BIG DATA

UNIT - 1

Introduction

feedback/corrections: vibha@pesu.pes.edu

VIBHA MASTI

Modelling Pipeline

Input data → Model → Analysis

Model: human construct to abstractify real-world systems / phenomena

Big Data Themes

1. Manage very large amount of data (eg: google search engine)
2. Extract value and knowledge (eg: recommendation system)

MACHINE TRANSLATION

- Translate from one language to another
- Two approaches: traditional approach & big data approach
- Traditional approach
  - rule-based
  - understanding of structure
- Big data
  - domain knowledge not necessary
  - patterns analysed using large amounts of data
  - https://www.google.co.in/amp/s/www.wired.com/2008/06/pb-theory/amp
- Domain knowledge required?
  - https://towardsdatascience.com/machine-translation-a-short-overview-91343ff39c9f
  - Peter Norvig: https://youtu.be/yvDCzhbjYWs

- Domain knowledge needed to check validity of model

PITFALLS

1. Spurious Correlation

- $C \to A$ & $C \to B$, does $A \to B$
- Eg: stork population & human birth rate — hidden variable: available nesting area

2. Gaps in the Data

- Due to selection bias, covenience

Error Checking of Models

- Nate Silver's book: *The Signal and the Noise: Why So Many Predictions Fail – but Some Don't* — weather forecasting with human adjustment

- All data: signal + noise

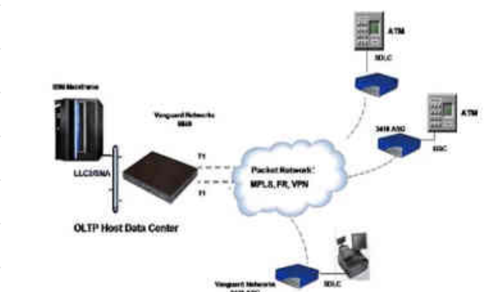
- Train on training set, estimate error on testing set

- Purely empirical estimation of errors

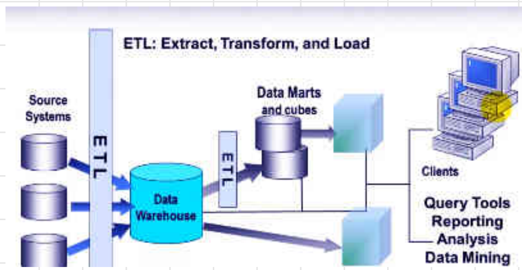
IBM — 4 V's of Big Data

https://www.sliceofbi.com/2015/09/basics-of-big-data.html?m=1

1. Volume



http://www.techbridge.solutions/home/solutions/solutions-ibm-applications/oltp-host-concentrator-solution/



- Old style data:
  - fixed format / schema
  - clean data
  - consistent data
  - batch processing, not real-time
- 44x increase from 2009 to 2020
- Generated by financial services, energy, media etc.

- BD: automated

2. Variety
   - variety of formats (video, photo, text etc)
   - Static data & streaming data
   - Extract knowledge → link various sources together

3. Velocity
   - So much data generated very fast
   - Real-time input and response

4. Veracity
   - How trustworthy the data is ; accuracy of data (lack thereof is due to hashtags, typos, abbreviations)

DATA FORMATS

1. Structured
   - described in a matrix/data structure format
   - relational databases (SQL)

2. Unstructured
   - no fixed structure for the data
   - documents, tweets, videos

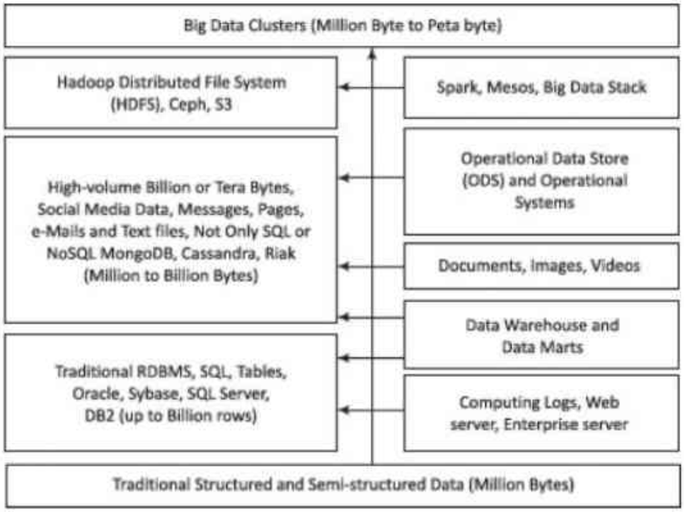
3. Semi-Structured
   - combination of the two
   - emails, XML

Data Architecture Design

Layer 5 Data consumption	Export of datasets to cloud, web etc.	Datasets usages: BPs, BIs, knowledge discovery	Analytics (real-time, near real-time, scheduled batches), reporting, visualization	
Layer 4 Data processing	Processing techn-ology: MapReduce, Hive, Pig, Spark	Processing in real-time, scheduled batches or hybrid	Synchronous or asynchronous processing	
Layer 3 Data storage	Considerations of types (historical or incremental), formats, compression, frequency of incoming data, patterns of querying and data consumption	Hadoop distributed file system (scaling, self-managing and self-healing), Spark, Mesos or S3	NoSQL data stores – Hbase, MongoDB, Cassandra, Graph database	
Layer 2 Data ingestion and acquisition	Ingestion using Extract Load and Transform (ELT)	Data semantics (such as replace, append, aggregate, compact, fuse)	Pre-processing (validation, transformation or transcoding) requirement	Ingestion of data from sources in batches or real time
Layer 1 Identification of internal and external sources of data	Sources for ingestion of data	Push or pull of data from the sources for ingestion	Data types for database, files, web or service	Data formats: structured, semi-structured or unstructured for ingestion

TI

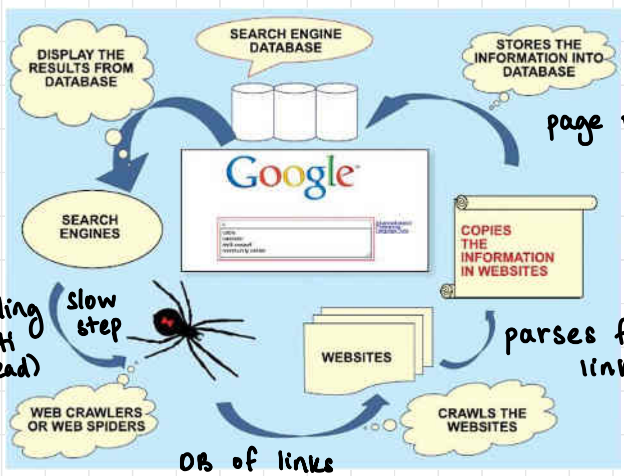
Data Storage for Traditional and Big Data



TI

Case Study: Working of Google Search Engine

- Page rank: sort pages by relevance (popularity) of page
  - initially took 5 days to compute
- Treats web as a graph



page rank computation

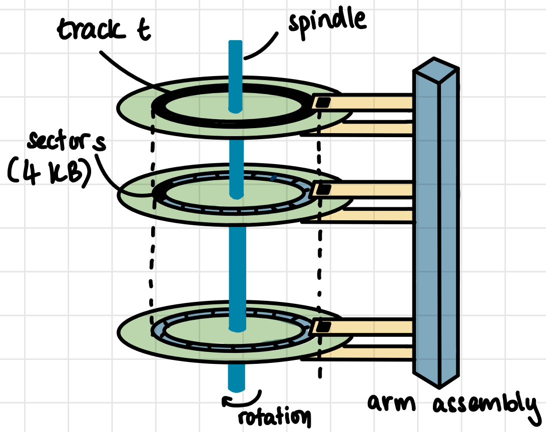
crawling (multi thread)

slow step

parses for links → indexing

DB of links

File Systems & Distributed File Systems



- Read time depends on
  - (i) seek time (position r/w head on track)
    - depends on current track & next track
    - mechanical
    - dominating delay
    - scheduling algorithm determines
  - (ii) rotational latency (position sector under head)
    - avg: half full rotational time
  - (iii) block transfer time
    - depends on electronics - fast, fixed
- File systems store mapping of files to blocks
- DFS: manages files on multiple machines
  - over LANs and WANs

Exercise

1. Consider that you have 1TB of data. Compare the time taken to read data in both the cases below:
   (i) Single machine (4 I/O channels, 100 mbps each)
   (ii) 10 machines (each having 4 I/O channels, 100 mbps each)

i) $\frac{10^6}{4 \times 10^2} = \frac{10^4}{4} = 2.5 \times 10^3 = 2500$ secs = 41 min 40 sec

ii) $\frac{10^6}{10 \times 4 \times 10^2} = \frac{10^3}{4} = 250$ secs = 4 mins 10 sec

HDFS - Hadoop Distributed File System

- HDFS inspired by Google File System (GFS) — 2003
- HDFS is a DFS, Open Source
- Origin: Apache Nutch search engine
- HDFS is DFS designed for storing very large files with streaming data access patterns (for analytics), running on clusters of commodity hardware
  - Large files: MB/GB/TB file sizes; PB clusters operational
  - Read mostly data: write once, read many most efficient
    * time to read whole dataset more imp than latency to read first record
    * each analysis involves large portion of dataset
  - Commodity hardware: inexpensive hardware
    * rack servers (present in CCBD)

Exercise

1. If you want to store a file on disk, what constitutes data & metadata?

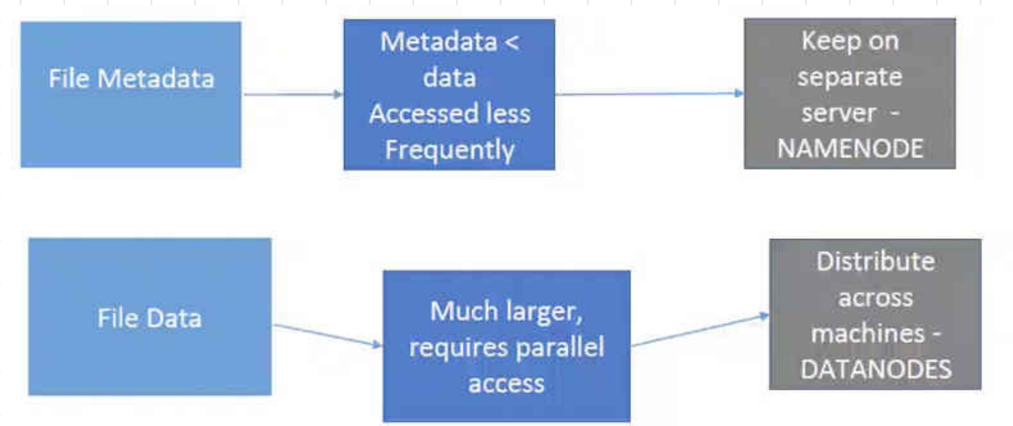
Data: file contents
Metadata: owner, creation time, file size, modified time, access rights, location on disk

