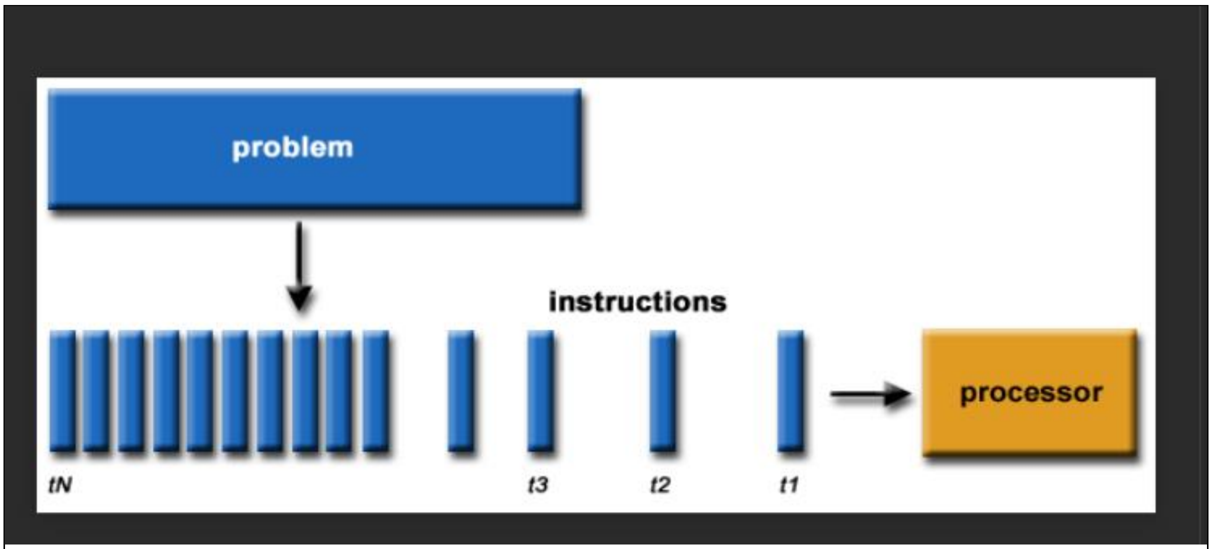
Introduction to parallel computing :



Parallel computing refers to the process of executing several processors an application or computation simultaneously. Generally, it is a kind of computing architecture where the large problems break into independent, smaller, usually similar parts that can be processed in one go. It is done by multiple CPUs communicating via shared memory, which combines results upon completion. It helps in performing large computations as it divides the large problem between more than one processor.

Parallel computing also helps in faster application processing and task resolution by increasing the available computation power of systems. The parallel computing principles are used by most supercomputers employ to operate. The operational scenarios that need massive processing power or computation, generally, parallel processing is commonly used there.

Typically, this infrastructure is housed where various processors are installed in a server rack; the application server distributes the computational requests into small chunks then the requests are processed simultaneously on each server. The earliest computer software is written for serial computation as they are able to execute a single instruction at one time, but parallel computing is different where it executes several processors an application or computation in one time.

There are many reasons to use parallel computing, such as save time and money, provide concurrency, solve larger problems, etc. Furthermore, parallel computing reduces complexity. In the real-life example of parallel computing, there are two queues to get a ticket of anything; if two cashiers are giving tickets to 2 persons simultaneously, it helps to save time as well as reduce complexity.

Types of parallel computing

From the open-source and proprietary parallel computing vendors, there are generally three types of parallel computing available, which are discussed below:

1. **Bit-level parallelism:** The form of parallel computing in which every task is dependent on processor word size. In terms of performing a task on large-sized data, it reduces the number of instructions the processor must execute. There is a need to split the operation into series of instructions. For example, there is an 8-bit processor, and you want to do an operation on 16-bit numbers. First, it must operate the 8 lower-order bits and then the 8 higher-order bits. Therefore, two instructions are needed to execute the operation. The operation can be performed with one instruction by a 16-bit processor.
2. **Instruction-level parallelism:** In a single CPU clock cycle, the processor decides in instruction-level parallelism how many instructions are implemented at the same time. For each clock cycle phase, a processor in instruction-level parallelism can have the ability to address that is less than one instruction. The software approach in instruction-level parallelism functions on static parallelism, where the computer decides which instructions to execute simultaneously.
3. **Task Parallelism:** Task parallelism is the form of parallelism in which the tasks are decomposed into subtasks. Then, each subtask is allocated for execution. And, the execution of subtasks is performed concurrently by processors.

Advantages of Parallel computing

Parallel computing advantages are discussed below:

- In parallel computing, more resources are used to complete the task that led to decrease the time and cut possible costs. Also, cheap components are used to construct parallel clusters.
- Comparing with Serial Computing, parallel computing can solve larger problems in a short time.
- For simulating, modelling, and understanding complex, real-world phenomena, parallel computing is much appropriate while comparing with serial computing.
- When the local resources are finite, it can offer benefit you over non-local resources.
- There are multiple problems that are very large and may impractical or impossible to solve them on a single computer; the concept of parallel computing helps to remove these kinds of issues.
- One of the best advantages of parallel computing is that it allows you to do several things in a time by using multiple computing resources.

- Furthermore, parallel computing is suited for hardware as serial computing wastes the potential computing power.

Disadvantages of Parallel Computing

There are many limitations of parallel computing, which are as follows:

- It addresses Parallel architecture that can be difficult to achieve.
- In the case of clusters, better cooling technologies are needed in parallel computing.
- It requires the managed algorithms, which could be handled in the parallel mechanism.
- The multi-core architectures consume high power consumption.
- The parallel computing system needs low coupling and high cohesion, which is difficult to create.
- The code for a parallelism-based program can be done by the most technically skilled and expert programmers.

