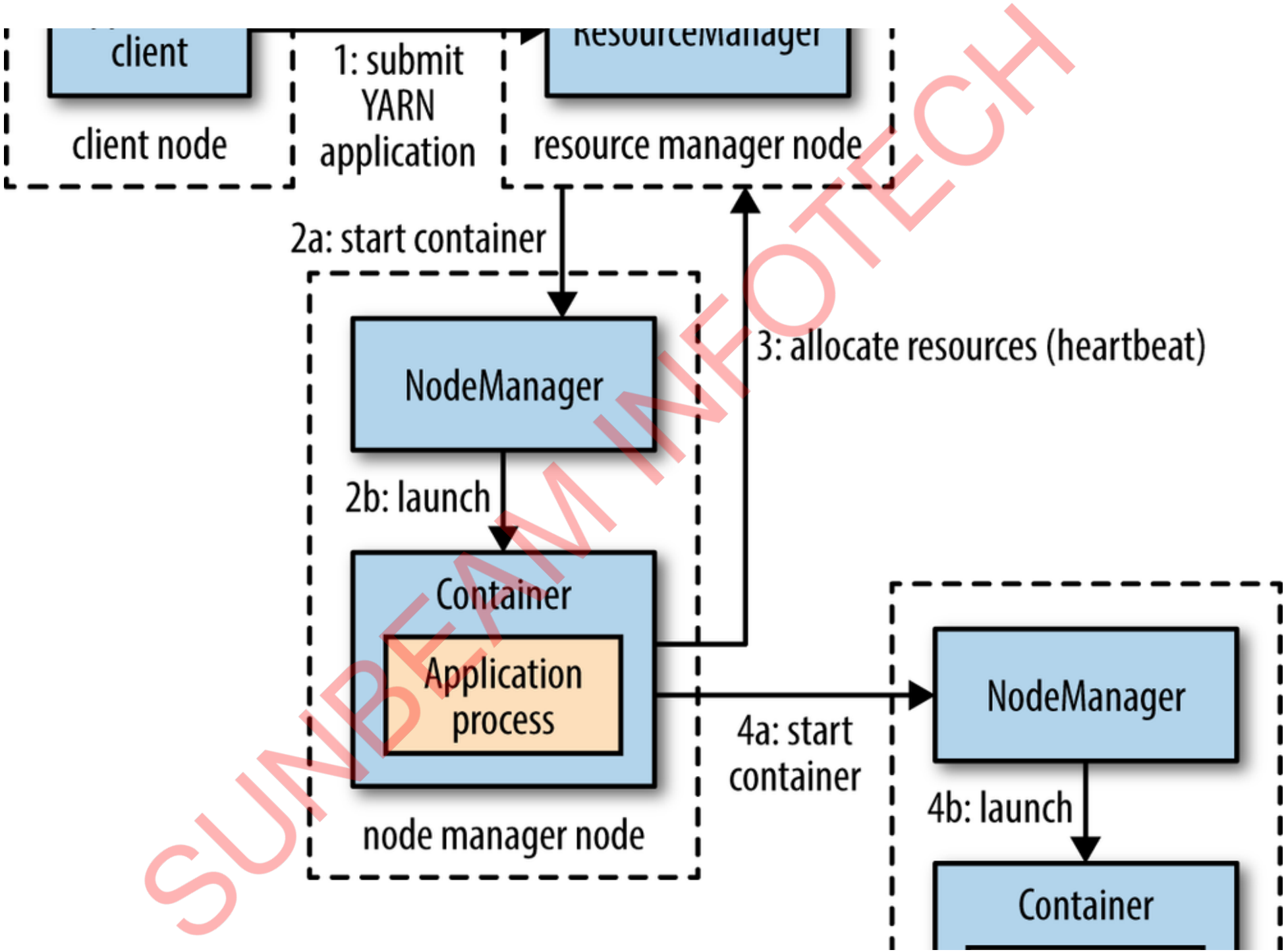
```
Job job2 = Job.getInstance(conf, "MaxTempJob");
job2.setMapperClass(MaxTempMapper.class);
// ...
job2.setReducerClass(MaxTempReducer.class);
```