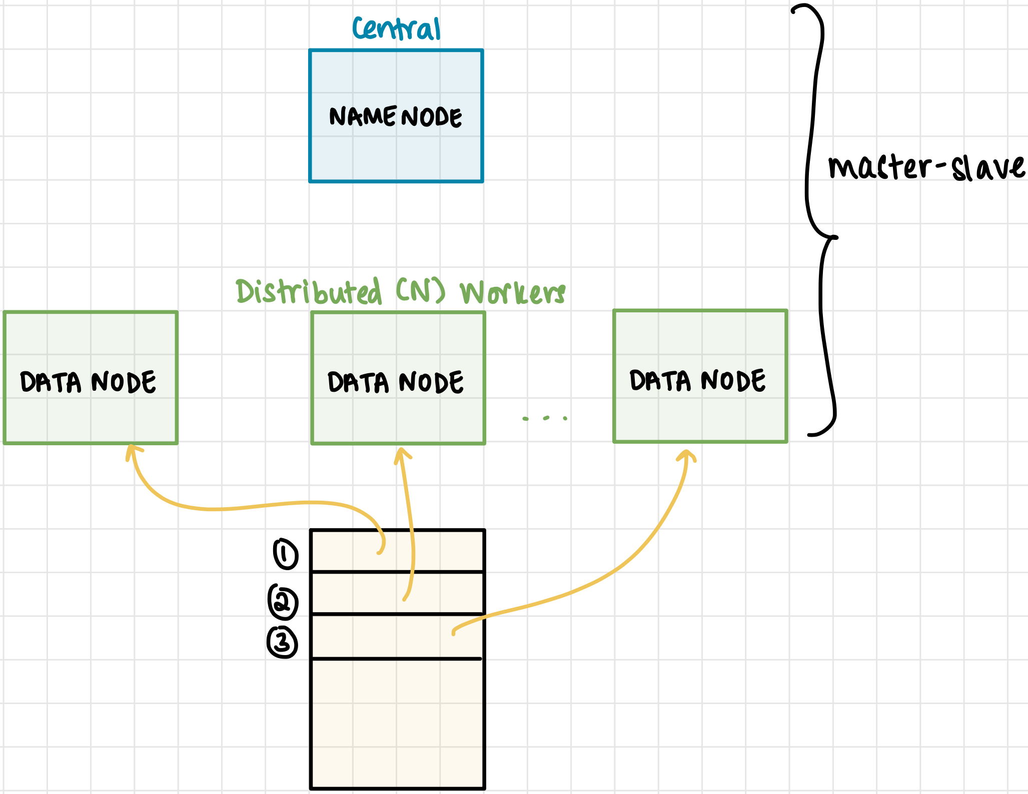
2. What are their access patterns? How often do you think each one would be accessed during a normal file read?

Data: accessed every time a file is opened & a line is read
Metadata: accessed every time a file is opened

3. How large are they, comparitively? Why is this important?

Data typically larger in size than metadata and is accessed more often than metadata





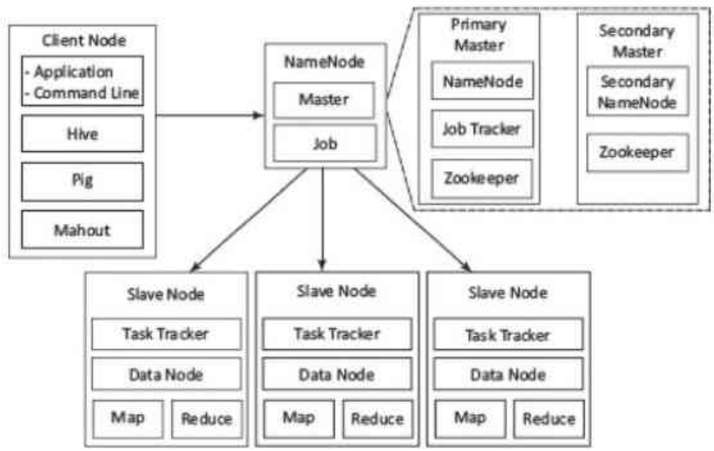
Scale Up vs Scale Out

- Scale up: buy a new machine with new specs
- Scale out: replicate resources to improve speed (better)

Commodity Servers - Issues

- Reliability is not very good
  - Solution: redundancy — data stored on 3 machines (so that if one fails, one can make a copy and another can serve)

Master-Slave Architecture



1. Writing a File

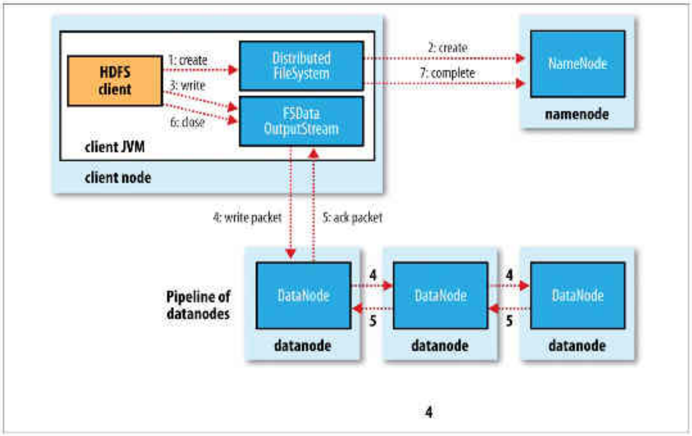
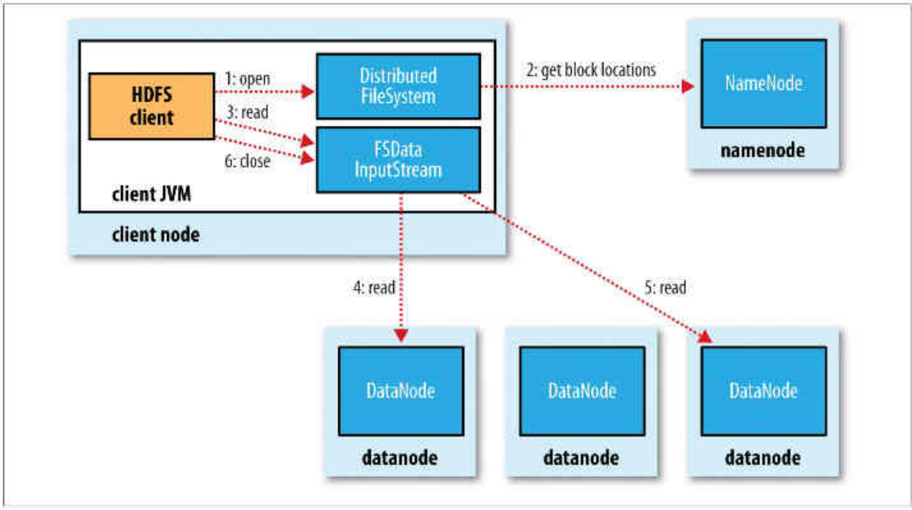


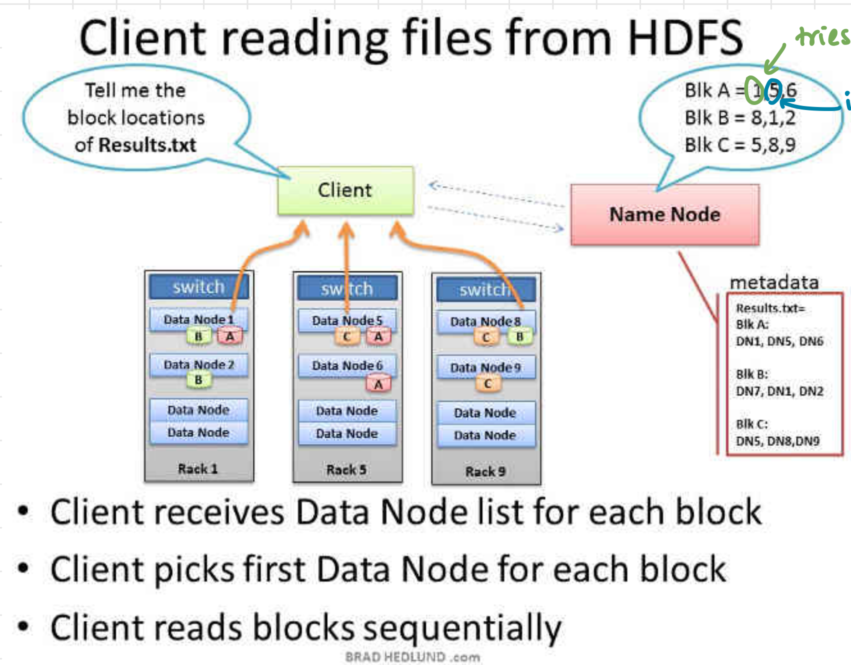
Fig 3-4, R3

- Write using pipeline — recursive request to make copies
- Direct write: iterative — requires more processing; not done (needs to send multiple copies)