Fundamentals of Parallel Computer Architecture

Parallel computer architecture is classified on the basis of the level at which the hardware supports parallelism. There are different classes of parallel computer architectures, which are as follows:

Multi-core computing

A computer processor integrated circuit containing two or more distinct processing cores is known as a multi-core processor, which has the capability of executing program instructions simultaneously. Cores may implement architectures like VLIW, superscalar, multithreading, or vector and are integrated on a single integrated circuit die or onto multiple dies in a single chip package. Multi-core architectures are classified as heterogeneous that consists of cores that are not identical, or they are categorized as homogeneous that consists of only identical cores.

Symmetric multiprocessing

In Symmetric multiprocessing, a single operating system handles multiprocessor computer architecture having two or more homogeneous, independent processors that treat all processors equally. Each processor can work on any task without worrying about the data for that task is available in memory and may be connected with the help of using on-chip mesh networks. Also, all processor contains a private cache memory.

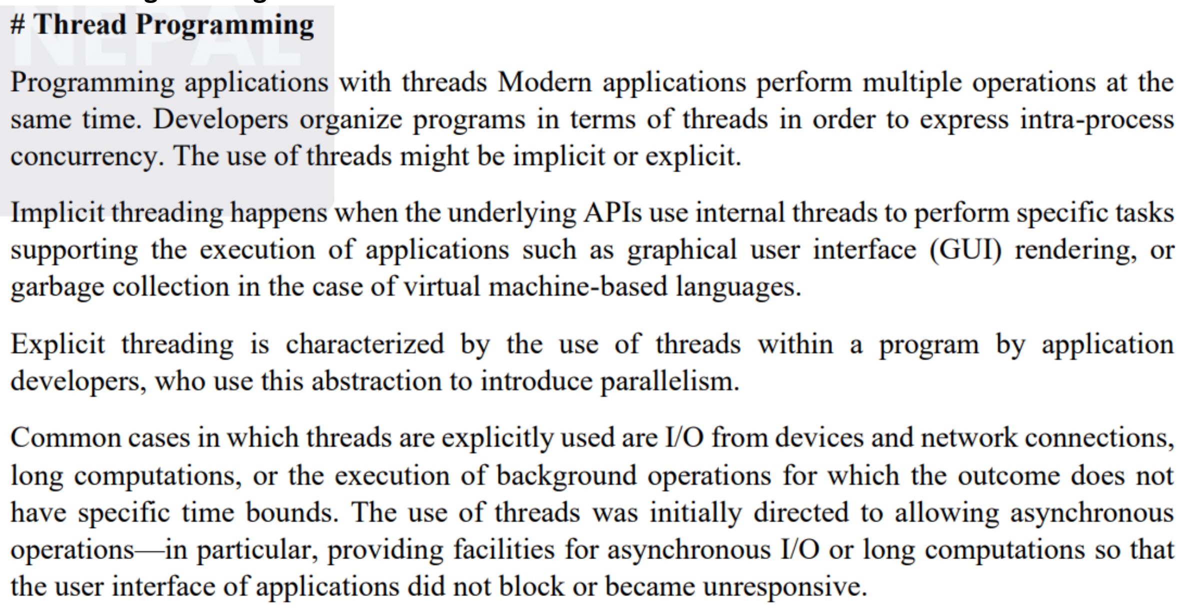
Distributed computing

On different networked computers, the components of a distributed system are located. These networked computers coordinate their actions with the help of communicating through HTTP, RPC-like message queues, and connectors. The concurrency of components and independent failure of components are the characteristics of distributed systems. Typically, distributed programming is classified in the form of peer-to-peer, client-server, n-tier, or three-tier architectures. Sometimes, the terms parallel computing and distributed computing are used interchangeably as there is much overlap between both.

Massively parallel computing

In this, several computers are used simultaneously to execute a set of instructions in parallel. Grid computing is another approach where numerous distributed computer system execute simultaneously and communicate with the help of the Internet to solve a specific problem.

Thread Programming :



# Thread Programming

Programming applications with threads Modern applications perform multiple operations at the same time. Developers organize programs in terms of threads in order to express intra-process concurrency. The use of threads might be implicit or explicit.

Implicit threading happens when the underlying APIs use internal threads to perform specific tasks supporting the execution of applications such as graphical user interface (GUI) rendering, or garbage collection in the case of virtual machine-based languages.

Explicit threading is characterized by the use of threads within a program by application developers, who use this abstraction to introduce parallelism.

Common cases in which threads are explicitly used are I/O from devices and network connections, long computations, or the execution of background operations for which the outcome does not have specific time bounds. The use of threads was initially directed to allowing asynchronous operations—in particular, providing facilities for asynchronous I/O or long computations so that the user interface of applications did not block or became unresponsive.

What is a thread?

A thread identifies a single control flow, which is a logical sequence of instructions, within a process. By logical sequence of instructions, we mean a sequence of instructions that have been designed to be executed one after the other one. More commonly, a thread identifies a kind of yarn that is used for sewing, and the feeling of continuity that is expressed by the interlocked fibers of that yarn is used to recall the concept that the instructions of thread express a logically continuous sequence of operations.

Operating systems that support multithreading identify threads as the minimal building blocks for expressing running code. This means that, despite their explicit use by developers, any sequence of instruction that is executed by the operating system is within the context of a thread. As a consequence, each process contains at least one thread but, in several cases, is composed of many threads having variable lifetimes. Threads within the same process share the memory space and the execution context; besides this, there is no substantial difference between threads belonging to different processes

Thread APIs

Even though the support for multithreading varies according to the operating system and the specific programming languages that are used to develop applications, it is possible to identify a minimum set of features that are commonly available across all the implementations.

POSIX Threads

Portable Operating System Interface for Unix (POSIX) is a set of standards related to the application programming interfaces for a portable development of applications over the Unix operating system flavors.

The POSIX standard defines the following operations: creation of threads with attributes, termination of a thread, and waiting for thread completion (join operation)

The model proposed by POSIX has been taken as a reference for other implementations that might provide developers with a different interface but a similar behavior. What is important to remember from a programming point of view is the following:

• A thread identifies a logical sequence of instructions.