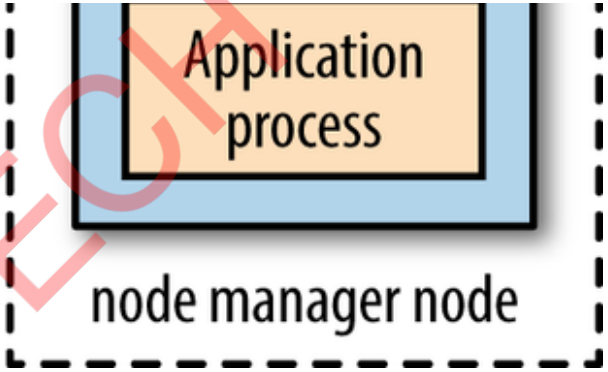
```
// ...
TextInputFormat.addInputPaths(job2, "/user/nilesh/ncdc/auxoutput");
TextOutputFormat.setOutputPath(job2, new Path("/user/nilesh/ncdc/output"));

job2.submit();
success = job2.waitForCompletition(true);
```

YARN

- Apache Hadoop YARN sits between HDFS and the processing engines being used to run applications.
- YARN combines a central resource manager with containers, application coordinators and node-level agents that monitor processing operations in individual cluster nodes.
- YARN schedulers
  - YARN supports multiple scheduling methods, all based on a queue format for submitting processing jobs.
  - The default FIFO Scheduler runs applications on a first-in-first-out basis. However, that may not be optimal for clusters that are shared by multiple users.
  - The Fair Scheduler tool assigns each job running at the same time its "fair share" of cluster resources, based on a weighting metric that the scheduler calculates.
  - The Capacity Scheduler, enables Hadoop clusters to be run as Multi-tenant systems shared by different units in one organization or by multiple companies, with each getting guaranteed processing capacity based on individual service-level agreements.
- Apache Hadoop YARN decentralizes execution and monitoring of processing jobs by separating the various responsibilities into these components:
  - A global ResourceManager that accepts job submissions from users, schedules the jobs and allocates resources to them.
  - A NodeManager slave that's installed at each node and functions as a monitoring and reporting agent of the ResourceManager.
  - An ApplicationMaster that's created for each application to negotiate for resources and work with the NodeManager to execute and monitor tasks.
  - Resource containers that are controlled by NodeManagers and assigned the system resources allocated to individual applications.

