DataBases NoSql SQL Big Data

Which NoSQL database to choose and the CAP theorem.

September 28, 2017 | Vasista Polali, Founder & Head of Engineering.

One of our clients recently asked me to evaluate different database systems to include in their Big Data Infrastructure.They are a conglomerate of businesses dealing with Industrial applications and sitting on vast amounts of inter-related data running into tens of Petabytes.IT infrastructure wise,as per their own confession,they are at least half a decade behind.Five years of lag in adapting the latest software designs and technologies,depending on how you look at it can be considered a generation gap in the current Digital Age.

Hence they wanted us to start with a Study paper to recommend the tools and technologies they should embrace to put the tons of data at their disposal to a productive use.Firstly,I would like to congratulate them for taking this leap of faith into the Digital world.

Out of the set of questions and requirements provided to us for the paper was a seemingly basic yet equally tricky query.What database should we be using? Aha!, I thought,ain't this a typical question that runs in every CTO,Software Architect, Manager or a Developers mind.But where to start from? Hence,I set about writing a few words about Relational as well as NoSql databases.I hope that would at least give some food for thought and set all those brilliant minds in some direction.

Let us now look into the CAP theorem and how it's properties influence the design of database systems and finally analyze the different types of NoSQL databases available in the market currently.

CAP Theorem:
CAP stands for Consistency,Availability and Partition Tolerance(read Horizontal Scalability),the three properties that any one primarily looks for in a Database system.Since,there is no Database,till date,that can satisfy all three properties completely,the age old game of Pros and Cons,finding the right balance begins.
RDBMS Systems had a virtual monopoly across Software implementations firstly because of robust design and also due to the lack of viable alternative systems and use cases to drive the innovation for an alternative.

If we consider a Travel application, typically used to book Air tickets from one destination to another and Hotel reservations thereon,scale is an important aspect to consider.With thousands of cities across the world,hundreds of service providers and millions of Hotels and not to mention the numerous combinations that exist in packaging these together,the traffic can hit immense proportions pretty quickly somewhere in the order of five-thousand TPS(Transactions per Second) which could result in three to five hundred SQL transactions simultaneously causing severe bottlenecks is a routine.The standard approach has been to attempt vertical scaling, increasing the memory,cpu cores etc,which in turn affects the application,every time there is a need to do some kind of vertical scaling on the RDBMS we need to bring down the web application affecting availability.Horizontal scaling in the form of sharding is also done to support high frequency of transactions, but now the onus is shifted to the web application to write the data to the right shard causing a level of disruption in its own right.
For Time series data like a Travel applications the need to perform frequent writes is a must.Apart from the load from the transactions that keep updating the data,the ticket prices,hotel reservations keep changing and updating the prices for them individually and for promotions and deals is a continuous process bringing down the read performance as result.Introducing a caching layer may improve read performance to a certain extent but puts undue pressure on the application to maintain some extra functionality,even then traditional cache layers have their own limitations in reference to high volumes of transactions and large volumes of data,which would be a topic of discussion for a future blog.Schema changes,satisfying ACID and limits with vertical scaling are also other aspects that hinder scaling with RDBMS.

With advent of NoSql databases the systems have become much more elastic.With a typical NoSQL database like Cassandra for example,the capacity can be scaled horizontally agnostic of the application using it.Even if the requirement increases to fifty-thousand TPS per second,extra nodes can be added to the cluster providing extra capability with out disrupting the application.Of course,this kind of elasticity means it can be wound down as well during off seasons,thereby saving costs.

Featured Posts

Lambda Transformations proposes an Integrated Analytics Platform with MicroServices and Big Data ecosystem.
October 17, 2017

Recent Posts

Lambda Transformations proposes an Integrated Analytics Platform with MicroServices and Big Data ecosystem.
October 17, 2017

