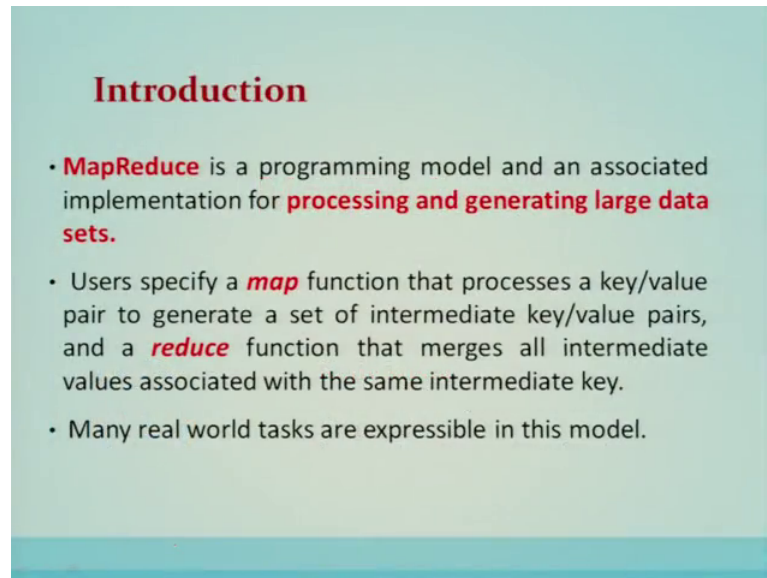


Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture - 21
Map Reduce

(Refer Slide Time: 00:16)



Introduction

- **MapReduce** is a programming model and an associated implementation for **processing and generating large data sets**.
- Users specify a **map** function that processes a key/value pair to generate a set of intermediate key/value pairs, and a **reduce** function that merges all intermediate values associated with the same intermediate key.
- Many real world tasks are expressible in this model.

Map reduce. Introduction: Map reduce is a programming model, and an associated implementation for processing and generating large datasets. Users is specify a map function that processes the key value pair to generate a set of intermediate key value pairs, and a reduced function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model.

(Refer Slide Time: 00:49)

Contd...

- Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines.
- The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication.
- This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The runtime system takes care of the details of partitioning of the input data; scheduling the programs execution across the set of machines, handling machines failures, and managing the required inter machine communication. This allows the programmers without any experience with parallel and distributed system to easily utilize the resources of a large distributed systems.

(Refer Slide Time: 01:26)

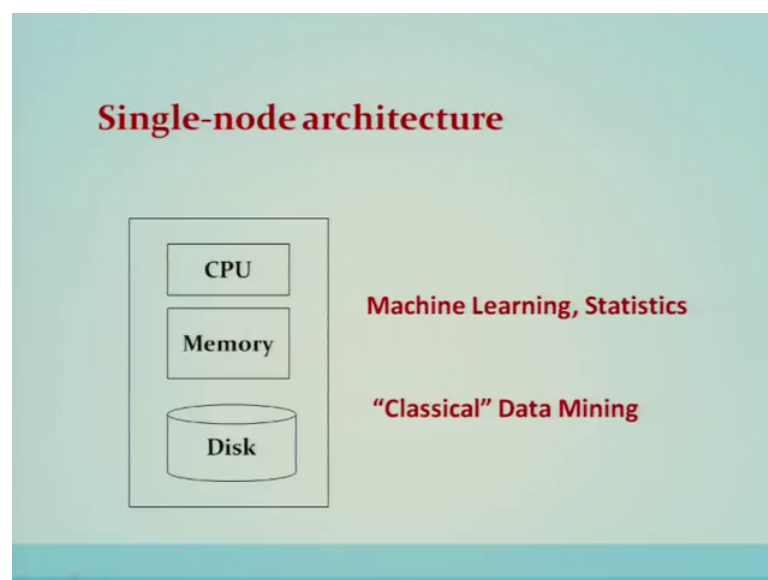
Contd...

- A typical MapReduce computation processes many terabytes of data on thousands of machines.
- Hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

A typical map reduce computation processes may have the data of terabytes of data on thousands of machines, hundreds of map reduce programs have been implemented, and upward of one thousand map reduce jobs are executed on a Google's cluster every day, meaning to say that a very large data set computation, which cannot be handled using the classical algorithms or the existing machines, how are they going to be computed, and how the Google is now doing it, that we are going to see in this lecture.

So, the Google has devised a programming paradigm, which exploits the cluster machines. So, using exploitation of the cluster machines, the distribution of large data set into small chunks on different nodes, which are distributed across the clusters on different nodes, will be used to compute. So, how the computation, how the programmer will write down those programs. So, Google has given this particular paradigm which is called a map reduce paradigm, or a programming paradigm which will allow the programmers to comfortably write the programs, for their application, without bothering the intricacies of underlying distributed system, and the distribution of program, or distribution of data programs, and also how the communication is going to take place. All these inter dependencies or intricacies are purely hidden, and an abstraction is available; that is in the form of a map reduce.

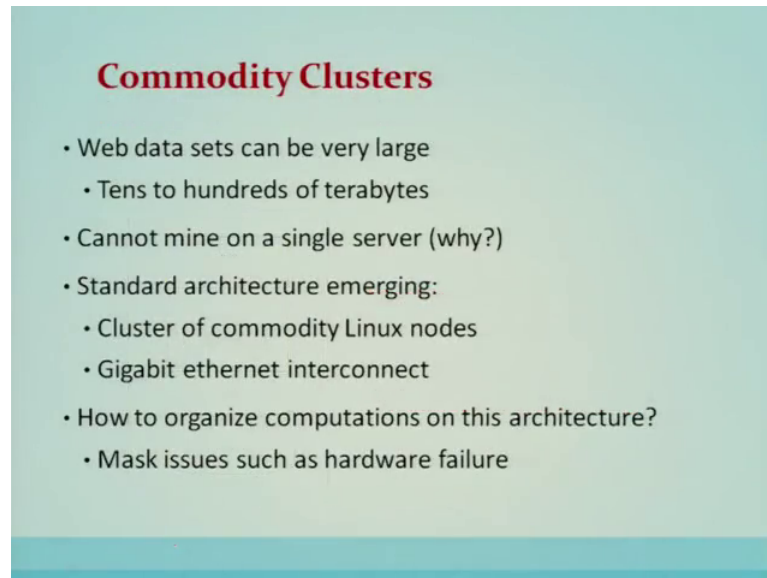
(Refer Slide Time: 03:35)



So, in this particular lecture we are going to introduce you this particular notion and we are going to give you several examples based on that. So, let us see the model or the

architecture, which is going to be used for map reduce programming. So, this is a single node architecture, where it consists or it is having one CPU, memory and a disk.

(Refer Slide Time: 03:57).