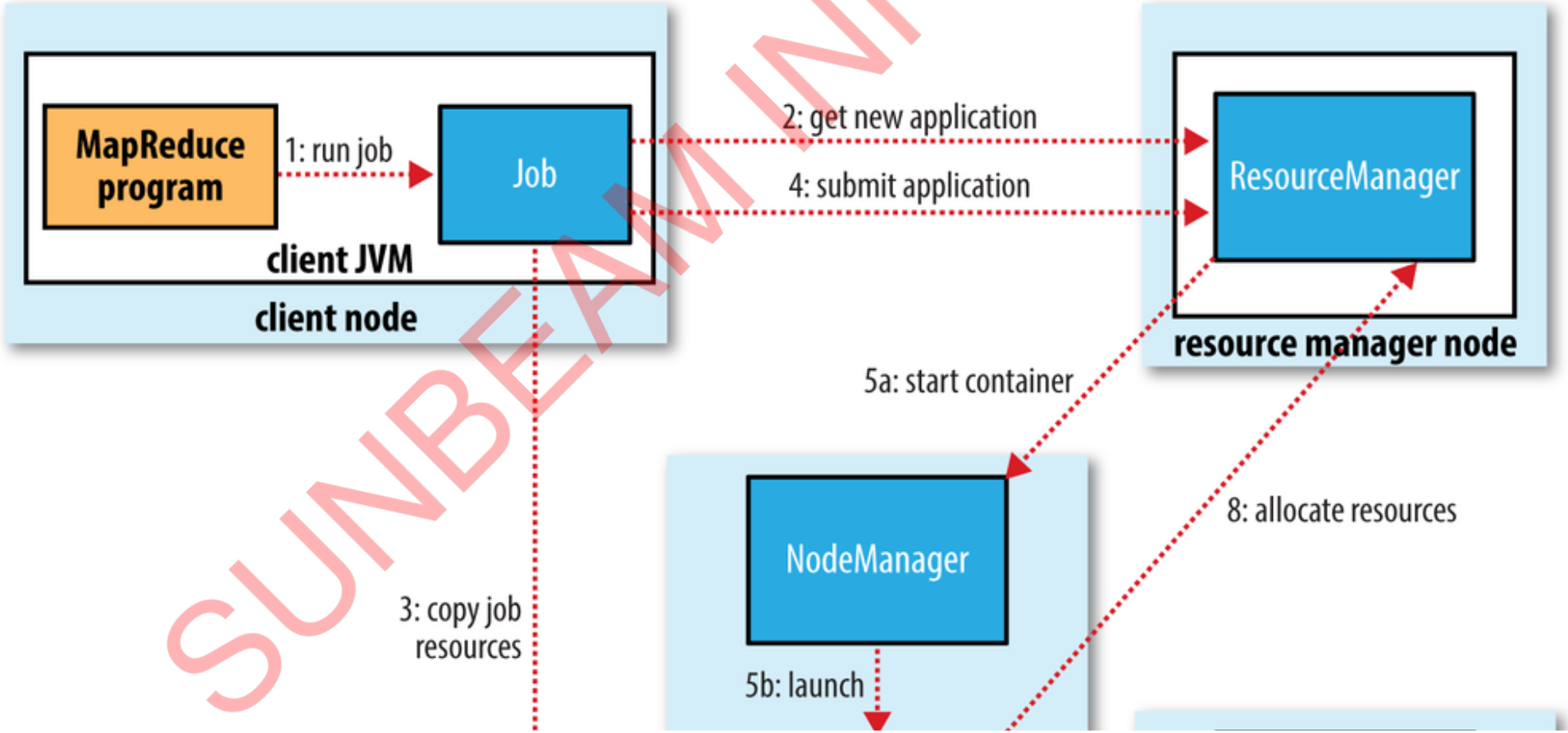
YARN Architecture

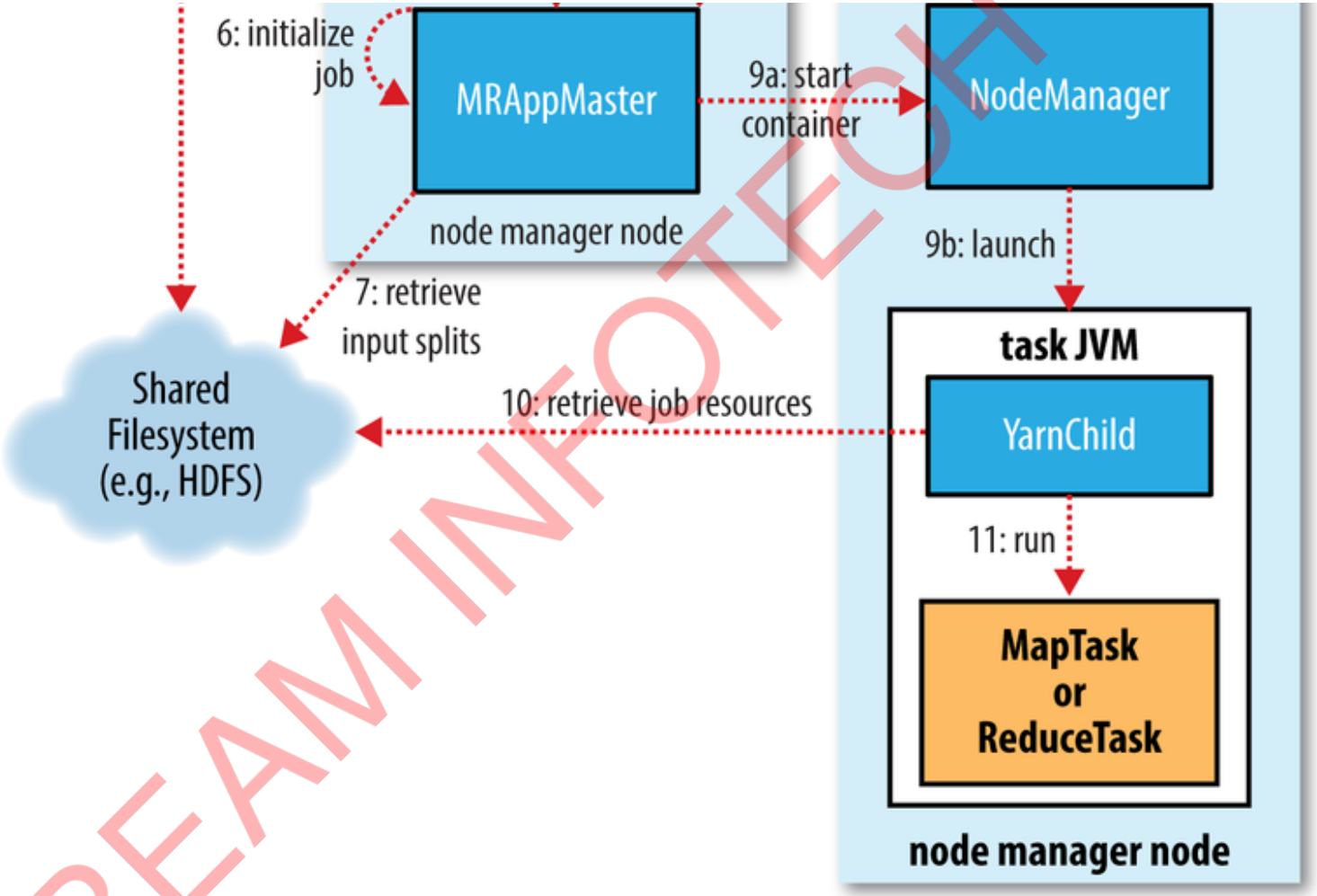




-

Map-Reduce Programs on YARN





-

Uber job

- By default for each MR job separate MRAppMaster, Mapper(s) and Reducer(s) processes (containers) are created.
- For small data processing this process creation and their communication is overhead.
- Uber mode allows running such small jobs in single container i.e. MRAppMaster.
- Mapper(s) & Reducer runs in same process. As no IPC involves, these small jobs are executed quickly.
- It is configured using settings
    - mapreduce.job.ubertask.enable (default: false)

- mapreduce.job.ubertask.maxmaps (default: 9)
- mapreduce.job.ubertask.maxreduces (default: 1)

Hadoop Streaming

- Hadoop Streaming enable writing Hadoop MR job in any programming language capable of reading input from terminal (stdin) and writing output on terminal (stdout).
- Internally, the HDFS data/Mapper output is redirected (input) to Mapper/Reducer task and their output will be redirected to Sort stage/HDFS.
  - HDFS --> Mapper --> Sort stage
  - Merge stage --> Reducer --> HDFS
- Word Count Map-Reduce using Python.
  - mapper.py

```
#!/usr/bin/python3
import sys

for line in sys.stdin:
    words = line.split()
    for word in words:
        print(f"{word}\t1")
```

  - reducer.py

```
#!/usr/bin/python3
import sys
di = dict()
for line in sys.stdin:
    (word,cnt) = line.split()
    newcnt = di.get(word,0) + cnt
    di[word] = newcnt
for word,total in di.items():
    print(f"{word}\t{total}")
```

- Execution

```
> hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.2.jar\
    -files mapper.py,reducer.py\
    -mapper mapper.py\
    -reducer reducer.py\
    -input /user/nilesh/wordcount/input\
