INTRODUCTION TO DISTRIBUTED DATA PROCESSING pdf

1: Distributed Processing

Introduction to Distributed Data Processing Distributed Database Systems.

Markos Sfikas Last month we introduced data Artisans Streaming Ledger , our new technology that brings serializable, distributed ACID transactions directly on data streams. In this blog post, we take a step back to introduce ACID semantics and transaction processing and lay the foundations based on which data Artisans Streaming Ledger operates. Transactions in modern enterprise systems are ubiquitous and necessary for providing data consistency even in highly concurrent environments. The transaction is independently executed for data retrieval or updates and must be treated as one entity: It has to either happen in full or not at all. Whether being a distributed systems engineer or an application developer, one needs to ensure that the system or application transactions are propagated both consistently and atomically to avoid data loss or deviated results. There is a long-standing debate about transactional processing guarantees in the industry. On the one side, industry experts argue that ACID guarantees see below are not necessary on all types of applications and that time gains outweigh any potential losses when ACID guarantees are in place. On the other side of the equation, IT professionals consider strong transactional guarantees the only way for application developers to focus on the application logic and write correct code, in a scalable manner, without the experience of a distributed systems engineer. In this section, we give a brief explanation of the ACID terminology as it relates to database transactions. The ACID acronym defines the properties of a database transaction that are characterized by four attributes: Atomicity Atomicity ensures that all changes made to data are executed as a single entity and operation. This means that the operation succeeds ONLY when all changes are performed. Consistency The consistency attribute ensures that all data changes are executed while maintaining a consistent state between the transaction start and end. This means that the transaction brings the tables, keys or any data type from one consistent state into another consistent state. Using again the example of the transfer of funds between accounts, the consistency attribute will guarantee that the transfer will only happen if the source account has sufficient funds. Isolation The isolation attribute certifies that each transaction executes as if it were the only transaction operating on the tables or keys. This means that the intermediate state of a transaction is invisible to other transactions that might be happening concurrently. Databases provide different isolation levels with different guarantees. Durability Durability guarantees that once a transaction is completed successfully, it will result in a permanent state change that will remain persistent even in the event of a system failure. In the example of the transfer of funds between accounts, it is the durability that will ensure that updates made to each account are persistent and can survive a potential system failure. With data Artisans Streaming Ledger, durability is ensured in the same way as in other Flink applications â€" through persistent sources and checkpoints. In the asynchronous nature of stream processing, the durability can only be assumed after a checkpoint for details, see the consistency model section of the data Artisans Streaming Ledger whitepaper. What is transaction processing and why is it important for the modern enterprise? Transaction processing has emerged as a necessary technology for modern enterprises dealing with real-time data and real-time applications. It helps in the reduction of errors, the provision of superior customer experience and the reliability of reporting and operations, especially in regulated industries such as the capital markets, investment banking, and financial services industries. Practically every business dealing with partners, suppliers, staff, stock or customers should think about adopting a transaction processing system in order to become more service-oriented and software-operated. How data Artisans Streaming Ledger came to fruition? At data Artisans, we have witnessed the exponential growth in the adoption of stream processing from the very early days. We always believed that stream processing is the new paradigm in data processing and we expect that with more and more companies adopting such technologies, stream processing frameworks will continue to evolve further in the future. When partnering with many world-class financial services organizations to develop their stream processing architecture we saw a gap in providing strong ACID guarantees directly on data streams. This was, in fact, one of the reasons that prevented some of those organizations to move more of their mission-critical applications to a streaming infrastructure which made us start thinking about how we could enable distributed serializable ACID transactions on streaming data. We encourage you to read more about Streaming Ledger by downloading our whitepaper or contacting us for more information or a consultation. Newsletter Sign up to receive news and updates. It only takes a click to unsubscribe.

2: Introduction to Distributed System Design - Google Code University - Google Code

The increased capabilities of a collection of logically interrelated databases distributed over a computer network enable scalable data processing. This course addresses the components of these systems, covering the following main topics: distributed database architectures, distributed data storage.

This is in contrast to a single, centralized server managing and providing processing capability to all connected systems. Computers that comprise the distributed data-processing network are located at different locations but interconnected by means of wireless or satellite links. Lower Cost Larger organizations invest in expensive mainframe and supercomputers to function as centralized servers. Each mainframe machine, for example, costs several hundred thousand dollars versus several thousand dollars for a few minicomputers, according to the University of New Mexico. Distributed data processing considerably lowers the cost of data sharing and networking across an organization by comprising several minicomputers that cost significantly less than mainframe machines. Reliable Hardware glitches and software anomalies can cause single-server processing to malfunction and fail, resulting in a complete system breakdown. Distributed data processing is more reliable, since multiple control centers are spread across different machines. A glitch in any one machine does not impact the network, since another machine takes over its processing capability. Faulty machines are quickly isolated and repaired. This makes distributed data processing more reliable than single-server processing systems. Improved Performance and Reduced Processing Time Single computers are limited in their performance and efficiency. An easy way to increase performance is by adding another computer to a network. Adding yet another computer will further augment performance, and so on. Distributed data processing works on this principle and holds that a job gets done faster if multiple machines are handling it in parallel, or synchronously. Complicated statistical problems, for example, are broken into modules and allocated to different machines where they are processed simultaneously. This significantly reduces processing time and improves performance. Flexible Individual computers that comprise a distributed network are present at different geographical locations. For example, an organizational-distributed network comprising of three computers can have each machine in a different branch. The three machines are interconnected via the Internet and are able to process data in parallel, even while at different locations. This makes distributed data-processing networks more flexible. The system is flexible also in terms of increasing or decreasing processing power. For example, adding more nodes or computers to the network increases processing power and overall system capability, while reducing computers from the network decreases processing power.

