Cloud Computing

Prof. Soumya Kanti Ghosh
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 14
Introduction to Map Reduce

(Refer Slide Time: 00:40)

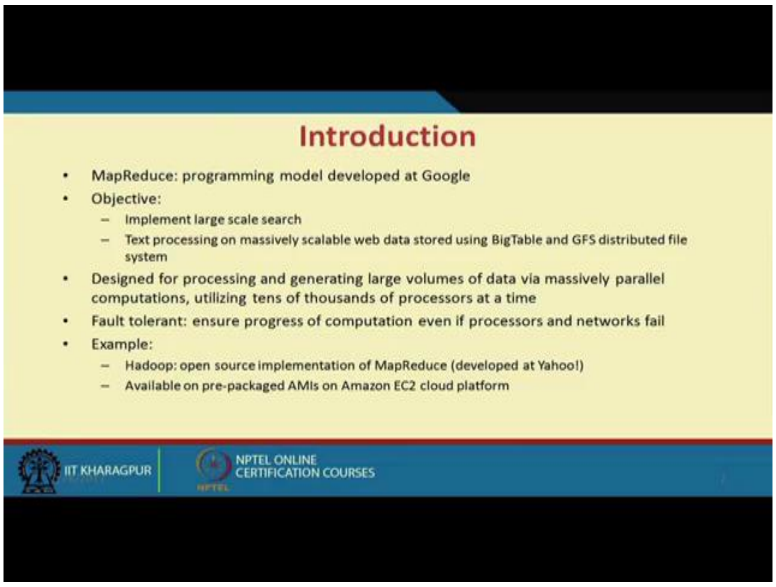
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES NPTEL

Cloud Computing :
Introduction to MapReduce

Prof. Soumya K Ghosh
Department of Computer Science and Engineering
IIT KHARAGPUR

Hello, so we will continue our discussion on cloud computing. As in our previous lecture we discussed about data store or data how to manage data in cloud having an overview of the things. Now, we like to see that another programming paradigm which is call MapReduce right a very popular programming paradigm which is primarily level out by Google, but now being used for different scientific purposes. So, Google primarily developed it for their large scale searches search engines primarily to search on huge amount of volumes of documents which their Google search engines chants, but it becomes a important paradigm programming paradigm for this scientific world to work on to exploit this philosophy to efficiently execute for different type of scientific problems.

(Refer Slide Time: 01:30)



Introduction

- MapReduce: programming model developed at Google
- Objective:
  - Implement large scale search
  - Text processing on massively scalable web data stored using BigTable and GFS distributed file system
- Designed for processing and generating large volumes of data via massively parallel computations, utilizing tens of thousands of processors at a time
- Fault tolerant: ensure progress of computation even if processors and networks fail
- Example:
  - Hadoop: open source implementation of MapReduce (developed at Yahoo!)
  - Available on pre-packaged AMIs on Amazon EC2 cloud platform

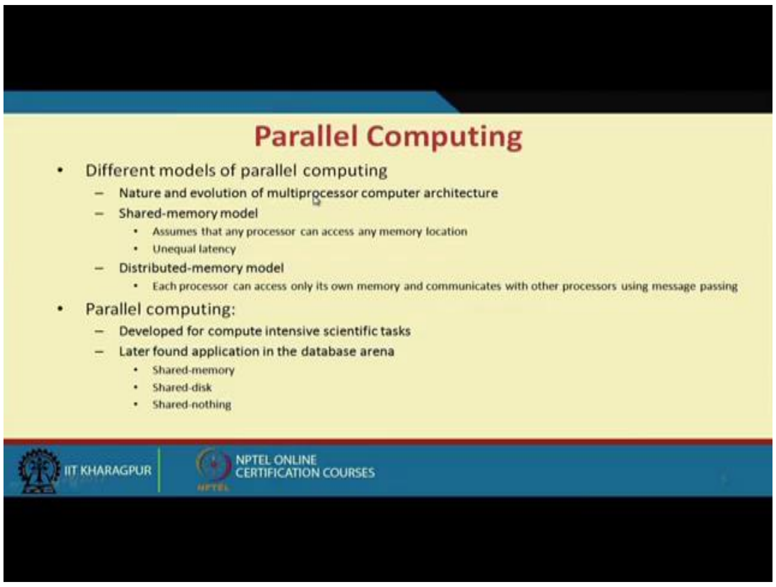
IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

So, map reduce is a programming model developed at Google, primarily objective was to implement large scale search, text processing on massively scalable web data stored in using big table or and GFS distributed file system. So, as we obtained that big data and GFS distributed file systems the data are stored. So, how to process this massively scalable web data that means, the huge volume of data are coming into play. Design for processing and generating large volumes of data via massively parallel computation utilizing tens of thousands of processor at a time.

So, I have a large pool of processors a huge pool of data and I want to do some analysis out of it. So, how can I do it? So, one very popular problem what we see is that if I have a huge volume of data and number of processors then how do say want to do some sort of word counting or counting the frequency of some of the words in that huge volume of data like I want to find out that how many times IIT Kharagpur appears in this a huge chunk of data, which are primarily stored in this HDFS or GFS or big table type of architecture.

So, and it should be fault tolerant, ensure progress of the computation even if processor fails and network fails right. So, because as there are huge volume huge number of processors and say underlining networks, so I do ensure fault tolerant. So, one of the example is Hadoop open source implementation of MapReduce developed at time volume had over initially developed at Yahoo and then became a open source. Available in a pre packaged AMIs on Amazon EC 2 platform, right. So, we are what we are looking at is trying to give a programming provide a programming platform or programming paradigm which can interact with data basis which are stored in this sort of cloud data stores right, it can be HDFS, GFS type or managed by big table and so on so forth.

(Refer Slide Time: 04:00)



Parallel Computing

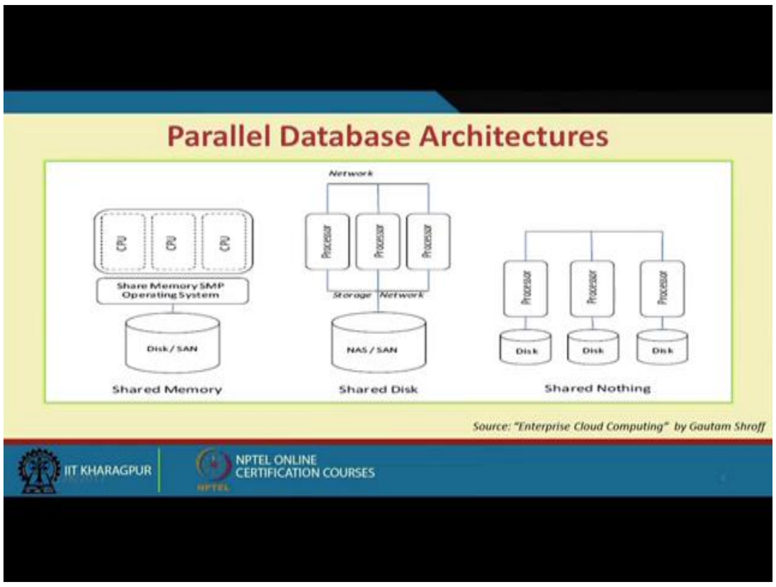
- Different models of parallel computing
  - Nature and evolution of multiprocessor computer architecture
  - Shared-memory model
    - Assumes that any processor can access any memory location
    - Unequal latency
  - Distributed-memory model
    - Each processor can access only its own memory and communicates with other processors using message passing
- Parallel computing:
  - Developed for compute intensive scientific tasks
  - Later found application in the database arena
    - Shared-memory
    - Shared-disk
    - Shared-nothing

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, if we look at again parallel computing as we have seen in our previous lectures, so different models of parallel computing it depends on the nature and evolution of the processor, multiprocessor computer architecture. So, it is shared memory model, distributed memory model, so these are the 2 popular thing. So, parallel computing developed for computing, intensive scientific tasks as we all know; later found application in data base arena or data base paradigm also, right.