    -output /user/nilesh/wordcount/output
```

Apache HBase

- HBase is an open-source non-relational distributed database.
- It is modeled after Google's Big-Table.
- HBase is developed in Java as part Apache Hadoop project.
- HBase contributed by developers at Facebook, Cloudera, Hortonworks, etc.
- It is cluster based database running on top of Hadoop HDFS.
- HBase data files are stored in HDFS.

HBase development:

- 2006: Google Big Table paper
- 2007: Hadoop's contrib
- 2008: Hadoop's sub-project
- 2010: Apache top level project
- 2011: HBase 0.92 release

HBase vs HDFS

- Distributed systems to scale to thousands of nodes.
- HDFS – Batch processing over big files

    - Not good for record lookup.
    - Not good for small incremental batches.
    - Not designed for update & delete.
    - Huge storage: 100s PB
- HBase – Distributed Columnar database
    - Low latency record lookup (by row id/key).
    - Support for inserting & updating records (CRUD operations).
    - Large storage: 10s PB

HBase vs RDBMS

- HBase
    - NoSQL database - Flexible schema
    - Commands in Ruby language
    - Connectivity from different programming languages: Java, Python, ...
    - Large storage: 10s PB
    - Column-wise storage
    - High speed lookup (by key)
    - Not fully ACID compliant
- RDBMS
    - Fixed schema
    - Commands in SQL language
    - Connectivity from different programming languages: Java, Python, ...
    - Relative less storage: few TBs
    - Row-wise storage
    - Good for grouping & joins
    - Fully ACID compliant

HBase Data Model

- HBase being NoSQL, is a schema-less (flexible) database. Columns can be added on the fly.
- Sparse tables have lot of null values and not stored by HBase to save disk space.

