

NoSQL – Factors Supporting the Adoption of Non-Relational Databases

Olli Sutinen

University of Tampere
Department of Computer Sciences
Computer Science
M.Sc. thesis
Supervisor: Marko Junkkari
December 2010

University of Tampere

Department of Computer Sciences

Computer science

Olli Sutinen: NoSQL – Factors Supporting the Adoption of Non-Relational Databases

M.Sc. thesis

November 2010

Relational databases have been around for decades and are still in use for general data storage needs. The web has created usage patterns for data storage and querying where current implementations of relational databases fit poorly. NoSQL is an umbrella term for various new data stores which emerged virtually simultaneously at the time when relational databases were the de facto standard for data storage. It is claimed that the new data stores address the changed needs better than the relational databases. The simple reason behind this phenomenon is the cost. If the systems are too slow or can't handle the load, the users will go to a competing site, and continue spending their time there watching 'wrong' advertisements. On the other hand, scaling relational databases is hard. It can be done and commercial RDBMS vendors have such systems available but it is out of reach of a startup because of the price tag included. This study reveals the reasons why many companies have found existing data storage solutions inadequate and developed new data stores.

Keywords: databases, database scalability, data models, nosql

Contents

1. Introduction.....	1
2. A retrospective.....	3
2.1. Changes in technology.....	3
2.1.1. Commodity hardware.....	4
2.2. Changes in operating environment and access patterns.....	5
2.3. New data storage markets.....	5
2.3.1. Business data processing.....	5
2.3.2. Data Warehouses.....	6
2.3.3. Text.....	6
3. Data models.....	7
3.1. Entity-relationship model.....	7
3.2. Structured Data.....	9
3.2.1. Hierarchical model.....	9
3.2.2. Network model.....	10
3.2.3. Relational model.....	11
3.2.4. Object-oriented model.....	12
3.2.5. Deductive model.....	14
3.3. Semistructured Data.....	15
3.3.1. XML.....	16
3.3.2. JSON.....	17
3.3.3. YAML.....	18
3.4. Unstructured Data.....	19
4. Features and characteristics of relational databases and relational database management systems.....	20
4.1. SQL language.....	21
4.1.1. Support for ad-hoc queries.....	21
4.2. ACID.....	22
4.3. Distributed transactions.....	22
4.4. Scalability.....	22
5. Distributed systems.....	23
5.1. CAP theorem.....	23
5.2. Yield and harvest.....	24
6. Features and characteristics of NoSQL databases.....	25
6.1. Eventual consistency.....	25
6.2. BASE.....	25

6.3. Paxos algorithm.....	26
6.4. Conflict resolution.....	26
6.4.1. Vector clocks.....	27
6.4.2. Multi-version concurrency control.....	28
6.5. Distribution and replication.....	28
6.5.1. Sharding.....	28
6.5.2. Consistent hashing.....	29
6.6. Setting fault-tolerance and optimizing for read or write operations.....	30
6.7. Map/Reduce querying.....	30
6.8. Shared nothing architecture.....	31
7. Notable implementations and business needs behind them.....	32
7.1. BigTable by Google.....	32
7.1.1. Data model.....	32
7.2. Dynamo by Amazon.....	35
7.3. Cassandra by Facebook.....	36
7.4. Summary.....	37
8. Discussion.....	38
9. Conclusion.....	40
References.....	41

1. Introduction

Relational databases are, *de facto*, the standard in data storage. They are implemented on the top of a universal data model, which can be applied to almost any kind of situation. Relational databases have offered a good mix of flexibility, performance, scalability, and compatibility in managing generic data. They also provide simplicity of development through strict consistency, which takes a lot of responsibility from application developers. That has created an "one size fits all" attitude and the selection of data store has been a choice between different relational databases such as Oracle¹, DB2², SQL Server³, MySQL⁴ or PostgreSQL⁵.

But in reality, one size doesn't fit all even in a planned economy. Technology, access patterns and operating environment have changed and created a need for specialized solutions for data storage. A number of specialized engines have emerged to address particular problems found in relational databases. The problems are listed here in an arbitrary order:

1. Scalability Relational databases scale well inside the boundaries of a single server. However, scalability needs beyond that point are hard to meet. Scalability is defined in this study as the ability to add physical computing resources to a system in order to gain better performance. This is similar to the definition by Nygard [2007]. Vertical scalability means adding the resources of a single computer. It might be more memory, faster processor or faster and larger disks. Horizontal scalability is the ability to add physical computing resources by adding more computers. In ideal case, addition of new computers provides linear increase in performance.