So, it was initially it is more of a doing a huge scientific task and later we have seen that it has a lot of application in the database domain too. And we have seen in our earlier lecture that we have three type of scenario one is shared memory, shared disk and shared nothing, right. So, whenever we want to do a programming paradigm or work on something which can work on this sort of parallel programming paradigm where the data stored in the different this sort of clouds storages, so we need to take care of that what sort of mechanism is there. Like, whether it is shared memory, shared disk or shared nothing type of configuration.

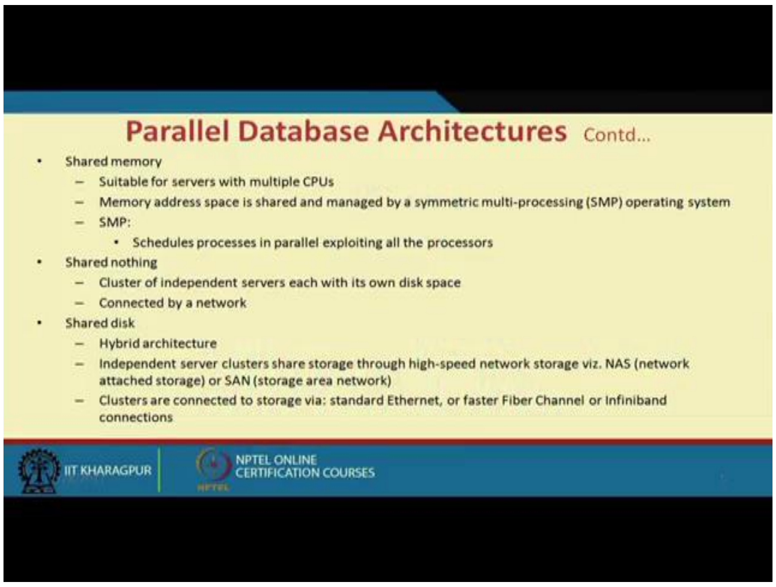
(Refer Slide Time: 05:28)


Parallel Database Architectures

Source: "Enterprise Cloud Computing" by Gautam Shroff

This is the picture already we have seen in our earlier lectures, so we do not want to repeat. So, it is a shared memory structure, shared disk and shared nothing, but the perspective we are looking at now is little different. There it is more of the storage where we are looking at. Now, we are trying to look at that how the programming can exploit this type of structure.

(Refer Slide Time: 05:52)


Parallel Database Architectures Contd...

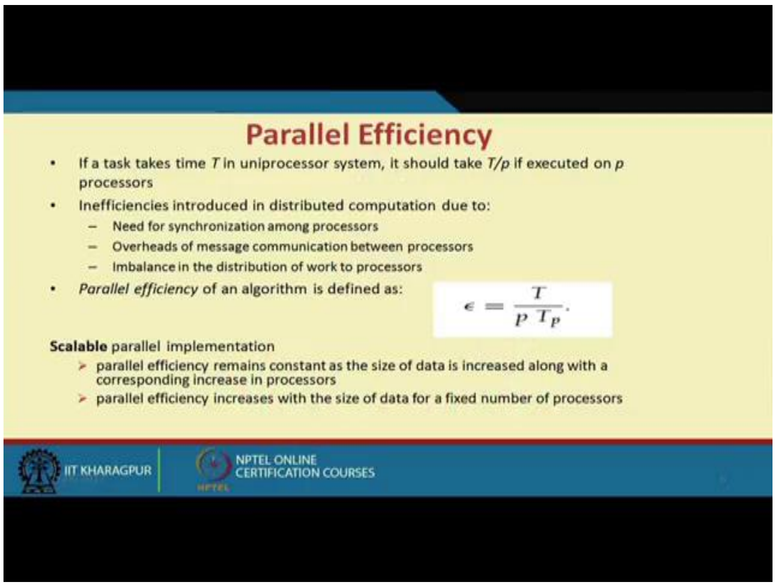
- Shared memory
  - Suitable for servers with multiple CPUs
  - Memory address space is shared and managed by a symmetric multi-processing (SMP) operating system
  - SMP:
    - Schedules processes in parallel exploiting all the processors
- Shared nothing
  - Cluster of independent servers each with its own disk space
  - Connected by a network
- Shared disk
  - Hybrid architecture
  - Independent server clusters share storage through high-speed network storage viz. NAS (network attached storage) or SAN (storage area network)
  - Clusters are connected to storage via: standard Ethernet, or faster Fiber Channel or Infiniband connections

So, this is already we have seen; so, shared memory suitable for servers with multiple CPUs. Shared nothing cluster of independent server each with its own hard disk, so connected by a high-speed network. And shared disk, so it is a hybrid architecture independent server cluster shares storage through a high speed network storage like NAS or SAN. Clusters are connected via to storage via standard Ethernet, fast fiber channel infini-band and so on and so forth.

So, whenever we do anything parallel or anything parallel computing or parallel storing and type of things, what is our back of the mind is to have efficiency right. So, you want to do parallism to one of the major aspect is to have a efficiency. There may be other aspects of fault tolerance and full proof and failure register and type of thing, but primarily it should be efficient. Now, first of all the type of work we are doing there should be inherent parallism into it. If there is no inherent parallism, then it may not be fruitful to do using always a parallel architecture.

So, first of all they should be inherent parallel it should not be a sequence of operations and then you try to do a parallel. So, it is job 1, job 2, job 3, job 4 a sequence is there or in between some parallism is there, but if you want to make a parallel operation, there may not be.

