

**CMR Institute of Technology  
Department of Computer Applications**

**IAT 4 – Answer Key**

1 a

**Describe any 4 characteristics (4 Vs of Big Data)**

Characteristics of Big Data:

1. Volume
2. Velocity
3. Variety
4. Value

**Volume:** The main characteristic that makes data “big” is the sheer volume. It makes no sense to focus on minimum storage units because the total amount of information is

growing exponentially every year.

**Variety:** is one the most interesting developments in technology as more and more information is digitized. Traditional data types (structured data) include things on a bank statement like date, amount, and time.

**Velocity** is the frequency of incoming data that needs to be processed.

**Value:** Analysis add value to your business is measured.

b

**Discuss the application of big data analytics**

**Table 1.1** Example Analytics Applications

Marketing	Risk Management	Government	Web	Logistics	Other
Response modeling	Credit risk modeling	Tax avoidance	Web analytics	Demand forecasting	Text analytics
Net lift modeling	Market risk modeling	Social security fraud	Social media analytics	Supply chain analytics	Business process analytics
Retention modeling	Operational risk modeling	Money laundering	Multivariate testing		
Market basket analysis	Fraud detection	Terrorism detection			
Recommender systems					
Customer segmentation					

c

**With a neat diagram, describe the working of analytical processing model**

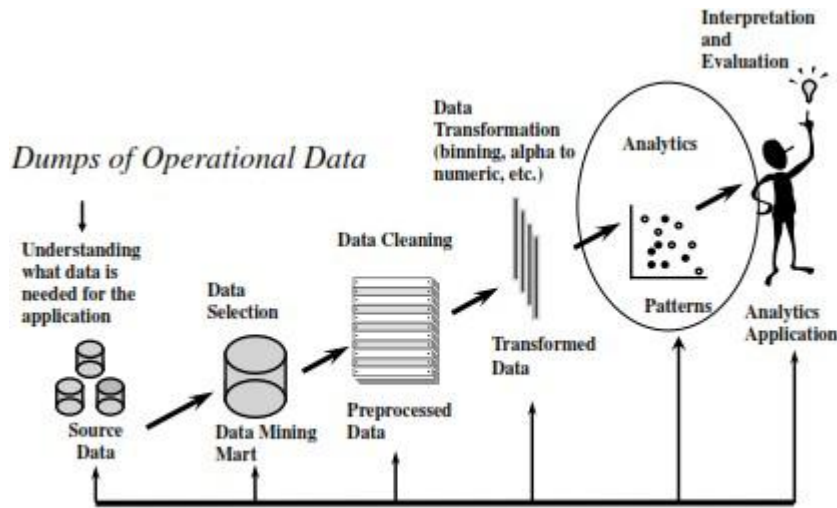


Figure 1.2 The Analytics Process Model

1. Define the business problems to be solved
2. All source-data need to be identified that could be of potential interest.
3. All data to be gathered in a staging area
4. Basic exploratory analysis will be considered.
5. Data cleaning step to get rid of all inconsistencies
6. In the analytics step, an analytical model will be estimated on the preprocessed and transformed data.

Once the model is built it will interpreted and evaluated by the business experts.

2 a **Mention the different types of data sources:**

**Transaction:** - Transactional data consists of structured, low-level, detailed information capturing the key characteristics of a customer transaction.

**Un-Structured data:** – are stored in form of text documents.

**Qualitative, expert based data:**-Subject matter expertise

**Data-Poolers:**- Dun & Bradstreet, Thomson Reuters

**Social Media:** Data from face book and twitter etc.

b **Calculate the Z-Score and detect the outlier for the following data. Where mean = 40 Standard deviation = 10 and Data= 30 50 10 40 60 80**

Observation	Mean	Standard Deviation	Z-Score
30	40	10	-1
50	40	10	1
10	40	10	-3
40	40	10	0
60	40	10	2
80	40	10	4

Any **z-score** greater than 3 or less than -3 is considered to be an **outlier**. Hence the data 80 is outlier.

**c Various factors required for analytical model:**

1. Business relevance
2. Statistical performance
3. Operational efficient
4. Economic cost
5. Local and International regulations and legislation