2. Availability Relational databases have the property to be always in a consistent state. That means refusing new write operations until the current write operation is finished. Gilbert and Lynch [2002] define

¹www.oracle.com/us/products/database/index.html

²www.ibm.com/software/data/db2/

³www.microsoft.com/sqlserver/

⁴www.mysql.com

⁵www.postgresql.org

availability as having every request received by a non-failing node to result in a response, and we follow this definition. A *node* is a physical computer which is a part of a system built of multiple computers. It is worth noting that this definition of availability applies only to non-failing nodes and doesn't make any limitations on how much time can elapse between the request and response. Usually availability requirements for software systems are set by negotiating a service level agreement. The technology-oriented people describe availability with 'nines', meaning the percentage of time when the system is available. For example, 99.99% availability or 'four nines' means 4.5 minutes of downtime a month.

3. Fault-tolerance Relational databases treat hardware failures as exceptions and special hardware is required to achieve fault-tolerance. While this is a shameless generalization and not really fair to systems such as MySQL cluster⁶ or DB2 PureScale⁷, fault-tolerance through replication is generally not a part of traditional relational database architecture. That roots from the days when hardware was really expensive and the cost of redundancy was too high. We have a vague definition for fault-tolerance. For the purposes of this study it is sufficient that a data storage software can continue operating after node failures if the number of failed nodes is significantly less than total number of nodes. This definition implies that local failures are not propagated over other nodes or entire system. While this definition takes only hardware failures into account, a reliable system should be prepared also for software failures. If a hardware failure strikes, the node usually doesn't respond at all, and that is the easy case. Software failures which cause a node to behave erroneously but continuing to create syntactically correct responses are a lot harder to detect [Lamport et al., 1982].

NoSQL⁸ is a term first used by a relational database which doesn't have SQL [Chamberlin and Boyce, 1974] interface, introduced in 1998. The term was redefined in early 2009 when an event was organized in San Francisco to discuss non-relational databases [Evans, 2009]. Today the NoSQL community describes the acronym as "Not Only SQL". One definition by Stefan Edlich is

⁶www.mysql.com/products/cluster

⁷www.ibm.com/software/data/db2/linux-unix-windows/editions-features-purescale.html

⁸http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page

“Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontal scalable. The original intention has been modern web-scale databases” [Edlich, 2009].

This study reveals the background and reasons for the fact that many companies have found existing data storage solutions inadequate and developed their own implementations. Even more businesses have adopted new open source non-relational data stores [Popescu, 2010]. A decision like this might look like a step back from the technological point of view because the data models of the new data stores are not even nearly as rich as the relational model. But when following the money, the conclusion is that in many cases availability, scalability and fault-tolerance are more important than strict consistency. While business decisions are economical and belong to the field of business economics, some knowledge of technology is required to understand the underlying issues.

This paper is organized as follows. Section 2 describes the historical background related to the development of database management systems. Section 3 describes different data models including the relational data model. Section 4 describes features and characteristics of relational databases in more detail. Distributed systems are discussed in section 5. Then some theoretical background of non-relational databases is described in section 6. Section 7 takes a look at some implementations of NoSQL databases and describes how the problems mentioned above are addressed in each particular implementation. Finally, the study is summarized and concluded in sections 8 and 9, respectively.

2. A retrospective

As stated in the introduction the world has changed and that affects also the field of data storage. Technology has changed in very fast pace, computers are used even in the poorest parts of the world⁹. Many of the poorest people can also afford a mobile phone. One explanation is that the field of technology has been enjoying a relatively low level of regulation by governments.

2.1. Changes in technology

A trend which has ended is processor cores becoming faster. Moore's law [Moore, 1965] states that the density of transistors in processors will double in

⁹www.laptop.org/en/

every two years. That has been mostly true in the last half a century, and it also reflected the clock frequency of the processor. A decade ago the trend encountered discontinuity. The typical clock frequency in manufactured processors had a peak in the first years of 21st century and have slightly declined since then. Moore's law is still valid, the density of transistors but instead of getting more out of a single core, multi-core processors are being produced. This requires software which is able to run in parallel [Sutter, 2005]. Applications including traditional relational database systems have been enjoying this progress to achieve better performance. Most software applications designed to run sequentially will need to be redesigned to take advantage of today's multi-core processors.