3: What is distributed data processing (DDP)? definition and meaning - www.enganchecubano.com

Definition of distributed data processing (DDP): Arrangement of networked computers in which data processing capabilities are spread across the network. In DDP, specific jobs are performed by specialized computers which may be far removed from the.

October 18, Today, the volume of data is often too big for a single server â€" node â€" to process. Therefore, there was a need to develop code that runs on multiple nodes. Writing distributed systems is an endless array of problems, so people developed multiple frameworks to make our lives easier. MapReduce is a framework that allows the user to write code that is executed on multiple nodes without having to worry about fault tolerance, reliability, synchronization or availability. Batch processing There are a lot of use cases for a system described in the introduction, but the focus of this post will be on data processing â€" more specifically, batch processing. Usually, the job will read the batch data from a database and store the result in the same or different database. An example of a batch processing job could be reading all the sale logs from an online shop for a single day and aggregating it into statistics for that day number of users per country, the average spent amount, etc. Doing this as a daily job could give insights into customer trends. MapReduce MapReduce is a programming model that was introduced in a white paper by Google in For MapReduce to be able to do computation on large amounts of data, it has to be a distributed model that executes its code on multiple nodes. This allows the computation to handle larger amounts of data by adding more machines â€" horizontal scaling. This is different from vertical scaling, which implies increasing the performance of a single machine. This way, the data stays on the same node, but the code is moved via the network. This is ideal because the code is much smaller than the data. Each node runs executes the given functions on the data it has in order the minimize network traffic shuffling data. The computation performance of MapReduce comes at the cost of its expressivity. The map phase generates key-value data pairs from the input data partitions , which are then grouped by key and used in the reduce phase by the reduce task. Everything except the interface of the functions is programmable by the user. Map Hadoop , along with its many other features, had the first open-source implementation of MapReduce. In order to increase parallelization, each directory is made up of smaller units called partitions and each partition can be processed separately by a map task the process that executes the map function. This is hidden from the user, but it is important to be aware of it because the number of partitions can affect the speed of execution. The map task mapper is called once for every input partition and its job is to extract key-value pairs from the input partition. The mapper can generate any number of key-value pairs from a single input including zero, see the figure above. The user only needs to define the code inside the mapper. Below, we see an example of a simple mapper that takes the input partition and outputs each word as a key with value 1. Map function, is applied on a partition def mapper key, value: Split the text into words and yield word,1 as a pair for word in value. All the grouped values entering the reducers are sorted by the framework. Reduce function, applied to a group of values with the same key def reducer key, values: The reducer can be called on all the values with the same key word , to create a distributed word counting pipeline. In the image below, we see that not every sorted group has a reduce task. This happens because the user needs to define the number of reducers, which is 3 in our case. After a reducer is done with its task, it takes another group if there is one that was not processed. Practical example In order for this post to not be only dry words and images, I have added these examples to a lightweight MapReduce in Python that you can run easily run on your local machine. The map and reduce functions are same as the ones above word counting. Run the command python2 example. Congrats, you just created a MapReduce word counting pipeline. Even if this does not sound impressive, the flexibility of MapReduce allows the user to do more complex data processing such as table joins, page rank, sorting and anything you can code inside the limitations of the framework. Conclusion MapReduce is a programming model that allows the user to write batch processing jobs with a small amount of code. It is flexible in the sense that you, the user, can write code to modify the behavior, but making complex data processing pipelines becomes cumbersome because every MapReduce job has to be managed and scheduled on its own. The intermediate output of map tasks is written to a file which allows the framework to recover easily if a node has a failure. This stability comes at a cost of performance, as the data could have been forwarded to reduce tasks with a small buffer instead, creating a stream. Keep in mind that this was a practical example of getting familiar with the MapReduce framework. Today, some databases and data processing systems allow the user to do computation over multiple machines without having to write the map and reduce functions. These systems offer higher-level libraries that allow the user to define the logic using SQL, Python, Scala, etc. The system translates the code written by the user into one or more MapReduce jobs, so the user does not have to write the actual map and reduce functions. This allows the users already familiar with those languages to utilize the power of the MapReduce framework with ease, using familiar tools. Knowing the inner workings for MapReduce might allow you to write more efficient code. Also, you might be excited to learn more about the inner workings of the systems you use.

4: Advantages of Distributed Data Processing | www.enganchecubano.com

1 G53DBC Distributed Data Processing Introduction to Distributed Data Processing (DDP) Â• Movement and structure of data around organisations Â• Range of data processing approaches.