- It persists all its data in underlying HDFS. Hence it is reliable, scalable, high performance at cost of distributed servers.
- Each record is associated with a key and is stored in sorted order of keys.
- Each row in table is indexed by row key.
- Data store can store one or more tables.
- Column oriented database:
    - Each table can have one or more column families.
    - Each column family have one or more columns.
    - Columns in family may be different for each row.
    - Columns can be added in family dynamically.
- Intersection of row and column is a cell.
- All cells, row ids, even table, column family & column names are stored as byte array.
- Thus any data type of any size can be stored in each cell.
- Empty cells aren't stored.
- Key & Version numbers are replicated with each column family.
- Multiple versions of the values (for each update) as per timestamps (by default).
    - With each edit, a new version of the cell is created with new time stamps.
    - Internally stores versions in desc order of time-stamps.

HBase Data Storage

- Different column families can have different properties and access patterns.
- Configurable column properties are cache usage, compression (none, gzip, LZO), version retention policies.
- Each column family is stored in a separate file (called HFile), which is then partitioned into regions.

HBase Compactions

- Compaction, the process by which HBase cleans up itself.
- Minor compactions combine (a configurable number of smaller HFiles into one larger HFile. Due to combining data together, disk access speed increases.
- Major compaction combine all HFiles into one large HFile. It also removes extra versions & deleted cells as per config.

HBase RegionServers

- They are software processes/daemons responsible for store and retrieve data in HBase. Running on each node in cluster.
- When a table grows beyond threshold (as per config), it is auto split & distributes the load to another node. This is called auto-sharding.
- Each split is separate region managed by a rgn server.
- Each column family store has a read cache called the BlockCache and a write cache called the MemStore.
- BlockCache helps with random read performance.
- The Write Ahead Log (WAL, for short) ensures that our Hbase writes are reliable.
- The design of HBase is to flush column family data stored in the MemStore to one HFile per flush. HFile is finally stored in HDFS blocks.

HBase Master

- Monitor region servers in cluster.
- Handle metadata operations.
- Assign regions (after split) & balance load.
- Manage region server failover.
- Manage and clean catalog tables.
- Clear the WAL (Write Ahead Logs).
- Usually a backup copy of master server is maintained to handle failover of master. Newer versions allow upto 9 backup masters.

HBase ZooKeeper

- ZooKeeper is a distributed cluster that provides reliable coordination & synchronization services for clustered applications.
- This team consists of ZooKeeper Leader and ZooKeeper Follower(s).
- HBase comes with an instance of ZK that coordinates operations of master server, region server(s) and client(s).

CAP and ACID properties

- CAP theorm
    - C : Consistency
        - HBase is consistent as same data is visible from any node (because of HDFS).
    - P : Partition Tolerance
        - HBase is tolerant i.e. if any node goes down the data can be still accessible (because of Replica).
    - A : Available

        - HBase doesn't guarantee 100% uptime i.e. each request may not get response (success/failure).
- HBase is not ACID compliant like RDBMS as it doesn't support Isolation.
    - A: Atomic - HBase guarantees update single record atomically
    - C: Consistent - HBase is in consistent stage - No PK/FK relations.
    - D: Durable - HBase changes are durable in HDFS.
    - I: Isolation - HBase doesn't guarantee changes sequentially.

HBase Access

- HBase shell - Ruby shell with set of commands
- Web interface - Browser on port 16010
- Java API - Native APIs for HBase
- REST (HTTP) - REST API on port 8080
    - hbase rest start -p
- Thrift - Enable access from other languages
- Hive/Pig for Analytics

User defined MR job

- We can implement MR job to process the data stored in HBase.
- For HBase data input & output, we should use TableInputFormat and TableOutputFormat (provided in HBase API).
- To create mapper (to read data from HBase table)
    - class MyHBaseTableMapper extends `TableMapper<KeyOut,ValueOut>`
        - Mapper input: Key=Row Key, Value=Column Values
        - All input data is available as byte-array.
- Implement driver class using TableMapReduceUtil
    - TableMapReduceUtil.initTableMapperJob("books", scan, MyHBaseTableMapper.class, Text.class, DoubleWritable.class, job);
    - It internally set up -- InputFormat as TableInputFormat.
- hadoop jar /path/to/job.jar pkg.HBaseBooksDriver /tmp/booksummary

HBase Shell Commands:

- https://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands/

HBase Applications

- HBase is preferred for random reads, random writes or both.
- Do not use HBase if random access is not required.
- HBase can perform thousands of operations per seconds on TBs of data.
- HBase performs well if access pattern is well known & simple.
- Used by: Facebook, Mozilla, Twitter, OpenLogic, Meetup, ...