2. Reading a File



https://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/

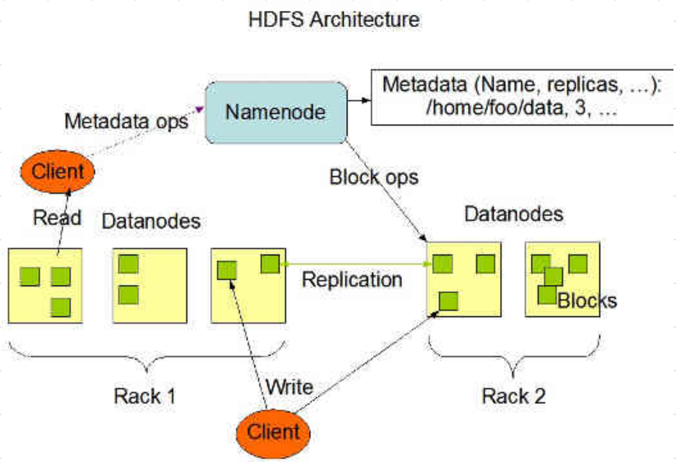


tries first

if 1 fails, 5, then 6

HDFS Architecture

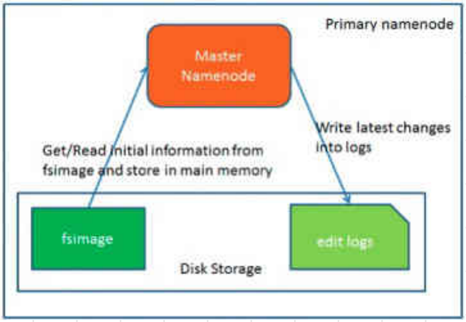
https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html



Namenode

- Manages directories, file system namespace
- Access rights regulation
- Open, close, rename files
- Mapping of blocks to data nodes
- Handles block failure
- Transaction log
- Metadata in memory

https://hrouhani.org/hdfs-file-system/

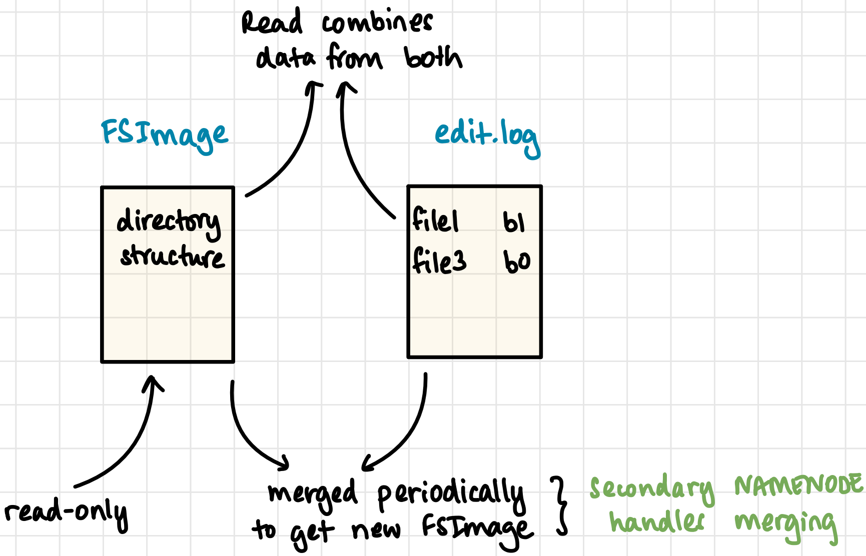


FSImage

- Serialised version of file system tree
- Not updated on every write (avoids recopying of data)
- Stores filename, access time, no. of blocks, blocks

Edit Logs

- Every write written to edit log
- Flushed and synced after every transaction
- Only appends allowed, no modifies



— Namenode Memory Requirements

- Rule of thumb: 1 GB for 1 million blocks

- Eg: 200 node cluster, 24 TB/node, 128 MB block size, replication factor 3, space=?

- no. of blocks = $\frac{200 \times 24 \times 2^{20}}{128 \times 3}$ = ~12 million = ~12,000 MB memory

  ← size of block is 3x due to replication

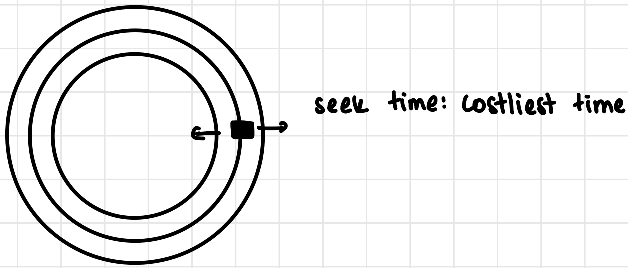
Why is Block Size 128 MB?

- Read time = seek time + transfer time + rotational latency

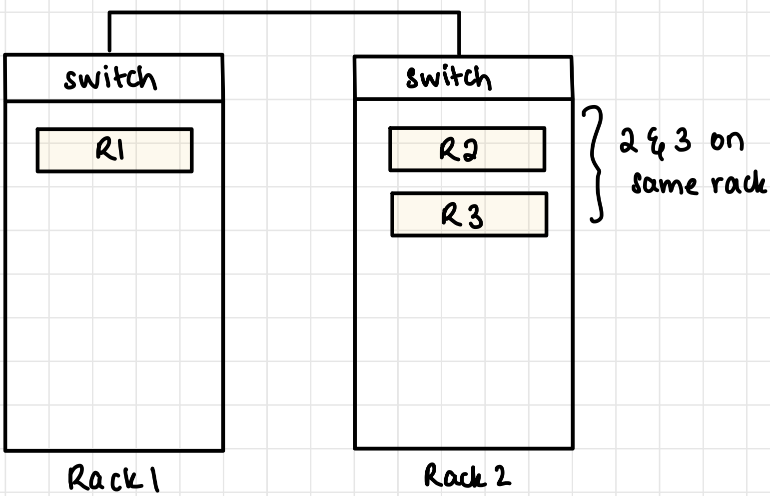
- Reduce seek time to improve performance of read; larger block sizes ⇒ transfer time >> seek time

- CPUs became faster, began waiting

- ∴ Hadoop V2, block size = 128 MB



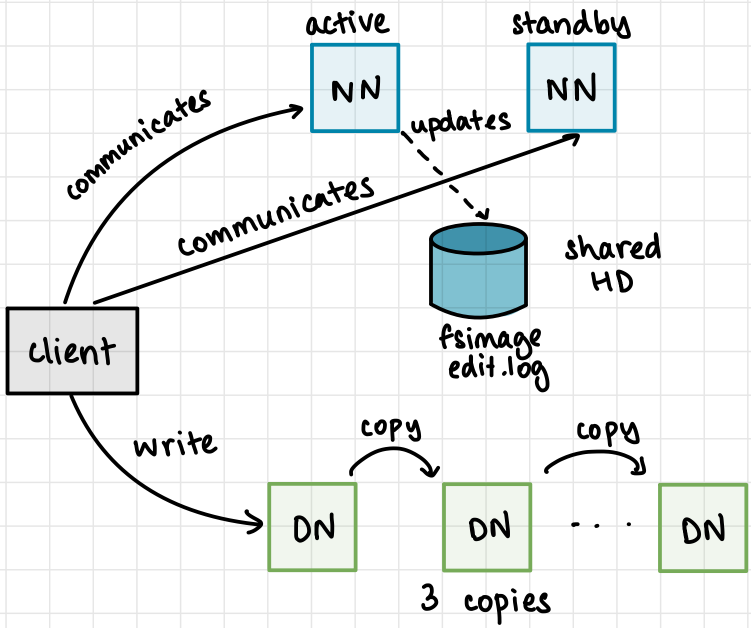
read such that read time >> seek time



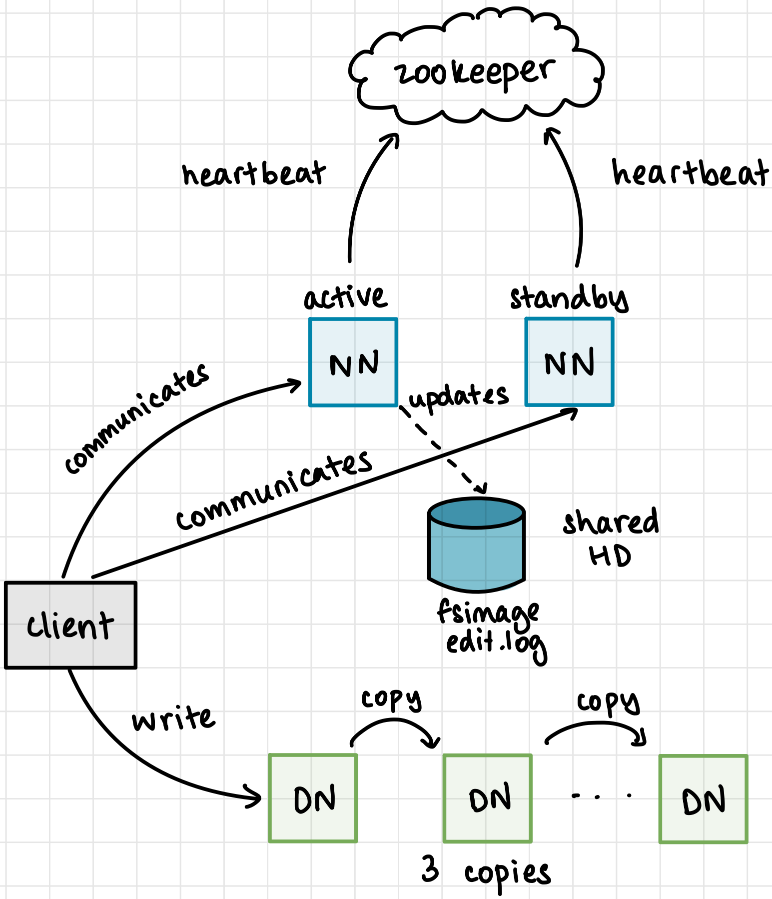
- Each block stored in 3 locations (copied) so that if one fails, one can copy while the other serves

Namenode Failure

- when name node fails, no requests can be handled
- Solutions will have trade-offs — based on use case
- One common solution: 2 NNs — Active NN & Standby NN both sharing a common HD
- HD stores FS Image and edit logs
- When client writes data to a DN, it communicates to both NNs
- If ANN fails, SNN changes state and becomes active (ANN) and can start updating shared HD
- Only the currently active NN can make updates to the shared HD



- How does SNN realise that ANN has failed?
- ANN & SNN both send periodic heartbeat messages to a third party (called zookeeper)
- If zookeper does not receive consecutive heartbeats from ANN, tells SNN to transition into ANN