• A thread is mapped to a function that contains the sequence of instructions to execute.

• A thread can be created, terminated, or joined.

• A thread has a state that determines its current condition, whether it is executing, stopped, terminated, waiting for I/O, etc.

• The sequence of states that the thread undergoes is partly determined by the operating system scheduler and partly by the application developers.

• Threads share the memory of the process, and since they are executed concurrently, they need synchronization structures.

Threading support in java and .NET

Languages such as Java and C# provide a rich set of functionalities for multithreaded programming by using an object-oriented approach.

Both Java and .NET express the thread abstraction with the class Thread exposing the common operations performed on threads: start, stop, suspend, resume, abort, sleep, join, and interrupt. Start and stop/abort are used to control the lifetime of the thread instance.

Thread Programming in .Net

The c# Thread class has properties and methods for creating and controlling threads. It may be found in System.Namespace for threading

C# Thread Properties

Property	Description
CurrentThread	returns the instance of currently running thread.
IsAlive	checks whether the current thread is alive or not. It is used to find the execution status of the thread.
IsBackground	is used to get or set value whether current thread is in background or not.
ManagedThreadId	is used to get unique id for the current managed thread.
Name	is used to get or set the name of the current thread.
Priority	is used to get or set the priority of the current thread.
ThreadState	is used to return a value representing the thread state.

C# Thread Methods

A list of important methods of Thread class are given below:

Method	Description
Abort()	is used to terminate the thread. It raises ThreadAbortException.
Interrupt()	is used to interrupt a thread which is in *WaitSleepJoin* state.
Join()	is used to block all the calling threads until this thread terminates.
ResetAbort()	is used to cancel the Abort request for the current thread.
Resume()	is used to resume the suspended thread. It is obselete.
Sleep(Int32)	is used to suspend the current thread for the specified milliseconds.
Start()	changes the current state of the thread to Runnable.
Suspend()	suspends the current thread if it is not suspended. It is obselete.
Yield()	is used to yield the execution of current thread to another thread.

Parallel Computation with threads

Creating parallel applications needs a comprehensive understanding of the problem and its logical structure. Understanding the interdependence and correlations of task inside an application is critical for developing the optimal software structure and adding parallelism when necessary. A decomposition is a useful approach for determining if a problem can be broken down into components that can be done concurrently. Domain and functional decompositions are the two basic decomposition approaches.

Multithreading with ANEKA

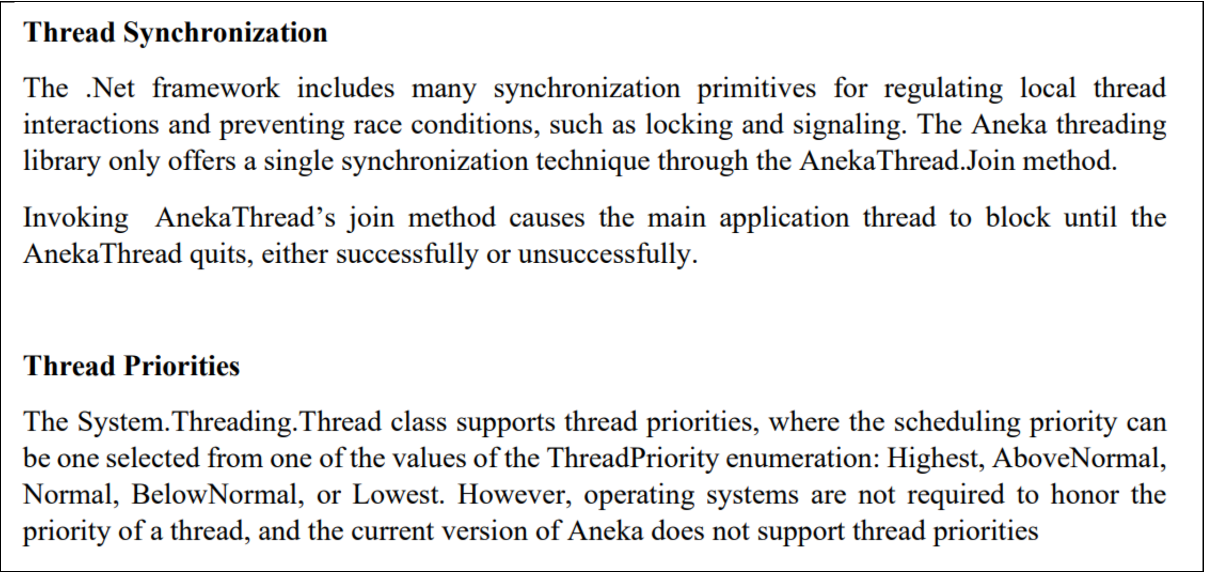
- Aneka offers the capability of implementing multi-threaded applications over the Cloud by means of *Thread Programming Model*.

- The *Thread Programming Model* has been designed to transparently porting high-throughput multi-threaded parallel applications over a distributed infrastructure and provides the best advantage in the case of embarrassingly parallel applications.

#Aneka Environment

Aneka Threads

An AnekaThread is a piece of work that may be run on a remote computer. Unlike ordinary threads, each AnekaThread runs in its process on the distant system. An AnekaThread provides a comparable interface to a normal thread and may be started and aborted in the same way.

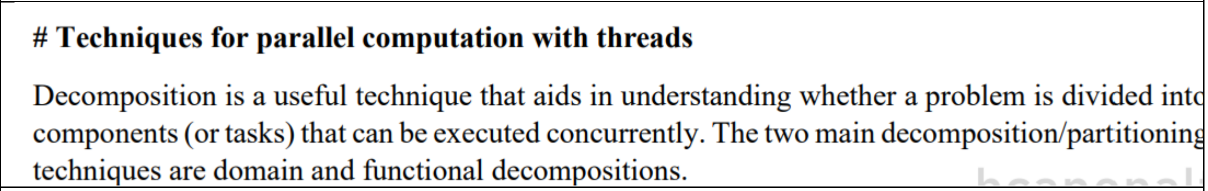

Thread Synchronization

The .Net framework includes many synchronization primitives for regulating local thread interactions and preventing race conditions, such as locking and signaling. The Aneka threading library only offers a single synchronization technique through the AnekaThread.Join method.

