Interesting Databases

Over the last decade, the volume and velocity with which data is generated within organizations has grown exponentially. Consequently, there has been an explosion of database technologies that have been developed to address these growing data needs.

autoauto- [Interesting Databases](#interesting-databases)auto - [CAP theorem](#cap-theorem)auto - [NoSQL - Types and Examples](#nosql---types-and-examples)auto - [Apache Kafka](#apache-kafka)auto - [Architecture](#architecture)auto - [MapReduce](#mapreduce)auto - [Overview](#overview)auto - [Apache Spark](#apache-spark)autoauto

- InfluxDB
- MariaDB / MySQL MySQL is an open-source relational database management system (RDBMS). Its name is a combination of co-founder Michael Widenius' daughter, and "SQL", the abbreviation for Structured Query Language. MySQL is free and open-source software under the terms of the GNU General Public License, and is also available under a variety of proprietary licenses. MySQL was owned and sponsored by the Swedish company MySQL AB, which was bought by Sun Microsystems (now Oracle Corporation). In 2010, when Oracle acquired Sun, Widenius forked the open-source MySQL project to create MariaDB. MySQL has stand-alone clients that allow users to interact directly with a MySQL database using SQL, but more often MySQL is used with other programs to implement applications that need relational database capability. MySQL is a component of the LAMP web development software stack (and others), which is an acronym for Linux, Apache, MySQL, Perl/PHP/Python.
- MariaDB MariaDB is a community-developed, commercially supported fork of the MySQL RDBMS, intended to remain free and open-source software under the GNU GPL. MariaDB intended to maintain high compatibility with MySQL, ensuring a drop-in replacement capability with library binary parity and exact matching with MySQL APIs and commands. However, new features diverge more.
- MongoDB MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License (SSPL).
- NoSQL A NoSQL (originally referring to "non-SQL" or "non-relational") database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. Such databases have existed since the late 1960s, but the name "NoSQL" was only coined in the early 21st century, triggered by the needs of Web 2.0 companies. NoSQL databases are increasingly used in big data and real-time web applications. NoSQL systems are also sometimes called "Not only SQL" to emphasize that they may support SQL-like query languages or sit alongside SQL databases in polyglot-persistent architectures.

CAP theorem

In theoretical computer science, the CAP theorem, also named Brewer's theorem after computer scientist Eric Brewer, states that it is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:

- Consitency: every read receives the most recent write or an error
- Availability: every request receives a (non-error) response, without the guaranteee that it contains the most recent write
- Partition tolerance: the system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes When a network partition failure happens should we decide to

- cancel the operation and thus decrease the availability but ensure consistency
- proceed with the operation and thus provide availability but risk inconsistency The CAP theorem implies that in the presence of a network partition, one has to chose between consistency and availability. Note that consistency as defined in the CAP theorem is quite different from the consistency guaranteed in ACID database transactions.

ACID = Atomicity, Consistency, Isolation, and Durability

-

NoSQL - Types and Examples

There are various ways to classify NoSQL databases, with different categories and subcategories, some of which overlap. What follows is a basic classification by data model, with examples:

- Wide Column Store: Accumulo, Amazon DynamoDB, Google Bigtable, Cassandra, Scylla, HBase
- Document Store: Apache CouchDB, Couchbase, IBM Domino, MongoDB
- Key-value: Apache Ignite, BerkeleyDB, Oracle NoSQL Database, ZooKeeper
- Graph: ArangoDB, Apache Giraph
- Couchbase Couchbase, originally known as Membase, is an open-source, distributed (shared-nothing architecture) multi-model NoSQL document-oriented database software package that is optimized for interactive applications. These applications may serve many concurrent users by creating, storing, retrieving, aggregrating, manipulating and presenting data. In support of these kinds of application needs, Couchbase Server is designed to provide easy-to-scale key-value or JSON document access with low latency and hugh sustained throughput. It is designed to be clustered from a single machine to very large-scale deployments spanning many machines. https://www.couchbase.com/
- Cassandra Apache Cassandra is a free and open-source, distributed, wide column store, NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Cassandra offers robust support for clusters spanning multiple datacenters, with asynchronous masterless replication allowing low latency operations for all clients. Cassandra offers the distribution design of Amazon Dynamo with the data model of Google's Bigtable. Avinash Lakshman, one of the author's of Amazon Dynamo, and Prashant Malik initially developed Cassandra at Facebook to power the Facebook inbox search feature.