Processors have become faster and cheaper, main memory has become cheaper, hard disks are bigger and cheap enough to keep essentially everything. However, relative time to seek data from disk compared to getting it from main memory has increased. This has lead to solutions where the whole database is decided to be kept in memory, one example is VoltDB¹⁰.

Solid State Disks have been said to be game changers in database markets because the technology makes disks a lot faster [Whitehorn, 2009]. Oracle even has a special tuning knob in their current flagship product, Oracle Database 11g, where the user can configure the database to take advantage of solid state disks. Whether it will really set the industry upside down depends naturally on the price of solid state disks compared to hard disks.

2.1.1. Commodity hardware

A term frequently found from papers describing distributed computing is commodity hardware. Large systems don't require special hardware, instead these systems are designed to be used with clusters of commodity hardware. Scalability is achieved adding nodes to clusters and sharing load between clusters. Main reason is economic, a commodity is something you can negotiate with multiple vendors and select the one offering the best value.

A small server with disks attached directly to it has actually a better disk-to-processor ratio than large servers leading to a more balanced system in terms of processor and disk speed. That comes directly from the fact that processor performance increases faster than disk performance. Power consumption

¹⁰www.voltdb.com

increases linearly when adding new servers to a cluster, but the increase is cubical when adding processor clock frequency [Hamilton, 2007]. At a certain point it is more efficient to add more servers instead of making the existing servers faster.

2.2. Changes in operating environment and access patterns

The cost of computers has decreased continuously while personnel costs have increased, it's a major change from the 1970's in operating environment. Today personnel costs are typically the largest expense in information technology companies. You won't see technicians wearing white lab coats nursing and feeding precious computers. Instead the trend is towards 'no knobs' operations where the operators aren't even allowed to fiddle with the systems [Brewer, 2001]. System-to-administrator ratio is a way to roughly measure the operating costs of a system. That being used as measurement tells a lot of the time we are living in. The number of systems isn't even counted, only the number of people operating the systems.

In the 1970's when first relational databases were developed, organizations had a single computer and access to the computer was via a dumb terminal. In contrast, today there are systems in production which operate at global scale and are accessed through the web. The number of simultaneous users is limited only by the network bandwidth. That creates workloads which must have been almost impossible to predict in the 1970's.

2.3. New data storage markets

When a technology becomes mainstream the normal evolution is that specialized needs emerge and "one size fits all" strategy doesn't work [Kotler and Keller, 2008]. In the cellular phone market this happened in very fast pace. In the 1990s people could go to a specialized cell phone store and buy a Nokia, Motorola or Ericsson phone. The capabilities of the phones were very similar to each other. A cell phone was really just a device to make calls on the road. Today the cell phone manufacturers use a lot of money to market segmentation and positioning with different models for each market segment. Yet others, such as Apple, has only one phone model. And both strategies could be very profitable. A similar development is in progress in the data storage market.

2.3.1. Business data processing

Business data processing is the traditional database market where majority of relational databases were designed. Still the largest part of business data is

stored to relational databases. Stonebraker and others [2007a] argue that relational databases are not the best choice even in their original market. As proof of their statement, Stonebraker et al. [2007b] created a business data processing database, which was called H-Store and developed in MIT Computer Science and Artificial Intelligence laboratory. The work continued commercially as VoltDB, a database available as open-source community edition or as proprietary enterprise edition with extra bells and whistles. The new implementation was almost two orders of magnitude faster than an RDBMS when running on the same hardware and the performance of RDBMS had been tweaked for several days by a professional database administrator.

2.3.2. Data Warehouses

Data warehouses collect data from various sources, usually importing from business data processing databases. The collected data is then processed and stored for on-line analytical processing, data mining and decision support systems. Business Intelligence is a fashion concept which includes all of the previous applications.

Column-oriented databases have shown performance benefits over traditional relational database management systems (RDBMSs) which are usually row-oriented. Column-orientation simply means that the system stores data to disk based on columns instead of rows. It speeds up compression because all data that belong to same column is of same type. It also speeds up aggregation because values to aggregate are collocated in the disk. Column-orientation is a good way to shift the load from disk to processor. Processors speed is increased faster than disk speed, column-oriented databases can reduce the amount of disk I/O by heavy compression.

2.3.3. Text

Search engines are the most obvious uses of text storage and indexing. The web has been driving this trend. BigTable [Chang et al., 2008] is an implementation targeted to this market. Brewer [2004] shares experiences from building the Inktomi search engine. Informix¹¹ had cluster support and text search features back in 1990s. However, performance tests revealed that the Informix database was ten times slower than Inktomi in-house implementation for text search. They did use Informix databases but not for text indexing.