Which NoSQL database to choose and the CAP theorem.
September 28, 2017

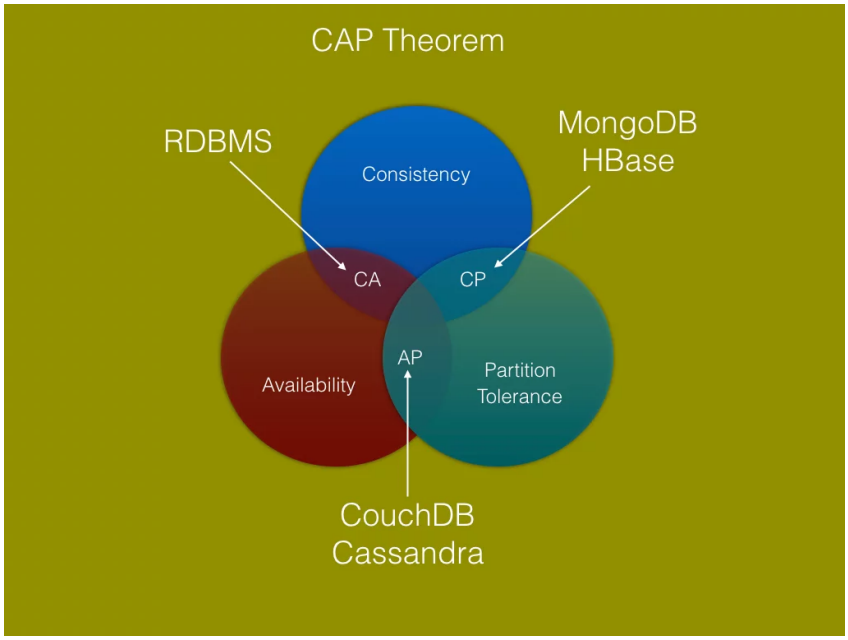
Archive

October 2017 (1)

September 2017 (1)

Search By Tags

Follow Us



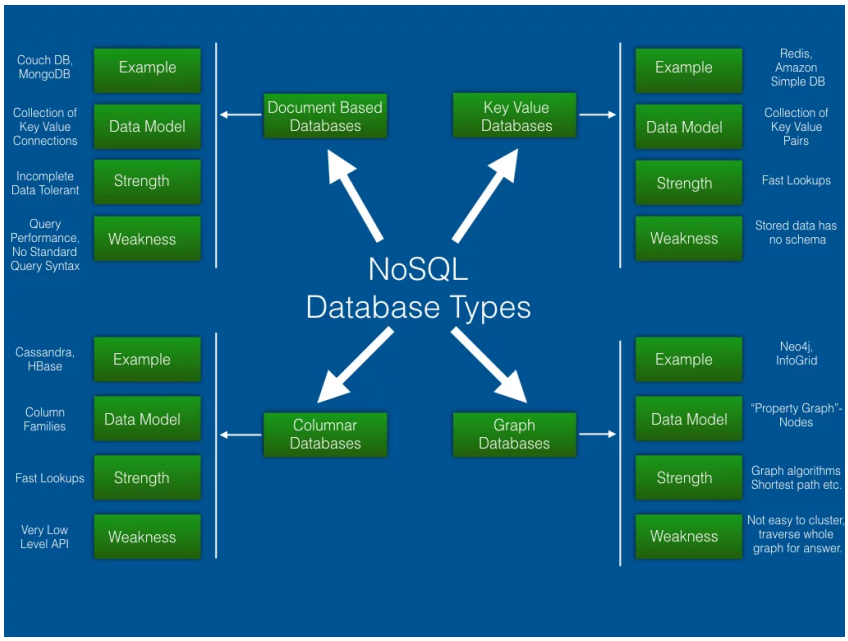
The above image depicts the quadrants that various databases fall into in relation to the CAP Theroem.

RDBMS falls into the CA quadrant.All your reads will be highly consistent as the data will not be available until the writes are properly committed.For Banking applications,relational systems are still the favorite as they provide high degree of reliability,consistency and security.

Databases like MongoDB and HBase fall in to the CP quadrant.Consistency needs a lot of control and the master slave model that these databases use give them a lot of control on how data is read and written which guarantees consistency and being a distributed database makes it easily scalable with negligible disruption if not any.But the problem lies in availability which is compromised for consistency.Here the master becomes a single point of failure without which the slaves are absolutely useless.Sure,multiple masters can be configured in the cluster introducing availability to a certain extent but at the end of the day these multiple masters are also a group of servers whose unavailability will render the salves incapacitated.
For cases where you need strong consistency like for example a Movie ticket booking application which has to be reliable and there is no scope for disparity in reads between different clients,such databases are very useful.Availability is always nice to have but can be toned down in favor of accurate and consistent writes to the data.

Next come databases like Cassandra and CouchDB that fall into AP quadrant.If you look at the use cases relating to time series data like the example we used above,strong consistency is not a big factor as opposed to availability and scaling and one can live with eventual consistency,which means multiple clients reading the same data at the same time may get different results as the writes may be affected by latency for all the replicas to be in sync in the cluster and across data centers.Say,if immediately after an update three clients are querying for the same data from three different regions,they may see different results,but eventually over a period of time all the replicas in different data centers sync up giving consistent results also called soft state.
Cassandra is one of those systems which provides what you call as a tunable or adjustable consistency where you can tune it for strong consistency but will degrade performance exponentially making it worse than RDBMS when large amounts of data are involved.Cassandra is a distributed model where all the nodes in the cluster are equal and have no single point of failure as seen in the master slave model giving it the flexibility to have a high degree of availability and at any point,depending on the replication factor,at least one node is available to service the request.
For Time series data which changes over time consistency is not important.If we go back to the Travel application example,if a customer is looking to buy an air ticket and sees it for a hundred dollars in the site and by the time he books it if it goes up or down by a couple of dollars he/she doesn't mind,but definitely expects the application to perform efficiently.The consistency part comes into play once the tickets are booked and confirmed as they should not be subjected to arbitrary changes henceforth.