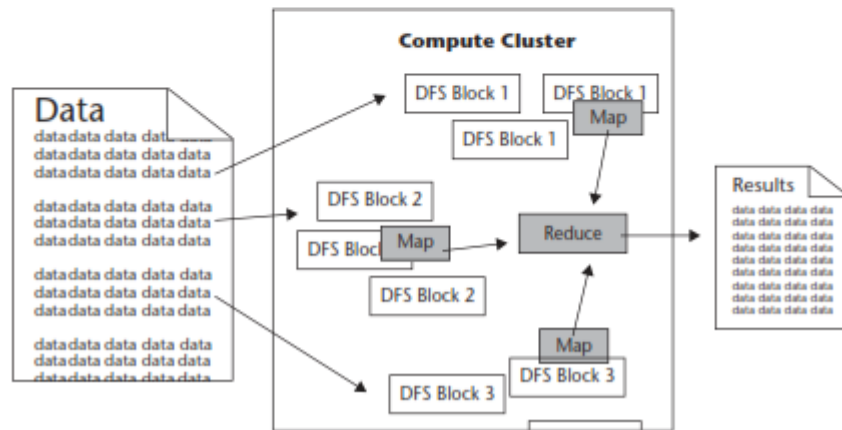
A good analytical model should satisfy several requirements, depending on the application area. A first critical success factor is business relevance. The analytical model should actually solve the business problem for which it was developed. It makes no sense to have a working analytical model that got sidetracked from the original problem statement. In order to achieve business relevance, it is of key importance that the business problem to be solved is appropriately defined, qualified, and agreed upon by all parties involved at the outset of the analysis.

A second criterion is statistical performance. The model should have statistical significance and predictive power. How this can be measured will depend upon the type of analytics considered. For example, in a classification setting (churn, fraud), the model should have good discrimination power. In a clustering setting, the clusters should be as homogenous as possible. In later chapters, we will extensively discuss various measures to quantify this.

Depending on the application, analytical models should also be interpretable and justifiable. *Interpretability* refers to understanding the patterns that the analytical model captures. This aspect has a certain degree of subjectivism, since interpretability may depend on the business user's knowledge. In many settings, however, it is considered to be a key requirement. For example, in credit risk modeling or medical diagnosis, interpretable models are absolutely needed to get good insight into the underlying data patterns. In other settings, such as response modeling and fraud detection, having interpretable models may be less of an issue. *Justifiability* refers to the degree to which a model corresponds to prior business knowledge and intuition.<sup>6</sup> For example, a model stating that a higher debt ratio results

	<p>in more creditworthy clients may be interpretable, but is not justifiable because it contradicts basic financial intuition. Note that both interpretability and justifiability often need to be balanced against statistical performance. Often one will observe that high performing analytical models are incomprehensible and black box in nature. A popular example of this is neural networks, which are universal approximators and are high performing, but offer no insight into the underlying patterns in the data. On the contrary, linear regression models are very transparent and comprehensible, but offer only limited modeling power.</p> <p>Analytical models should also be <i>operationally efficient</i>. This refers to the efforts needed to collect the data, preprocess it, evaluate the model, and feed its outputs to the business application (e.g., campaign management, capital calculation). Especially in a real-time online scoring environment (e.g., fraud detection) this may be a crucial characteristic. Operational efficiency also entails the efforts needed to monitor and backtest the model, and reestimate it when necessary.</p> <p>Another key attention point is the <i>economic cost</i> needed to set up the analytical model. This includes the costs to gather and preprocess the data, the costs to analyze the data, and the costs to put the resulting analytical models into production. In addition, the software costs and human and computing resources should be taken into account here. It is important to do a thorough cost-benefit analysis at the start of the project.</p> <p>Finally, analytical models should also comply with both local and international <i>regulation and legislation</i>. For example, in a credit risk set-</p>	
3 a	<p><b>Discuss the critical components of hadoop with neat diagram</b></p> <p>The two critical components of Hadoop are:</p> <ol style="list-style-type: none"> <li><b>1. The Hadoop Distributed File System (HDFS).</b> HDFS is the storage system for a Hadoop cluster. When data lands in the cluster, HDFS breaks it into pieces and distributes those pieces among the different servers participating in the cluster. Each server stores just a small fragment of the complete data set, and each piece of data is replicated on more than one server.</li> <li><b>2. MapReduce.</b> Because Hadoop stores the entire dataset in small pieces across a collection of servers, analytical jobs can be distributed, in parallel, to each of the servers storing part of the data. Each server evaluates the question against its local fragment simultaneously and reports its results back for collation into a comprehensive answer. MapReduce is the agent that distributes the work and collects the results.</li> </ol>	

Both HDFS and MapReduce are designed to continue to work in the face of system failures. HDFS continually monitors the data stored on the cluster. If a server becomes unavailable, a disk drive fails, or data is damaged, whether due to hardware or software problems, HDFS automatically restores the data from one of the known good replicas stored elsewhere on the cluster. Likewise, when an analysis job is running, MapReduce monitors progress of each of the servers participating in the job. If one of them is slow in returning an answer or fails before completing its work, MapReduce automatically starts another instance of that task on another server that has a copy of the data. Because of the way that HDFS and MapReduce work, Hadoop provides scalable, reliable, and fault-tolerant services for data storage and analysis at very low cost.



b What is predictive analysis? Why are they required? Discuss the leading trends of predictive analysis.