¹¹www.ibm.com/software/data/informix/

3. Data models

According to Ullman [1988] a *data model* is a mathematical formalism consisting of two parts, (1) a notation for describing data and (2) a set of operations used to manipulate the data. Understanding different data models and their capabilities is essential to make an informed decision what model to use. Some of the models described in this section are already obsolete. A particularly interesting thing is that hierarchical data model was treated as obsolete when relational model conquered the world of data storage. Today various semi-structured data models are widely used in data exchange and virtually all of these models are hierarchical.

When discussing data models a distinction between intensional and extensional level should be considered. The intensional level is the schema level. It defines the data types intensionally, by describing necessary and sufficient set of attributes which can be used to decide if an entity or object belongs to the type. In practice, however, the universe of discourse must be restricted to achieve meaningful definitions. Extensional level is the instance level. For example, in the object-oriented model intensional level contains classes while extensional level contains objects.

Data modelling is done in separate stages. A *conceptual schema* is constructed in the highest level of abstraction. It consists of concepts and their relationships to other concepts. A conceptual schema can be transformed to a *logical schema* by applying specific rules to complete the transformation. A *physical schema* is the layout of data in physical storage media. Examples of different physical shcemas are row-oriented and column-oriented databases. The logical schema can be identical but the layout of data in the physical media is different.

3.1. Entity-relationship model

Entity-relationship model [Chen, 1976] is used to create a conceptual schema of the data. It doesn't make any assumptions of the physical or logical schema. When the conceptual model is first constructed it is usually straightforward to transform it to a logical schema which can be implemented in a database management system. Chen describes the entity-relationship model as generalization to relational, network and entity-set models and each of the three models can be derived from the entity-relationship model.

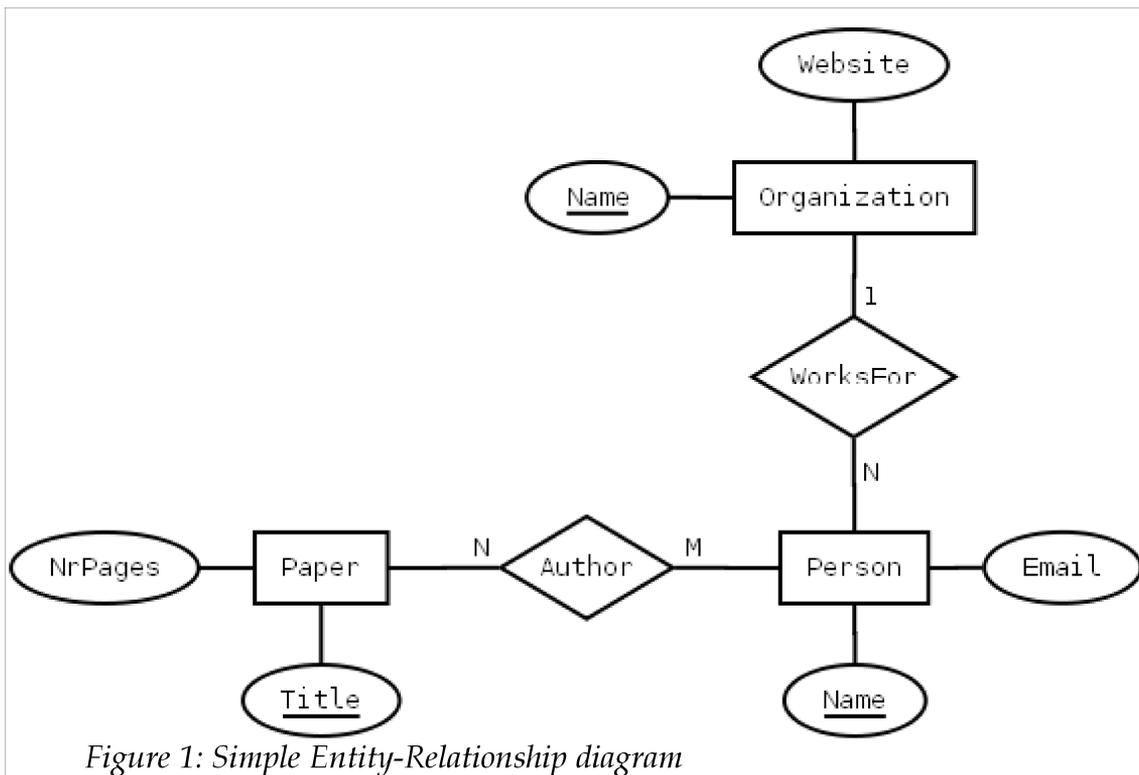


Figure 1: Simple Entity-Relationship diagram

The entity-relationship model has three basic concepts, entity, relationship and attribute. Entities can have relationships with other entities and both entities and relationships can have attributes. Entity is actually the extensional level concept while entity type is the intensional level concept in the entity-relationship model. When constructing entity-relationship diagrams, only entity types are shown.