Models[edit] Many tasks that we would like to automate by using a computer are of questionâ€”answer type: In theoretical computer science , such tasks are called computational problems. Formally, a computational problem consists of instances together with a solution for each instance. Instances are questions that we can ask, and solutions are desired answers to these questions. Theoretical computer science seeks to understand which computational problems can be solved by using a computer computability theory and how efficiently computational complexity theory. Traditionally, it is said that a problem can be solved by using a computer if we can design an algorithm that produces a correct solution for any given instance. Such an algorithm can be implemented as a computer program that runs on a general-purpose computer: Formalisms such as random access machines or universal Turing machines can be used as abstract models of a sequential general-purpose computer executing such an algorithm. However, it is not at all obvious what is meant by "solving a problem" in the case of a concurrent or distributed system: Three viewpoints are commonly used: Parallel algorithms in shared-memory model All processors have access to a shared memory. The algorithm designer chooses the program executed by each processor. One theoretical model is the parallel random access machines PRAM that are used. Shared-memory programs can be extended to distributed systems if the underlying operating system encapsulates the communication between nodes and virtually unifies the memory across all individual systems. A model that is closer to the behavior of real-world multiprocessor machines and takes into account the use of machine instructions, such as Compare-and-swap CAS , is that of asynchronous shared memory. There is a wide body of work on this model, a summary of which can be found in the literature. Models such as Boolean circuits and sorting networks are used. Similarly, a sorting network can be seen as a computer network: Distributed algorithms in message-passing model The algorithm designer only chooses the computer program. All computers run the same program. The system must work correctly regardless of the structure of the network. A commonly used model is a graph with one finite-state machine per node. In the case of distributed algorithms, computational problems are typically related to graphs. Often the graph that describes the structure of the computer network is the problem instance. This is illustrated in the following example. Different fields might take the following approaches: Centralized algorithms[citation needed] The graph G is encoded as a string, and the string is given as input to a computer. The computer program finds a coloring of the graph, encodes the coloring as a string, and outputs the result. Parallel algorithms Again, the graph G is encoded as a string. However, multiple computers can access the same string in parallel. Each computer might focus on one part of the graph and produce a coloring for that part. The main focus is on high-performance computation that exploits the processing power of multiple computers in parallel. Distributed algorithms The graph G is the structure of the computer network. There is one computer for each node of G and one communication link for each edge of G. Initially, each computer only knows about its immediate neighbors in the graph G; the computers must exchange messages with each other to discover more about the structure of G. Each computer must produce its own color as output. The main focus is on coordinating the operation of an arbitrary distributed system. For example, the Coleâ€”Vishkin algorithm for graph coloring [39] was originally presented as a parallel algorithm, but the same technique can also be used directly as a distributed algorithm. Moreover, a parallel algorithm can be implemented either in a parallel system using shared memory or in a distributed system using message passing. Complexity measures[edit] In parallel algorithms, yet another resource in addition to time and space is the number of computers. Indeed, often there is a trade-off between the running time and the number of computers: If a decision problem can be solved in polylogarithmic time by using a polynomial number of processors, then the problem is said to be in the class NC. Perhaps the simplest model of distributed computing is a synchronous system where all nodes operate in a lockstep fashion. In such systems, a central complexity measure is the number of synchronous communication rounds required to complete the task. Let D be the diameter of the network. On the one hand, any computable problem can be solved trivially in a synchronous distributed system in approximately 2D communication rounds: On the other hand, if the running time of the algorithm is much smaller than D communication rounds, then the nodes in the network must produce their output without having the possibility to obtain information about distant parts of the network. In other words, the nodes must make globally consistent decisions based on information that is available in their local D-neighbourhood. Many distributed algorithms are known with the running time much smaller than D rounds, and understanding which problems can be solved by such algorithms is one of the central research questions of the field [44]. Typically an algorithm which solves a problem in polylogarithmic time in the network size is considered efficient in this model. Another commonly used measure is the total number of bits transmitted in the network cf. Other problems[edit] Traditional computational problems take the perspective that we ask a question, a computer or a distributed system processes the question for a while, and then produces an answer and stops. However, there are also problems where we do not want the system to ever stop. Examples of such problems include the dining philosophers problem and other similar mutual exclusion problems. In these problems, the distributed system is supposed to continuously coordinate the use of shared resources so that no conflicts or deadlocks occur. There are also fundamental challenges that are unique to distributed computing. The first example is challenges that are related to fault-tolerance. Examples of related problems include consensus problems , [46] Byzantine fault tolerance , [47] and self-stabilisation. Synchronizers can be used to run synchronous algorithms in asynchronous systems. Before the task is begun, all network nodes are either unaware which node will serve as the "coordinator" or leader of the task, or unable to communicate with the current coordinator. After a coordinator election algorithm has been run, however, each node throughout the network recognizes a particular, unique node as the task coordinator. For that, they need some method in order to break the symmetry among them. For example, if each node has unique and comparable identities, then the nodes can compare their identities, and decide that the node with the highest identity is the coordinator. The algorithm suggested by Gallager, Humblet, and Spira [54] for general undirected graphs has had a strong impact on the design of distributed algorithms in general, and won the Dijkstra Prize for an influential paper in distributed computing. Many other algorithms were suggested for different kind of network graphs , such as undirected rings, unidirectional rings, complete graphs, grids, directed Euler graphs, and others. A general method that decouples the issue of the graph family from the design of the coordinator election algorithm was suggested by Korach, Kutten, and Moran. The coordinator election problem is to choose a process from among a group of processes on different processors in a distributed system to act as the central coordinator. Several central coordinator election algorithms exist. A complementary research problem is studying the properties of a given distributed system. The halting problem is undecidable in the general case, and naturally understanding the behaviour of a computer network is at least as hard as understanding the behaviour of one computer. In particular, it is possible to reason about the behaviour of a network of finite-state machines. One example is telling whether a given network of interacting asynchronous and non-deterministic finite-state machines can reach a deadlock.