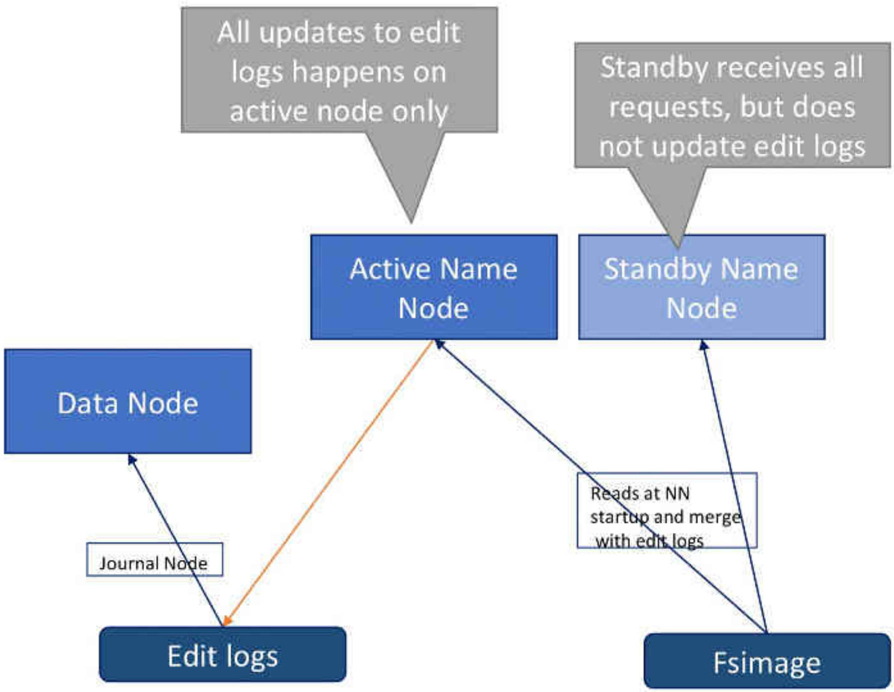
- Heartbeat not received can be due to problem at source or problem in network
- Zookeeper cannot determine where the problem is

- If problem is in the network and ANN is fine, but ZK has told SNN that it is now active, two NNs will be updating the HD at the same time

- Solution: enforce that only one NN can write to HD

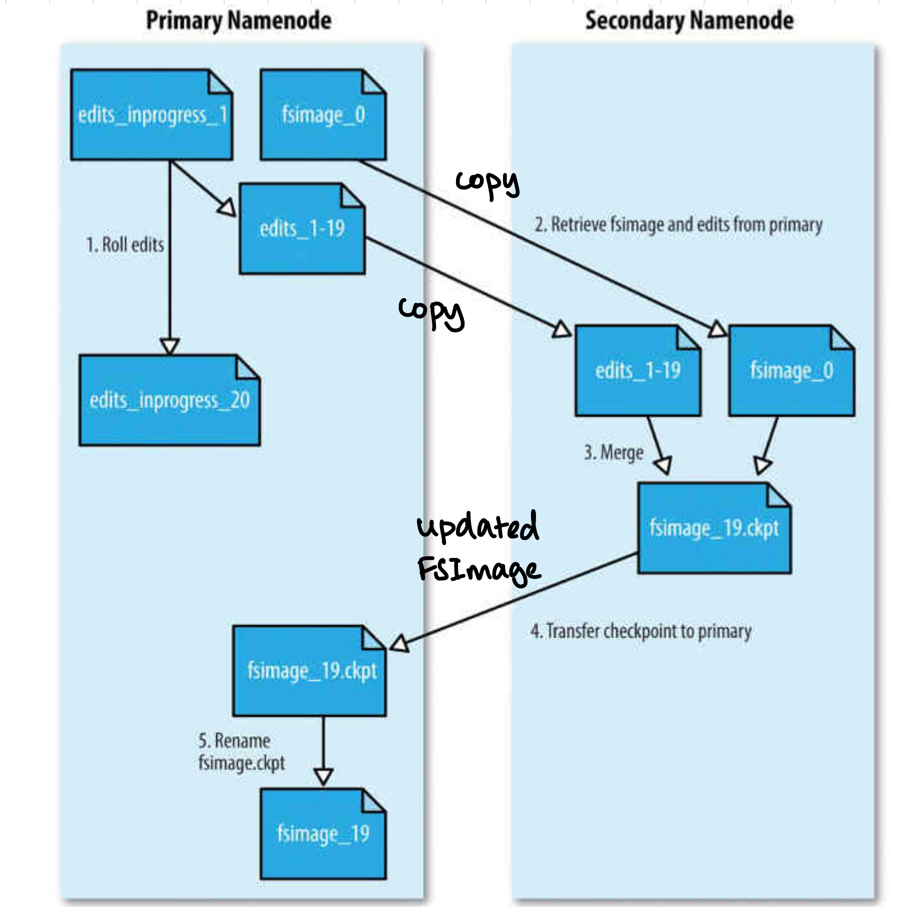
- Fencing: currently active NN virtually fences the HD from the other NN by blocking the network (STONITH- shoot the other node in the head)

- What if hard disk fails?



Secondary Namenode

- Primary NN can focus on serving requests
- Secondary NN tasked with merging FSImage with edits
- Secondary NN keeps a copy of Primary NN metadata



Example in HDFS

```
PES1UG19CS565@PES1UG19CS565:~/dfsdata/namenode/current $ l
edits_0000000000000000001-0000000000000000002  edits_0000000000000001613-0000000000000001749
edits_0000000000000000003-0000000000000000307  edits_0000000000000001750-0000000000000001789
edits_0000000000000000308-0000000000000000309  edits_0000000000000001790-0000000000000001833
edits_0000000000000000310-0000000000000000311  edits_0000000000000001834-0000000000000002332
edits_0000000000000000312-0000000000000000313  edits_0000000000000002333-0000000000000002376
edits_0000000000000000314-0000000000000000314  edits_0000000000000002377-0000000000000002464
edits_0000000000000000315-0000000000000000315  edits_0000000000000002465-0000000000000002510
edits_0000000000000000316-0000000000000000870  edits_0000000000000002511-0000000000000002634
edits_0000000000000000871-0000000000000001281  edits_0000000000000002635-0000000000000002636
edits_0000000000000001282-0000000000000001283  edits_0000000000000002637-0000000000000002638
edits_0000000000000001284-0000000000000001285  edits_0000000000000002639-0000000000000002640
edits_0000000000000001286-0000000000000001369  edits_0000000000000002641-0000000000000002642
edits_0000000000000001370-0000000000000001453  edits_0000000000000002643-0000000000000002644
edits_0000000000000001454-0000000000000001455  edits_0000000000000002645-0000000000000002758
edits_0000000000000001456-0000000000000001457  edits_0000000000000002759-0000000000000002798
edits_0000000000000001458-0000000000000001459  edits_0000000000000002799-0000000000000003147
edits_0000000000000001460-0000000000000001461  edits_inprogress_0000000000000003148
edits_0000000000000001462-0000000000000001463  fsimage_0000000000000002798
edits_0000000000000001464-0000000000000001465  fsimage_0000000000000002798.md5
edits_0000000000000001466-0000000000000001467  fsimage_0000000000000003147
edits_0000000000000001468-0000000000000001604  fsimage_0000000000000003147.md5
edits_0000000000000001605-0000000000000001610  seen_txid
edits_0000000000000001611-0000000000000001612  VERSION
```

BLOCKS IN HDFS

- File sizes vary (can be larger than a single disk in the network)
- Replication of blocks
- **%hadoop fsck -files –blocks** lists blocks that make up a file in the filesystem

Example:

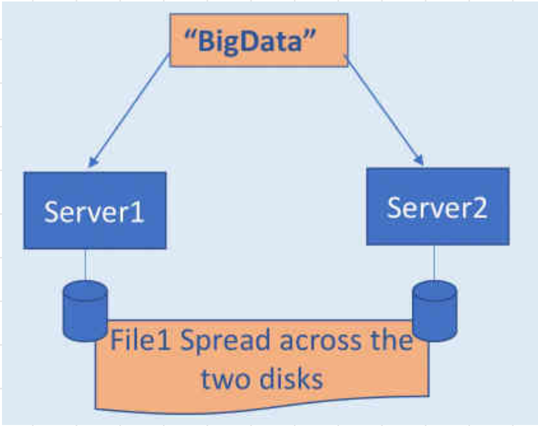
seek = 10 ms
transfer rate = 100 MBPS

We need to make seek time = 1% of transfer time (~100 MB)

- Hadoop v1 default — 64 MB
- Hadoop v2 default — 128 MB

MAP REDUCE PROGRAMMING MODEL

- Fundamental way to process large amounts of data
- Google, OSDI '04 (Operating System Design and Implementation)
- Runs on large set of commodity machines in a distributed manner, with checks for failures (high availability)
- Consider very large text file distributed over two machines and we need to search for word "BigData" in the file



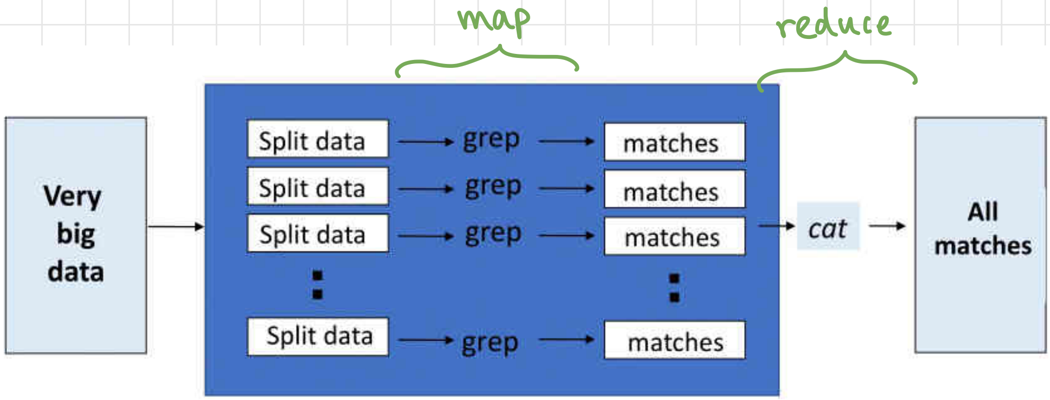
- Approach 1: run search in parallel on both machines — more efficient

- Approach 2: use 3rd machine to copy file (merge) and run grep — involves large network transfer