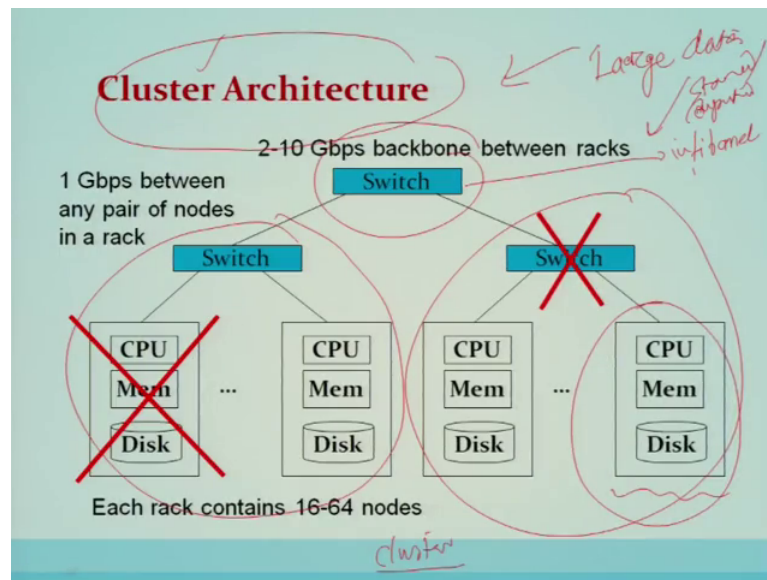
Commodity Clusters

- Web data sets can be very large
 - Tens to hundreds of terabytes
- Cannot mine on a single server (why?)
- Standard architecture emerging:
 - Cluster of commodity Linux nodes
 - Gigabit ethernet interconnect
- How to organize computations on this architecture?
 - Mask issues such as hardware failure

This particular node, collection of such nodes we are going to form a cluster, and that will be commodity cluster if the normal machines like laptops and the desktops are used, then it is called commodity cluster.

So, web data sets can be very large tens to the thousands of terabytes. So, standard architectures which are emerging to accommodate the large data sets are like cluster of commodity nodes, gigabit Ethernet interconnect will be used in it, how to organize the computation in this architecture.

(Refer Slide Time: 04:40)



So, this is an example of a cluster architecture. So, earlier we have seen one such node. So, several such nodes, which are connected through a switch, and further another switch will connect, the complete set of different nodes. So, I no doubt thousand, a cluster of a thousand nodes is also available for this particular computation. So, these switches are a very fast switch, and infiniband switch is a gigabit switch, or gigabit internet is used for a very high speed interconnect across these nodes.

So, this is an architecture for the cluster, which consists of 64 nodes shown here in this particular figure. So, given this particular architecture how a program, can be written which will exploit this kind of architecture. So, a large data set can be stored, and can be computed, which is now not possible to be utilized with the single node. So, this is the current state of the art, which the Google and other big companies are now using it. So, we are going to see this new technology.

(Refer Slide Time: 06:21)

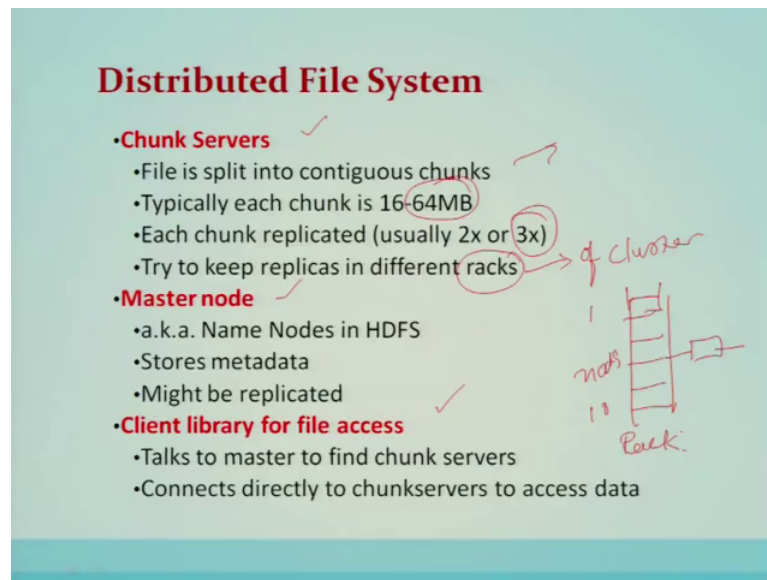
Stable storage

- First order problem: if nodes can fail, how can we store data persistently? (The word 'persistently' is circled in red, and 'norm' is written above it with an arrow pointing to the circle.)
- Answer: Distributed File System (Circled in red)
 - Provides global file namespace ✓
 - Google GFS; Hadoop HDFS; Kosmix KFS ✓
- Typical usage pattern
 - Huge files (100s of GB to TB) ✓
 - Data is rarely updated in place ✓
 - Reads and appends are common ✓

So, to utilize this cluster architecture, we require something which is called a distributed file system, which we have already covered in the previous lectures. So, distributed file system will provide a single abstraction of a system to the programmer, and that will be in the form of Google file system and Hadoop HDFS file system, and Kosmix KFS, which will give the global file name a space. So, typical usage patterns is, it is able to accommodate huge files that is hundreds of terabytes, and data is rarely updated in place, and reads and appends are very common operations, when we are dealing with the large data set.

These commodity nodes, they also are failing. So; obviously, the norm failure is becoming a norm in such kind of commodity clusters. So, how can we store these particular stores and compute these data persistently. So, we assume there is a distributed file system; like Google file system or HDFS file system is in place which will ensure the stable storage.

(Refer Slide Time: 07:56)



That is the faults are fault tolerance systems are already in place within that particular system, and the failures and faults will be handled accordingly.

Distributed file system comprises of file chunk servers, master nodes and a client library for file access. Chunk servers is nothing, but where the files is split into a contiguous chunks and they are being stored. So, each chunk is of size 64 Mb. So, each chunk is replicated three times, and try to keep these replicas on a different racks of the cluster. So, the rack in a cluster consists of, or a stores more than one nodes, and this is being interconnected with the fast switch. So, this is called a rack, which is located at in one unit.

(Refer Slide Time: 09:12)

Motivation for Map Reduce (Why)

- **Large-Scale Data Processing**
 - Want to use 1000s of CPUs
 - But don't want hassle of managing things
- **MapReduce Architecture provides**
 - Automatic parallelization & distribution *of large data sets*
 - Fault tolerance *(nodes) can fail.*
 - I/O scheduling ✓
 - Monitoring & status updates ✓

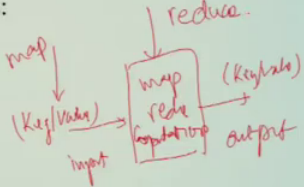
So, motivation for map reduce, is to support the large scale data processing, which otherwise is not possible on a single node, or on a supercomputer, because they have the finite space, and finite computing power. Now if we are considering a cluster, a big cluster of thousand nodes, which has the storage and processing capabilities now how can this particular cluster of a very big cluster can be utilized, to store the data and to compute the data, and what will be the programming model for the programmers without dealing with the intricacies. So, we are going to see that environment, programming environment which is called a map reduce, which will be used for a large scale data processing. So, here the programs are quite simple, but the data is very large. So, that becomes a challenge, how a very large program can be computed.

The map reduce architecture will provide automatic parallelization, and the distribution of the large data sets. It also ensures the fault tolerance why, because the nodes can fail at any point of time, and I O scheduling is also required and monitoring, and status updates, how that is all done in that particular architecture.

(Refer Slide Time: 11:04)

Programming Model

- The computation takes a set of **input key/value pairs**, and produces a set of **output key/value pairs**.
- The user of the MapReduce library expresses the computation as two functions:
 - (i) The Map
 - (ii) The Reduce

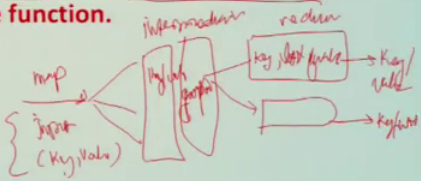


So, programming model, the computation takes a set of input key value pairs, and produces a set of output key value pairs. So, the user of a map reduce library expresses the computation as two functions, which is called a map and the other function is called a reduce.