(Refer Slide Time: 07:22)



Parallel Efficiency

- If a task takes time *T* in uniprocessor system, it should take *T/p* if executed on *p* processors
- Inefficiencies introduced in distributed computation due to:
  - Need for synchronization among processors
  - Overheads of message communication between processors
  - Imbalance in the distribution of work to processors
- *Parallel efficiency* of an algorithm is defined as:

$$\epsilon = \frac{T}{p\,T_p}.$$

Scalable parallel implementation

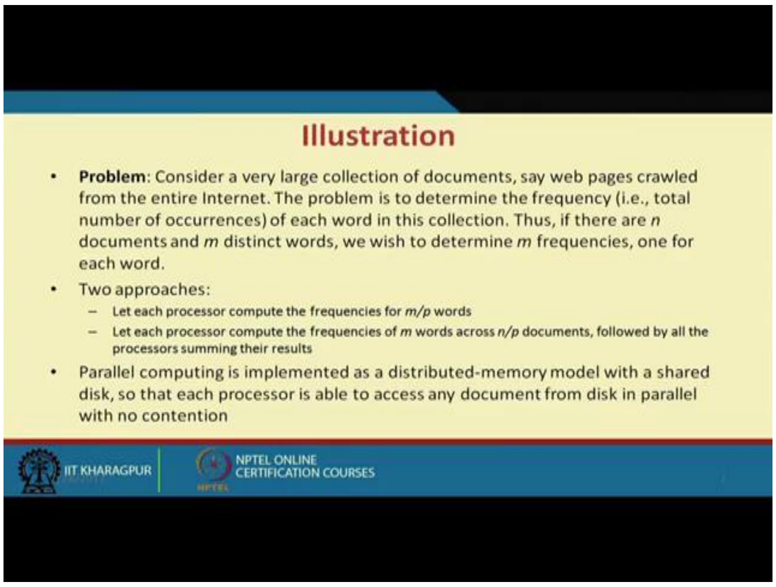
- parallel efficiency remains constant as the size of data is increased along with a corresponding increase in processors
- parallel efficiency increases with the size of data for a fixed number of processors

So, if a task takes time T in uni-processor, it should take T/p if executed in P processor ideally if the parallelism is there, and we are thinking that there is no cost in dividing distributing and type of things. So, it ideally T/p is a something ideal condition we can have. So, inefficiencies introduced in distributed computation due to need of synchronization among the processors. So, I need to synchronize among the processor, it is not like that all processor has you may have the individual clocks and you need to synchronize that where things will be there. Otherwise if you if you divide the job into 2 where one executed now and one executed after couple of hours then it is it could have been better that is execute to in a one system. So, synchronization in between the processor is one of the important aspects. So, need to synchronize.

Overheads of message communication between the processors another aspect; imbalance in the distribution of work to the processors another, so it may not be equally divided and type of things. So, these are the different aspects which indirectly affect this efficiency or bring about inefficiency into this parallel implementation. So, parallel efficiency of an algorithm can be defined as $\frac{T}{p \cdot T/p}$. So, if it is scalable we say this is scalable or scalable parallel efficiency remains constant as the size of the data increased along with a corresponding increase of the processor.

So, what is happening when more data is coming, so you go on deploying more processor or you go on requesting from the cloud more processor and then your efficiency remains constant, the efficiency values does not change? So, then what we say that it is scalable. So, that if I increase both say for example, for linearly then it goes on in the constant thing. Parallel efficiency increases with the size of the data for a fixed number of processor, it increases with the size of the data; and if it is a fixed number of processor then we can have effectively more efficiency.

(Refer Slide Time: 09:46)



Illustration

- **Problem**: Consider a very large collection of documents, say web pages crawled from the entire Internet. The problem is to determine the frequency (i.e., total number of occurrences) of each word in this collection. Thus, if there are *n* documents and *m* distinct words, we wish to determine *m* frequencies, one for each word.
- Two approaches:
  - Let each processor compute the frequencies for *m/p* words
  - Let each processor compute the frequencies of *m* words across *n/p* documents, followed by all the processors summing their results
- Parallel computing is implemented as a distributed-memory model with a shared disk, so that each processor is able to access any document from disk in parallel with no contention

Now, the example, which is there in that book you are referring also you will find the example in different literature, this sort of example not the same. Consider a very large collection of documents say the web document crawled by the entire internet. So, it is a pretty large it is large every day it is growing. The problem is to determine the frequency that is total number of occurrences of each word in this collection right. So, I want to determine the total number of what is the frequency of occurrences of each word in this document d. So, thus if there are n documents and m distinct words, we use to determine m frequency one for each word right. So, this is a simple problem may be true or may be more relevant for search engines and type of things.

So, we have two approaches let each processor compute the frequency for m/p words. So, each processors if there are p processors, if the m frequencies I need to calculate, I divide m by p, so many, so for example, I want to look for I have ten processors and I have no I am look for some 90 words, m equal to ninety. So, every processor does it chunk of ten right roughly if it is not divisible then you have to make some asymmetric division. So, it makes things and that at the end of things they again report the things together or in some through some system. So, other way let each processor compute the frequency of m words across n by p documents.

So, total number of documents say 10,000. So, 10,000 number of words I am looking for 90 number of processor I am having 10. So, one is 90 by 10 is the 9 words on an average given to the every processor and they count on the things.

Other thing what we are telling that each processor compute the frequency for all the 90 words, but on n by p document if that 10,000 words and then p 10 processors; so, some thousand document so each take 100 documents and do the processing and once that frequency of this m words by individual professors processors come out then I sum up this thing and aggregate and show the result that this is the thing right, by followed by all processors summing their results right. Parallel computing, now which one will be efficient based on this parallel computing paradigm, we need to look at right. So, parallel computing is implemented as a distributed memory model with a shared disk, so that each processor is able to access any document from the disk in parallel with no contention. So, this can be one of the implementation mechanisms.

(Refer Slide Time: 13:01)

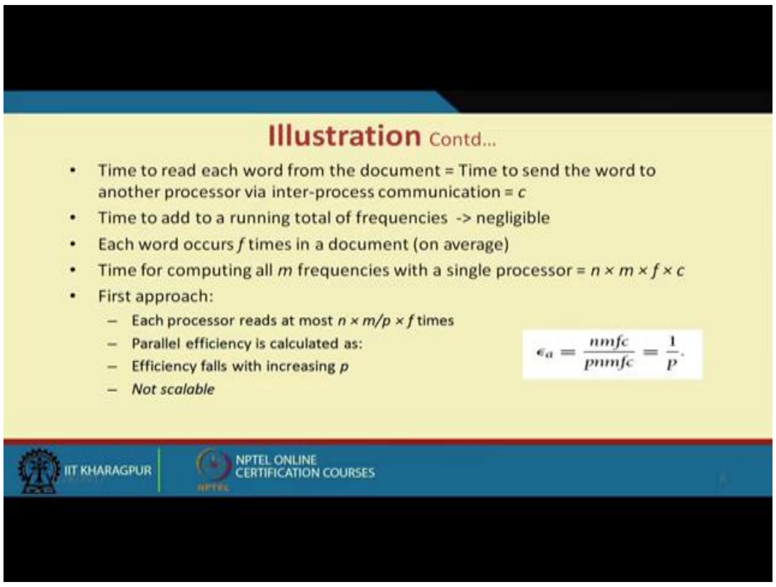


Illustration Contd...

- Time to read each word from the document = Time to send the word to another processor via inter-process communication = *c*
- Time to add to a running total of frequencies -> negligible
- Each word occurs *f* times in a document (on average)
- Time for computing all *m* frequencies with a single processor = $n \times m \times f \times c$
- First approach:
  - Each processor reads at most $n \times m/p \times f$ times
  - Parallel efficiency is calculated as:
  - Efficiency falls with increasing *p*
  - *Not scalable*

$$\epsilon_a = \frac{nmfc}{pnmfc} = \frac{1}{p}.$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, time to read each word from the document say if let us assume that time to read each word from the document equal to time to send the word to another processor via inter processor communication and equals to c. So, making thing simple so it may be it should be means ideally in ideal case or in a real life case it will be different, but we make these scenarios. So, first approach so time for so time to add to a running total of the frequencies negligible, so summing up is negligible. Once I find the frequencies of this m word then summing up is negligible.