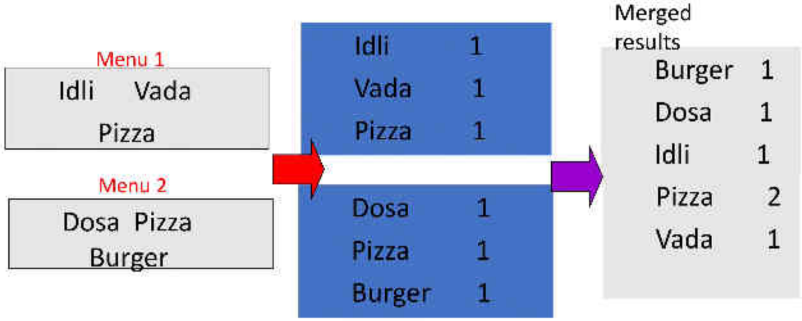
Distributed Grep Solution

- Less to transfer over network

- grep executed on all machines, merged together and the results are sent over the network



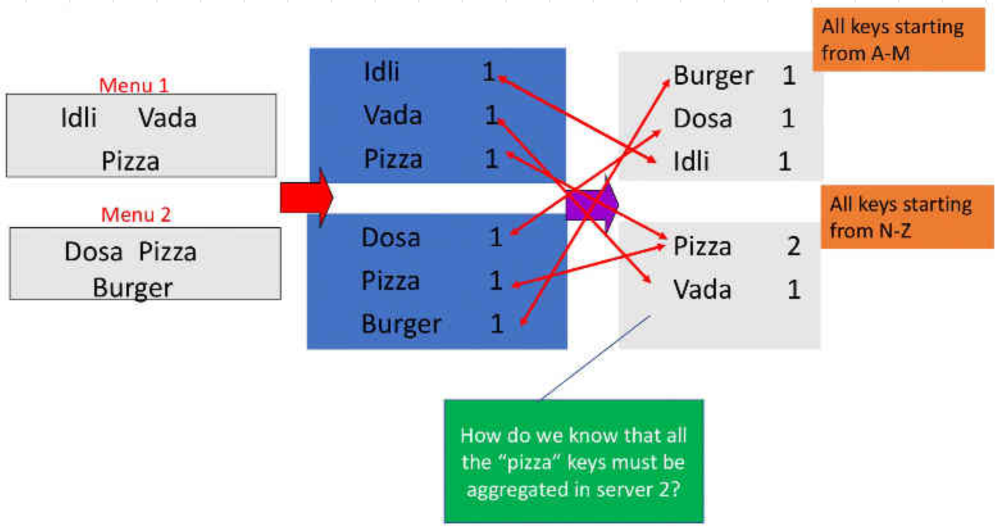
Eg: Find the number of restaurants offering each item



- Map: convert input to key-value pairs

- Reduce: merges intermediate key-value pairs to form final key-value pairs

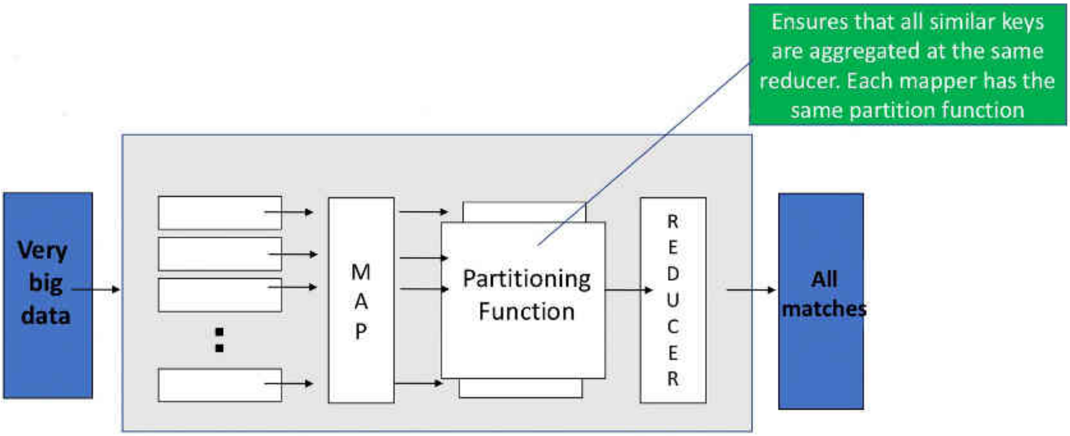
- How to parallelise the merge problem?
  - Assign keys to a particular machine based on rules
  - Eg: A-M on one machine and N-Z in another (range partitions)
  - Which reducer should receive which keys
  - Default Hadoop: Hash Partitioning (r = no. of reducers)
  - Range partitioning suffers from skewness problems



- Mappers accept input k/v pair and emits intermediate k/v pairs

- All mappers have same partitioning function

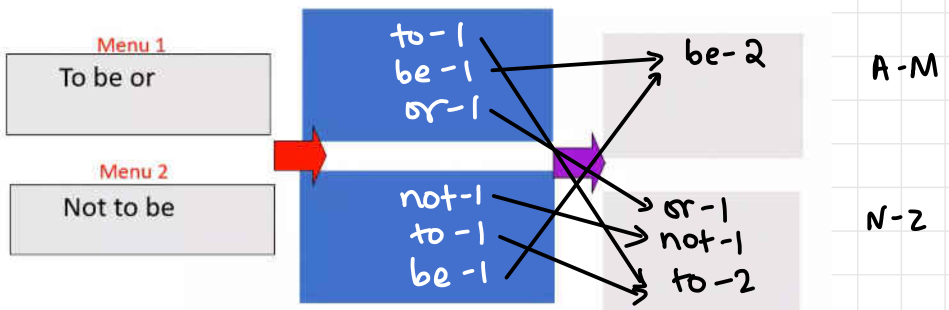
- Each reducer gets a partition



- Map:
  - Accepts *input* key/value pair
  - Emits *intermediate* key/value pair

- Reduce :
  - Accepts *intermediate* key/value* pair
  - Emits *output* key/value pair

Q: show map-reduce for word count (case-insensitive)



- A-M in first reducer
- N-Z in second reducer

- Note: if M1 has "to be or to", the map looks like
  to - [1, 1], be - [1], or - [1]

- Can add reducer code can be pushed to each mapper as a combiner (separate function)

- Combiner works on mapper like a mini reducer on the mapper
  - eg: can convert idli-[1,1] to idli-[2]

- For the count use case, intermediate k/v will have a list of 1's, but it is dependent on the application

- Each reducer must be responsible for a certain key (every key assigned to a reducer; reducer can be responsible for multiple keys)

- Input to reducer — pizza [1,1] (from 2 diff machines)

Map Reduce Programming Model

- Map: $(K_{in}, V_{in}) \rightarrow \mathrm{list}(K_{int}, V_{int})$

- Reduce: $(K_{int}, \mathrm{list}(V_{int})) \rightarrow \mathrm{list}(K_{out}, V_{out})$

Mapper for Word Count

```
public static class TokenizerMapper
     extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
```

(Key, value)

List (Key, value)

break sentence into words

like reading from stdin

Reducer for word count

```
public static class IntSumReducer
     extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}
```

Key ,List (value)

List (Key ,value)