(Refer Slide Time: 12:01)

(i) Map Abstraction

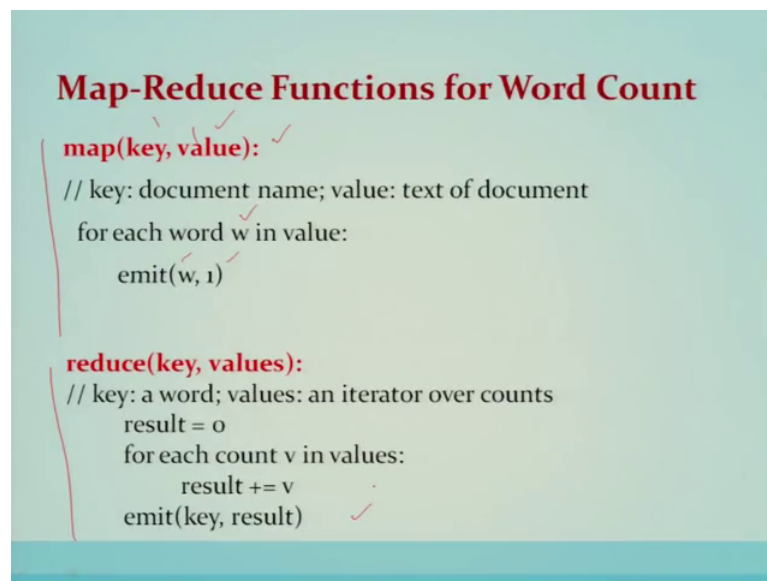
- Map, written by the user, takes an input pair and produces a set of **intermediate key/value pairs**.
- The MapReduce library groups together all intermediate values associated with the same **intermediate key 'I'** and passes them to the **Reduce function**.



So, map function is written by the user takes the input pair and produces the intermediate key value pairs. The map reduce library grouped together, all the intermediate values associated with the same intermediate keys, and passes them to the next page; that is the

reducer phase. Meaning to say that the input is in the form of a key value pair, where the map function will be applied on this particular data set. This is going to generate an intermediate key value pair, which is grouped together with the group together. So, that all the intermediate values associated with the same intermediate key I, is now being passed same key, and there is a list of values. These are passed to the reducer and that will give an output again in the form of the key value pair.

(Refer Slide Time: 13:39)



So, the key value; the key value, using the key value pair the map reduce functions for word count, is given in this particular manner. We can see there is a function which is called a map, which will take key value pair, the map will read each word, which is given as a part of the value, and for each word, it emits one similarly as far as the reducer is another program, which programmer has write down, which is based on the output or the output of the mapper is called the intermediate results, and these intermediate values are taken by the reducer to produce the final output; that is in the form of a key value pair. So, we will explain this particular example in this lecture.

(Refer Slide Time: 14:34)

Map-Reduce Functions

- **Input:** a set of key/value pairs ✓
- User supplies two functions:
 - ✓ $\text{map}(k, v) \rightarrow \text{list}(k_1, v_1) \rightarrow$
 - ✓ $\text{reduce}(k_1, \text{list}(v_1)) \rightarrow v_2$ ✓ (k_1, v_2)
- (k_1, v_1) is an intermediate key/value pair
- **Output** is the set of (k_1, v_2) pairs ✓

So, map reduce functions as I told you that input will be in the form of a set of key value pairs. So, the user has to supply two different functions map, and reduce based on the application, which they have in the mind. So, the map are based on the input set of key value pairs, it will form a list of key value pairs, and it will group according to a particular key, all the values and that will be given to the reducer. So, reducer will take the key and all group values as the input for the reducer, and it will generate an output which will be the key value pair, again the values based on a particular key.

So, output again will be that key k_1 , and its value k_2 will be the output. So, key value pair will be the output. So, this is a simple explanation that map reduce function provides this simple constructs of map, and reduce. Now programmer has to fill in, what is the map, and what is the reducer reduce functions as per the application is concerned.

(Refer Slide Time: 15:58)

Applications

- Here are a few simple applications of interesting programs that can be easily expressed as **MapReduce computations**.
- **Distributed Grep**: The map function emits a line if it matches a supplied pattern. The reduce function is an identity function that just copies the supplied intermediate data to the output.
- **Count of URL Access Frequency**: The map function processes logs of web page requests and outputs (URL; 1). The reduce function adds together all values for the same URL and emits a (URL; total count) pair.
- **ReverseWeb-Link Graph**: The map function outputs (target; source) pairs for each link to a target URL found in a page named source. The reduce function concatenates the list of all source URLs associated with a given target URL and emits the pair: (target; list(source))

Let us see different applications, where this map reduce paradigm or a model of programming is going to be used. There are a few simple applications of interesting programs that can easily be expressed as map reduce computations, distributed grep count of URL access frequency reverse web link graph term vector per host and so on, inverted index distributed sort.

(Refer Slide Time: 16:20)

Contd...

- **Term-Vector per Host**: A term vector summarizes the most important words that occur in a document or a set of documents as a list of (word; frequency) pairs.
- The map function emits a (hostname; term vector) pair for each input document (where the hostname is extracted from the URL of the document).
- The reduce function is passed all per-document term vectors for a given host. It adds these term vectors together, throwing away infrequent terms, and then emits a final (hostname; term vector) pair

(Refer Slide Time: 16:24)

Contd...

- **Inverted Index:** The map function parses each document, and emits a sequence of (word; document ID) pairs. The reduce function accepts all pairs for a given word, sorts the corresponding document IDs and emits a (word; list(document ID)) pair. The set of all output pairs forms a simple inverted index. It is easy to augment this computation to keep track of word positions.
- **Distributed Sort:** The map function extracts the key from each record, and emits a (key; record) pair. The reduce function emits all pairs unchanged.

Let us see the implementation. So, many different implementations of the map reduce interface are provided are possible.

(Refer Slide Time: 16:31)

Implementation Overview

- Many different implementations of the MapReduce interface are possible. The right choice depends on the environment.
- For example, one implementation may be suitable for a small shared-memory machine, another for a large NUMA multi-processor, and yet another for an even larger collection of networked machines.
- Here we describes an implementation targeted to the computing environment in wide use at Google: large clusters of commodity PCs connected together with switched Ethernet.

The right choice depends on the environment. For example, one implementation may be suitable for a small shared memory machine, and another for a large NUMA multiprocessor, and yet another machine for even bigger collection of networked machines. Here we describe an implementation targeted to the computing environment, which is in wide use at Google.

(Refer Slide Time: 17:07)

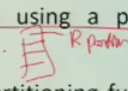
Contd...

- (1) Machines are typically dual-processor x86 processor running Linux, with 2-4 GB of memory per machine.
- (2) Commodity networking hardware is used . Typically either 100 megabits/second or 1 gigabit/second at the machine level, but averaging considerably less in overall bisection bandwidth.
- (3) A cluster consists of hundreds or thousands of machines, and therefore machine failures are common.
- (4) Storage is provided by inexpensive IDE disks attached directly to individual machines.
- (5) Users submit jobs to a scheduling system. Each job consists of a set of tasks, and is mapped by the scheduler to a set of available machines within a cluster.

So, Google, large clusters of commodity PCs connected together through a fast Ethernet switch. So, this is the description of the cluster machine that we have already explained.