Cassandra Main Features

- Distributed
- Supports replication and multi data center replication
- Scalability

- Fault-tolerant
- Tunable consistency
- MapReduce support
- Query language
- Eventual Consistency

- Apache Hadoop Hadoop is a collection of open-source software utilities that facilitate using a network of many computers to solve problems involving massive amounts of data and computation. It provides a software framework for distributed storage and processing of big data using the MapReduce programming model. Originally designed for computer clusters built from community hardware - still the common use - it has also found use on clusters of higher-end hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common occurances and should be automatically handled by the framework. The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part which is a MapReduce programming model. Hadoop splits files into large blocks and distributes them across nodes in a cluster. It then transfers packaged code into nodes to process the data in parallel. This approach takes advantage of data locality, where nodes manipulate the data they have access to. This allows the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking. The base Apache Hadoop framework is composed of the following modules:

    - Hadoop Common – contains libraries and utilities needed by other Hadoop modules;
    - Hadoop Distributed File System (HDFS) – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster;
    - Hadoop YARN – (introduced in 2012) a platform responsible for managing computing resources in clusters and using them for scheduling users' applications;
    - Hadoop MapReduce – an implementation of the MapReduce programming model for large-scale data processing. The term Hadoop is often used for both base modules and sub-modules and also the ecosystem, or collection of additional software packages that can be installed on top of or alongside Hadoop, such as Apache Pig, Apache Hive, Apache HBase, Apache Phoenix, Apache Spark, Apache ZooKeeper, Cloudera Impala, Apache Flume, Apache Sqoop, Apache Oozie, and Apache Storm.

  Apache Hadoop's MapReduce and HDFS components were inspired by Google papers on MapReduce and Google File System.

  The Hadoop framework itself is mostly written in the Java programming language, with some native code in C and command line utilities written as shell scripts. Though MapReduce Java code is common, any programming language can be used with Hadoop Streaming to implement the map and reduce parts of the user's program. Other projects in the Hadoop ecosystem expose richer user interfaces.

- HBase = Hadoop Database

- ZooKeeper Apache ZooKeeper is a software project of the Apache Software Foundation. It is essentially a service for distributed systems offering a hierarchical key-value store, which is used to provide a distributed configuration service, synchronization service, and naming registry for large distributed

systems (see Use cases). ZooKeeper was a sub-project of Hadoop but is now a top-level Apache project in its own right.

ZooKeeper's architecture supports high availability through redundant services. The clients can thus ask another ZooKeeper leader if the first fails to answer. ZooKeeper nodes store their data in a hierarchical name space, much like a file system or a tree data structure. Clients can read from and write to the nodes and in this way have a shared configuration service. ZooKeeper can be viewed as an atomic broadcast system, through which updates are totally ordered. The ZooKeeper Atomic Broadcast (ZAB) protocol is the core of the system.

ZooKeeper is used by companies including Yelp, Rackspace, Yahoo!, Odnoklassniki, Reddit, NetApp SolidFire, Facebook, Twitter and eBay as well as open source enterprise search systems like Solr.

ZooKeeper is modeled after Google's Chubby lock service and was originally developed at Yahoo! for streamlining the processes running on big-data clusters by storing the status in local log files on the ZooKeeper servers. These servers communicate with the client machines to provide them the information. ZooKeeper was developed in order to fix the bugs that occurred while deploying distributed big-data applications.

Some of the prime features of Apache ZooKeeper are:

Reliable System: This system is very reliable as it keeps working even if a node fails. Simple Architecture: The architecture of ZooKeeper is quite simple as there is a shared hierarchical namespace which helps coordinating the processes. Fast Processing: ZooKeeper is especially fast in "read-dominant" workloads (i.e. workloads in which reads are much more common than writes). Scalable: The performance of ZooKeeper can be improved by adding nodes.