- MR framework (Hadoop) implemented to do all the heavylifting

Driver Program

```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  String[] otherArgs = new GenericOptionsParser(conf, args).
                           getRemainingArgs();
  if (otherArgs.length < 2) {
    System.err.println("Usage: wordcount <in> [<in>...] <out>");
    System.exit(2);
  }
  Job job = new Job(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);

  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  for (int i = 0; i < otherArgs.length - 1; ++i) {
    FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
  }
  FileOutputFormat.setOutputPath(job,
    new Path(otherArgs[otherArgs.length - 1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

can have combiner here

Set Mapper and Reducer class

Q: What will be mapper and reducer? What will be keys?

Input: file (lineNumber, line) records and pattern

Output: lines matching a given pattern

Map: for line in file:
    if line matches pattern:
        write line to context

Reducer: no job or identity function

Keys: pattern

Q: Sort function: mapper? reducer? partition?

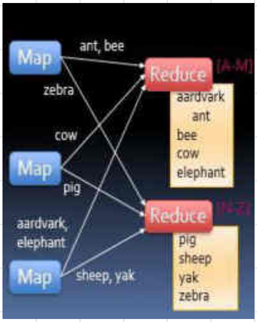
Input: (key, value) records

Output: same records sorted by key

Map: identity (output of mapper always sorted by key)

Reducer: identity (concat over multiple reducers)

Partition: pick $p(k)$ such that $p(k_1) < p(k_2)$ if $k_1 < k_2$



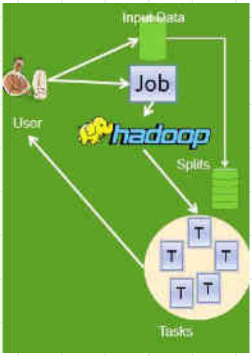
Hadoop Flow

- User submits job
  - input data, MR program, config info (# of reducers, mem to be allocated)

- Job split into smaller map tasks and reduce tasks

- Job splits I/P data into smaller chunks called splits

- One map task per split ; parallelisation



1. Single Reducer Task

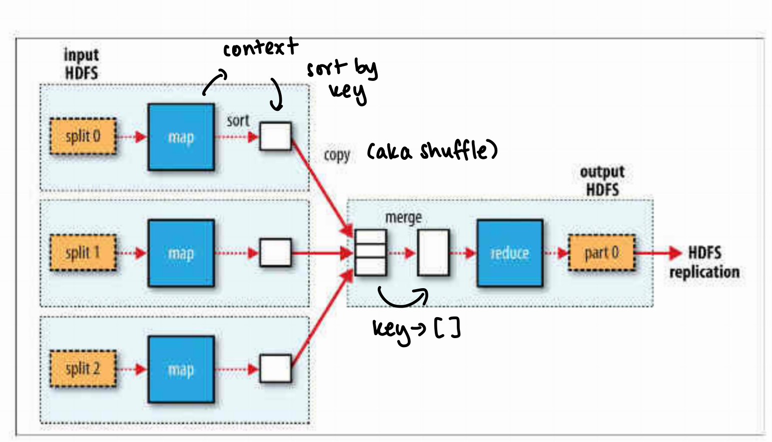


Figure 2-3. MapReduce data flow with a single reduce task

2. Multiple Reducers

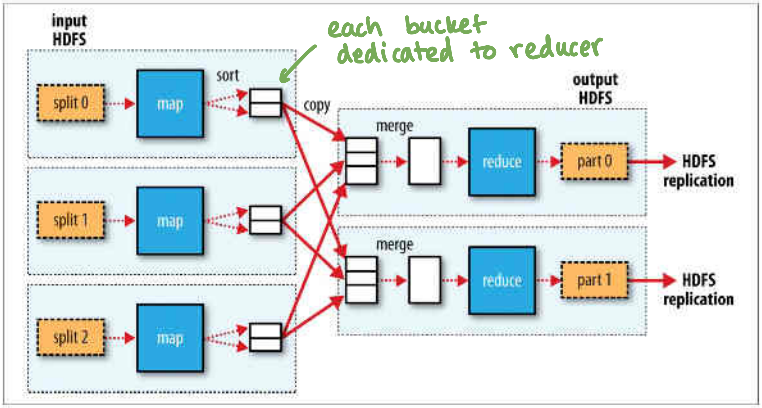


Figure 2-4. MapReduce data flow with multiple reduce tasks

MR Split Size Considerations

- Smaller splits ⇒ more parallelism
- Small split size advantages
  - large # of splits
  - increased parallelism
  - increased load balancing
- Small split size disadvantages
  - overhead of managing splits and map task creation
  - less time to execute job (this dominates)
- Optimal split size = HDFS block size (128 MB on v2)

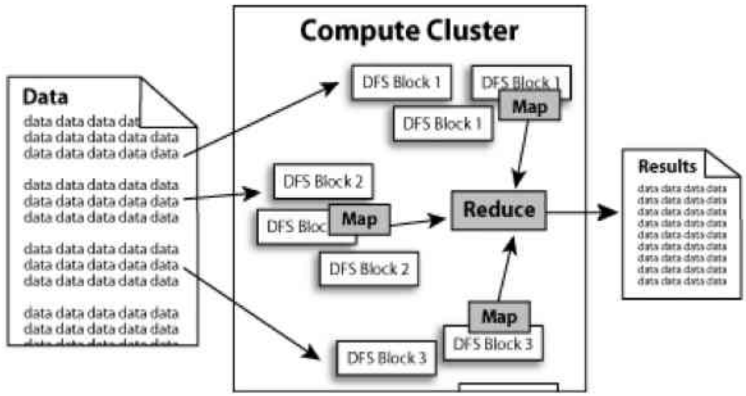
Split == Block size

- All data required for Map
  - In the same node
  - No inter-node data transfer is required

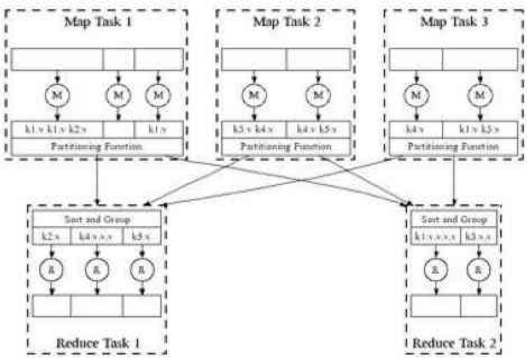
Split != block size

- Data transfer across multiple nodes
- Impacts performance

Traditional Compute



Big Data : move compute to machines



Map output

- Written to local disk, not to HDFS
- Local data

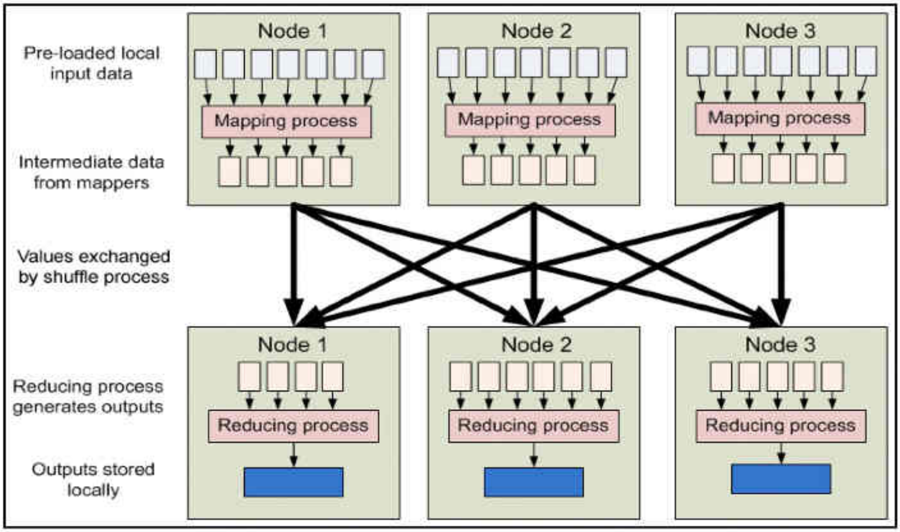
Failure of Map Task

- If node fails while performing map and before sending data to reduce, it is re-run on another map node

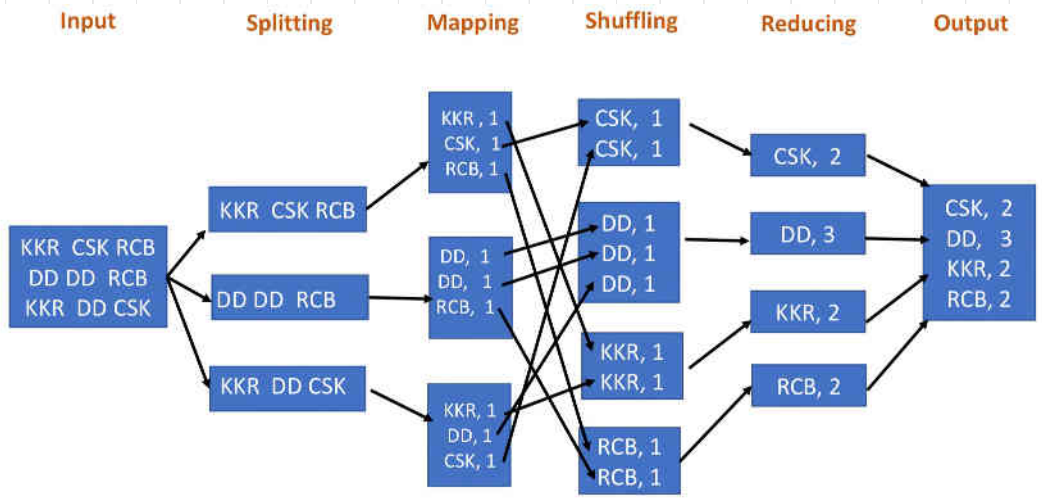
Reduce Tasks

- O/P stored in HDFS
- Sorted map O/Ps have to transfer over network
- 1 copy stored on reduce node where reduce task happens
- Copies stored on off-rack nodes

High Level View

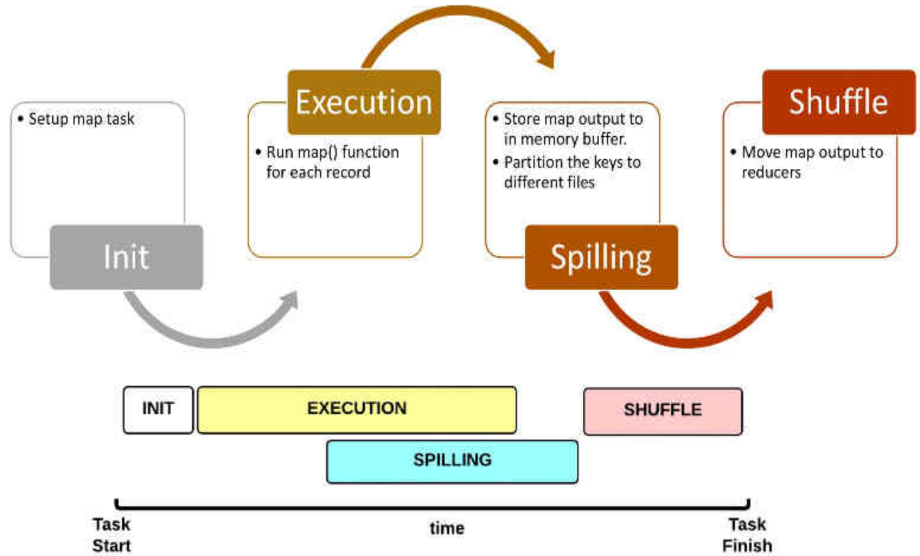


Shuffling example

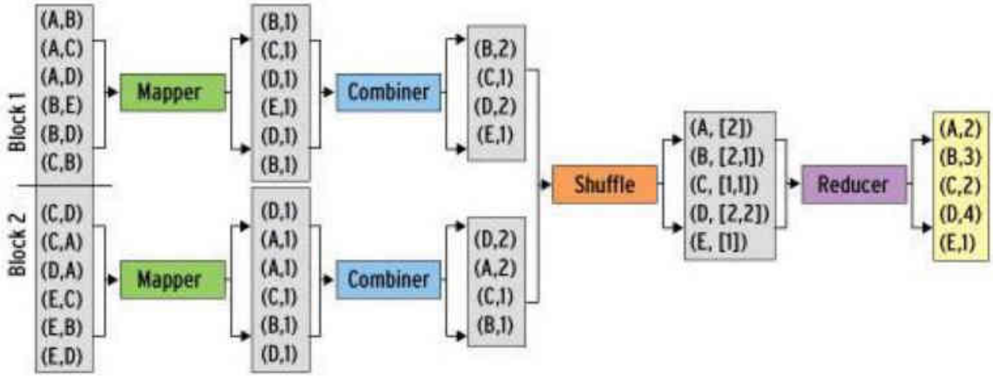


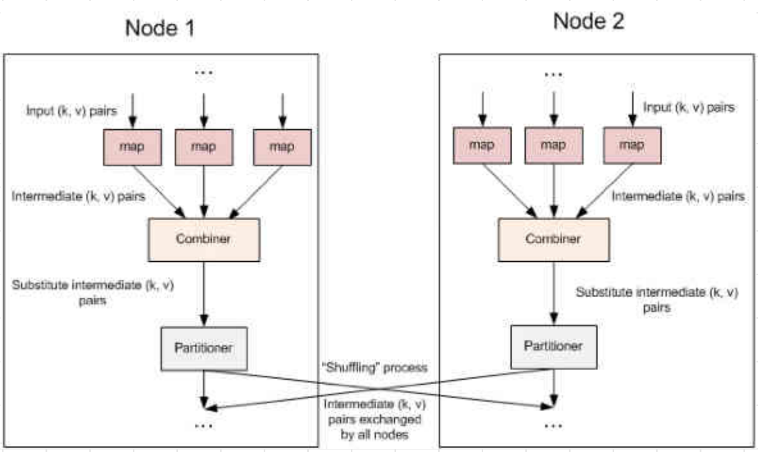
The overall Map-Reduce word count Process

Closer look at Map



Combiners


Block 1
(A,B)
(A,C)
(A,D)
(B,E)
(B,D)
(C,B)
Block 2
(C,D)
(C,A)