(Refer Slide Time: 17:15)

Distributed Execution Overview

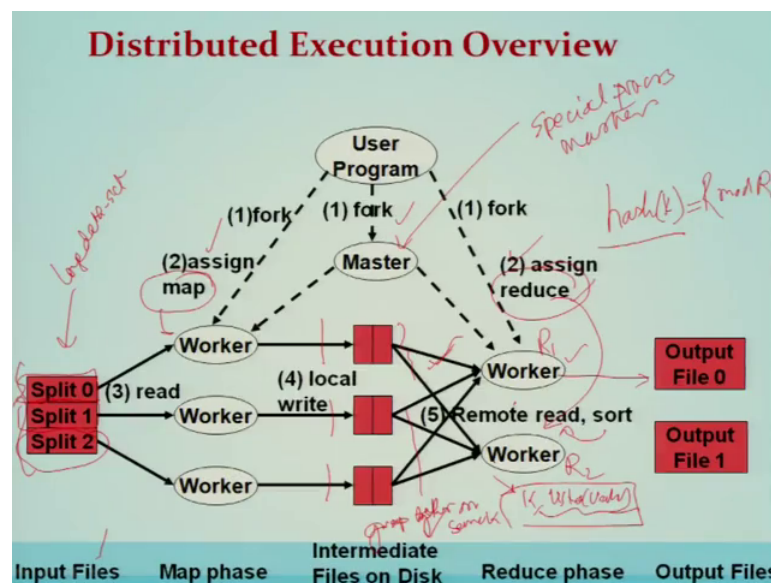
- The **Map invocations** are distributed across multiple machines by automatically partitioning the input data into a set of M splits.
- The input splits can be processed in parallel by different machines.
- **Reduce invocations** are distributed by partitioning the intermediate key space into R pieces using a partitioning function (e.g., $\text{hash}(\text{key}) \bmod R$). 
- The number of partitions (R) and the partitioning function are specified by the user.
- Figure 1 shows the overall flow of a MapReduce operation.

Let us consider the distributed environment or distributed execution, which basically takes place on invocation of map and reduce functions. So, the map invocations are distributed across multiple machines by automatically partitioning the input data into the set of M splits. The input splits can be processed in parallel by different machines; that is all taken care, when the map is invoke reduced function is invoked; that is called reduce

invocations here, that the distributed these reducers, the reducing invocations are distributed by partitioning the intermediate keyvalue space into R pieces using the partitioning function, which is nothing, but a hash function on the mod R.

So, number of partitions are n the number and the partitioning functions are specified by the user figure shows, the overall flow of the map reduce operations.

(Refer Slide Time: 18:26)



So, this is going to be a distributed execution. Now the user program is in the form of different threads. So, three different processes here it is shown will be created, and the first one is a special process, which is called a master, and all other processors, all other processes will be assigned, either the map function or the reduced function. So, the map function will on a particular worker, the map function will take the input in the form of a splits input is a big file, which is splitted into some multiple of chunks, which can be stored on of different nodes. So, these splits of all data set, and which is given as an input in the form of input file is split.

So, here it is shown as three splits. So, three is splits, means three workers will be reading it in parallel, and applying the map function on these splits, then afterwards the next step is that the output of the map function will be written locally, or that is being buffered locally, then as far as these intermediate key value pairs are concerned. So, these values which are read from the local buffer, and then basically they are grouped together on the same key. So, for a particular key, all the values which are being grouped,

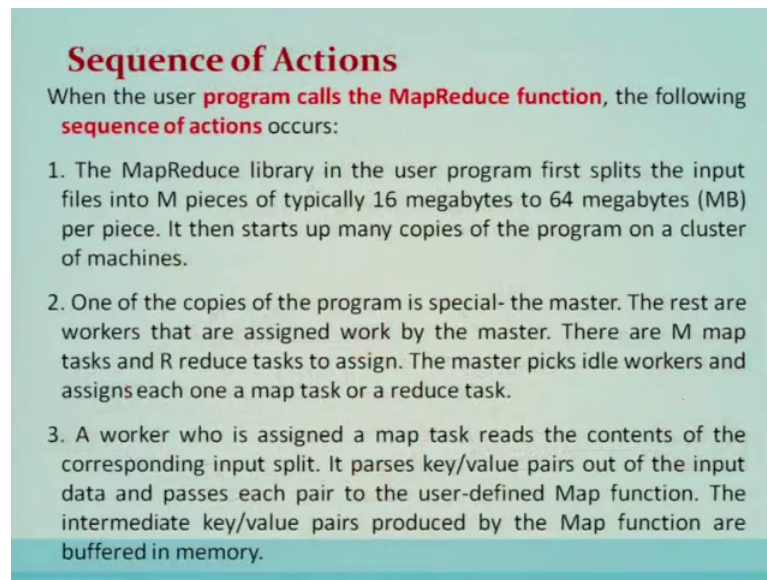
they are being given now to the server. So, serve to the worker which basically will be running the reduce functions.

So, the reducer which is running on a worker, will remote read to four performing read, it will do a sort also. So; that means, for a particular key all the list of values; that means, according to the key, it will be sorted, and all the values for a particular key. These particular values will be given in the form of a list to the reducer. So, reducer will perform the reduced functions on that particular intermediate key, and its corresponding list of values and the same program will happen on another worker. So, it depends upon how many workers are there, this particular data will be now targeted on this particular worker.

So, it depends upon the hash function on a particular key, and this hash will be divided into R different workers, this is $R_1 R_2$. So, it will be something $\text{mod } R$. So, the hash; that means, these intermediate values which are coming, they are to be hashed on a worker of size R with using $\text{mod } R$. So, that is we have explained here in the reducer invocation by partitioning the input intermediate key space into R pieces, using partitioning function $\text{hash key mod } R$. So, R different partitions of the intermediate values will now take place, and each partition is given to the worker. So, how many workers are assigned with a reducer program is fixed by R . So, many; that means, the intermediate key value pair, is partitioned further into R s plates, and they are being assigned to the reducer in this particular form.

So, the partitioning is done at two levels; one is at the input data, the other partitioning is done for the intermediate values. So, after applying the reducer on each different partition, it will produce an output file in the form of the key value pairs.

(Refer Slide Time: 23:59)



Sequence of Actions

When the user **program calls the MapReduce function**, the following **sequence of actions** occurs:

1. The MapReduce library in the user program first splits the input files into M pieces of typically 16 megabytes to 64 megabytes (MB) per piece. It then starts up many copies of the program on a cluster of machines.
2. One of the copies of the program is special- the master. The rest are workers that are assigned work by the master. There are M map tasks and R reduce tasks to assign. The master picks idle workers and assigns each one a map task or a reduce task.
3. A worker who is assigned a map task reads the contents of the corresponding input split. It parses key/value pairs out of the input data and passes each pair to the user-defined Map function. The intermediate key/value pairs produced by the Map function are buffered in memory.

So, this sequence of action whatever I have explained through the figure, is again going to be explained. So, when a user program also the map reduce function, the following sequence of action occurs; first actions is that the map reduce library in the user program first is splits, the input file into M pieces, of typically 64 Mb. If you recall 64 MB is the size of the chunk. Then it, it then starts up many copies of the program on a cluster machine. So, the first step is over, once the input is split.

Now, one of the copies of the program is a special that is called a master and the rest of the workers are assigned by the master. There are M map tasks and are reduce tasks to assign. So, the map picks the ideal workers, and assign one map task and one reduce task. So, that is what I have explained that, user program will create the threads. One thread will be master, and all other threads will be a sign the worker job, and these workers will be assigned, either to do a map task, or basically to do the reduced task. So, the worker who is assigned the map task reads the contents of the corresponding input split, it parses key value pairs, out of the input data and passes each pair of user defined map function.

(Refer Slide Time: 25:49)

Contd...

4. Periodically, the buffered pairs are written to local disk, partitioned into R regions by the partitioning function.
 - The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the reduce workers.
5. When a reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers. When a reduce worker has read all intermediate data, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together.
 - The sorting is needed because typically many different keys map to the same reduce task. If the amount of intermediate data is too large to fit in memory, an external sort is used.