	<ul style="list-style-type: none"> <li>■ Recommendation engines similar to those used in Netflix and Amazon that use past purchases and buying behavior to recommend new purchases.</li> <li>■ Risk engines for a wide variety of business areas, including market and credit risk, catastrophic risk, and portfolio risk.</li> <li>■ Innovation engines for new product innovation, drug discovery, and consumer and fashion trends to predict potential new product formulations and discoveries.</li> <li>■ Customer insight engines that integrate a wide variety of customer-related info, including sentiment, behavior, and even emotions. Customer insight engines will be the backbone in online and set-top box advertisement targeting, customer loyalty programs to maximize customer lifetime value, optimizing marketing campaigns for revenue lift, and targeting individuals or companies at the right time to maximize their spend.</li> <li>■ Optimization engines that optimize complex interrelated operations and decisions that are too overwhelming for people to systematically handle at scales, such as when, where, and how to seek natural resources to maximize output while reducing operational costs—or what potential competitive strategies should be used in a global business that takes into account the various political, economic, and competitive pressures along with both internal and external operational capabilities.</li> </ul>	
4 a	<p><b>List and Explain the technical features of Hadoop</b></p> <p>Hadoop is an open source, Scalable, and Fault tolerant framework written in Java. It efficiently processes large volumes of data on a cluster of commodity hardware. Hadoop is not only a storage system but is a platform for large data storage as well as processing.</p> <p><b>1. Open Source</b></p> <p>Apache Hadoop is an open source project. It means its code can be modified according to business requirements.</p> <p><b>2. Distributed Processing</b></p> <p>As data is stored in a distributed manner in HDFS across the cluster, data is processed in parallel on a cluster of nodes.</p> <p><b>3. Fault Tolerance</b></p> <p>By default 3 replicas of each block is stored across the cluster in Hadoop and it can be changed also as per the requirement. So if any node goes down, data on that node can be recovered from other nodes easily. Failures of nodes or tasks are recovered automatically by the framework. This is how Hadoop is fault tolerant.</p>	



	<p>4. Reliability</p> <p>Due to replication of data in the cluster, data is reliably stored on the cluster of machine despite machine failures. If your machine goes down, then also your data will be stored reliably.</p> <p>5. High Availability</p> <p>Data is highly available and accessible despite hardware failure due to multiple copies of data. If a machine or few hardware crashes, then data will be accessed from another path.</p>	
b	<p><b>Write notes on Crowdsourcing and Mobile business intelligence.</b></p> <p><b>Crowdsourcing</b> is a great way to capitalize on the resources that can build algorithms and predictive models.</p> <p>Crowd sourcing is a cost and time effective method for moderating and curating data. It has no over head costs and produces high quality results with little investment.</p> <p>Crowdsourcing is a disruptive business model whose roots are in technology but is extending beyond technology to other areas.</p> <p>There are various types of crowdsourcing, such as crowd voting, crowd purchasing, wisdom of crowds, crowd funding, and contests.</p> <p>Take for example:</p> <ul style="list-style-type: none"> <li>■ <a href="http://99designs.com/">99designs.com/</a> , which does crowdsourcing of graphic design</li> <li>■ <a href="http://agentanything.com/">agentanything.com/</a> , which posts “missions” where agents vie for to run errands</li> <li>■ <a href="http://33needs.com/">33needs.com/</a> , which allows people to contribute to charitable programs that make a social impact</li> </ul> <p><b>Mobile Intelligence:</b></p> <p>Analytics on mobile devices is what some refer to as putting BI in your pocket. Mobile drives straight to the heart of simplicity and ease of use that has been a major barrier to BI adoption since day one. Mobile devices are a great leveling field where making complicated actions easy is the name of the game.</p> <p>Three elements that have impacted the viability of mobile BI:</p> <ol style="list-style-type: none"> <li>1. Location—the GPS component and location . . . know where you are in time as well as the movement.</li> <li>2. It 's not just about pushing data; you can transact with your smart phone based on information you get.</li> </ol>	

	<p>3. Multimedia functionality allows the visualization pieces to really come into play.</p> <p>Three challenges with mobile BI include:</p> <ol style="list-style-type: none"> <li>1. Managing standards for rolling out these devices.</li> <li>2. Managing security (always a big challenge).</li> <li>3. Managing “bring your own device,” where you have devices both owned by the company and devices owned by the individual, both contributing to productivity.</li> </ol>	
5 a	<p><b>Explain the various open source technologies of Hadoop Eco-System.</b></p> <p><b>Common</b> A set of components and interfaces for distributed filesystems and general I/O (serialization, Java RPC, persistent data structures).</p> <p><b>Avro</b> A serialization system for efficient, cross-language RPC, and persistent data storage.</p> <p><b>MapReduce</b> A distributed data processing model and execution environment that runs on large clusters of commodity machines.</p> <p><b>HDFS</b> A distributed filesystem that runs on large clusters of commodity machines.</p> <p><b>Pig</b> A data flow language and execution environment for exploring very large datasets. Pig runs on HDFS and MapReduce clusters.</p> <p><b>Hive</b> A distributed data warehouse. Hive manages data stored in HDFS and provides a query language based on SQL (and which is translated by the runtime engine to MapReduce jobs) for querying the data.</p> <p><b>HBase</b> A distributed, column-oriented database. HBase uses HDFS for its underlying storage, and supports both batch-style computations using MapReduce and point queries (random reads).</p> <p><b>ZooKeeper</b> A distributed, highly available coordination service. ZooKeeper provides primitives such as distributed locks that can be used for building distributed applications.</p> <p><b>Sqoop</b> A tool for efficiently moving data between relational databases and HDFS.</p>	
b	<p><b>Discuss the difficulties of implementing storage and analysis support for big data.</b></p>	