The simple entity-relationship diagram shown in Figure 1 is used to model scientific papers. Rectangles are entity types, diamond shapes are relationships and ovals are attributes. Each paper has a title, which is selected as identifier and shown underlined in the diagram. There is also a plain attribute called NrPages which means the number of pages in a paper. A paper can have any number of authors. Similarly a person can be an author of any number of papers. That is called a many-to-many or M:N relationship and is shown in the diagram around the author relationship. The relationship between person and

organization is one-to-many which means that a person works for at most one organization while the organization can have any number of employees.

3.2. Structured Data

Structured data has strict constraints for the format of the data. Some kind of taxonomy exists and each chunk of data belongs to exactly one section defined by the taxonomy. Structured data has several advantages. One advantage is that there is no surprises in the data from the application developer's viewpoint. Another advantage is that the data storage can enforce the structure to the data in order to preserve data quality and usefulness. The downside is the lack of flexibility.

3.2.1. Hierarchical model

Data is organized into a tree structure in the hierarchical model [Ullman, 1988]. The hierarchical model contains *record types* and a record type may have any number of attributes which are called *fields*. In the extensional level there is one special record, the root record which doesn't have a parent record. All other records have a parent and any number of child records. The hierarchical model doesn't directly support many-to-many relationships and in the implementations this restriction is bypassed with a concept of virtual record. A virtual record is simply a pointer or link to a real record.

