Database Systems
NoSQL

Observations or Existing Problems:-

There are many technical limitations of majority of Relational Database Implementations because of which they are not able to meet the requirements of big companies like Amazon Dynamo, Facebook Cassandra, Yahoo! PNUTS, etc. The three major issues with SQL are :-
Unprecedented transaction volumes, expectations of Low-latency access to massive datasets and nearly perfect service availability while operating in an unreliable environment.

Initially companies tried many approaches for fixing the problem such as they tried to scale the existing relational solutions by: 1) simplifying their database schema, 2) de-normalizing the schema, 3) relaxing durability, 4) referential integrity, 5) Introducing various query caching layers, 6) separating read-only from write-dedicated replicas and 7) Data partitioning.
None of the mentioned solutions are able to resolve the core limitations of SQL. Thus, needed a solution that would scale, be resilient, and be operationally efficient.

SQL databases are also not best fit for hierarchical data storage. As SQL databases are vertically scalable. Thus, amount of data that can be stored mainly depends on the Physical memory of the system. As the amount of data increases, it is no longer that easy to use. We can say, they don't scale out very well in a distributed system. And all of the big sites like Google, Yahoo, Facebook and Amazon, etc have lots of data and all of them store the data in distributed systems.

Key contributions and basic idea of key contributions

Sql databases emphasizes on ACID properties where A stands for Atomicity, C stands for Consistency, I stands for Isolation and D stands for Durability. Whereas NoSql database follows the Brewer's CAP theorem where C stands for Consistency, A stands for Availability and P stands for Partition tolerance. And according to the theorem we can not have all the three at the same time.

For achieving scalability it was necessary to relax or redefine some of the fundamental elements of transactions (ACID), particularly consistency and durability. NoSql relaxes the idea of complete consistency to something called “eventual consistency”. In eventual consistency there will be an unbounded delay in propagating a change made on one system to all the other copies. Thus, in the case of eventual consistency all systems will not be totally consistent at all times. But, it is sure that they will reach the consistent state after some short period of time.

Nosql uses optimistic concurrency control while sql uses pessimistic concurrency control. In pessimistic concurrency control to provide a Strict Serializable transaction outcome, Read/Write locks are used. Locks are known to create contention especially between long read transactions and update transactions.

MVCC aims at solving the problem by keeping multiples copies of each data item. In this way, each user connected to the database sees a *snapshot* of the database at a particular instant in time. Any changes made by a writer will not be seen by other users of the database until the changes have been completed.

In both eventual consistency and MVCC, the data that will be read in a transaction will be consistent but may not be up-to-date.

SQL databases are table based databases whereas NoSQL databases can be classified on the basis of way of storing data as

1) document based
2) key-value pairs
3) graph databases.

This means that SQL databases represent data in form of tables which consists of n number of rows of data whereas NoSQL databases are the collection of key-value pair, documents, graph databases which do not have standard schema definitions which it needs to stick to.

Sql ensures durability by ensuring that before a response is returned to the client, the write has been flushed to disk. But as writing to disks slows down the database, there is huge impact on performance. Thus NoSql, came up with a solution of matching durability against speed. It consider update durable when it has been written into the memory of some number of systems. But, yes that could be risky if in case those systems loss power before the update is written to disk because in this case that commit would be lost. NoSql deals with this problem by ensuring that data is stored on more than one systems before returning to client. And as network latencies are much faster than disk latencies, it will improve the speed and at the same time ensure that data will not be lost due to system failure.

One of the important category of NoSQL databases is document based databases which stores data in formats that are more native to the languages and systems that interact with them. It has dynamic schema. Mongodb is one of many examples, it stores data in JSON like documents.
With the massive growth of social networking sites Neo4j has received a lots of attention. It is a graph-based NoSQL database that stores information about nodes and edges and provides simple, highly optimized interfaces to examine the connectedness of any part of the graph.

Most leading NoSql solutions are open-source, which decreases some of the risks like sponsoring company changing hands or going out of business. And one should avoid moving from one NoSql solution to another as it is very time consuming.

Generally, NoSQL database are highly preferred for large data set. In most typical situations, SQL databases are vertically scalable. We can manage increasing load by increasing the CPU, RAM, SSD, etc,

on a single server. On the other hand, NoSQL databases are horizontally scalable. We can just add few more servers easily in our NoSQL database infrastructure to handle the large traffic. Other than scalability, it also provides reliability. And it provides additional reliability, when partitions overlap, by keeping redundant copies of the same data at
multiple nodes in the system. The drawback of this is there will be some duplicated data and huge costs associated with managing consistency across these partitions.

A good NoSql solution will prove itself resilient even when operating in unfavourable or punishing environments, whereas the less mature one may lose data or stop operating when too many unpredicted conditions arise. In addition to distributing data, many NoSQL solutions also offer distributed processing, generally based on MapReduce.