## Data Storage and Analysis

The problem is simple: although the storage capacities of hard drives have increased massively over the years, access speeds—the rate at which data can be read from drives—have not kept up. One typical drive from 1990 could store 1,370 MB of data and had a transfer speed of 4.4 MB/s,<sup>4</sup> so you could read all the data from a full drive in around five minutes. Over 20 years later, 1-terabyte drives are the norm, but the transfer speed is around 100 MB/s, so it takes more than two and a half hours to read all the data off the disk.

This is a long time to read all data on a single drive—and writing is even slower. The obvious way to reduce the time is to read from multiple disks at once. Imagine if we had 100 drives, each holding one hundredth of the data. Working in parallel, we could read the data in under two minutes.

Using only one hundredth of a disk may seem wasteful. But we can store 100 datasets, each of which is 1 terabyte, and provide shared access to them. We can imagine that the users of such a system would be happy to share access in return for shorter analysis times, and statistically, that their analysis jobs would be likely to be spread over time, so they wouldn't interfere with each other too much.

There's more to being able to read and write data in parallel to or from multiple disks, though.

The first problem to solve is hardware failure: as soon as you start using many pieces of hardware, the chance that one will fail is fairly high. A common way of avoiding data loss is through replication: redundant copies of the data are kept by the system so that in the event of failure, there is another copy available. This is how RAID works, for instance, although Hadoop's filesystem, the Hadoop Distributed Filesystem (HDFS), takes a slightly different approach, as you shall see later.

The second problem is that most analysis tasks need to be able to combine the data in some way, and data read from one disk may need to be combined with data from any of the other 99 disks. Various distributed systems allow data to be combined from multiple sources, but doing this correctly is notoriously challenging. MapReduce provides a programming model that abstracts the problem from disk reads and writes, transforming it into a computation over sets of keys and values. We look at the details of this model in later chapters, but the important point for the present discussion is that there are two parts to the computation—the map and the reduce—and it's the interface between the two where the “mixing” occurs. Like HDFS, MapReduce has built-in reliability.

6 a **List the differences between map reduce and RDBMs**

## Relational Database Management Systems

Why can't we use databases with lots of disks to do large-scale analysis? Why is Hadoop needed?

The answer to these questions comes from another trend in disk drives: seek time is improving more slowly than transfer rate. Seeking is the process of moving the disk's head to a particular place on the disk to read or write data. It characterizes the latency of a disk operation, whereas the transfer rate corresponds to a disk's bandwidth.

If the data access pattern is dominated by seeks, it will take longer to read or write large portions of the dataset than streaming through it, which operates at the transfer rate. On the other hand, for updating a small proportion of records in a database, a traditional B-Tree (the data structure used in relational databases, which is limited by the rate at which it can perform seeks) works well. For updating the majority of a database, a B-Tree is less efficient than MapReduce, which uses Sort/Merge to rebuild the database.

In many ways, MapReduce can be seen as a complement to a Relational Database Management System (RDBMS). (The differences between the two systems are shown in Table 1-1.) MapReduce is a good fit for problems that need to analyze the whole dataset in a batch fashion, particularly for ad hoc analysis. An RDBMS is good for point queries or updates, where the dataset has been indexed to deliver low-latency retrieval and update times of a relatively small amount of data. MapReduce suits applications where the data is written once and read many times, whereas a relational database is good for datasets that are continually updated.<sup>2</sup>

Table 1-1. RDBMS compared to MapReduce

	Traditional RDBMS	MapReduce
Data size	Gigabytes	Petabytes
Access	Interactive and batch	Batch
Updates	Read and write many times	Write once, read many times
Transactions	ACID	None

## b Grid Computing:

The High Performance Computing (HPC) and Grid Computing communities have been doing large-scale data processing for years, using such APIs as Message Passing Interface (MPI). Broadly, the approach in HPC is to distribute the work across a cluster of machines, which access a shared filesystem, hosted by a SAN. This works well for predominantly compute-intensive jobs, but becomes a problem when nodes need to access larger data volumes (hundreds of gigabytes, the point at which MapReduce really starts to shine), since the network bandwidth is the bottleneck and compute nodes become idle.