Now,how does NoSql come to the rescue,there are few aspects I would like to dwell on.
Shared-nothing architecture, scale out, means nodes in the cluster are absolutely independent of each other and the data is distributed into pieces and spread across the nodes.
Non locking concurrency control mechanism.In case of RDBMS writes,typically its a three step process where it does a seek on a particular key,updates the particular object and writes it back to the database.But with NoSQL databases like Cassandra, its just one step process,its not bothered about whats existing,it just writes and the validation part to see if the update is valid or not happens in the background resulting in your writes to be very very fast.
Scalable replication and distribution.Due to the distributed nature they can be scaled horizontally to thousands of nodes and data distribution is done automatically easing the maintenance unless the user wants to control it.Apart from the writes being fast due to the one step process,reads are even faster because of the load being distributed across multiple nodes akin to ten entities doing the same job as compared to one entity handling it previously in relational systems.
Schema less design provides sparse data models,if some system does not need any column thats fine,while an other one needs ten more just add it,and yet everything can be saved at the same place,the entire data model is de-normalized and absolutely non relational and schema less which means you can have varied number of columns for the same row i.e in general terminology I can have a table with rows that have varied number of columns.For example in the travel application if a Hotel wants to advertise some new features it has put in its rooms or added any other facilities,it needs to reflect them as new columns in the table which can be added dynamically in a NoSQL database,while altering the schema hitherto was very painful in a relational system due the existing data in the table,de-populating nulls for that particular column,etc.



There are nearly 150 NoSQL databases out there for one to choose from,but there is no one solution fits all.The above image tries to split them broadly into four main categories,which are Document based,Key Value based,Columnar storage,Graph databases and look at their strengths and weaknesses.For example,Facebook uses Cassandra,Neo4j and lot of other things,but the purpose for each is different.So the key is to find the right database the suits a particular application or find the right balance between the various requirements and zero in on the one that suits your purpose aptly.

Now lets look at the most popular NoSQL databases.Cassandra,MongoDB,HBase and talk a bit about there respective strengths and weaknesses.
Cassandra is a query modeling system where one looks at the most commonly run queries on the data and models the database based on that,hence it is not suitable for dynamic querying.It is simple to setup and maintain,the administrator has to do nothing as most of the maintenance is done automatically.Scale up and Scale down is very fast in Cassandra at it provides a simple tool to perform these operations and one doesn't have to worry about re-synching,distribution of data and balancing as well as its automatically done.Cassandra has a very high velocity of random reads and writes compared to other NoSQl databases because of its columnar storage capability and distributed decentralized architecture.Flexible Sparse Wide Column requirements means that you can alter the schema on as per need basis and increase the number of columns for a specific type as and when you need.But,its only suitable for systems where secondary Index needs are less i.e all the information to serve a query is sitting in one single table and is not spread across tables,in short its suitable for non-group by kind of systems.Cassandra shows good performance on reads and writes even with Peta bytes and Peta Bytes of data as compared to MongoDB and HBase which start deteriorating in the high GB's.

Tags: NoSql DataBase Which DataBase to use Big Data CAP Theorem

HBase is one of the best systems used for map-reduce kind of models,hence was built to work well with Hadoop and is well optimized for reads and is well suited for Range based scans.The data in

n l he ge ch ıer and block size can also be
da ng oc or your range falls in a single memc

block the reads are pretty fast. Map reduce is done by picking up a block of data and running map and reduce operations on it hence these kind of databases are not suitable for querying small amounts of data.Facebook uses it for messenger to store user status,photos and chat messages.Strong consistency is provided because of the master slave model in the form of controller node and worker nodes and for the very same reason it suffers from availability issues.Full table scans are a no-no in HBase as the data is saved across the blocks and scanning all the blocks and collecting data would dip the performance considerably.

MongoDB can be safely called a RDBMS replacement for web applications,with the option of high scalability added in.Its a suitable system for semi-structured content management and real time analytics and also really good for high speed logging. Craigslist uses MongoDB to archive posts and the number currently stands at two billion.Foursquare uses Mongo as well,for storing venues and check-ins.So if some one is looking for a simple transition from the relational world to NoSQL then MongoDB would be the ideal choice,but should be mindful that it should not be used with high transactional applications where there is foreign key constraints,lot of joins etc.

For our Travel application one can use Casandra for data before the actual booking takes place as it is time series in nature and for all things beyond the booking MongoDB can be used for consistency.

So whats the conclusion,which database is the best? Like for all things in life there is no perfect answer,but it should be a fine balance between the pressing requirements driving the system design and the individual characteristics of each database system.