5: Introduction to Distributed Data Processing: Harry Katzan: www.enganchecubano.com: Books

Enter your mobile number or email address below and we'll send you a link to download the free Kindle App. Then you can start reading Kindle books on your smartphone, tablet, or computer - no Kindle device required.

I am an entrepreneur and technology agnostic. People from all walks of life have started to interact with data storages and servers as a part of their daily routine. Therefore we can say that dealing with big data in the best possible manner is becoming the main area of interest for businesses, scientists and individuals. For instance an application launched for achieving certain business goals will be more successful if it can efficiently handle the queries made by customers and serve their purpose well. The rapid growth of social media applications, cloud based systems, Internet of things and an unending spree of innovations has made it important for a developer or a data scientist to take well calculated decisions while launching, upgrading or troubleshooting an enterprise application. Although it has been widely accepted and understood that using a modular approach to build an application has multiple advantages and long term benefits, the pursuit for selecting the right data processing architecture still keeps putting question marks in front of many proposals related to existing and upcoming enterprise software. Lambda Architecture Lambda architecture is a data processing technique that is capable of dealing with huge amount of data in an efficient manner. The efficiency of this architecture becomes evident in the form of increased throughput, reduced latency and negligible errors. While we mention data processing we basically use this term to represent high throughput, low latency and aiming for near-real-time applications. Which also would allow the developers to define delta rules in the form of code logic or natural language processing NLP in event-based data processing models to achieve robustness, automation and efficiency and improve the data quality. Moreover, any change in the state of data is an event to the system and as a matter of fact it is possible to give a command, queried or expected to carry out delta procedures as a response to the events on the fly. Event sourcing is a concept of using the events to make prediction as well as storing the changes in a system on the real time basis a change of state of a system, an update in the databases or an event can be understood as a change. For instance if someone interact with a web page or a social network profile, the events like page view, likes or Add as a Friend request etcâ€¦ are triggering events that can be processed or enriched and the data stored in a database. Data processing deals with the event streams and most of the enterprise software that follow the Domain Driven Design use the stream processing method to predict updates for the basic model and store the distinct events that serve as a source for predictions in a live data system. To handle numerous events occurring in a system or delta processing, Lambda architecture enabling data processing by introducing three distinct layers. At every instance it is fed to the batch layer and speed layer simultaneously. Any new data stream that comes to batch layer of the data system is computed and processed on top of a Data Lake. When data gets stored in the data lake using databases such as in memory databases or long term persistent one like NoSQL based storages batch layer uses it to process the data using MapReduce or utilizing machine-learning ML to make predictions for the upcoming batch views. The data streams processed in the batch layer result in updating delta process or MapReduce or machine learning model which is further used by the stream layer to process the new data fed to it. Speed layer provides the outputs on the basis enrichment process and supports the serving layer to reduce the latency in responding the queries. As obvious from its name the speed layer has low latency because it deals with the real time data only and has less computational load. Here is a basic diagram of what Lambda Architecture model would look like: The symbols used in this equation are known as Lambda and the name for the Lambda architecture is also coined from the same equation. This function is widely known to those who are familiar with tidbits of big data analysis. Applications of Lambda Architecture Lambda architecture can be deployed for those data processing enterprise models where: User queries are required to be served on ad-hoc basis using the immutable data storage. Quick responses are required and system should be capable of handling various updates in the form of new data streams. None of the stored records shall be erased and it should allow addition of updates and new data to the database. Lambda architecture can be considered as near real-time data processing architecture. As mentioned above, it can withstand the faults as well as allows scalability. It uses the functions of batch layer and stream layer and keeps adding new data to the main storage while ensuring that the existing data will remain intact. Companies like Twitter, Netflix, and Yahoo are using this architecture to meet the quality of service standards. Pros and Cons of Lambda Architecture Pros Batch layer of Lambda architecture manages historical data with the fault tolerant distributed storage which ensures low possibility of errors even if the system crashes. It is a good balance of speed and reliability. Fault tolerant and scalable architecture for data processing. Cons It can result in coding overhead due to involvement of comprehensive processing. Re-processes every batch cycle which is not beneficial in certain scenarios. A data modeled with Lambda architecture is difficult to migrate or reorganize. Kappa Architecture In Jay Kreps started a discussion where he pointed out some discrepancies of Lambda architecture that further led the big data world to another alternate architecture that used less code resource and was capable of performing well in certain enterprise scenarios where using multi layered Lambda architecture seemed like extravagance. Kappa Architecture cannot be taken as a substitute of Lambda architecture on the contrary it should be seen as an alternative to be used in those circumstances where active performance of batch layer is not necessary for meeting the standard quality of service. This architecture finds its applications in real-time processing of distinct events. Here is a basic diagram for the Kappa architecture that shows two layers system of operation for this data processing architecture. It also signifies that that the stream processing occurs on the speed layer in kappa architecture. Applications of Kappa architecture Some variants of social network applications, devices connected to a cloud based monitoring system, Internet of things IoT use an optimized version of Lambda architecture which mainly uses the services of speed layer combined with streaming layer to process the data over the data lake. Kappa architecture can be deployed for those data processing enterprise models where: Multiple data events or queries are logged in a queue to be catered against a distributed file system storage or history. The order of the events and queries is not predetermined. Stream processing platforms can interact with database at any time. It is resilient and highly available as handling Terabytes of storage is required for each node of the system to support replication. The above mentioned data scenarios are handled by exhausting Apache Kafka which is extremely fast, fault tolerant and horizontally scalable. It allows a better mechanism for governing the data-streams. A balanced control on the stream processors and databases makes it possible for the applications to perform as per expectations. Kafka retains the ordered data for longer durations and caters the analogous queries by linking them to the appropriate position of the retained log. LinkedIn and some other applications use this flavor of big data processing and reap the benefit of retaining large amount of data to cater those queries that are mere replica of each other. Re-processing is required only when the code changes. It can be deployed with fixed memory. It can be used for horizontally scalable systems. Fewer resources are required as the machine learning is being done on the real time basis. Cons Absence of batch layer might result in errors during data processing or while updating the database that requires having an exception manager to reprocess the data or reconciliation. Conclusion In short the choice between Lambda and Kappa architectures seems like a tradeoff. On the other hand if you want to deploy big data architecture by using less expensive hardware and require it to deal effectively on the basis of unique events occurring on the runtime then select the Kappa architecture for your real-time data processing needs.