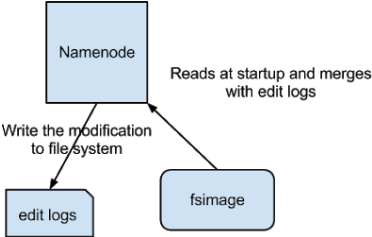
MapReduce tries to collocate the data with the compute node, so data access is fast since it is local. This feature, known as data locality, is at the heart of MapReduce and is the reason for its good performance. Recognizing that network bandwidth is the most precious resource in a data center environment (it is easy to saturate network links by copying data around), MapReduce implementations go to great lengths to conserve it by explicitly modelling network topology. Notice that this arrangement does not preclude high-CPU analyses in MapReduce

## Volunteer Computing:

Volunteer computing projects work by breaking the problem they are trying to solve into chunks called work units, which are sent to computers around the world to be analyzed. For example, a SETI@home work unit is about 0.35 MB of radio telescope data, and takes hours or days to analyze on a typical home computer. When the analysis is completed, the results are sent back to the server, and the client gets another work unit. As a precaution to combat cheating, each work unit is sent to three different machines and needs at least two results to agree to be accepted. Although SETI@home may be superficially similar to MapReduce (breaking a problem into independent pieces

	<p>to be worked on in parallel), there are some significant differences. The SETI@home problem is very CPU-intensive, which makes it suitable for running on hundreds of thousands of computers across the world,8 since the time to transfer the work unit is dwarfed by the time to run the computation on it. Volunteers are donating CPU cycles, not bandwidth. MapReduce is designed to run jobs that last minutes or hours on trusted, dedicated hardware running in a single data center with very high aggregate bandwidth interconnects.</p>	
7 a	<p><b>What is a memory block in HDFS? Explain block report, replication factor and rack awareness with respect to data node.</b></p> <p><b>HDFS concepts:</b></p> <p>A disk has a block size, which is the minimum amount of data that it can read or write. Filesystems for a single disk build on this by dealing with data in blocks, which are an integral multiple of the disk block size. Filesystem blocks are typically a few kilobytes in size, while disk blocks are normally 512 bytes. This is generally transparent to the filesystem user who is simply reading or writing a file – of whatever length. However, there are tools to perform filesystem maintenance, such as df and fsck, that operate on the filesystem block level.</p> <p>HDFS, too, has the concept of a block, but it is a much larger unit–64 MB by default. Like in a filesystem for a single disk, files in HDFS are broken into block-sized chunks, which are stored as independent units.</p> <p><b>Name nodes and Data nodes:</b></p> <p>An HDFS cluster has two types of node operating in a master-worker pattern: a namenode (the master) and a number of datanodes (workers). The namenode manages the filesystem namespace. It maintains the filesystem tree and the metadata for all the files and directories in the tree. This information is stored persistently on the local disk in the form of two files: the namespace image and the edit log. The namenode also knows the datanodes on which all the blocks for a given file are located, however, it does not store block locations persistently, since this information is reconstructed from datanodes when the system starts.</p> <p>A client accesses the filesystem on behalf of the user by communicating with the namenode and datanodes. The client presents a POSIX-like filesystem interface, so the user code does not need to know about the namenode and datanode to function. Datanodes are the workhorses of the filesystem. They store and retrieve blocks when they are told to (by clients or the namenode), and they report back to the namenode periodically with lists of blocks that they are storing.</p>	
b	<p><b>Discuss any 4 HDFS commands.</b></p>	

	<p><b>appendToFile</b></p> <p>Usage: hdfs dfs -appendToFile &lt;localsrc&gt; ... &lt;dst&gt;</p> <p><b>cat</b></p> <p>Usage: hdfs dfs -cat URI [URI ...]</p> <p><b>chgrp</b></p> <p>Usage: hdfs dfs -chgrp [-R] GROUP URI [URI ...]</p> <p><b>chmod</b></p> <p>Usage: hdfs dfs -chmod [-R] &lt;MODE[,MODE]...   OCTALMODE&gt; URI [URI ...]</p>	
--	---	--

<p>8 a</p>	<p><b>Explain the architectural changes that are needed while replacing active name node with stand by name node.</b></p> <p>Namenode: Namenode holds the meta data for the HDFS like Namespace information, block information etc. When in use, all this information is stored in main memory. But these information also stored in disk for persistence storage.</p>  <p>The above image shows how Name Node stores information in disk. Two different files are fsimage - Its the snapshot of the filesystem when namenode started</p>	
------------	--	--

Edit logs - Its the sequence of changes made to the filesystem after namenode started

Only in the restart of namenode , edit logs are applied to fsimage to get the latest snapshot of the file system. But namenode restart are rare in production clusters which means edit logs can grow very large for the clusters where namenode runs for a long period of time. The following issues we will encounter in this situation.