Each what word occurs f times on the document on an average. So, if I for our calculation sake that each word that on an average, workers some f time. Time for compute all m frequencies with the single processor equal to then I have $n \times m \times f \times c$. So, this is the time to compute m frequencies with a single processor, if I have a single process this could have been the thing. So, if we do the first approach, first approach was this one, let each processor compute the frequency of m/p words, so that is a first approach.

So, each processor reads at most $\frac{n \times m}{p \times f}$. So, parallel efficiency is calculated as $\frac{n \times m \times f \times c}{p \times n \times m \times f \times c}$, so 1 by p very vanilla type consideration. So, we take that all are doing all are morally same frequencies, all are negligible time for the any aggregation then the all time for the means read and write another operations we have to consider c, considering this we are getting $\frac{1}{p}$. So, efficiency falls with increasing p. So, if we increase the p, then the efficiency falls. So, it is not constant. So, it is not scalable, it is one of the major problem is that though it is what we say easy to conceptualize etcetera, but there is a problem in the scalability so of the things. This one that let each processor compute frequencies per n by m words n by m words is not scalable.

(Refer Slide Time: 15:36)

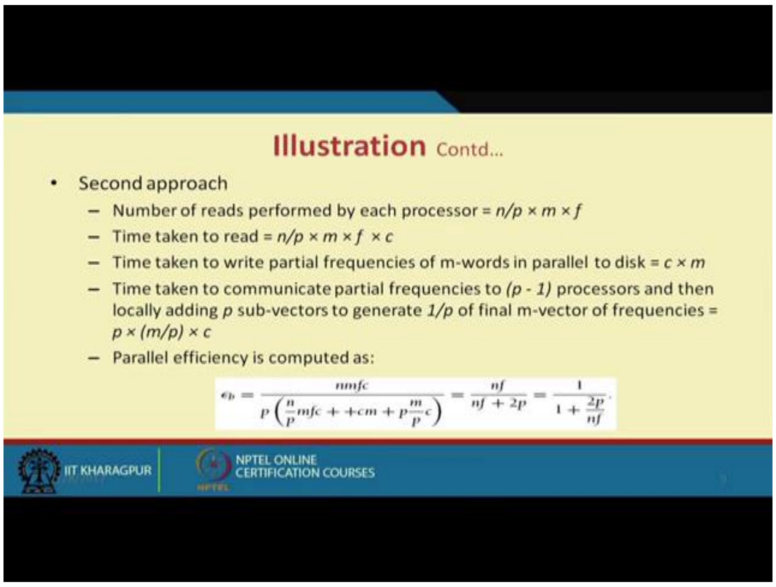


Illustration Contd...

- Second approach
  - Number of reads performed by each processor = $n/p \times m \times f$
  - Time taken to read = $n/p \times m \times f \times c$
  - Time taken to write partial frequencies of m-words in parallel to disk = $c \times m$
  - Time taken to communicate partial frequencies to $(p - 1)$ processors and then locally adding p sub-vectors to generate $1/p$ of final m-vector of frequencies = $p \times (m/p) \times c$
  - Parallel efficiency is computed as:

$$\epsilon_p = \frac{nmfc}{p\left(\frac{n}{p}mfc + +cm + p\frac{m}{p}c\right)} = \frac{nf}{nf + 2p} = \frac{1}{1 + \frac{2p}{nf}}.$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Whereas, in the second approach, where that m words we divide into the different processes oh sorry we divide that document d, whereas every processor compute this for all the m words and then aggregate. So, apparently what it looks that this could me more costly. So, it is there is a aggregation thing then you are doing clubbing those processor, club means dividing the m set into different this whole documents set into different partitions and doing that, this could be in efficient than the first one. But let us see what is there. So, the number of read performs for each processor is $n/p \times m \times f$ right the time taken to read is $n/p \times m \times f$. It is because you are having n/p amount of volume of the data and then want to calculate for $m \times f \times c$, so that number of time taken to calculate this read. Time taken to write partial frequency on of m words in parallel to disk is $m.(c \times m)$.

So, once you are done you need to write on the parallel to the disk and that is that comes to be $(c \times m)$ time taken to communicate partial frequency right to $p-1$ processors. And then locally adding sub p sub vectors to generate $1/p$ of the final m vector of frequencies then what we have $p\frac{m}{p}c$. So, what you need to do we are time taken to communicate partial frequencies right because you do not have the whole frequencies. So, partial frequency by different processor and $p-1$ processor and then locally adding p sub vectors to generate 1 by p here of the final m vector frequencies is this one. So, individually need to do.

So, if we adopt all those things in case of this second approach what we have this parallel frequency as this structure, so $\frac{1}{1+2p/nf}$, so that is if you if you look at it little minutely if you consult the book, it is not a very difficult problem difficult to deduce. It is pretty easy just have to go by step by step. Now, this is an interesting phenomena. So, the term we are having here is $1+2p/nf$. So, in this case a p is many, many times less than nf. Efficiency of the second approach is higher than that of the first right here if it is p is many, many times less than nf, then this term that this will be tending towards one. And it can be seemed that there is much efficiency is much higher.

(Refer Slide Time: 18:57)

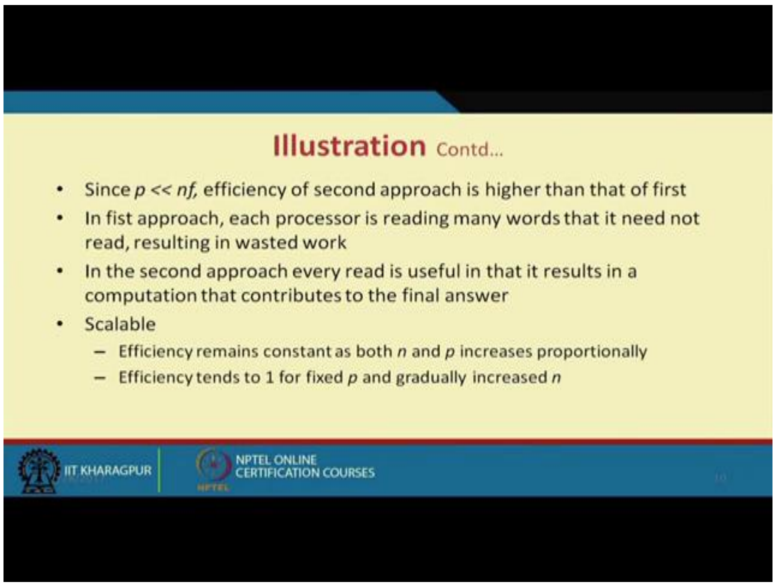


Illustration Contd...