Invoking AnekaThread's join method causes the main application thread to block until the AnekaThread quits, either successfully or unsuccessfully.

Thread Priorities

The System.Threading.Thread class supports thread priorities, where the scheduling priority can be one selected from one of the values of the ThreadPriority enumeration: Highest, AboveNormal, Normal, BelowNormal, or Lowest. However, operating systems are not required to honor the priority of a thread, and the current version of Aneka does not support thread priorities


# Techniques for parallel computation with threads

Decomposition is a useful technique that aids in understanding whether a problem is divided int
components (or tasks) that can be executed concurrently. The two main decomposition/partitioning
techniques are domain and functional decompositions.

Map Reduce Model

A MapReduce is a data processing tool which is used to process the data parallelly in a distributed form. It was developed in 2004, on the basis of paper titled as "MapReduce: Simplified Data Processing on Large Clusters," published by Google.

The MapReduce is a paradigm which has two phases, the mapper phase, and the reducer phase. In the Mapper, the input is given in the form of a key-value pair. The output of the Mapper is fed to the reducer as input. The reducer runs only after the Mapper is over. The reducer too takes input in key-value format, and the output of reducer is the final output.

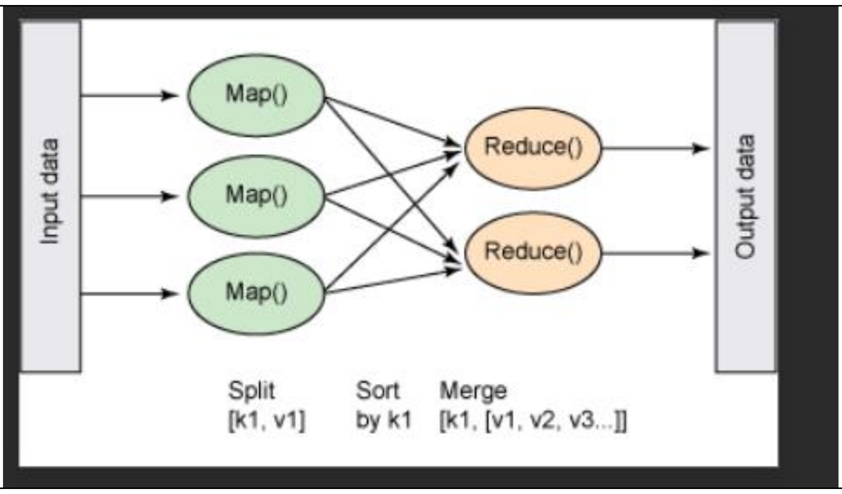
MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into *mappers* and *reducers* is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

Google released a paper on MapReduce technology in December 2004. This became the genesis of the Hadoop Processing Model. So, MapReduce is a programming model that allows us to perform parallel and distributed processing on huge datasets.

The Algorithm

- Generally MapReduce paradigm is based on sending the computer to where the data resides!
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
  - **Map stage** − The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
  - **Reduce stage** − This stage is the combination of the Shuffle stage and the Reduce stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.
- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.



Inputs and Outputs (Java Perspective)

The MapReduce framework operates on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.

The key and the value classes should be in serialized manner by the framework and hence, need to implement the Writable interface. Additionally, the key classes have to implement the Writable-Comparable interface to facilitate sorting by the framework. Input and Output types of a **MapReduce job** − (Input) <k1, v1> → map → <k2, v2> → reduce → <k3, v3>(Output).

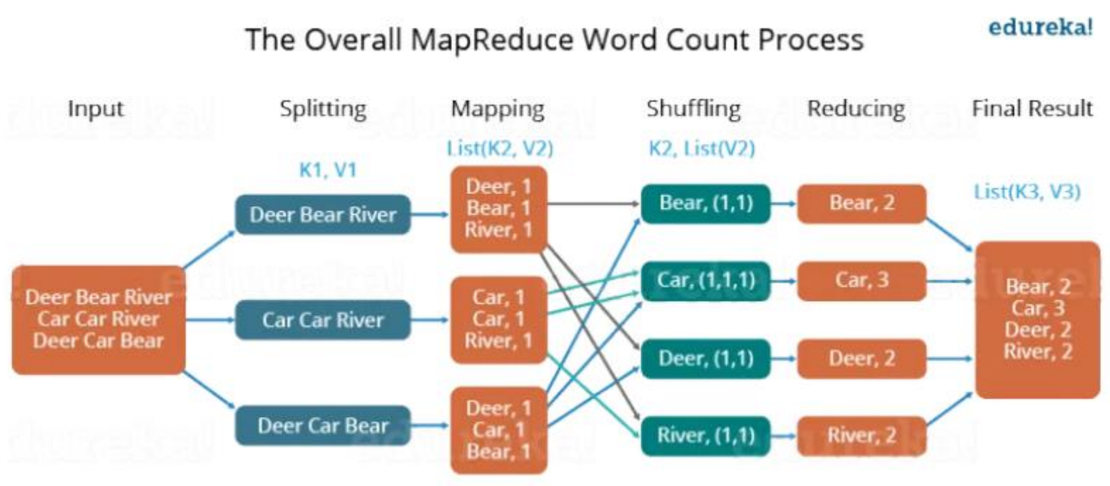
	Input	Output
Map	<k1, v1>	list (<k2, v2>)
Reduce	<k2, list(v2)>	list (<k3, v3>)

A Word Count Example of MapReduce

Let us understand, how a MapReduce works by taking an example where I have a text file called example.txt whose contents are as follows:

Dear, Bear, River, Car, Car, River, Deer, Car and Bear

Now, suppose, we have to perform a word count on the sample.txt using MapReduce. So, we will be finding unique words and the number of occurrences of those unique words.