6: A brief introduction to two data processing architectures â€” Lambda and Kappa for Big Data

Introduction to Distributed Database Management Systems (Distributed DBMSs) Database technology has taken us from a paradigm of data processing in which each application defined and maintained its own data, to one in which data is defined and administered centrally.

The benefits of MapReduce programming 9. How I failed at designing distributed processing Once, while working on an eDiscovery system, before Hadoop was born, I had to scale up my computations: I chose to install JBoss on every machine, only to use its JMS messaging, and my computers were talking to each other through that. It was working, but it had its drawbacks: It had a concept of master and workers, and the master was dividing the job into tasks for the workers, but this preparation, which happened at the start of the job, took a long time. The system was not stable: If a worker went down, he stopped working, that is, he did not pick up more work, but the work left undone was in an unknown stage. All of the data resided on a central file server. Had my system worked properly, I would have discovered other problems, which I did not get far enough to see: IO and network bottlenecks. That was quite upsetting. I started having dreams about stacks of Linux servers, piled one upon another. Then I read about the Fallacies of distributed computing and realized that I had violated all of them. I will trust you that did not cheat by looking ahead. Whether you do this or not, looking at the MapReduce solution gives you an appreciation of how much it provides. MapReduce has a master and workers, but it is not all push or pull, rather, the work is a collaborative effort between them. The master assigns a work portion to the next available worker; thus, no work portion is forgotten or left unfinished. Workers send periodic heartbeats to the master. If the worker is silent for a period of time usually 10 minutes , then the master presumes this worker crashed and assigns its work to another worker. The master also cleans up the unfinished portion of the crashed worker. All of the data resides in HDFS, which avoids the central server concept, with its limitations on concurrent access and on size. MapReduce never updates data, rather, it writes new output instead. This is one of the features of functional programming, and it avoids update lockups. MapReduce is network and rack aware, and it optimizes the network traffic. How MapReduce really does it In the previous section I have shown how MapReduce resolves the common instability problems found in homegrown distributed systems. But I really just hinted at it, so now let us explain this in a little more detail. Masters and slaves Nobody likes to be a slave, and up until now we avoided this terminology, calling them workers. In that, we followed the remark from the movie Big Lebowski": MapReduce has a master and slaves, and they collaborate on getting the work done. The master is listed in the "masters" configuration file, and the slaves are listed in the "slaves", and in this way they know about each other. Furthermore, to be a real "master", the node must run a daemon called the "Job Tracker" daemon. The slave, to be able to do its work, must run another daemon, called the "Tasktracker" daemon. The master does not divide all the work beforehand, but has an algorithm on how to assign the next portion of the work. Thus, no time is spent up front, and the job can begin right away. This division of labor, how much to give to the next Tasktracker, is called "split", and you have control over it. By default, the input file is split into chunks of about 64MB in size. About, because complete lines in the input file have to be preserved. MapReduce is stable Recall that in my system I gave the responsibility for selecting the next piece of work to the workers. This created two kinds of problems. When a worker crashed, nobody knew about it. Of course, the worker would mark the work as "done" after it was completed, but when it crashed, there was nobody to do this for him, so it kept hanging. You needed watchers over watchers, and so on. Another problem would be created when two overzealous workers wanted the same portion. There was a need to somehow coordinate this effort. My solution was a flag in the database, but then this database was becoming the real-time coordinator for multiple processes, and it is not very good at that. You can image multiple scenarios when this would fail. By contrast, in MapReduce the Job Tracker doles out the work. There is no contention: If a Tasktracker crashes, it stops sending heartbeats to the Job Tracker. Thus the problem of a central files server, with its limited capacity, is eliminated. Moreover, MapReduce never updates data, rather, it writes a new output instead. This is one of the principles of functional programming , and it avoids update lockups. It also avoids the need to coordinate multiple processes writing to the same file; instead, each Reducer writes to its own output file in an HDFS directory, designated as output for the given job. In further processing, MapReduce will treat all of the files in the input directory as its input, and thus having multiple files either in the input or the output directory is no problem. MapReduce optimizes network traffic As it turns out, network bandwidth is probably the most precious and scarce resource in a distributed system and should be used with care. It is a problem which I have not seen even in my eDiscovery application, because it needs to be correct and stable before optimizing, and getting there is not an easy task. MapReduce, however, notes where the data is by using the IP address of the block of data that needs to be processed and it also knows where the Task Tracker is by using its IP address. If it can, MapReduce assigns the computation to the server which has the data locally, that is, whose IP address is the same as that of the data. If local computation is not possible, MapReduce can select the server that is at least closest to the data, so that the network traffic will go through the least number of hops. It does it by comparing the IPs, which have the distance information encoded. Naturally, servers in the same rack are considered closer to each other than servers on different racks. This property of MapReduce to follow the network configuration is called "rack awareness". You set the rack information in the configuration files and reap the benefits. They are called Mappers and Reducers. Next section will illustrate this concept. Understanding Mappers and Reducers MapReduce is not like the usual programming models we grew up with. To illustrate the MapReduce model, lets look at an example. Say there is a election in progress. People are voting at the polling places. It is a fancy way of saying they interview voters exiting the polling place and ask them how they voted. So for our problem, say we want to understand how different age groups voted. We want to understand how people aged 20s, 30s and 40s voted. We are going to divide the problem into two phases Phase one: Interview each age group and see how they voted. The following image explains this. There are few subtle things happening here: The result for one age group is not influenced by the result of other age group. So they can be processed in parallel. For example, all 20 somethings are in the group 20s. If the mapper did her job right, this would be the case. With these assumptions, the guy in bowtie can produce a result for a particular age group, indepedently. To prove the point, here are some "facts" about Jeff: Jeff Dean once failed a Turing test when he correctly identified the rd Fibonacci number in less than a second. Jeff Dean compiles and runs his code before submitting it, but only to check for compiler and CPU bugs. The speed of light in a vacuum used to be about 35 mph. Then Jeff Dean spent a weekend optimizing physics. You can read the complete article by the two Google engineers, entitled MapReduce: Simplified Data Processing on Large Clusters and decide for yourself. As you can see, it summarizes a lot of the experiences of scientists and practitioners in the design of distributed processing systems. It resolves or avoids several complications of distributed computing. It allows unlimited computations on an unlimited amount of data. And, although it looks deceptively simple, it is very powerful, with a great number of sophisticated and profitable applications written in this framework. In the other sections of this book we will introduce you to the practical aspects of MapReduce implementation. Then you will be able to see whether or not Hadoop is for you, or even invent a new framework. Keep in mind though that other developers are also busy inventing new frameworks, so hurry to read more.