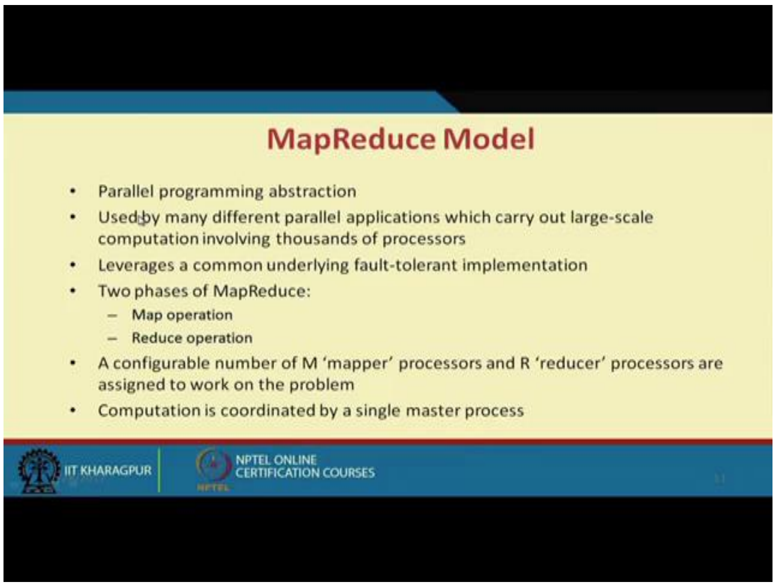
- Since $p << nf$, efficiency of second approach is higher than that of first
- In fist approach, each processor is reading many words that it need not read, resulting in wasted work
- In the second approach every read is useful in that it results in a computation that contributes to the final answer
- Scalable
  - Efficiency remains constant as both n and p increases proportionally
  - Efficiency tends to 1 for fixed p and gradually increased n

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

In the first approach, so there is a type it should be, let us in the in first approach each processor is reading many words than it needs to read resulting in wastage of time. What we have done in the first approach this many processor we have divided this m into different chunk. So, the processor say as we as we have taken the example that if I am having m as ninety and number of processor is p, so 90 by p is 10. So, everybody is getting 10, but when it is searching the whole document, so number of documents is reading where there is no hit, it is no success.

So, efficiency, so in the second approach every read is useful right. As it results in a computation and distributes to the final results. So, for in the second approach, every read is likely to be useful where it contribute to this result. So, it is scalable also. The efficiency remains constant at both n and p increases potentially, they proportionally. So, what we see what we have done there that if my data load increases I will increase the processor. So, if I proportionally increase the data processor then my efficiency remains constant in this case in the second case. Efficiency tends to one for fixed p and gradually increasing n. So, efficiency tends to 1, if the number of processor is fixed and gradually increased we are increasing n that means we are increasing the data load, number of processor fixed and it will basically approaches one.

(Refer Slide Time: 20:57)



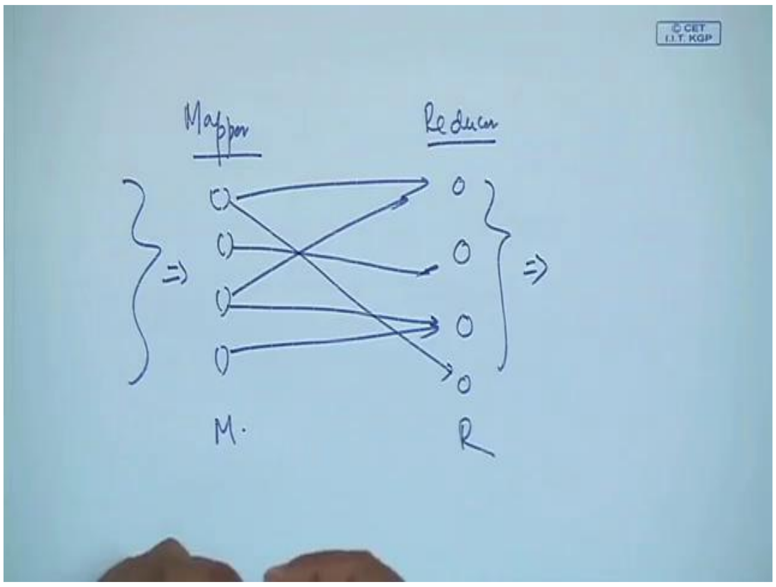
MapReduce Model

- Parallel programming abstraction
- Used by many different parallel applications which carry out large-scale computation involving thousands of processors
- Leverages a common underlying fault-tolerant implementation
- Two phases of MapReduce:
  - Map operation
  - Reduce operation
- A configurable number of M 'mapper' processors and R 'reducer' processors are assigned to work on the problem
- Computation is coordinated by a single master process

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

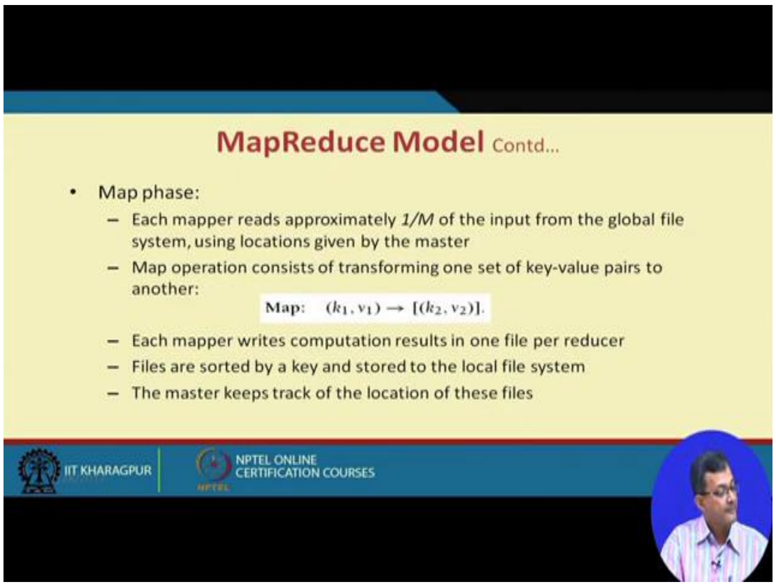
So, with these context or with these background of that which can be that this doing that individually then aggregating is becoming more efficient with this things, we look at that your map reduce model. So, it is a parallel programming abstraction used by many different parallel applications which carry out large scale computations involving thousands of processors; leverages a common underlining fault tolerant implementation. Two phases of map reduce map operation and reduce operation. A configurable number of M mapper - mapper processor and R reducer processors are assigned to work on the problem. Computation is coordinated by a single master process. So, what we are having now? There are different mapper processors like and there is a different reducer processor. So, whole process, I divide into two things.

(Refer Slide Time: 22:16)



Like I have a mapper, so different mapper processor, so there are M processor and there is reducer. So, there are different reducer processor. So, what we does it when the data come here it basically do some execution and then this reducer may be based on the type of problem it will go on different reduce things and do the execution. So, reducer will generate is more of aggregated results right. So, what it tries to do it is a parallel programming abstraction used by mineral parallel applications which carryout large scale computation involving thousands of processors. So, here the application come into play. So, it is a two phase process, one is a map operation, another is a reduce operation. So, that the configurable number of M mapper processor, R reducer processors, so it is configurable; that means, you can have more etcetera mapper and reducer.