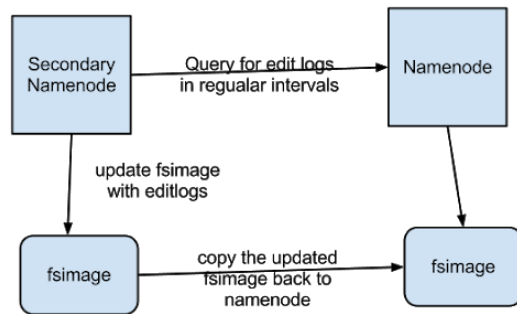
Editlog become very large , which will be challenging to manage it  
Namenode restart takes long time because lot of changes has to be merged.

In the case of crash, we will lost huge amount of metadata since fsimage is very old  
So to overcome this issues we need a mechanism which will help us reduce the edit log size which is manageable and have up to date fsimage ,so that load on namenode reduces . It's very similar to Windows Restore point, which will allow us to take snapshot of the OS so that if something goes wrong , we can fallback to the last restore point.

So now we understood NameNode functionality and challenges to keep the meta data up to date. So what is this all have to with Secondary Namenode?

Secondary Namenode:

Secondary Namenode helps to overcome the above issues by taking over responsibility of merging editlogs with fsimage from the namenode.



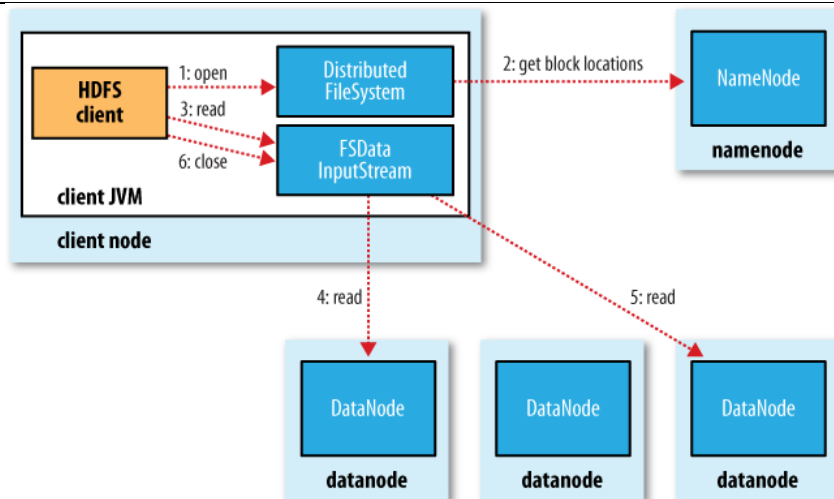
The above figure shows the working of Secondary Namenode

1. It gets the edit logs from the namenode in regular intervals and applies to fsimage
2. Once it has new fsimage, it copies back to namenode
3. Namenode will use this fsimage for the next restart, which will reduce the startup time

Secondary Namenode whole purpose is to have a checkpoint in HDFS. Its just a helper node for namenode. That's why it also known as checkpoint node inside the community.

b **With a neat diagram, explain the anatomy of reading data from a file in HDFS.**

To get an idea of how data flows between the client interacting with HDFS, the namenode and the datanodes, consider Figure which shows the main sequence of events when reading a file.



The client opens the file it wishes to read by calling `open()` on the `FileSystem` object, which for HDFS is an instance of `DistributedFileSystem` (step 1 in Figure).

`DistributedFileSystem` calls the `namenode`, using RPC, to determine the locations of the blocks for the first few blocks in the file (step 2).

For each block, the `namenode` returns the addresses of the `datanodes` that have a copy of that block. Furthermore, the `datanodes` are sorted according to their proximity to the node. If the client is itself a `datanode` (in the case of a MapReduce task, for instance), then it will read from the local `datanode`, if it hosts a copy of the block.

The `DistributedFileSystem` returns an `FSDataInputStream` (an input stream that supports file seeks) to the client for it to read data from. `FSDataInputStream` in turn wraps a `DFSInputStream`, which manages the `datanode` and `namenode` I/O.