MapReduce Example - MapReduce Tutorial

- First, we divide the input into three splits as shown in the figure. This will distribute the work among all the map nodes.

- Then, we tokenize the words in each of the mappers and give a hardcoded value (1) to each of the tokens or words. The rationale behind giving a hardcoded value equal to 1 is that every word, in itself, will occur once.

- Now, a list of key-value pair will be created where the key is nothing but the individual words and value is one. So, for the first line (Dear Bear River) we have 3 key-value pairs — Dear, 1; Bear, 1; River, 1. The mapping process remains the same on all the nodes.

- After the mapper phase, a partition process takes place where sorting and shuffling happen so that all the tuples with the same key are sent to the corresponding reducer.

- So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. For example, Bear, [1,1]; Car, [1,1,1].., etc.

- Now, each Reducer counts the values which are present in that list of values. As shown in the figure, reducer gets a list of values which is [1,1] for the key Bear. Then, it counts the number of ones in the very list and gives the final output as — Bear, 2.

- Finally, all the output key/value pairs are then collected and written in the output file.

Advantages of MapReduce

The two biggest advantages of MapReduce are:

1. Parallel Processing:

In MapReduce, we are dividing the job among multiple nodes and each node works with a part of the job simultaneously. So, MapReduce is based on Divide and Conquer paradigm which helps us to process the data using different machines. As the data is processed by multiple machines instead of a single machine in parallel, the time taken to process the data gets reduced by a tremendous amount.

2. Data Locality:

Instead of moving data to the processing unit, we are moving the processing unit to the data in the MapReduce Framework. In the traditional system, we used to bring data to the processing unit and process it. But, as the data grew and became very huge, bringing this huge amount of data to the processing unit posed the following issues:

- Moving huge data to processing is costly and deteriorates the network performance.
- Processing takes time as the data is processed by a single unit which becomes the bottleneck.
- Master node can get over-burdened and may fail.

Now, MapReduce allows us to overcome the above issues by bringing the processing unit to the data. So, as you can see in the above image that the data is distributed among multiple nodes where each node processes the part of the data residing on it. This allows us to have the following advantages:

- It is very cost effective to move the processing unit to the data.
- The processing time is reduced as all the nodes are working with their part of the data in parallel.
- Every node gets a part of the data to process and therefore, there is no chance of a node getting overburdened.

Key Features of MapReduce

The following advanced features characterize MapReduce:

1. Highly scalable

A framework with excellent scalability is Apache Hadoop MapReduce. This is because of its capacity for distributing and storing large amounts of data across numerous servers. These servers can all run simultaneously and are all reasonably priced.

By adding servers to the cluster, we can simply grow the amount of storage and computing power. We may improve the capacity of nodes or add any number of nodes (horizontal scalability) to attain high computing power. Organizations may execute applications from massive sets of nodes, potentially using thousands of terabytes of data, thanks to Hadoop MapReduce programming.

2 Versatile

Businesses can use MapReduce programming to access new data sources. It makes it possible for companies to work with many forms of data. Enterprises can access both organized and unstructured data with this method and acquire valuable insights from the various data sources.

Since Hadoop is an open-source project, its source code is freely accessible for review, alterations, and analyses. This enables businesses to alter the code to meet their specific needs. The MapReduce framework supports data from sources including email, social media, and clickstreams in different languages

3. Secure

The MapReduce programming model uses the HBase and HDFS security approaches, and only authenticated users are permitted to view and manipulate the data. HDFS uses a replication technique in Hadoop 2 to provide fault tolerance. Depending on the replication factor, it makes a clone of each block on the various machines. One can therefore access data from the other devices that house a replica of the same data if any machine in a cluster goes down. Erasure coding has taken the role of this replication technique in Hadoop 3. Erasure coding delivers the same level of fault tolerance with less area. The storage overhead with erasure coding is less than 50%.

4. Affordability

With the help of the MapReduce programming framework and Hadoop’s scalable design, big data volumes may be stored and processed very affordably. Such a system is particularly cost-effective and highly scalable, making it ideal for business models that must store data that is constantly expanding to meet the demands of the present.

In terms of scalability, processing data with older, conventional relational database management systems was not as simple as it is with the Hadoop system. In these situations, the company had to minimize the data and execute classification based on presumptions about how specific data could be relevant to the organization, hence deleting the raw data. The MapReduce programming model in the Hadoop scale-out architecture helps in this situation.

5. Fast-paced

The Hadoop Distributed File System, a distributed storage technique used by MapReduce, is a mapping system for finding data in a cluster. The data processing technologies, such as MapReduce programming, are typically placed on the same servers that enable quicker data processing.

Thanks to Hadoop’s distributed data storage, users may process data in a distributed manner across a cluster of nodes. As a result, it gives the Hadoop architecture the capacity to process data exceptionally quickly. Hadoop MapReduce can process unstructured or semi-structured data in high numbers in a shorter time.

6. Based on a simple programming model

Hadoop MapReduce is built on a straightforward programming model and is one of the technology's many noteworthy features. This enables programmers to create MapReduce applications that can handle tasks quickly and effectively. Java is a very well-liked and simple-to-learn programming language used to develop the MapReduce programming model.
Java programming is simple to learn, and anyone can create a data processing model that works for their company. Hadoop is straightforward to utilize because customers don't need to worry about computing distribution. The framework itself does the processing.

7. Parallel processing-compatible

The parallel processing involved in MapReduce programming is one of its key components. The tasks are divided in the programming paradigm to enable the simultaneous execution of independent activities. As a result, the program runs faster because of the parallel processing, which makes it simpler for the processes to handle each job. Multiple processors can carry out these broken-down tasks thanks to parallel processing. Consequently, the entire software runs faster.