Apache Kafka

Apache Kafka is an open-source stream-processing software platform developed by the Apache Software Foundation, written in Scala and Java. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. Kafka can connect to external systems (for data import/export) via Kafka Connect and provides Kafka Streams, a Java stream processing library. Kafka uses a binary TCP-based protocol that is optimized for efficiency and relies on a "message set" abstraction that naturally groups messages together to reduce the overhead of the network roundtrip. This "leads to larger network packets, larger sequential disk operations, contiguous memory blocks [...] which allows Kafka to turn a bursty stream of random message writes into linear writes." Kafka was originally developed by LinkedIn, and was subsequently open sourced in early 2011. Graduation from the Apache Incubator occurred on 23 October 2012. Jay Kreps chose to name the software after the author Franz Kafka because it is "a system optimized for writing", and he liked Kafka's work.

Architecture

Kafka stores key-value messages that come from arbitrarily many processes called producers. The data can be partitioned into different "partitions" within different "topics". Within a partition, messages are strictly ordered by their offsets (the position of a message within a partition), and indexed and stored together with a timestamp. Other processes called "consumers" can read messages from partitions. For stream processing, Kafka offers the Streams API that allows writing Java applications that consume data from Kafka and write results back to Kafka. Apache Kafka also works with external stream processing systems such as Apache Apex, Apache Flink, Apache Spark, Apache Storm and Apache NiFi.

Kafka runs on a cluster of one or more servers (called brokers), and the partitions of all topics are distributed across the cluster nodes. Additionally, partitions are replicated to multiple brokers. This architecture allows Kafka to deliver massive streams of messages in a fault-tolerant fashion and has allowed it to replace some of the conventional messaging systems like Java Message Service (JMS), Advanced Message Queuing Protocol (AMQP), etc. Since the 0.11.0.0 release, Kafka offers transactional writes, which provide exactly-once stream processing using the Streams API.

Kafka supports two types of topics: Regular and compacted. Regular topics can be configured with a retention time or a space bound. If there are records that are older than the specified retention time or if the space bound is exceeded for a partition, Kafka is allowed to delete old data to free storage space. By default, topics are configured with a retention time of 7 days, but it's also possible to store data indefinitely. For compacted topics, records don't expire based on time or space bounds. Instead, Kafka treats later messages as updates to older message with the same key and guarantees never to delete the latest message per key. Users can delete messages entirely by writing a so-called tombstone message with null-value for a specific key.

MapReduce

MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster.

A MapReduce program is composed of a map procedure, which performs filtering and sorting (such as sorting students by first name into queues, one queue for each name), and a reduce method, which performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance.

The model is a specialization of the split-apply-combine strategy for data analysis. It is inspired by the map and reduce functions commonly used in functional programming, although their purpose in the MapReduce framework is not the same as in their original forms. The key contributions of the MapReduce framework are not the actual map and reduce functions (which, for example, resemble the 1995 Message Passing Interface standard's reduce and scatter operations), but the scalability and fault-tolerance achieved for a variety of applications by optimizing the execution engine[citation needed]. As such, a single-threaded implementation of MapReduce is usually not faster than a traditional (non-MapReduce) implementation; any gains are usually only seen with multi-threaded implementations on multi-processor hardware. The use of this model is beneficial only when the optimized distributed shuffle operation (which reduces network communication cost) and fault tolerance features of the MapReduce framework come into play. Optimizing the communication cost is essential to a good MapReduce algorithm.

MapReduce libraries have been written in many programming languages, with different levels of optimization. A popular open-source implementation that has support for distributed shuffles is part of Apache Hadoop. The name MapReduce originally referred to the proprietary Google technology, but has since been genericized. By 2014, Google was no longer using MapReduce as their primary big data processing model, and development on Apache Mahout had moved on to more capable and less disk-oriented mechanisms that incorporated full map and reduce capabilities.

Overview

MapReduce is a framework for processing parallelizable problems across large datasets using a large number of computers (nodes), collectively referred to as a cluster (if all nodes are on the same local network and use similar hardware) or a grid (if the nodes are shared across geographically and administratively distributed systems, and use more heterogeneous hardware). Processing can occur on data stored either in a filesystem (unstructured) or in a database (structured). MapReduce can take advantage of the locality of data, processing it near the place it is stored in order to minimize communication overhead.