So, the map function will generate the intermediate key value pairs, and that is buffered in the memory. Periodically the buffered pairs are written to the local disk, then they are partitioned into R regions by a partitioning function that I have explained, which is nothing, but an hashing with a mod R. So, the locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the reduce workers.

So, the information about partitioning of intermediate values is basically given to the master. And when reduced worker is notified by the master about these locations, it uses remote procedure call to read the buffer data from the local disk of the map workers, when a reduced worker has read all the intermediate data is sort sheet by the intermediate key. So, that all the occurrences of the same key are grouped together, that I have also explained before giving the intermediate split to the reducer.

Sorting is needed, because typical many different keys map to the same reduce task. If the amount of intermediate data is too large to fit in the memory and external sorting is used

(Refer Slide Time: 27:08)

Contd...

6. The reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function.
 - The output of the Reduce function is appended to a final output file for this reduce partition.
7. When all map tasks and reduce tasks have been completed, the master wakes up the user program.
 - At this point, the MapReduce call in the user program returns back to the user code.

Sixth's steps says that the reduce workers iterates over the sorted intermediate data, and for each unique intermediate key and counter, it passes the key and the corresponding set of intermediate values to the users the reduced function the output of the reduced function is appended to the output file, for this reduce partition, when all the map tasks and reduce have been completed the master wakes up the user program. At this point the map reduce call in the user program returned back to the user code.

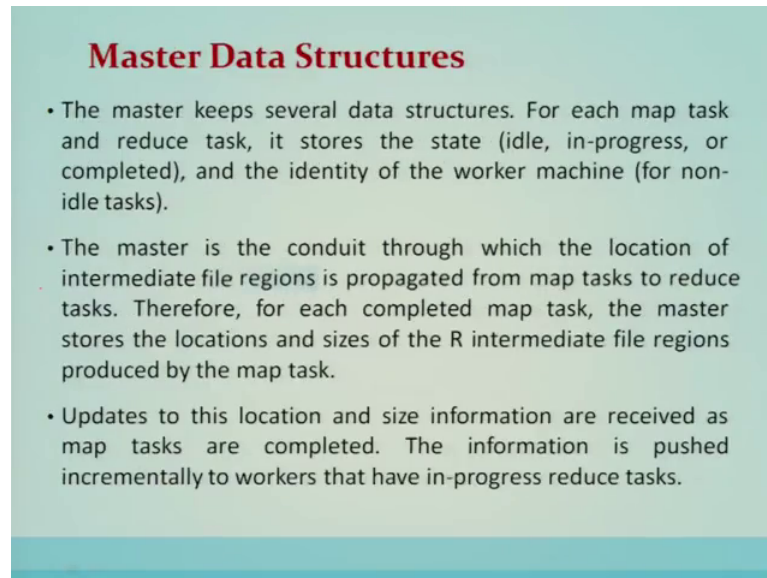
(Refer Slide Time: 27:42)

Contd...

- After successful completion, the output of the mapreduce execution is available in the R output files (one per reduce task, with file names as specified by the user).
- Typically, users do not need to combine these R output files into one file- they often pass these files as input to another MapReduce call, or use them from another distributed application that is able to deal with input that is partitioned into multiple files.

So, after successful completion the output of the map reduce execution is available in R output files; one per reduced task with the file names is specified by the typically users do not need to combine these R output file into one file. They often pass these files as input to another map reduce call, or use them from another distributed application; that is able to deal with the input; that is partition into multiple files.

(Refer Slide Time: 28:11)



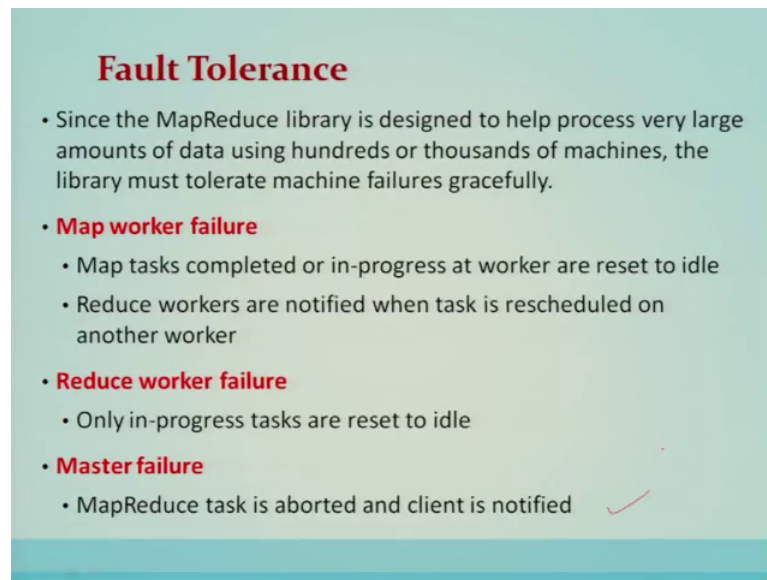
Master Data Structures

- The master keeps several data structures. For each map task and reduce task, it stores the state (idle, in-progress, or completed), and the identity of the worker machine (for non-idle tasks).
- The master is the conduit through which the location of intermediate file regions is propagated from map tasks to reduce tasks. Therefore, for each completed map task, the master stores the locations and sizes of the R intermediate file regions produced by the map task.
- Updates to this location and size information are received as map tasks are completed. The information is pushed incrementally to workers that have in-progress reduce tasks.

So, here let us go into more details of the master data structure. The master keeps several data structures for each map task and reduce tasks it is stores, the state that is idle in progress or completed, and the identity of the worker machine for non ideal tasks. So, you know that master has the complete global view of all the workers and. So, the states of all the processes are maintained, and also the identity of the worker machines, because they are going to be used in this particular computation.

So, the master is the conduit through, which the location of the intermediate regions is propagated from map tasks to the reduce task. Therefore, each completed map tasks the master stores the location and sizes of R intermediate file regions produced by the map tasks that we have already explained. Updates to this location in size information are received as the map tasks are completed. The information is pushed incrementally to the workers, that have been progress reduce tasks.

(Refer Slide Time: 29:22)

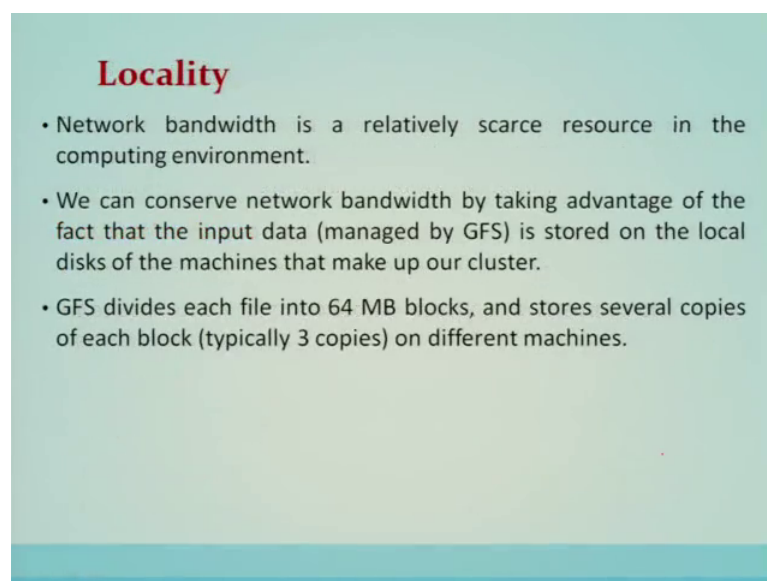


Fault Tolerance

- Since the MapReduce library is designed to help process very large amounts of data using hundreds or thousands of machines, the library must tolerate machine failures gracefully.
- **Map worker failure**
 - Map tasks completed or in-progress at worker are reset to idle
 - Reduce workers are notified when task is rescheduled on another worker
- **Reduce worker failure**
 - Only in-progress tasks are reset to idle
- **Master failure**
 - MapReduce task is aborted and client is notified ✓

Now, another aspect is called fault tolerance. Since map reduce library, is designed to help process, very large amount of data using hundreds of thousands of machines. The library must tolerate machine failures gracefully. When a map worker fails, so the map task completed or in progress at the worker are reset to the idle, reduced worker are notified when the task is rescheduled on another worker. Reduce worker failure only in progress tasks are reset to the idle, master failure, so map reduce task is aborted and the client is notified.