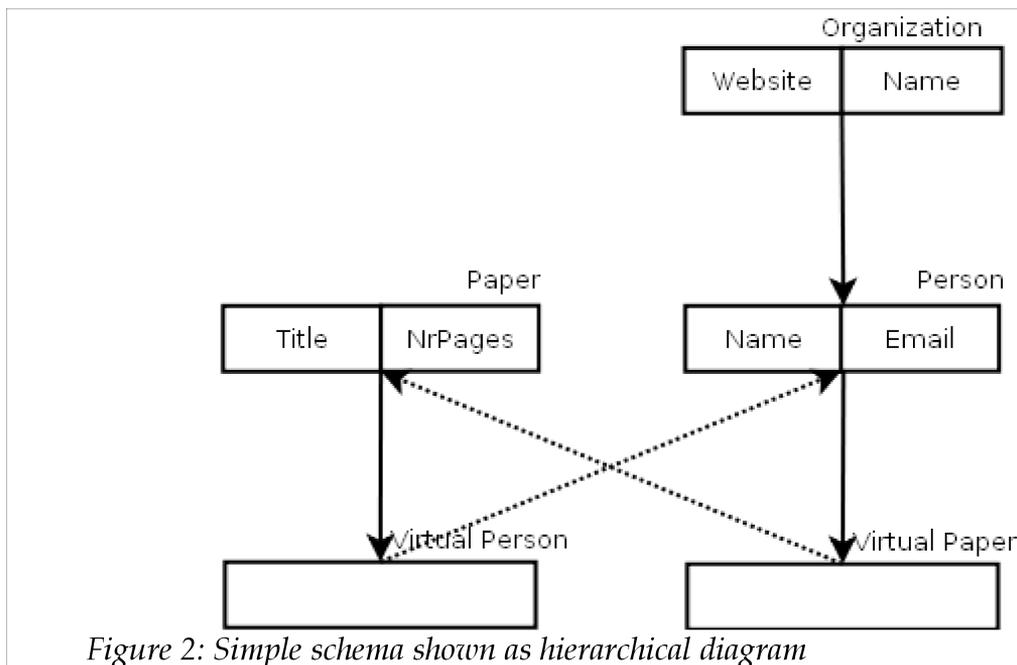


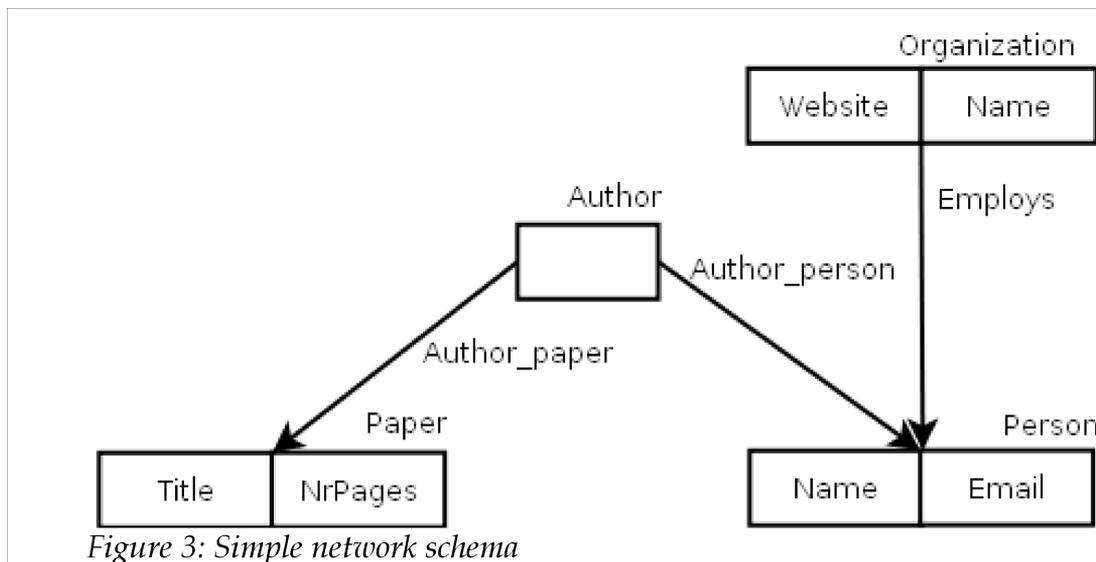
Figure 2: Simple schema shown as hierarchical diagram

The ER example shown in Figure 1 is converted to the hierarchical diagram in Figure 2. Because of the limitations of hierarchical model, two separate tree structures are formed. An organization is shown as a root node which can have any number of persons as employees. Persons can have any numbers of papers, but the record is represented by a virtual paper, which contains a pointer to a real paper record. The paper record is a root node of another tree structure having any number of persons as authors. Those persons are again represented by a virtual person record which points to the real person record.

3.2.2. Network model

The network model [Ullman, 1988] has similar concepts as the hierarchical model. Data is represented as a collection of records and each record consists of any number of fields. Instead of having hierarchical parent or child records, each record may have any number of links to predecessor and successor records. Links represent a one-to-many relationship between records. Links have names to make the links of the same record identifiable. The model forms a directed graph structure which may contain cycles.

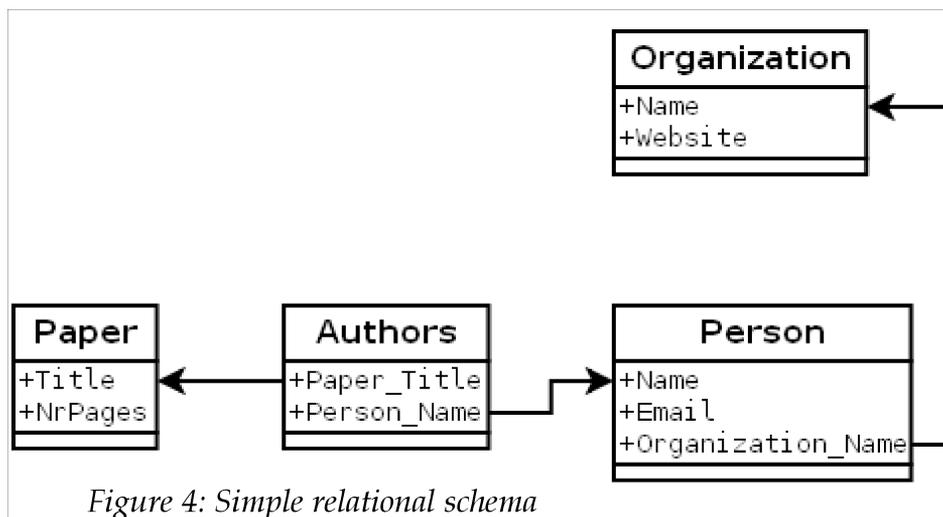
Figure 3 contains the scientific paper schema converted to network database diagram. An organization can employ any number of persons. Network model supports record types with no fields at all, because a record is more than the field values. The Author record type exists only because many-to-many relationships must be handled by an intermediate record which has links to the records attending to the relationship.