Top 5 Uses of MapReduce

By spreading out processing across numerous nodes and merging or decreasing the results of those nodes, MapReduce has the potential to handle large data volumes. This makes it suitable for the following use cases:

1. Entertainment

Hadoop MapReduce assists end users in finding the most popular movies based on their preferences and previous viewing history. It primarily concentrates on their clicks and logs. Various OTT services, including Netflix, regularly release many web series and movies. It may have happened to you that you couldn't pick which movie to watch, so you looked at Netflix's recommendations and decided to watch one of the suggested series or films. Netflix uses Hadoop and MapReduce to indicate to the user some well-known movies based on what they have watched and which movies they enjoy. MapReduce can examine user clicks and logs to learn how they watch movies.

2. E-commerce

Several e-commerce companies, including Flipkart, Amazon, and eBay, employ MapReduce to evaluate consumer buying patterns based on customers' interests or historical purchasing patterns. For various e-commerce businesses, it provides product suggestion methods by analyzing data, purchase history, and user interaction logs.
Many e-commerce vendors use the MapReduce programming model to identify popular products based on customer preferences or purchasing behavior. Making item proposals for e-commerce inventory is part of it, as is looking at website records, purchase histories, user interaction logs, etc., for product recommendations.

3. Social media

Nearly 500 million tweets, or about 3000 per second, are sent daily on the microblogging platform Twitter. MapReduce processes Twitter data, performing operations such as tokenization, filtering, counting, and aggregating counters.

- **Tokenization:** It creates key-value pairs from the tokenized tweets by mapping the tweets as maps of tokens.
- **Filtering:** The terms that are not wanted are removed from the token maps.
- **Counting:** It creates a token counter for each word in the count.
- **Aggregate counters:** A grouping of comparable counter values is prepared into small, manageable pieces using aggregate counters.

4. Data warehouse

Systems that handle enormous volumes of information are known as data warehouse systems. The star schema, which consists of a fact table and several dimension tables, is the most popular data warehouse model. In a shared-nothing architecture, storing all the necessary data on a single node is impossible, so retrieving data from other nodes is essential.
This results in network congestion and slow query execution speeds. If the dimensions are not too big, users can replicate them over nodes to get around this issue and maximize parallelism. Using MapReduce, we may build specialized business logic for data insights while analyzing enormous data volumes in data warehouses.

5. Fraud detection

Conventional methods of preventing fraud are not always very effective. For instance, data analysts typically manage inaccurate payments by auditing a tiny sample of claims and requesting medical records from specific submitters. Hadoop is a system well suited for handling large volumes of data needed to create fraud detection algorithms. Financial businesses, including banks, insurance companies, and payment locations, use Hadoop and MapReduce for fraud detection, pattern recognition evidence, and business analytics through transaction analysis.

Parallel efficiency in map reduce

Parallel efficiency in the context of MapReduce refers to the effectiveness of utilizing parallel processing to perform a computation or process a dataset. MapReduce is a programming model and processing framework designed for large-scale data processing across distributed computing clusters. It divides a task into two main phases: the map phase and the reduce phase.

Here's a brief overview of the MapReduce process:

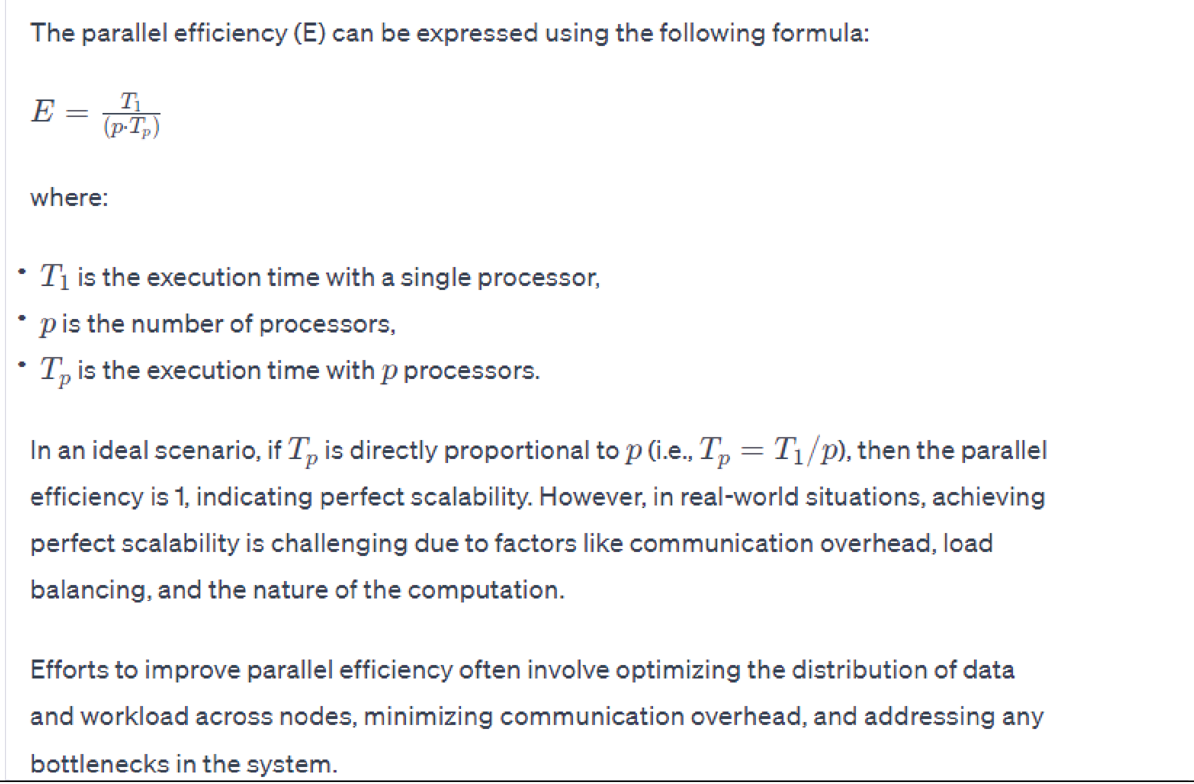
1. **Map Phase:**

- Input data is divided into smaller chunks.
- The map function is applied to each chunk independently, producing intermediate key-value pairs.