A MapReduce framework (or system) is usually composed of three operations (or steps):

Map: each worker node applies the map function to the local data, and writes the output to a temporary storage. A master node ensures that only one copy of the redundant input data is processed. Shuffle: worker nodes redistribute data based on the output keys (produced by the map function), such that all data belonging to one key is located on the same worker node. Reduce: worker nodes now process each group of output data, per key, in parallel. MapReduce allows for the distributed processing of the map and reduction operations. Maps can be performed in parallel, provided that each mapping operation is independent of the others; in practice, this is limited by the number of independent data sources and/or the number of CPUs near each source. Similarly, a set of 'reducers' can perform the reduction phase, provided that all outputs of the map operation that share the same key are presented to the same reducer at the same time, or that the reduction function is associative. While this process often appears inefficient compared to algorithms that are more sequential (because multiple instances of the reduction process must be run), MapReduce can be applied to significantly larger datasets than a single "commodity" server can handle – a large server farm can use MapReduce to sort a petabyte of data in only a few hours. The parallelism also offers some possibility of recovering from partial failure of servers or storage during the operation: if one mapper or reducer fails, the work can be rescheduled – assuming the input data are still available.

Another way to look at MapReduce is as a 5-step parallel and distributed computation:

Prepare the Map() input – the "MapReduce system" designates Map processors, assigns the input key K1 that each processor would work on, and provides that processor with all the input data associated with that key. Run the user-provided Map() code – Map() is run exactly once for each K1 key, generating output organized by key K2. "Shuffle" the Map output to the Reduce processors – the MapReduce system designates Reduce processors, assigns the K2 key each processor should work on, and provides that processor with all the Map-generated data associated with that key. Run the user-provided Reduce() code – Reduce() is run exactly once for each K2 key produced by the Map step. Produce the final output – the MapReduce system collects all the Reduce output, and sorts it by K2 to produce the final outcome. These five steps can be logically thought of as running in sequence – each step starts only after the previous step is completed – although in practice they can be interleaved as long as the final result is not affected.

In many situations, the input data might have already been distributed ("sharded") among many different servers, in which case step 1 could sometimes be greatly simplified by assigning Map servers that would process the locally present input data. Similarly, step 3 could sometimes be sped up by assigning Reduce processors that are as close as possible to the Map-generated data they need to process.

Apache Spark

Apache Spark is an open-source distributed general-purpose cluster-computing framework. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Originally developed at the University of California, Berkeley's AMPLab, the Spark codebase was later donated to the Apache Software Foundation, which has maintained it since. Apache Spark has its architectural foundation in

the Resilient Distributed Dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way. The Dataframe API was released as an abstraction on top of the RDD, followed by the Dataset API. In Spark 1.x, the RDD was the primary application programming interface (API), but as of Spark 2.x use of the Dataset API is encouraged even though the RDD API is not deprecated. The RDD technology still underlies the Dataset API.

Spark and its RDDs were developed in 2012 in response to limitations in the MapReduce cluster computing paradigm, which forces a particular linear dataflow structure on distributed programs: MapReduce programs read input data from disk, map a function across the data, reduce the results of the map, and store reduction results on disk. Spark's RDDs function as a working set for distributed programs that offers a (deliberately) restricted form of distributed shared memory.

Spark facilitates the implementation of both iterative algorithms, which visit their data set multiple times in a loop, and interactive/exploratory data analysis, i.e., the repeated database-style querying of data. The latency of such applications may be reduced by several orders of magnitude compared to Apache Hadoop MapReduce implementation. Among the class of iterative algorithms are the training algorithms for machine learning systems, which formed the initial impetus for developing Apache Spark.

Apache Spark requires a cluster manager and a distributed storage system. For cluster management, Spark supports standalone (native Spark cluster, where you can launch a cluster either manually or use the launch scripts provided by the install package. It is also possible to run these daemons on a single machine for testing), Hadoop YARN, Apache Mesos or Kubernetes. For distributed storage, Spark can interface with a wide variety, including Alluxio, Hadoop Distributed File System (HDFS), MapR File System (MapR-FS), Cassandra, OpenStack Swift, Amazon S3, Kudu, Lustre file system, or a custom solution can be implemented. Spark also supports a pseudo-distributed local mode, usually used only for development or testing purposes, where distributed storage is not required and the local file system can be used instead; in such a scenario, Spark is run on a single machine with one executor per CPU core.