3.2.3. Relational model

The relational model [Codd, 1970] was among the first attempts to use abstraction as method for managing software complexity. It decoupled the physical storage of data from its logical structure. The hierarchical and network data models contain pointers or links to physical addresses which ties the models to the implementation.

Relational databases have been around since the introduction of IBM System R [Astrahan et al., 1976] in 1970s inspired by Codd's paper. Relational databases maintain a collection of flat two dimensional tables. Intuitively the dimensions are called rows and columns. You can think rows as objects and columns as object properties. So the data located at a particular cell is the value of the column property for that row object. In order to avoid data duplication a normal practice is to model the problem domain over multiple tables. This process is known as database normalization. A table has a primary key column and related tables have foreign key columns where the values of these key columns are used to join the data between tables. A join combines two tables when foreign key column of one table refer to primary key column of another table. Maintaining these references in a consistent state is known as a referential integrity.



To better illustrate how the relational model treats data, Figure 4 contains the entity-relationship diagram shown in Figure 1 converted to the relational schema. The result of the conversion is four tables, Paper, Authors, Person and Organization. The arrows visualize references from foreign key field to the

reference primary-key field. The paper table has two fields, Title and NrPages. Author information is stored in a relationship table called Authors which has two foreign-key fields, one pointing to the person and other pointing to the paper. That is the way the relational model handles many-to-many relationships. The one-to-many relationship between Person and Organization is handled with foreign-key reference Organization_Name from Person to Organization.

The schema actually looks very similar to the network database diagram shown in Figure 3. However, a fundamental difference exists between the models. Relational model is value-oriented while network model is object-oriented, at least to the extent that it supports object identity [Ullman, 1988]. The relationship between person and paper is a good example. In the relational model, foreign-key fields are used instead of links.

3.2.4. Object-oriented model

Object-oriented databases were once predicted to replace relational databases as the dominant solution of structured data storage [Baker, 1992]. The breakthrough of object-oriented programming languages was reality and the new programming model wasn't actually compatible with relational model. However, object-oriented databases never grow out of marginality.

The object-oriented model is based on the notion of object. The aim is to bridge the semantic gap between real world and a data model. Each data object should have corresponding real world object, meaning that objects are extensional level concepts. Unlike the relational model, object-oriented model is not value-oriented, objects are more than collection of attribute values. The concept of object identity makes it possible to have multiple objects with identical attribute values. An object belongs to a class, which is the intensional level concept. A class contains the definition of object attributes and operations. Operations are called methods in the object-oriented model. Classes are organized in hierarchies. Some implementations allow only single inheritance while others may allow multiple inheritance. An object may inherit attributes and methods. Methods and attributes may be defined as private or public. Private methods are visible only to the object itself, public methods are visible also to other objects. Methods are called by sending messages. If the method is defined in the object receiving message, it will execute the method, otherwise it will pass the message to superclass, i.e. the next direct ancestor class.