2. **Shuffle and Sort Phase:**
   - Intermediate key-value pairs from the map phase are shuffled and sorted based on keys. This step is crucial for grouping related data together.
3. **Reduce Phase:**
   - The reduce function is applied to each group of intermediate key-value pairs, producing the final output.

The parallel efficiency of a MapReduce job is influenced by how well the computation can be divided into independent tasks that can be executed simultaneously. The goal is to achieve a linear speedup, meaning that doubling the number of processors should ideally result in a halving of the processing time.

The parallel efficiency (E) can be expressed using the following formula:



The parallel efficiency (E) can be expressed using the following formula:

$$E = \frac{T_1}{(p \cdot T_p)}$$

where:

- T_1 is the execution time with a single processor,
- p is the number of processors,
- T_p is the execution time with p processors.

In an ideal scenario, if T_p is directly proportional to p (i.e., $T_p = T_1/p$), then the parallel efficiency is 1, indicating perfect scalability. However, in real-world situations, achieving perfect scalability is challenging due to factors like communication overhead, load balancing, and the nature of the computation.

Efforts to improve parallel efficiency often involve optimizing the distribution of data and workload across nodes, minimizing communication overhead, and addressing any bottlenecks in the system.

Q. Explain Parallel Efficiency of MapReduce.

It is worth trying to figure out what is the parallel efficiency of MapReduce. Now let us assume that the data produced after the map phase is σ times the original data size D (σD), further, we assume that there are P processors which in turn perform map and reduce operations depending on which phase they are used in, so we do not have any wastage of processors. Also, the algorithm itself is assumed to do wD useful work, w is not necessarily a constant it could be D^2 or something like that, but the point is that there is some amount of useful work being done even if you have a single processor and that's wD. Now let us look at the overheads of doing the computation wD using MapReduce. After the map operation, instead of D data items, we have σD data items in P pieces, so each mapper writes data to their local disk. So, P there is some overhead associated with writing this data. Next, this data has to be read by each reducer before it can begin 'reduce operations, so each reducer has to read that one Pth of the data from a particular mapper. Since there are P different reducers, one Pth of the data in a map goes to each reducer and it has to read P of these from P different

mappers once again getting us the communication time that a reducer spends getting the data it requires from the different mappers as P

Overheads: σD/P

So, the total overhead is work that would not have to be done if we did not have parallelism, that is writing data to disk and reading data from remote mappers is 2D/P. Now if you look at the efficiency using this overhead, we get the wD is the time it takes on one processor which is the useful work that needs to be done wD/P; if we had P processors but we have this extra overhead of 2σD/P. We have got a constant c which essentially measures how much time it takes to write one data item to disk or to read a data item remotely from a different mapper. Now you assumed both of these to be the same constant.

loud Program

$$\cdot \mathrm{MR} = \frac{wD}{P\left(\frac{wD}{P} + 2C\frac{sD}{P}\right)} = \frac{1}{1 + \frac{2c}{w}s}$$

Simplifying this parallel efficiency, we get an expression that is surprisingly independent of P, dependent on w and σ in a MapReduce important way though. It is nice that it is independent of P because it indicates that the is scalable and you can get large efficiencies even with a large number of processors, on the processors with the efficiency. However, it is dependent on σ and w. In process dependency particular, as long as the amount of useful work that you do per data item grows, the efficiency ends up there is no ing close to 1, which is nice. On the other hand, if the extra data that you produce after the map phase grows then efficiency can suffer and reduce considerably. So, it is very important to use things like Combiners to ensure that too much data does not get blown up after the map phase is done. In particular, if σ up being dependent on P then the statement here that efficiency is independent of P no longer holds so be careful when computing this expression. σ is quite likely to be dependent on P even though the expression hides that dependence.

This is quite insightful and will compute the parallel efficiency of our word-counting problem using this formula. When we were counting words we had n documents, m total possible words, each occurring say, f times per document on average, so the total volume of data to be processed is nmf. Now, after the map phase using combiners, we produce P into m partial counts, that is m different partial counts per mapper and their P mappers. At worst, there will be mP partial counts, some mappers may not produce counts for words that do not occur

there, but we do not worry about that. We assume that at worst there will be mP partial counts which give us σ as the data after the map, as compared to the data before it and there you get P/nf.

Extra

What Is MapReduce Architecture?

MapReduce Architecture is a programming model and a software framework utilized for preparing enormous measures of data. MapReduce program works in two stages, to be specific, Map and Reduce. Map requests that arrange with mapping and splitting of data while Reduce tasks reduce and shuffle the data.

Hadoop MapReduce Architecture is fit for running MapReduce programs written in different languages: C, Python, Ruby, and Java. The projects of MapReduce in cloud computing are equal, accordingly help to perform an enormous scope of data examination utilizing various machines in the cluster.

1. MAPREDUCE ARCHITECTURE

HDFS and MapReduce architecture are the two significant parts of Hadoop that make it so efficient and powerful to utilize. MapReduce is a programming model utilized for proficient handling in equal over huge data collections in a conveyed way. The data is first to part and afterward consolidated to deliver the eventual outcome.