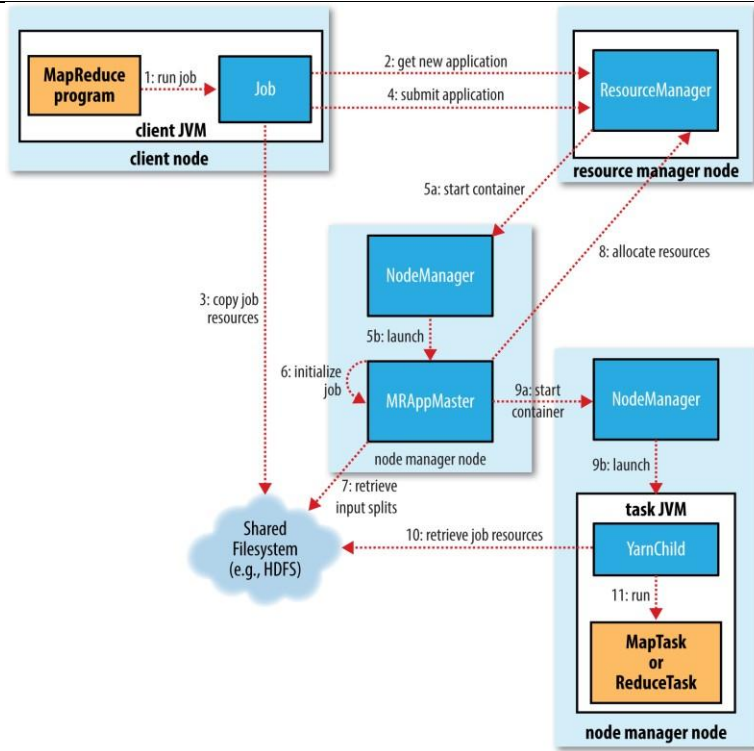
The client then calls `read()` on the stream (step 3). `DFSInputStream`, which has stored the `datanode` addresses for the first few blocks in the file, then connects to the first (closest) `datanode` for the first block in the file. Data is streamed from the `datanode` back to the client, which calls `read()` repeatedly on the stream (step 4).

When the end of the block is reached, `DFSInputStream` will close the connection to the `datanode`, then find the best `datanode` for the next block (step 5). This happens transparently to the client, which from its point of view is just reading a continuous stream.

Blocks are read in order with the `DFSInputStream` opening new connections to `datanodes` as the client reads through the stream. It will also call the `namenode` to retrieve the `datanode` locations for the next batch of blocks as needed. When the client has finished reading, it calls `close()` on the `FSDataInputStream`(step 6)



9 a



- One map task is created for each split which then executes map function for each record in the split.
- It is always beneficial to have multiple splits, because time taken to process a split is small as compared to the time taken for processing of the whole input. When the splits are smaller, the processing is better load balanced since we are processing the splits in parallel.
- However, it is also not desirable to have splits too small in size. When splits are too small, the overload of managing the splits and map task creation begins to dominate the total job execution time.
- For most jobs, it is better to make split size equal to the size of an HDFS block (which is 64 MB, by default).
- Execution of map tasks results into writing output to a local disk on the respective node and not to HDFS.
- Reason for choosing local disk over HDFS is, to avoid replication which takes place in case of HDFS store operation.
- Map output is intermediate output which is processed by reduce tasks to produce the final output.
- Once the job is complete, the map output can be thrown away. So, storing it in HDFS with replication becomes overkill.
- In the event of node failure before the map output is consumed by the reduce task, Hadoop reruns the map task on another node and re-creates the map output.
- Reduce task don't work on the concept of data locality. Output of every map task is fed to the reduce task. Map output is transferred to the machine where

- reduce task is running.
- On this machine the output is merged and then passed to the user defined reduce function.
- Unlike to the map output, reduce output is stored in HDFS (the first replica is stored on the local node and other replicas are stored on off-rack nodes). So, writing the reduce output

b **Write Short notes on Map Reduce UI and Hadoop Logs:**

## The MapReduce Web UI

Hadoop comes with a web UI for viewing information about your jobs. It is useful for following a job's progress while it is running, as well as finding job statistics and logs after the job has completed. You can find the UI at `http://resource-manager-host:8088/`.

## Retrieving the Results

Once the job is finished, there are various ways to retrieve the results. Each reducer produces one output file, so there are 30 part files named `part-r-00000` to `part-r-00029` in the `max-temp` directory.



As their names suggest, a good way to think of these “part” files is as parts of the `max-temp` “file.”

If the output is large (which it isn't in this case), it is important to have multiple parts so that more than one reducer can work in parallel. Usually, if a file is in this partitioned form, it can still be used easily enough—as the input to another MapReduce job, for example. In some cases, you can exploit the structure of multiple partitions to do a map-side join, for example (see “Map-Side Joins” on page 269).

This job produces a very small amount of output, so it is convenient to copy it from HDFS to our development machine. The `-getmerge` option to the `hadoop fs` command is useful here, as it gets all the files in the directory specified in the source pattern and merges them into a single file on the local filesystem:

Table 5-2. Types of Hadoop logs

Logs	Primary audience	Description	Further information
System daemon logs	Administrators	Each Hadoop daemon produces a logfile (using <code>log4j</code> ) and another file that combines standard out and error. Written in the directory defined by the <code>HADOOP_LOG_DIR</code> environment variable.	“System log-files” on page 307 and “Logging” on page 349.
HDFS audit logs	Administrators	A log of all HDFS requests, turned off by default. Written to the namenode's log, although this is configurable.	“Audit Logging” on page 344.