(D,A)
(E,C)
(E,B)
(E,D)
Mapper
Mapper
(B,1)
(C,1)
(D,1)
(E,1)
(D,1)
(B,1)
(D,1)
(A,1)
(A,1)
(C,1)
(B,1)
(D,1)
Combiner
Combiner
(B,2)
(C,1)
(D,2)
(E,1)
(D,2)
(A,2)
(C,1)
(B,1)
Shuffle
(A, [2])
(B, [2,1])
(C, [1,1])
(D, [2,2])
(E, [1])
Reducer
(A,2)
(B,3)
(C,2)
(D,4)
(E,1)


Node 1
Node 2
Input (k, v) pairs
map
map
map
Intermediate (k, v) pairs
Combiner
Substitute intermediate (k, v) pairs
Partitioner
Input (k, v) pairs
map
map
map
Intermediate (k, v) pairs
Combiner
Substitute intermediate (k, v) pairs
Partitioner
"Shuffling" process
Intermediate (k, v) pairs exchanged by all nodes

Q: Suppose
- We have a 2GB file
- Split size is 128 MB
- We have 4 disks

(i) How many splits are there?

$$2 \times 1024/128 = 2^{11}/2^{7} = 2^{4}$$

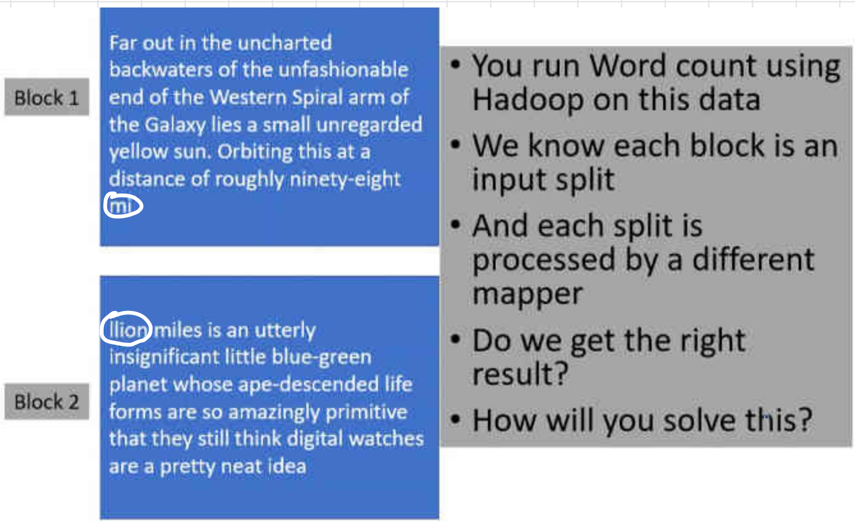
$= 16$ splits

(ii) How many splits per disk?

$16/4 = 4$ splits / disk

(iii) How many map tasks? 16
(iv) How many map tasks per node? 4
(v) How many reduce tasks? user specified

Q:



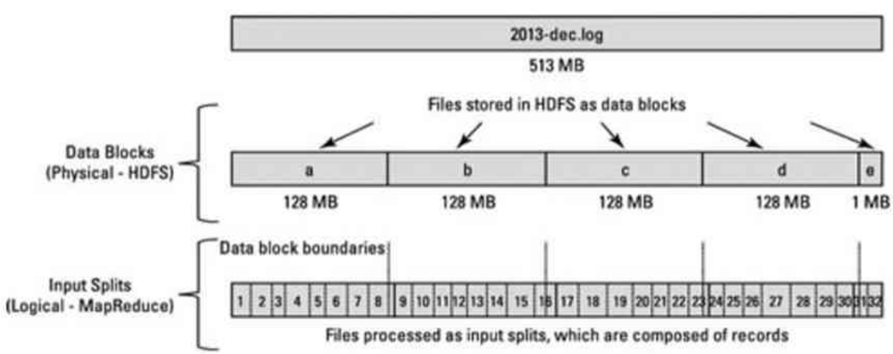
• No; word is split across blocks (million)

- First & last words are issue
- From second block, start computing from second word (EOR separator)
- Map 1 processes last word of first block (million) by looking for end-of-record (EOR) separator

Each split further broken into records

In this case the records would look like
- Far
- Out
- in. ...

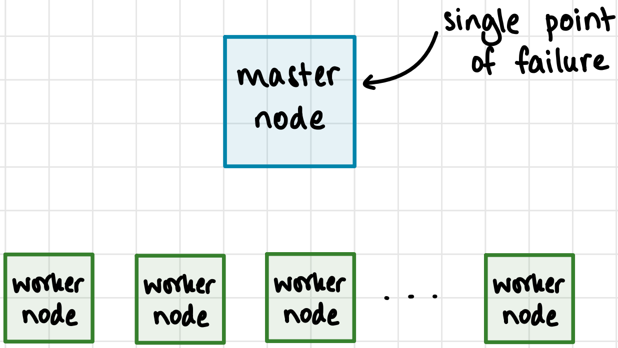
How to handle incomplete records?
- Mapper 1 will process million
- By looking for EOR separator.



map: (K1, V1) → list(K2, V2)
reduce: (K2, list(V2)) → list(K3, V3)

JOB MANAGEMENT

- How to allocate machines for map & reduce tasks?
- Who allocates and monitors tasks?

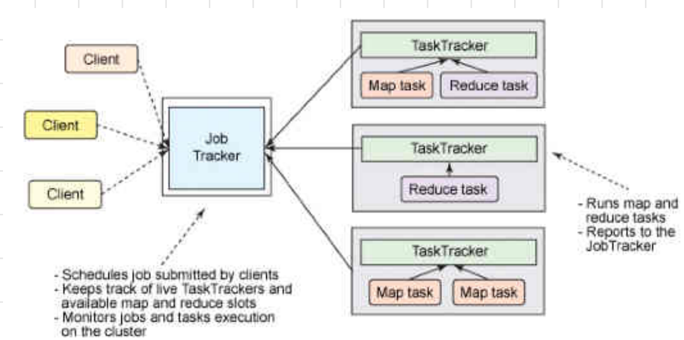


Hadoop 1.0 Job Management

- Job tracker (master-slave)

- Client submits job to job tracker, job tracker sends mapper and reducer jobs to available nodes

- Nodes have task trackers that can receive tasks from the job tracker



failures

- Job tracker handles fault tolerance, cluster resource management and scheduling

availability

allocation

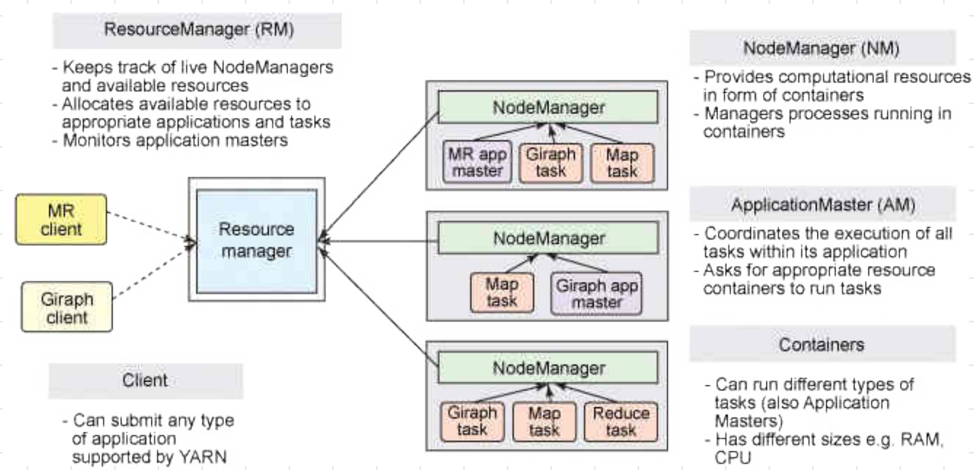
Issues

1. Limits scalability (only 4000 nodes per cluster)
2. Availability — single point of failure
3. Resource utilisation problems
4. Limitation in running only MR applications

YARN

- Yet Another Resource Negotiator
- MR and other tasks can use YARN for resources

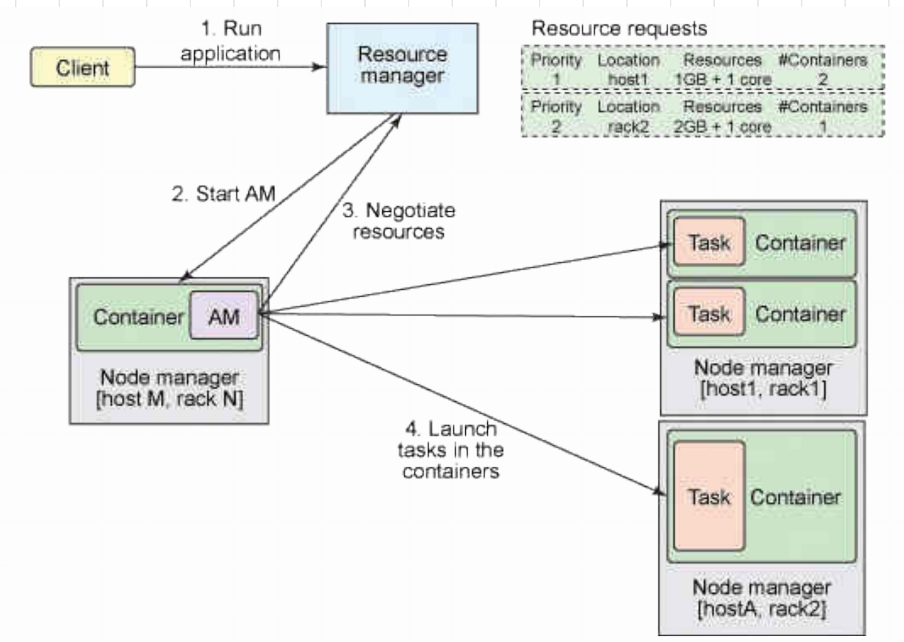
YARN Architecture



- Application master manages jobs from within node
- Node manager manages node (all tasks on a node)

Resource Manager	• Arbitrates resources amongst all applications of the system
Node Manager	• Per machine slave • Responsible for launching application containers • Monitors resource usage
Application Master	• Negotiate appropriate resource containers from the scheduler • Track and monitor the progress of the containers
Container	• Unit of allocation incorporating resources such as memory, CPU, disk

YARN Working



- From R3

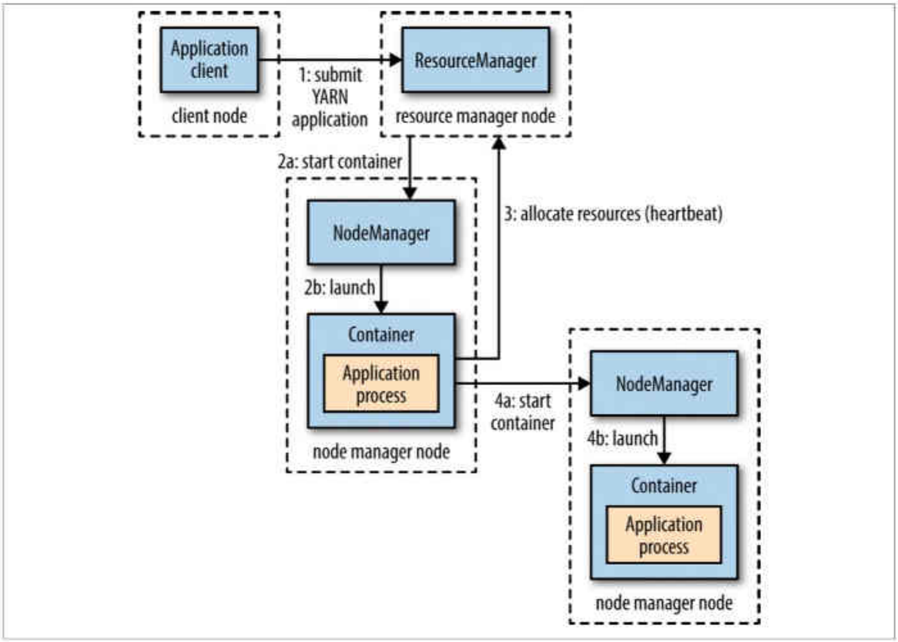


Figure 4-2. How YARN runs an application

Data Locality in Map Reduce

- Best if map task runs on same node as the input data's location (in HDFS)

- If all nodes hosting input data are busy, looks for a free map slot on a node in the same rack as one of the blocks

- Occasionally, inter-rack network transfer required (off-rack node)

- From R3

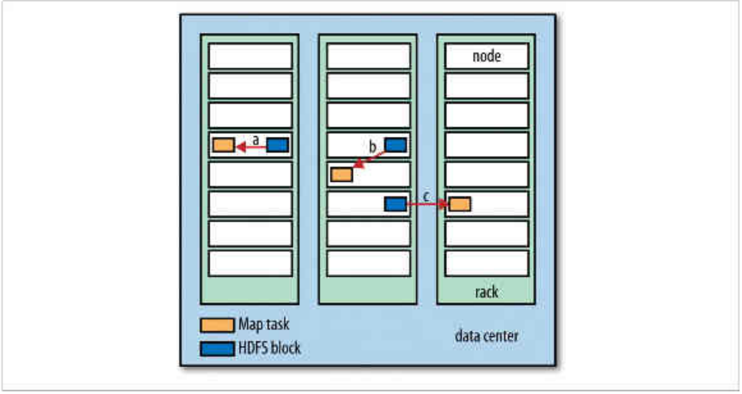