(Refer Slide Time: 30:10)

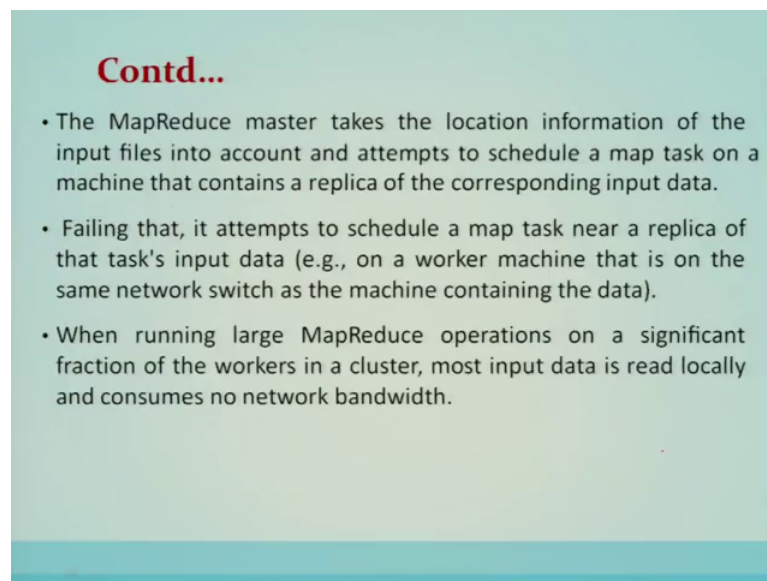


Locality

- Network bandwidth is a relatively scarce resource in the computing environment.
- We can conserve network bandwidth by taking advantage of the fact that the input data (managed by GFS) is stored on the local disks of the machines that make up our cluster.
- GFS divides each file into 64 MB blocks, and stores several copies of each block (typically 3 copies) on different machines.

So, if the master is failed, then the entire setup is aborted, and the client is notified. Locality network bandwidth is relatively scarce resource in the computing environment, we can conserve the network bandwidth by taking advantage of the fact that input data is stored on a local disk of the machines that can make up our cluster. GFS divides each file into 64 MB chunks, and stores several copies of each block; that is 3 copies on different machines.

(Refer Slide Time: 30:36)



Contd...

- The MapReduce master takes the location information of the input files into account and attempts to schedule a map task on a machine that contains a replica of the corresponding input data.
- Failing that, it attempts to schedule a map task near a replica of that task's input data (e.g., on a worker machine that is on the same network switch as the machine containing the data).
- When running large MapReduce operations on a significant fraction of the workers in a cluster, most input data is read locally and consumes no network bandwidth.

So, map reduce master takes the location information of the input into the information attempt to schedule the map tasks on the machines. Failing that it attempt to schedule map tasks, near the replica of the task input, when running the large map reduce operation on a significant, fraction of the workers in a cluster most input, is read locally and consumes no network bandwidth.

(Refer Slide Time: 31:04)

Task Granularity

- The **Map phase is subdivided into M pieces and the reduce phase into R pieces.**
- Ideally, M and R should be much larger than the number of worker machines.
- Having each worker perform many different tasks improves dynamic load balancing, and also speeds up recovery when a worker fails: the many map tasks it has completed can be spread out across all the other worker machines.
- There are practical bounds on how large M and R can be, since the master must make **$O(M + R)$ scheduling decisions** and keeps **$O(M * R)$ state in memory.**
- Furthermore, R is often constrained by users because the output of each reduce task ends up in a separate output file.

So, the next task is. So, task granularity. So, the map phase is subdivided into M pieces and reduce phase in R pieces. Ideally M and R should be much larger than the number of workers nodes. So, here there are the bounds, practical bounds on how large M and R can be, that can be a granularity. So, granularity is mentioned here that the, there is a master must take order M plus R scheduling decisions, and keeps order M star M multiplied by R state in the merge in the memory. Further R is often constraint by the users, because the output of each reduce task ends in a separate output file.

(Refer Slide Time: 31:56)

Partition Function

- Inputs to map tasks are created by contiguous splits of input file
- For reduce, we need to ensure that records with the same intermediate key end up at the same worker
- System uses a default partition function e.g., **$\text{hash}(\text{key}) \bmod R$**
- Sometimes useful to override
 - E.g., **$\text{hash}(\text{hostname}(\text{URL})) \bmod R$** ensures URLs from a host end up in the same output file

(Refer Slide Time: 31:59)

Ordering Guarantees

- We guarantee that within a given partition, the intermediate key/value pairs are processed in increasing key order.
- This ordering guarantee makes it easy to generate a sorted output file per partition, which is useful when the output file format needs to support efficient random access lookups by key, or users of the output find it convenient to have the data sorted.

So, partition function that I have already discussed, ordering guarantees are also taken care of. So, and combiner function is also we have discussed.

(Refer Slide Time: 32:05)

Combiners Function (1)

- In some cases, there is significant repetition in the intermediate keys produced by each map task, and the user specified Reduce function is commutative and associative.
- A good example of this is the word counting example. Since word frequencies tend to follow a Zipf distribution, each map task will produce hundreds or thousands of records of the form <the, 1>.
- All of these counts will be sent over the network to a single reduce task and then added together by the Reduce function to produce one number. We allow the user to specify an optional Combiner function that does partial merging of this data before it is sent over the network.

Let us go ahead with few examples. The first example is called word count.

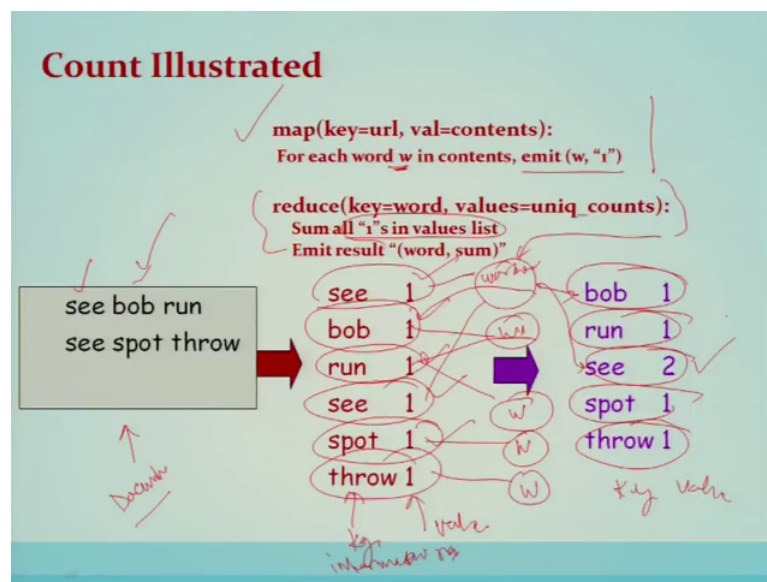
(Refer Slide Time: 31:12)

Example: 1 Word Count using MapReduce

```
map(key, value):  
  // key: document name; value: text of document  
  for each word w in value:  
    emit(w, 1)  
  
reduce(key, values):  
  // key: a word; values: an iterator over counts  
  result = 0  
  for each count v in values:  
    result += v  
  emit(key, result)
```

So, if a document is given, and we are going to count how many frequency, or what is the frequency of a particular word; that is called a word count, and how this is going to be used using map reduce program. Let us explain through this illustrative example.