7: An introduction to ACID guarantees and transaction processing - data Artisans

Distributed data processing is a computer-networking method in which multiple computers across different locations share computer-processing capability. This is in contrast to a single.

In a distributed database, there are a number of databases that may be geographically distributed all over the world. A distributed DBMS manages the distributed database in a manner so that it appears as one single database to users. In the later part of the chapter, we go on to study the factors that lead to distributed databases, its advantages and disadvantages. A distributed database is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network. Features Databases in the collection are logically interrelated with each other. Often they represent a single logical database. Data is physically stored across multiple sites. Data in each site can be managed by a DBMS independent of the other sites. The processors in the sites are connected via a network. They do not have any multiprocessor configuration. A distributed database is not a loosely connected file system. A distributed database incorporates transaction processing, but it is not synonymous with a transaction processing system. Distributed Database Management System A distributed database management system DDBMS is a centralized software system that manages a distributed database in a manner as if it were all stored in a single location. Features It is used to create, retrieve, update and delete distributed databases. It synchronizes the database periodically and provides access mechanisms by the virtue of which the distribution becomes transparent to the users. It ensures that the data modified at any site is universally updated. It is used in application areas where large volumes of data are processed and accessed by numerous users simultaneously. It is designed for heterogeneous database platforms. It maintains confidentiality and data integrity of the databases. Each unit requires its own set of local data. Thus, the overall database of the organization becomes distributed. This demands common databases or replicated databases that should be used in a synchronized manner. Distributed database systems aid both these processing by providing synchronized data. Replication of data automatically helps in data recovery if database in any site is damaged. Users can access data from other sites while the damaged site is being reconstructed. Thus, database failure may become almost inconspicuous to users. DDBMS provides a uniform functionality for using the same data among different platforms. Advantages of Distributed Databases Following are the advantages of distributed databases over centralized databases. However, in distributed databases, the work simply requires adding new computers and local data to the new site and finally connecting them to the distributed system, with no interruption in current functions. However, in distributed systems, when a component fails, the functioning of the system continues may be at a reduced performance. On the other hand, in centralized systems, all queries have to pass through the central computer for processing, which increases the response time. This is not feasible in centralized systems. Adversities of Distributed Databases Following are some of the adversities associated with distributed databases. Improper data distribution often leads to very slow response to user requests.

8: Distributed computing - Wikipedia

Distributed database systems aid both these processing by providing synchronized data. Database Recovery â˜' One of the common techniques used in DDBMS is replication of data across different sites. Replication of data automatically helps in data recovery if database in any site is damaged.