(Refer Slide Time: 23:28)



MapReduce Model Contd...

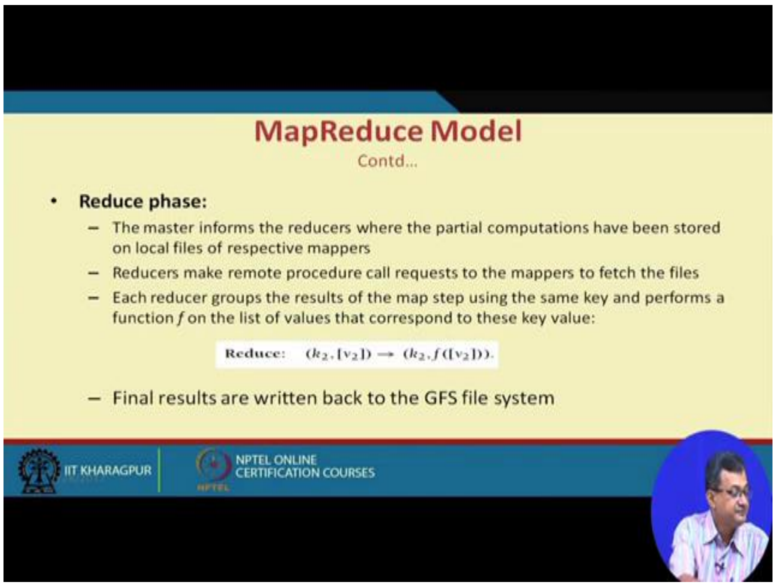
- Map phase:
  - Each mapper reads approximately *1/M* of the input from the global file system, using locations given by the master
  - Map operation consists of transforming one set of key-value pairs to another:

    $$\textbf{Map:} \quad (k_1, v_1) \rightarrow [(k_2, v_2)].$$

  - Each mapper writes computation results in one file per reducer
  - Files are sorted by a key and stored to the local file system
  - The master keeps track of the location of these files

So, map reduce phase. So, if we look at the map phase each mapper read approximately $1/M$ of the input from the global file. So, it is not the whole data d, but a chunk of the data read. Map operation consists of transforming one set of value key value pair to another set of key value pair. So, what map does, it is a one set of key value pair to another set of key value pair. So, map $(k_1.v_1) \rightarrow [(k_2.v_2)]$. So, each mapper writes computational results in one file per reducer. So, what it does, it basically for every reducer it produces a file. So, it says if there are reducers R 1, R 2, R 3 a mapper m, I create three files based on the corresponding the reducer. So, the files are sorted by a key and stored in a local file systems right. The master keeps tracks of the location of these files. So, there is a master map reduce master, so which takes care of this location of the file, each mapper produces a one file for every reducers and the master takes care where the files are stored in the local disk etcetera.

(Refer Slide Time: 24:55)



MapReduce Model

Contd...

- **Reduce phase:**
  - The master informs the reducers where the partial computations have been stored on local files of respective mappers
  - Reducers make remote procedure call requests to the mappers to fetch the files
  - Each reducer groups the results of the map step using the same key and performs a function *f* on the list of values that correspond to these key value:

    Reduce: $(k_2, [v_2]) \rightarrow (k_2, f([v_2]))$.

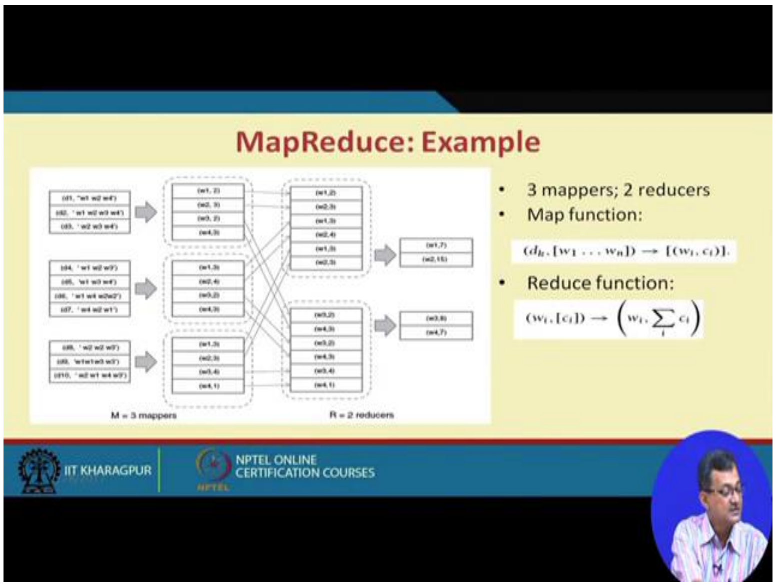
  - Final results are written back to the GFS file system

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

In the reduce phase, the master informs the reducers where the partial computation have been stored on local file systems of respective mappers; so that means, in the reducer phase the reducer consult this master which informs that where its related files are stored corresponding to the every mapper functions. Reducer makes remote procedure call to the mappers to fetch the files. So, reducer in turn make a remote procedure call for the mapper. So, mapper it is somewhere in the disk and the reducer there may be in different structure with different types of VMs etcetera running on the things ideally it is not far not geographically distributed then the things will not work. So, nevertheless it is working on that particular data which are produced by the mapper.

So, each reducer groups the results of the map step using the same key value key value function f etcetera, so $(k_2.[v_2]) \rightarrow (k_2.f([v_2]))$. So, here the aggregated functions in comes into play. In other sense, if we remember our problem. So, what we do that every doc, every key or every word we want to calculate the frequency, so the functional model is summing up the frequencies of the things, it can be different for different type of things. So, it does a k_2v etcetera. So, it goes for another key value up here. Final results are return back to the GFS file system Google file system.

(Refer Slide Time: 26:36)



MapReduce: Example

- 3 mappers; 2 reducers
- Map function:

$$(d_k, [w_1 \ldots w_n]) \rightarrow [(w_i, c_i)].$$

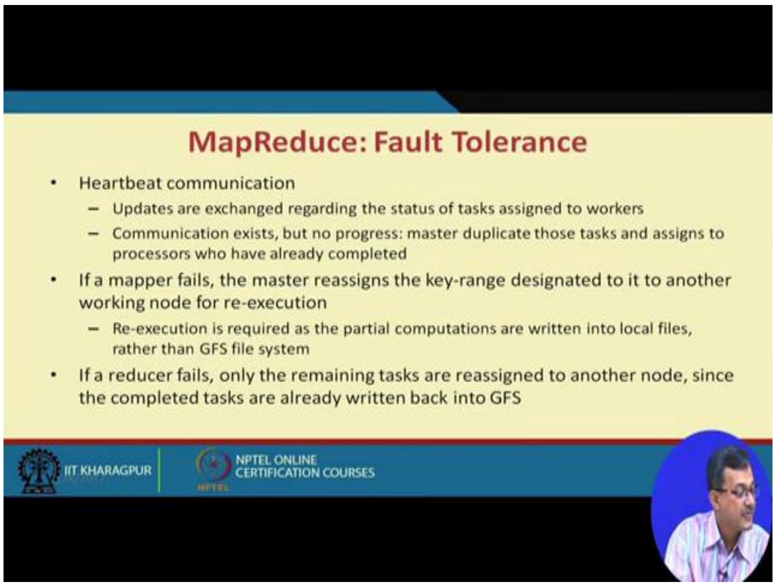
- Reduce function:

$$(w_i, [c_i]) \rightarrow \left(w_i, \sum_i c_i\right)$$

So, map reduce example. So, if we see there are 3 mapper, 2 reducer. So, map function in this in our case is that is the data d there are the set of word w_1, w_2, w_n and it produce for every w_i the count of the things, how much count the portion of the mapper it is having. So, every w_i, it counts the thing. So, if you see if d_1, it has w_1, w_2, w_4; d_2 these are the things and it counts this. So, every mapper does it, and then it basically stored in a intermediate space where the reducer reads. So, it generates every file for every reducer like this particular things is generate a particular file for a reducer. So, there are two reducer.

So, for two reducer every mapper generates the file. So, and the reducer in turns basically accumulate those. So, it says that w it has the thing w_1, w_2, so w_1 as 7, w_2 as something 15. In this case, w_3, w_4 are the other two. So, the reducers reduces the thing from the inputs of the or from the outputs of the mapper getting the input from the mappers output.

(Refer Slide Time: 28:08)



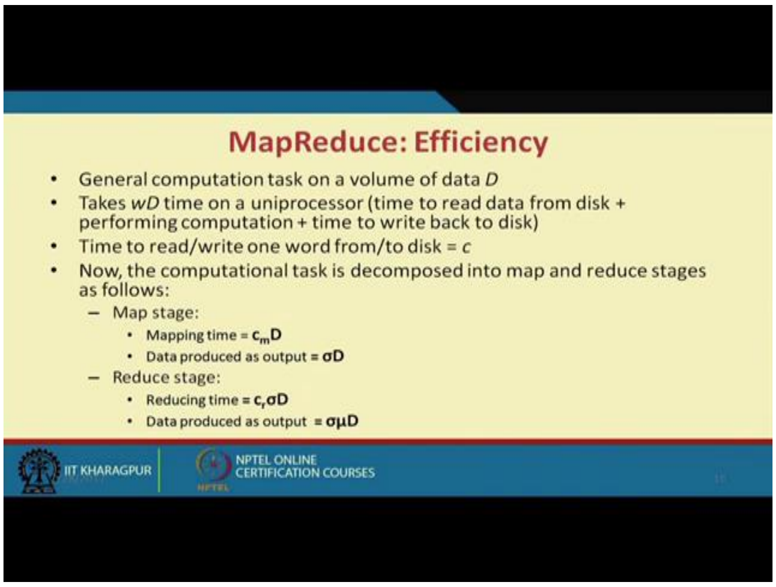
MapReduce: Fault Tolerance

- Heartbeat communication
  - Updates are exchanged regarding the status of tasks assigned to workers
  - Communication exists, but no progress: master duplicate those tasks and assigns to processors who have already completed
- If a mapper fails, the master reassigns the key-range designated to it to another working node for re-execution
  - Re-execution is required as the partial computations are written into local files, rather than GFS file system
- If a reducer fails, only the remaining tasks are reassigned to another node, since the completed tasks are already written back into GFS

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, map reduce model is fault tolerance; there are different way to look at it, one is heart beat message. So, every particular time period, it says that whether it is a live and type of things. Communication exists, but no progress master if there are communication exists, but no progress master duplicate those tasks and assign the processor who are already completed or some free processors. If the mapper fails, the mapper reassigns key value designated to it to another work node on the re-execution. So, if it is a failure then it re-execute the thing. If the reducer fails only the remaining task need to be reassigned to another node. Since the completed tasks are already written back to Google file system. So, if the completed tasks are there, they are already in Google file systems only the remaining tasks need to be reassigned.

(Refer Slide Time: 29:04)



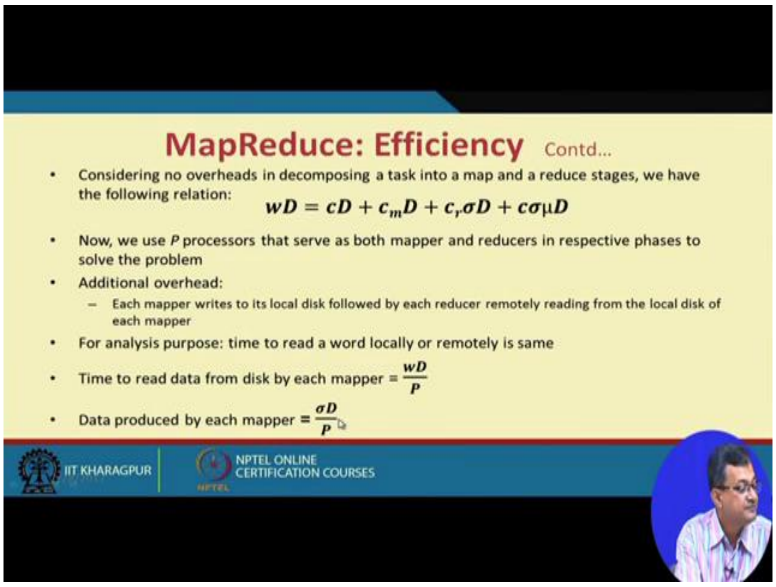
MapReduce: Efficiency

- General computation task on a volume of data *D*
- Takes *wD* time on a uniprocessor (time to read data from disk + performing computation + time to write back to disk)
- Time to read/write one word from/to disk = *c*
- Now, the computational task is decomposed into map and reduce stages as follows:
  - Map stage:
    - Mapping time = c_mD
    - Data produced as output = σD
  - Reduce stage:
    - Reducing time = $c_r\sigma D$
    - Data produced as output = $\sigma\mu D$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, if you want to calculate the efficiency of the MapReduce, so the general computation task on a volume of data D. So, takes w D time to uni-processor read time to read data from disk performing computation write back to the disk. Time to read write one word from to disk is c. Now, the computation task is decomposed into map reduce stages like map stage mapping time c_mD data producing and output σD, reduce stage reduce time $c_r\sigma D$ and data produced at the output is $\sigma\mu D$. So, this is not that difficult. So, mapping time how much that with D every mapper is doing data produced time is from the particular mapper which is how much time it is producing reduce reducers time in calculated with the every c_r and that finally, we have that reducer output.