(Refer Slide Time: 32:28)



So, let us say this file, contains these words see bob run, see spot throw, when it is given to the map function. So, map as in the function you see that after for each word, each word is parsed, and then it will do an emit with 1. So, c will be emitted with the value 1,

bob will be emitted with the value 1, run is emitted. So, all the words, they are called key, there will be emitted with the value 1. So, this is the key, and this is a value.

So, this particular output will be generated as the intermediate results. Now, these intermediate results will be sorted according to the keys. For example, see how many, 2 times see is there. So, it will go to the 1 worker, with the reduced function, this reduce function on it. Similarly bob will have another worker, and run will also have another worker, and spot is also another worker, and throw is also another worker. So, just see that, as far as see is concerned, see will output to why, because it will just add both of them. So, here you see that, it will sum all once in this list. So, in this list there are 2 times 1 is coming. So, see will be output as 2; whereas, bob is only one occurrence. So, bob will be output as bob 1, run also will do the same thing, spot also and throw also. So, just see that the output also is in the form of a key value pair.

So, input was the document, and the output is the key value pair. So, you just see that the word count has happened. So, for every word, how many the frequency of that particular word in the document; that is being counted using map reduce program.

(Refer Slide Time: 34:52)

Example 2: Counting words of different lengths

- The map function takes a value and outputs key:value pairs.
- For instance, if we define a map function that takes a string and outputs the length of the word as the key and the word itself as the value then
- map(steve) would return 5:steve and
- map(savannah) would return 8:savannah.

This allows us to run the map function against values in parallel and provides a huge advantage.

Handwritten notes:

- map(s, document)
- for w in document
- emit((length, w))
- key: value
- 5:steve
- 8:savannah
- Map
- word → 3
- or → 2

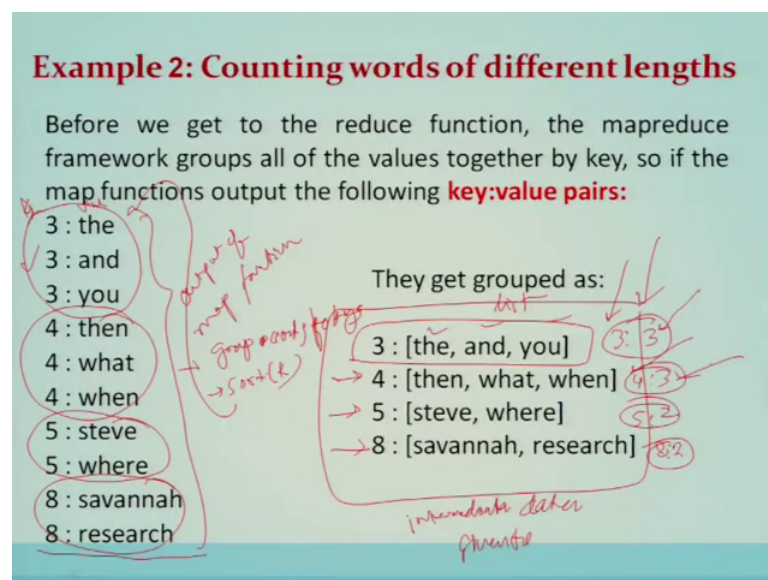
Now, another program we are going to see; that is for counting words of different length. For example, and or. So, and is of length 3, R is of length 2 and so on. So, we have to count how many words of length 3 are appearing in the document, how many words of length 2 are appearing in the document. So, for that we are going to write down a map

reduce function. So, the map function takes the value and output the key value, and outputs the key colon value pair.

So, first key will be output and its corresponding value will be given. So, for instance if you define the map function that takes the string and outputs the length of the word as the key, and the word itself as the value. So, if we give map to a Steve, this particular map will return. The length of this particular word as a key, and the value will be the word itself. So, this particular map function will emit, for a word, in the document it will emit the length of the word colon, the word itself, and this is basically the output.

So, this will allow us to run the map function against the values in parallel, and it provides a huge advantage.

(Refer Slide Time: 36:57)



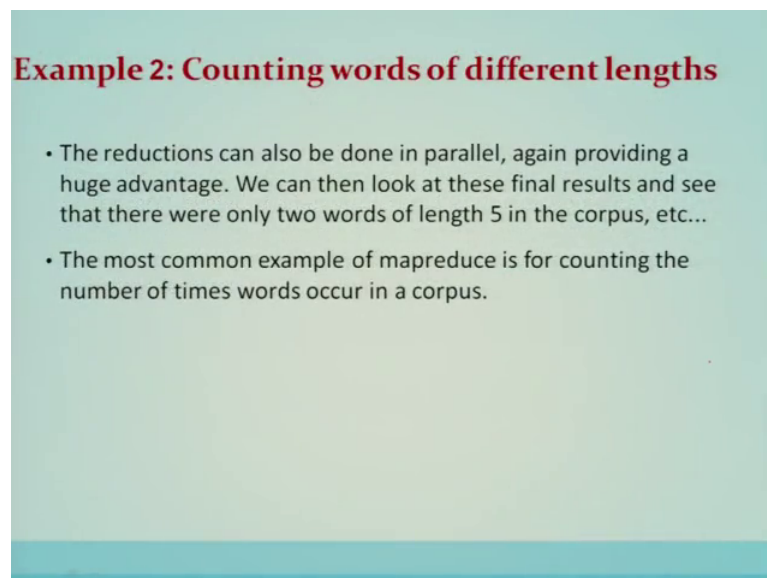
Let us see what happens. So, if a file is given with these words. So, the map function will output in this form. This is basically the length of the key; this is the key value pair. This is the output of the map function. Now once this is given as an output of the map function, then they are grouped according to the key values. So, you see that this is in the sorted order for to group, according to the keys values. We have to do the sorting, according to the key. So, this particular list is sorted according to the key.

Now, then we are going to group the same keys, and the list of words we are going to generate like this. So, for the value, for the key 3, here this is the list of words. For the

key 4, this is the list of words. For key 5, these two are the list of words, and for key 8, these are the two lists of words. These lists of words, this is an intermediate data, which is basically given to the reducer for each key there will be a reduce function, and this will count that is all, how many elements are there in that particular list. So, for 3 it will say 3, for 4 it will say 3, for 5 it will say 2, for 8 it will output 2; that means, the words of length 3, there are 3 frequencies 3; that means, 3 times it is appearing the length of word. 4, how many times they are appearing is 3 times and so on.

So, this was the example of a map reduce programs. So, that is what we have explained that using map and reduce function.

(Refer Slide Time: 39:23)



Example 2: Counting words of different lengths

- The reductions can also be done in parallel, again providing a huge advantage. We can then look at these final results and see that there were only two words of length 5 in the corpus, etc...
- The most common example of mapreduce is for counting the number of times words occur in a corpus.

You can solve a bigger problems also. Let us take another problem as an example.