Logs	Primary audience	Description	Further information
MapReduce job history logs	Users	A log of the events (such as task completion) that occur in the course of running a job. Saved centrally on the jobtracker, and in the job's output directory in a <code>_logs/history</code> subdirectory.	<a href="#">"Job History" on page 166.</a>
MapReduce task logs	Users	Each tasktracker child process produces a logfile using log4j (called <code>syslog</code> ), a file for data sent to standard out ( <code>stdout</code> ), and a file for standard error ( <code>stderr</code> ). Written in the <code>userlogs</code> subdirectory of the directory defined by the <code>HADOOP_LOG_DIR</code> environment variable.	This section.

10  
a

```

Example 2-3. Mapper for the maximum temperature example
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MaxTemperatureMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {

    private static final int MISSING = 9999;

    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

        String line = value.toString();
        String year = line.substring(15, 19);
        int airTemperature;
        if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs
            airTemperature = Integer.parseInt(line.substring(88, 92));
        } else {
            airTemperature = Integer.parseInt(line.substring(87, 92));
        }
        String quality = line.substring(92, 93);

        if (airTemperature != MISSING && quality.matches("[01459]")) {
            context.write(new Text(year), new IntWritable(airTemperature));
        }
    }
}

```

	<p><i>Example 2-4. Reducer for the maximum temperature example</i></p> <pre> import java.io.IOException;  import org.apache.hadoop.io.IntWritable; import org.apache.hadoop.io.Text; import org.apache.hadoop.mapreduce.Reducer;  public class MaxTemperatureReducer     extends Reducer&lt;Text, IntWritable, Text, IntWritable&gt; {      @Override     public void reduce(Text key, Iterable&lt;IntWritable&gt; values, Context context)         throws IOException, InterruptedException {          int maxValue = Integer.MIN_VALUE;         for (IntWritable value : values) {             maxValue = Math.max(maxValue, value.get());         }         context.write(key, new IntWritable(maxValue));     } } </pre> <p><i>Example 2-5. Application to find the maximum temperature in the weather dataset</i></p> <pre> import org.apache.hadoop.fs.Path; import org.apache.hadoop.io.IntWritable; import org.apache.hadoop.io.Text; import org.apache.hadoop.mapreduce.Job; import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  public class MaxTemperature {      public static void main(String[] args) throws Exception {         if (args.length != 2) {             System.err.println("Usage: MaxTemperature &lt;input path&gt; &lt;output path&gt;");             System.exit(-1);         }          Job job = new Job();         job.setJarByClass(MaxTemperature.class);         job.setJobName("Max temperature");          FileInputFormat.addInputPath(job, new Path(args[0]));         FileOutputFormat.setOutputPath(job, new Path(args[1]));          job.setMapperClass(MaxTemperatureMapper.class);         job.setReducerClass(MaxTemperatureReducer.class);          job.setOutputKeyClass(Text.class);         job.setOutputValueClass(IntWritable.class);          System.exit(job.waitForCompletion(true) ? 0 : 1);     } } </pre>	
b	<b>How does a Map Reduce Model works with a Single Reduce task?</b>	

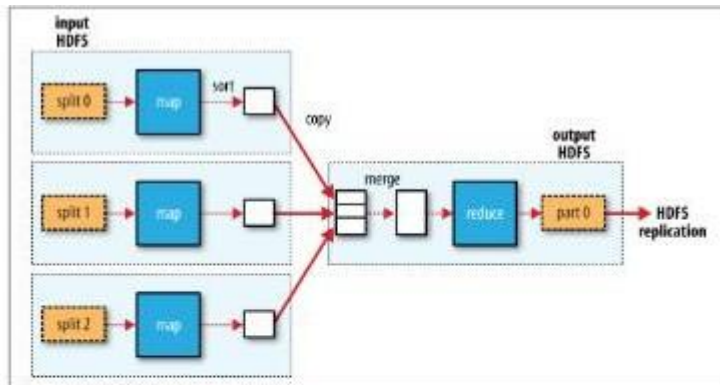


Figure 2-3. MapReduce data flow with a single reduce task

The number of reduce tasks is not governed by the size of the input, but instead is specified independently. In “The Default MapReduce Job” on page 214, you will see how to choose the number of reduce tasks for a given job.

When there are multiple reducers, the map tasks *partition* their output, each creating one partition for each reduce task. There can be many keys (and their associated values) in each partition, but the records for any given key are all in a single partition. The partitioning can be controlled by a user-defined partitioning function, but normally the default partitioner—which buckets keys using a hash function—works very well.