For example, the database systems described earlier in the chapter can operate over datasets that are stored across multiple machines. No single machine may contain the data necessary to respond to a query, and so communication is required to service requests. This section investigates a typical big data processing scenario in which a data set too large to be processed by a single machine is instead distributed among many machines, each of which process a portion of the dataset. To coordinate this distributed data processing, we will discuss a programming framework called MapReduce. Creating a distributed data processing application with MapReduce combines many of the ideas presented throughout this text. An application is expressed in terms of pure functions that are used to map over a large dataset and then to reduce the mapped sequences of values into a final result. Familiar concepts from functional programming are used to maximal advantage in a MapReduce program. MapReduce requires that the functions used to map and reduce the data be pure functions. In general, a program expressed only in terms of pure functions has considerable flexibility in how it is executed. Sub-expressions can be computed in arbitrary order and in parallel without affecting the final result. A MapReduce application evaluates many pure functions in parallel, reordering computations to be executed efficiently in a distributed system. The principal advantage of MapReduce is that it enforces a separation of concerns between two parts of a distributed data processing application: The map and reduce functions that process data and combine results. The communication and coordination between machines. The coordination mechanism handles many issues that arise in distributed computing, such as machine failures, network failures, and progress monitoring. While managing these issues introduces some complexity in a MapReduce application, none of that complexity is exposed to the application developer. Instead, building a MapReduce application only requires specifying the map and reduce functions in 1 above; the challenges of distributed computation are hidden via abstraction. For instance, each input may be a line of text in some vast corpus. Computation proceeds in three steps. A map function is applied to each input, which outputs zero or more intermediate key-value pairs of an arbitrary type. All intermediate key-value pairs are grouped by key, so that pairs with the same key can be reduced together. A reduce function combines the values for a given key k; it outputs zero or more values, which are each associated with k in the final output. To perform this computation, the MapReduce framework creates tasks perhaps on different machines that perform various roles in the computation. A map task applies the map function to some subset of the input data and outputs intermediate key-value pairs. A reduce task sorts and groups key-value pairs by key, then applies the reduce function to the values for each key. All communication between map and reduce tasks is handled by the framework, as is the task of grouping intermediate key-value pairs by key. In order to utilize multiple machines in a MapReduce application, multiple mappers run in parallel in a map phase, and multiple reducers run in parallel in a reduce phase. In between these phases, the sort phase groups together key-value pairs by sorting them, so that all key-value pairs with the same key are adjacent. Consider the problem of counting the vowels in a corpus of text. We can solve this problem using the MapReduce framework with an appropriate choice of map and reduce functions. The map function takes as input a line of text and outputs key-value pairs in which the key is a vowel and the value is a count. Zero counts are omitted from the output: In the following section, we will use the open-source Hadoop implementation. In this section, we develop a minimal implementation using built-in tools of the Unix operating system. The Unix operating system creates an abstraction barrier between user programs and the underlying hardware of a computer. It provides a mechanism for programs to communicate with each other, in particular by allowing one program to consume the output of another. In their seminal text on Unix programming, Kernigham and Pike assert that, ""The power of a system comes more from the relationships among programs than from the programs themselves. The input to a Unix program is an iterable object called standard input and accessed as sys. Iterating over this object yields string-valued lines of text. The output of a Unix program is called standard output and accessed as sys. The built-in print function writes a line of text to standard output. The following Unix program writes each line of its input to its output, in reverse: First, we need to tell the operating system that we have created an executable program: Input to a program can come from another program. This effect is achieved using the symbol called "pipe" which channels the output of the program before the pipe into the program after the pipe. This version has only a single map task and single reduce task, which are both Unix programs implemented in Python. We run an entire MapReduce application using the following command: This module also includes an interface to the Hadoop distributed implementation of MapReduce. Finally, assume that we have the following input file called haiku. Its streaming interface allows arbitrary Unix programs to define the map and reduce functions. Hadoop offers several advantages over our simplistic local MapReduce implementation. The first is speed: The second is fault tolerance: The third is monitoring: In order to run the vowel counting application using the provided mapreduce. For more information on the Hadoop streaming interface and use of the system, consult the Hadoop Streaming Documentation.

9: Distributed Computing: An Introduction - ExtremeTech

Distributed computing is a field of computer science that studies distributed systems. A distributed system is a system whose components are located on different networked computers, which then communicate and coordinate their actions by passing messages to one another.