(Refer Slide Time: 39:29)

Example 3: Finding Friends

using MapReduce

- Facebook has a list of friends (note that friends are a bi-directional thing on Facebook. If I'm your friend, you're mine).
- They also have lots of disk space and they serve hundreds of millions of requests everyday. They've decided to pre-compute calculations when they can to reduce the processing time of requests. One common processing request is the "You and Joe have 230 friends in common" feature.
- When you visit someone's profile, you see a list of friends that you have in common. This list doesn't change frequently so it'd be wasteful to recalculate it every time you visited the profile (sure you could use a decent caching strategy, but then we wouldn't be able to continue writing about mapreduce for this problem).
- We're going to use mapreduce so that we can calculate everyone's common friends once a day and store those results. Later on it's just a quick lookup. We've got lots of disk, it's cheap.

Which will find out the friends, using map reduce programming paradigm. So, finding friends face book has a list of friends. Note that the friends are the bi directional thing on the Facebook. So, if I am your friend you are mine. So, they also have a lot of disk a space, and they serve hundreds of millions of requests every day, they have decided to pre compute the calculation, when they can to reduce the processing time of the request. So, one common processing request is, the you and Joe have 230 friends in the common feature.

So, when you visit someone's profile you see the list of friends that you have in common. This list does not change frequently, so it would be wasteful to recalculate it every time you visit the profile. So, we are going to use this map reduce function, so that we can calculate everyone's common friend once in a day, and store those results.

(Refer Slide Time: 40:45)

Example 3: Finding Friends *using MapReduce*

- Assume the friends are stored as Person->[List of Friends], our friends list is then:

Person *List of friends*

- A -> B C D
- B -> A C D E
- C -> A B D E
- D -> A B C E
- E -> B C D

input

So, later on it just a quick lookup we can use it, and this will save lot of disk space, and it is cheap also. So, let us see how map reduce, can be used to find out the common friends. So, let us assume the friends are stored in this particular form. The person and having a list of friends, this is given as an input to the program. So, this is the person and then followed by the list of friends.

(Refer Slide Time: 41:37)

Example 3: Finding Friends

For map(A -> B C D) :

Person: A
map for each w in friend list
emit (A w) (1st)

- (A B) -> B C D ✓
- (A C) -> B C D ✓
- (A D) -> B C D ✓

For map(B -> A C D E) : (Note that A comes before B in the key)

- (A B) -> A C D E ✓
- (B C) -> A C D E ✓
- (B D) -> A C D E ✓
- (B E) -> A C D E ✓

Now, let us see what next, how the map and reduce function will do. So, map function will take this particular person and its friend list and will emit that particular person with

a pair one of these friends and so on. So, that way there are three friends, then the map function will emit for each word in friend list. What it will do? It will emit, let us say person is A and W. So, this way, and followed by that list to emit. So, it will emit A B B C D, then it will emit A C then B C D, then it will emit A D B C D. Similarly for person B with the list A C D E, it will emit A B. So, order is to be changed here. Note that A comes before B in the key. So, A B then same list, then BC, then same list of persons B D and so on.

(Refer Slide Time: 43:16)

Example 3: Finding Friends

For map(C -> A B D E) :

(A C) -> A B D E	
(B C) -> A B D E	
(C D) -> A B D E	
(C E) -> A B D E	

And finally for map(E -> B C D):

	(B E) -> B C D
	(C E) -> B C D
	(D E) -> B C D

For map(D -> A B C E) :

(A D) -> A B C E
(B D) -> A B C E
(C D) -> A B C E
(D E) -> A B C E

So, for all persons, this particular map will emit this key value pair. Now let us see what happens next. Now before we send these key value pair to the reducer, we group them by their key and get these values.

(Refer Slide Time: 43:24)

Example 3: Finding Friends

- Before we send these key-value pairs to the reducers, we group them by their keys and get:

(A B) -> (A C D E) (B C D)

(A C) -> (A B D E) (B C D)

(A D) -> (A B C E) (B C D)

(B C) -> (A B D E) (A C D E)

(B D) -> (A B C E) (A C D E)

(B E) -> (A C D E) (B C D)

(C D) -> (A B C E) (A B D E)

(C E) -> (A B D E) (B C D)

(D E) -> (A B C E) (B C D)

Handwritten notes: intersection, AB -> (C D), AC -> (B D)

So, here you see that this is one group, this is the another group. So, all pairs will be in a separate group. So, the reducer what it will do? It will take the intersection of these two lists. So, it will generate A B followed by the intersection.

So, the intersection is C D. Similarly here in AC, it will generate if you take the intersection B and D will come and so on.

(Refer Slide Time: 44:24)

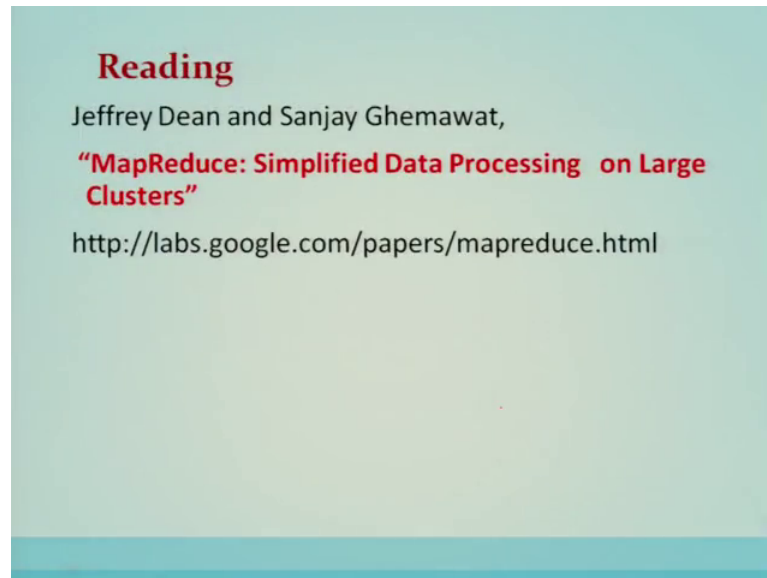
Example 3: Finding Friends

- Each line will be passed as an argument to a reducer.
- The **reduce function will simply intersect the lists of values** and output the same key with the result of the intersection.
- For example, **reduce((A B) -> (A C D E) (B C D))** will output **(A B) : (C D)**
- and means that friends A and B have C and D as common friends.

So, each line will be passed as an argument to the reducer, and the reduce function will simply do the intersection of the list of these values, and hence they will output the

common friends between two people. So, when A visits another person's side; that is B side, then he can find out that C and D is a common friends between A and B this way.

(Refer Slide Time: 45:05)



So, with these three different examples, we are sure that we can solve the data processing, large data set computing using map and reduce functions, and Google is also running several such programs daily, every day

Now, further reading on this particular topic, you can refer this particular reference, which is the map reduce simplified data processing on a large clusters, and that is available on the Google site.

(Refer Slide Time: 45:39)

Conclusion

- The MapReduce programming model has been successfully used at Google for many different purposes.
- The model is easy to use, even for programmers without experience with parallel and distributed systems, since it hides the details of parallelization, fault-tolerance, locality optimization, and load balancing.
- A large variety of problems are easily expressible as MapReduce computations.
- For example, MapReduce is used for the generation of data for Google's production web search service, for sorting, for data mining, for machine learning, and many other systems.

Conclusion. The map reduce programming model has been successfully used at the Google for many different purposes. The model is easy to use, even for the programmers without experience with parallel and distributed systems; since it hides the details of parallelization or tolerance, locality, optimization and load balancing. A variety of problems are easily expressible as map reduce configuration. For example, map reduce is used for the generation of data for the Google's production, web search service for sorting, for data mining, for machine learning and many other systems.

Thank you.