Figure 2-2. Data-local (a), rack-local (b), and off-rack (c) map tasks

Scheduling in YARN

- Earlier versions — FIFO scheduler
  - each job used entire cluster
  - jobs had to wait for turn
- Balance between production (periodic) jobs and ad-hoc jobs
- Configured in config file

1. FAIR SCHEDULER

- Jobs placed in pools

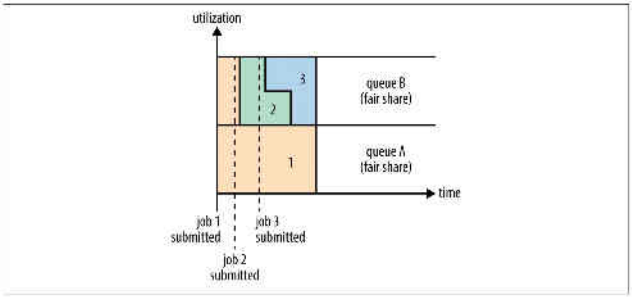


Figure 4-1. Fair sharing between user queues

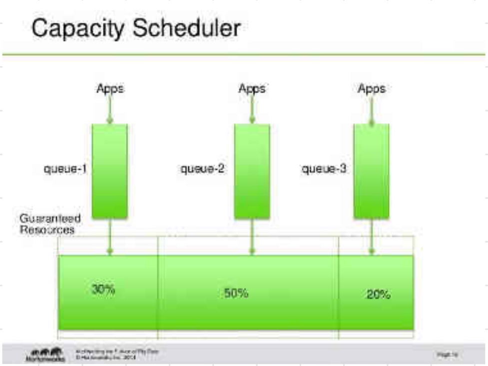
(R3)

- Each user gets own pool (default)
- Single job ⟶ full cluster
- Free task slots given to jobs in a fair way (each user gets fair share)
- Long & short jobs
- Scheduler ensures that a single user does not hog the cluster by submitting too many jobs
- Custom pools: guaranteed minimum capacities with map/reduce slots
- Fair Scheduler supports preemption

2. CAPACITY SCHEDULER

- Certain number of queues (like pools in Fair Scheduler)
  - allocated capacity (eg: max 30%)
  - can be hierarchial
  - FIFO within each queue

- https://www.slideshare.net/Hadoop_Summit/w-525hall1shenv2



- Cannot use free spare capacity even if it exists
- Break up clusters into smaller clusters

Handling Failures

- What can fail
  - task
  - app manager
  - resource manager
  - node manager

1. Task Failure

Due to runtime exceptions	• JVM reports error back to parent application master
Hanging tasks	• Progress updates not happening for 10 mins • Timeout value can be set.
Killed tasks	• Speculative duplicates can be killed
Recovery	• AM tries restarting task on a different node

2. Application Master Failure

When can failure occur?	• Due to hardware or network failures
How to detect for failures?	• AM sends periodic heartbeats to Resource Manager
Restart	• Max-attempts to restart application • Default = 2

3. Node Manager Fail

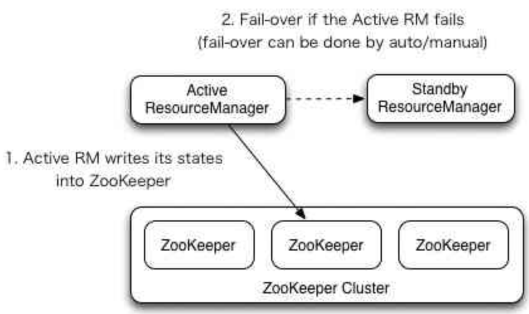
When can failure occur?	• Hardware, crashing, slow network
How to detect for failures?	• When a heartbeat is not received by RM for 10mins
Restart	• Tasks of incomplete jobs will be rerun – maybe on different node

4. Resource Manager Failure

How is failure handled?	• Active Standby configuration
Impact	• More serious as all tasks fail
Restart	• Handled by failover controller

- Zookeeper: manages cluster (coordination b/w machines; distributed locking, heartbeats)

- RM: within cluster, allocates resources



- https://hadoop.apache.org/docs/current/hadoop- yarn/hadoop-yarn-site/ResourceManagerHA.html

Benefits of YARN

- YARN manages very large cluster at Yahoo
  - Scalable
  - Flexible (Hadoop, Storm, Spark in same cluster using YARN)

- Read
  https://www.techrepublic.com/article/why-the-worlds-largest-hadoop-installation-may-soon-become-the-norm/

Q: A 1000 node YARN cluster has no jobs running. Two pools are configured with max of 50% of the resources. A new job requiring 600 nodes is submitted and on starting consumes all 600 nodes. Which YARN scheduler is active?

- FIFO scheduler or Fair Scheduler
- Can use entire cluster

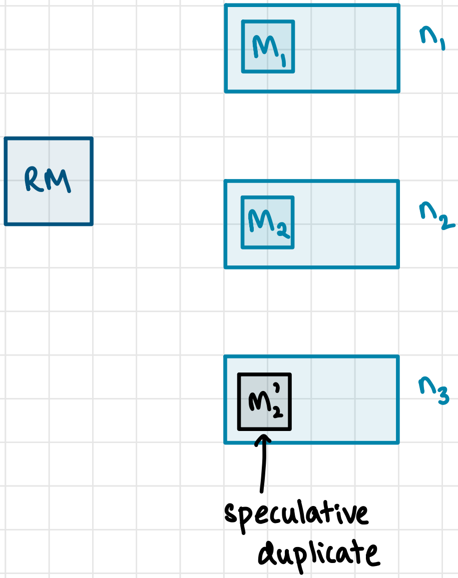
Q: Will the failure of task result in failure of the entire job?

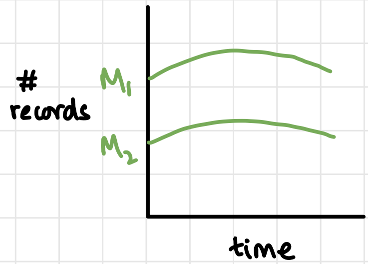
- No it will be restarted

Q: What are speculative duplicates?

- Tasks started when AM determines that there is a slow running task

Speculative Duplicates





- Monitor M_1 & M_2. If M_2 is not processing fast enough, M_2' created
- If M_2' does better, M_2 is killed
- If M_2 does better, M_2' is killed