BEGG Introduction to Distributed Database Management Systems Distributed DBMSs Database technology has taken us from a paradigm of data processing in which each application defined and maintained its own data, to one in which data is defined and administered centrally. During recent times, we have seen the rapid developments in network and data communication technology, epitomized by the Internet, mobile and wireless computing, intelligent devices, and grid computing. Now with the combination of these two technologies, distributed database technology may change the mode of working from centralized to decentralized. This combined technology is one of the major developments in the database systems area. In previous chapters we have concentrated on centralized database systems, that is systems with a single logical database located at one site under the control of a single DBMS. In this chapter we discuss the concepts and issues of the Distributed Database Management System DDBMS , which allows users to access not only the data at their own site but also data stored at remote sites. Introduction A major motivation behind the development of database systems is the desire to integrate the operational data of an organization and to provide controlled access to the data. Although integration and controlled access may imply centralization, this is not the intention. In fact, the development of computer networks promotes a decentralized mode of work. This decentralized approach mirrors the organizational structure of many companies, which are logically distributed into divisions, departments, projects, and so on, and physically distributed into offices, plants, factories, where each unit maintains its own operational data Date, The shareability of the data and the efficiency of data access should be improved by the development of a distributed database system that reflects this organizational structure, makes the data in all units accessible, and stores data proximate to the location where it is most frequently used. Distributed DBMSs should help resolve the islands of information problem. Databases are sometimes regarded as electronic islands that are distinct and generally inaccessible places, like remote islands. This may be a result of geographical separation, incompatible computer architectures, incompatible communication protocols, and so on. Integrating the databases into a logical whole may prevent this way of thinking. Distributed database A logically interrelated collection of shared data and a description of this data physically distributed over a computer network. Distributed DBMS The software system that permits the management of the distributed database and makes the distribution transparent to users. Each fragment is stored on one or more computers under the control of a separate DBMS, with the computers connected by a communications network. Each site is capable of independently processing user requests that require access to local data that is, each site has some degree of local autonomy and is also capable of processing data stored on other computers in the network. Users access the distributed database via applications, which are classified as those that do not require data from other sites local applications and those that do require data from other sites global applications. Distributed database management system. Thus, the fact that a distributed database is split into fragments that can be stored on different computers and perhaps replicated, should be hidden from the user. The objective of transparency is to make the distributed system appear like a centralized system. This requirement provides significant functionality for the end-user but, unfortunately, creates many additional problems that have to be handled by the DDBMS, as we discuss in Section Distributed processing It is important to make a distinction between a distributed DBMS and distributed processing. Distributed processing A centralized database that can be accessed over a computer network. If the data is centralized, even though other users may be accessing the data over the network, we do not consider this to be a distributed DBMS, simply distributed processing. We illustrate the topology of distributed processing in Figure Compare this figure, which has a central database at site 2, with Figure Parallel DBMS A DBMS running across multiple processors and disks that is designed to execute operations in parallel, whenever possible, in order to improve performance. Parallel DBMSs are again based on the premise that single processor systems can no longer meet the growing requirements for cost-effective scalability, reliability, and performance. Parallel DBMSs link multiple, smaller machines to achieve the same throughput as a single, larger machine, often with greater scalability and reliability than single-processor DBMSs. To provide multiple processors with common access to a single database, a parallel DBMS must provide for shared resource management. Shared memory is a tightly coupled architecture in which multiple processors within a single system share system memory. Known as symmetric multiprocessing SMP , this approach has become popular on platforms ranging from personal workstations that support a few microprocessors in parallel, to large RISC Reduced Instruction Set Computer based machines, all the way up to the largest mainframes. This architecture provides high-speed data access for a limited number of processors, but it is not scalable beyond about 64 processors when the interconnection network becomes a bottleneck. Shared disk is a loosely-coupled architecture optimized for applications that are inherently centralized and require high availability and performance. Each processor can access all disks directly, but each has its own private memory. Like the shared nothing architecture, the shared disk architecture eliminates the shared memory performance bottleneck. Unlike the shared nothing architecture, however, the shared disk architecture eliminates this bottleneck without introducing the overhead associated with physically partitioned data. Shared disk systems are sometimes referred to as clusters. Shared nothing, often known as massively parallel processing MPP , is a multiple processor architecture in which each processor is part of a complete system, with its own memory and disk storage. The database is partitioned among all the disks on each system associated with the database, and data is transparently available to users on all systems. This architecture is more scalable than shared memory and can easily support a large number of processors. However, performance is optimal only when requested data is stored locally. While the shared nothing definition sometimes includes distributed DBMSs, the distribution of data in a parallel DBMS is based solely on performance considerations. Further, the nodes of a DDBMS are typically geographically distributed, separately administered, and have a slower interconnection network, whereas the nodes of a parallel DBMS are typically within the same computer or within the same site. Parallel technology is typically used for very large databases possibly of the order of terabytes bytes , or systems that have to process thousands of transactions per second. These systems need access to large volumes of data and must provide timely responses to queries. A parallel DBMS can use the underlying architecture to improve the performance of complex query execution using parallel scan, join, and sort techniques that allow multiple processor nodes automatically to share the processing workload. We discuss this architecture further in Chapter 31 on data warehousing. Suffice it to note here that all the major DBMS vendors produce parallel versions of their database engines.

INTRODUCTION TO DISTRIBUTED DATA PROCESSING pdf

The Kings bastard Sql gr teradata aster paper Yeshiva fundamentalism in the Haredi community in Israel Recent advances in respiratory care for motor neuron disease Daniele Lo Coco, Santino Marchese and Albino Taking My Letters Back A day in the life of an architect Blueprints ob gyn 7th edition Crack the code worksheet The rhythm and blues story Learn malayalam in 30 days through telugu The world of the Egyptians Romanticism, realism, and the modernist turn Is it the Christians duty to fight for the faith? Discuss the distinctive features of the american novel Net system management services H-rs880-uatx manual Eers to open up files Writing : webpage design Safety Advancement for Employees Act of 1997 Proceedings of the Ninth International Conference on General Relativity and Gravitation, Jena, 14-16 July Close encounters of the ambiguous kind : when Crusaders and locals meet Sprung from Some Common Source The latest illusion Counseling and self-esteem Liability to third parties Nuclear Energy in Latin America Jennifer probst searching for disaster Short fiction by Irish women writers And of Building a Travois, 13 Rituals of the Imagination The Valmiki Ramayana Part II: Cooking with Sugar. Chapter 5: Keeping Track of the Sweet Things in Life Garfield Sobers : the greatest by Keith Sandiford. The history of Remington Firearms An Analysis of Early Military Attrition Behavior (Rand Corporation//Rand Report) Old Izaak Walton, or, Tom Moore of Fleet Street, the silver trout, and the seven sisters of Tottenham If you really loved me- Maureen the Detective 5th edition magic items How To Distinguish Between Dreams And Astral Projection