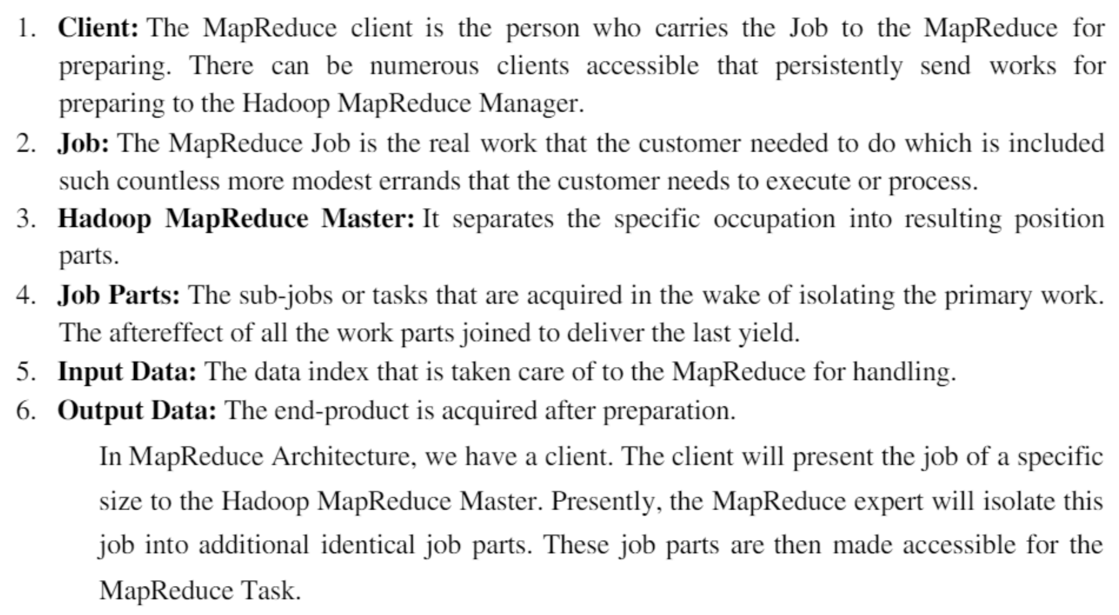
The MapReduce task is predominantly isolated into 2 phases:

1. Map Phase
2. Reduce Phase

The libraries for MapReduce are written in so many programming languages with different diverse various improvements. The motivation behind Map Reduce in Hadoop is to Map every one of the positions, and afterward, it will decrease it to comparable undertakings for giving less overhead over the cluster network and to diminish the preparing power.

2. COMPONENTS OF MAPREDUCE ARCHITECTURE

- **Components of MapReduce Architecture are:**
- Client
- Job
- Hadoop MapReduce Master
- Job Parts
- Input Data
- Output Data



1. **Client:** The MapReduce client is the person who carries the Job to the MapReduce for preparing. There can be numerous clients accessible that persistently send works for preparing to the Hadoop MapReduce Manager.
2. **Job:** The MapReduce Job is the real work that the customer needed to do which is included such countless more modest errands that the customer needs to execute or process.
3. **Hadoop MapReduce Master:** It separates the specific occupation into resulting position parts.
4. **Job Parts:** The sub-jobs or tasks that are acquired in the wake of isolating the primary work. The aftereffect of all the work parts joined to deliver the last yield.
5. **Input Data:** The data index that is taken care of to the MapReduce for handling.
6. **Output Data:** The end-product is acquired after preparation.

In MapReduce Architecture, we have a client. The client will present the job of a specific size to the Hadoop MapReduce Master. Presently, the MapReduce expert will isolate this job into additional identical job parts. These job parts are then made accessible for the MapReduce Task.

Map Reduce Infrastructure

The MapReduce infrastructure is a distributed computing framework that provides a scalable and fault-tolerant solution for processing large datasets across a cluster of machines. It was popularized by Google and is widely used for big data processing. The open-source Apache Hadoop project is a well-known implementation of the MapReduce framework.

Here are the key components of the MapReduce infrastructure:

1. **Job Tracker (or Master Node):**
   - The Job Tracker is the central coordinator that manages the entire MapReduce job.
   - It accepts job submissions, schedules tasks, and monitors the progress of tasks across the cluster.
   - It keeps track of the availability of Task Trackers and assigns tasks to them.
2. **Task Trackers (or Worker Nodes):**
   - Task Trackers run on individual worker nodes in the cluster.
   - They are responsible for executing map and reduce tasks assigned by the Job Tracker.
   - Task Trackers periodically send heartbeat signals to the Job Tracker to indicate their availability.
3. **Map Tasks:**
   - The input data is divided into splits, and each split is processed by a separate map task.
   - Map tasks run in parallel across multiple nodes, processing different portions of the input data concurrently.
   - The output of the map tasks is a set of intermediate key-value pairs.

4. **Shuffle and Sort:**
    - After the map phase, the framework performs a shuffle and sort phase to organize the intermediate key-value pairs.
    - It ensures that all values for a given key are grouped together and sorted, preparing them for the reduce phase.
5. **Reduce Tasks:**
    - The reduce phase takes the output of the shuffle and sort phase as input.
    - Each reduce task processes a subset of the sorted key-value pairs, producing the final output.
    - Reduce tasks run in parallel, and their output is typically written to a distributed file system.
6. **Distributed File System:**
    - MapReduce jobs often use a distributed file system (e.g., Hadoop Distributed File System - HDFS) to store input data, intermediate results, and final output.
    - The distributed file system provides fault tolerance by replicating data across multiple nodes.
7. **Input and Output Formats:**
    - MapReduce supports various input and output formats, allowing users to customize how data is read from and written to the distributed file system.
8. **Combiners:**
    - Combiners are optional functions in the MapReduce framework that can be used to perform a local aggregation of the map output before sending it over the network to the reduce tasks.
    - Combiners help in reducing the amount of data shuffled between nodes, improving overall performance.

The MapReduce infrastructure abstracts the complexities of distributed computing and allows developers to focus on writing map and reduce functions for their specific processing needs. While it has been widely used, newer data processing frameworks like Apache Spark have gained popularity due to their improved performance and additional features, such as in-memory processing and support for iterative algorithms.