(Refer Slide Time: 30:11)

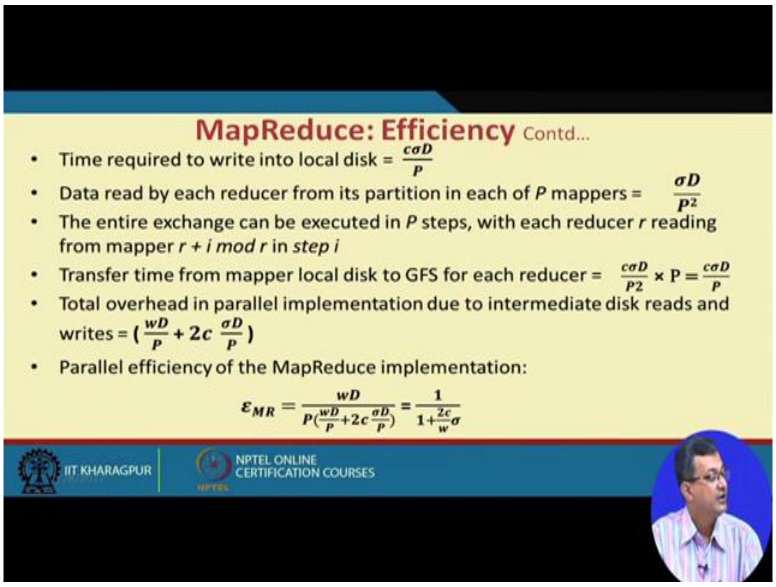


MapReduce: Efficiency Contd...

- Considering no overheads in decomposing a task into a map and a reduce stages, we have the following relation: $wD = cD + c_m D + c_r \sigma D + c\sigma\mu D$
- Now, we use *P* processors that serve as both mapper and reducers in respective phases to solve the problem
- Additional overhead:
  - Each mapper writes to its local disk followed by each reducer remotely reading from the local disk of each mapper
- For analysis purpose: time to read a word locally or remotely is same
- Time to read data from disk by each mapper = $\frac{wD}{P}$
- Data produced by each mapper = $\frac{\sigma D}{P}$

So, considering no overheads in decomposing the task into map and reduce stages, we can have the following relationship. So, if we forget the overhead in decomposing in mapping and reducing, so we can have this summation of the things. Now, if we had P processors that serve as both mapper and reducer right irrespective of the phases to solve problem. So, if we use P processor sometimes it acts as a mapper, sometimes act as a reducer. Then we have additional overhead each mapper writes to local we have some additional overheads writes to local disk followed by each reducer remotely reading to the disk. For analysis purpose time to read to a word locally or remotely, let us consider as same. Time to read a data from the disk is for each mapper is wD by number of with an if the number of processor is P wD/P data producer is mapper is $\sigma D/P$.

(Refer Slide Time: 31:12)



MapReduce: Efficiency Contd...

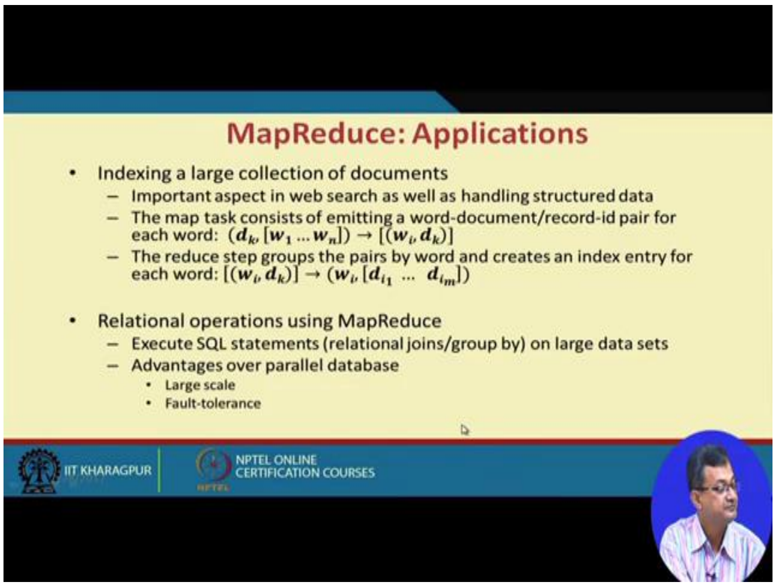
- Time required to write into local disk = $\frac{c\sigma D}{P}$
- Data read by each reducer from its partition in each of *P* mappers = $\frac{\sigma D}{P^2}$
- The entire exchange can be executed in *P* steps, with each reducer *r* reading from mapper *r + i mod r* in *step i*
- Transfer time from mapper local disk to GFS for each reducer = $\frac{c\sigma D}{P2} \times P = \frac{c\sigma D}{P}$
- Total overhead in parallel implementation due to intermediate disk reads and writes = $(\frac{wD}{P} + 2c\,\frac{\sigma D}{P})$
- Parallel efficiency of the MapReduce implementation:

$$\varepsilon_{MR} = \frac{wD}{P(\frac{wD}{P}+2c\frac{\sigma D}{P})} = \frac{1}{1+\frac{2c}{w}\sigma}$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, time required to write back to the disk because once you read then you have to after computation, you have to write back to that is that much. So, similarly data read by each reducer from its partition to each mappers P mappers are $\sigma D/P/P$. So, $\frac{\sigma D}{P^2}$. So, if we calculate like that we say that the parallel efficiency of the map reduce implementation comes as this one, $\frac{1}{1+\frac{2c}{w}\sigma}$.

(Refer Slide Time: 31:50)



MapReduce: Applications

- Indexing a large collection of documents
  - Important aspect in web search as well as handling structured data
  - The map task consists of emitting a word-document/record-id pair for each word: $(d_k, [w_1 \ldots w_n]) \rightarrow [(w_i, d_k)]$
  - The reduce step groups the pairs by word and creates an index entry for each word: $[(w_i, d_k)] \rightarrow (w_i, [d_{i_1} \ldots d_{i_m}])$
- Relational operations using MapReduce
  - Execute SQL statements (relational joins/group by) on large data sets
  - Advantages over parallel database
    - Large scale
    - Fault-tolerance

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, so this is what we get a parallel efficiency out here. Now, if the indexing map reduce there are several type of applications one is indexing a large collection of documents right, so that which is primarily one of the major motivation for Google. So, important aspect for web search as well as handling structured data. So, map task consists of emitting a word document, record id pair for the each word like as we have seen $wdk.w_1.n$ into map to one its map w_1 into every word dks. So, I can have some sort of indexing reduce step groups the pair of words and creates entry into the thing.

So, there are applications in relational operations using map reduce. Execute SQL statements relational join, group by on large set of data. Advantages of parallel data base large scale fault tolerance we want to exploit and I can have those type of function like as we have seen that it is a group by clause and type of etcetera we can do, so that some sort of relational operations we can execute.

So, with these we come to this end of today's talk. So, what we try to do here to give you a overview of a MapReduce paradigm that how a problem can be divided into a set of parallel executions, which is a mapper node which creates intermediate results. And there is a set of reducer nodes which takes this data and create the final results right. And what we can which there are some of the things which is interesting that the mapper creates data file for every reducer. So, it is the data is created per reducer. So, the reducer knows that where the data is there.

Over and above there is a master controller or the map reducer master things which come to which knows where there things where the data is stored by the mapper and how the reducer will read. Not only that if the mapper node fails how to reallocate the things; if the reducer node fails, how to reallocate because the things or the reallocate the not executed data not executed things not executed yet to be executed operations and so on and so forth. So, with this we will stop our lecture today.

Thank you.