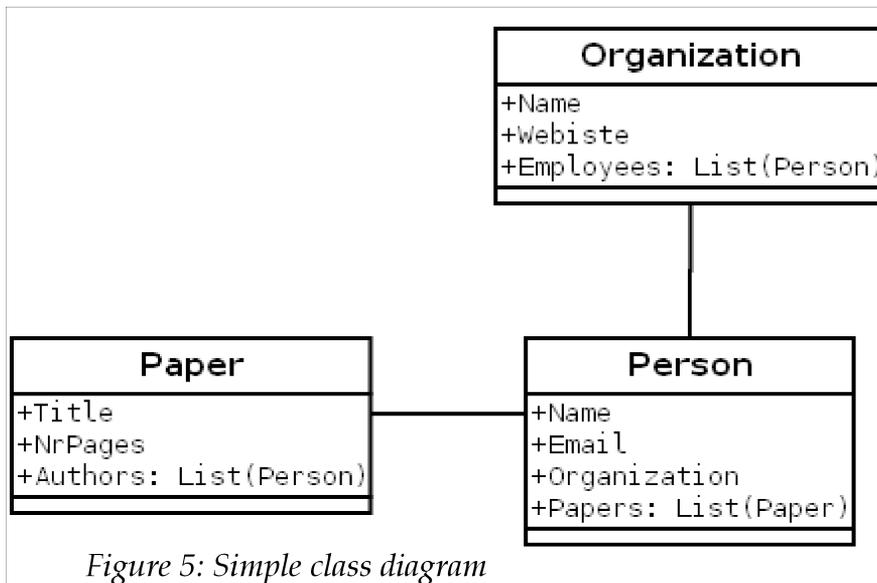


Figure 5 shows the example schema in an object-oriented way as a class diagram. Because the relationships are represented as references to objects, no relationship classes are present as was the case with the relational model. A person has simply a list of papers she has authored. Each item in the list is a reference to an object of type paper. The way of referencing is simple, but it is not bidirectional in nature. We have to define a similar list in the paper class if there is a need to find the author when the paper object is known. So, the relational model has bidirectional foreign-key references, but the object-oriented model has one-way object references. For bidirectional manipulation, inverse references must be defined. Unified modelling language [Rumbaugh et al. 1999] also has a diagram for extensional level modelling, called object diagram. That is meant to be used to model the state of a system at a particular moment.

The real reason why object oriented databases have never been in a mainstream use might have something to do with the fact that the object model combines behavioural aspects to data. Each data type can have its own set of supported operations, and the number of data types is infinite. Sharing objects in the sense that an object contains both data and the set of legal operations requires both sides of the data transmission to know the data type and the supported operations. It requires all the type definitions to be somehow accessible to everybody and systems which know how to use the data.

Compare all the trouble described above to the value-oriented relational model. It has a limited set of universally known data types. All the processes can use the same data and apply its own operations to the data. Then the result of the operations are usually another set of data which can be stored back to the database. After that yet another process, which knows nothing about the operations the previous process has done, can take the same data and use it as input to its own operations. In reality, software development is very much integration of different systems to work together. Relational databases have served well in this use case. Object-orientation is a step backwards in the data interchange and the reality has fortunately prevented object-oriented databases to become mainstream.

Another thing worth mentioning is the interoperability of unix system tools. All the tools take in ASCII text, do some kind of transformations, and output the result as ASCII text. Unix system tools work very well, because the integration of different tools is straightforward. Prolog has good means for integration, too. Prolog programmers can use top-down approach to software development because they have very powerful yet simple glue at their disposal. There is just a handful of types Prolog supports but arbitrary data structures can be defined by combining them properly. When it is time to compose the big system from parts, the data interchange usually works as expected because it is done using simple types. This is the area where the object-oriented model fails, developers use their time creating such oxymorons as data transfer objects [North, 2010] and doing type conversions between them and 'real' objects while they should be doing productive work.

Software developers found that combining object-oriented programming with relational databases is hard and error-prone so object-relational mappers were introduced. It was an attempt to hide the fact that the underlying database is relational. That mostly works well, but when a system becomes more complex the generalizations of object-relational mappers start to constrain development.

3.2.5. Deductive model

The deductive model [Elmasri and Navathe, 2004] is an extension of the relational model. Deductive databases have similar features as logic programming languages. Data is stored extensionally as facts or intensionally as rules. Rules can be applied to facts in runtime to deductively create new knowledge of the data.

3.3. Semistructured Data

A property that differentiates structured data from semi-structured or unstructured data is that the intensional level, the schema or form of data, is separated from the data itself. Semistructured data typically has no predefined schema. Many new non-relational databases are document databases and classified as containing semistructured data. A document database stores objects as self-contained documents. It is actually a lot easier for a typical user to understand the relationships in the database when it contains self-contained documents than normalized database tables. Some redundancy exists in document databases that can be avoided in relational databases, but the redundancy is justified by the ability to store the self-contained documents in loosely-coupled manner across a number of computers.

```

<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="papers">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="paper" type="paperInfo" />
        <xs:element name="person" type="personInfo"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="paperInfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="paperID" type="xs:ID"/>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="nrPages" type="xs:positiveInteger"/>
        <xs:element name="authorIDREF" type="xs:IDREF"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="personInfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="personID" type="xs:ID"/>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="email" type="xs:string"/>
        <xs:element name="organization" type="organizationInfo"/>
        <xs:element name="paperIDREF" type="xs:IDREF"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="organizationInfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="website" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

Listing 1: The example converted to XML schema

XML is an acronym of eXtensible Markup Language and it is defined by World Wide Web Consortium [W3C, 2008]. XML has become the de facto standard for data interchange over the internet. XML data can be queried in data-oriented or document-oriented way. Data-oriented query over XML data requires that the structure of the document is known. Document-oriented query is similar to the keyword search used in information retrieval [Junkkari, 2007].

Schema validation methods exist for XML such as document type declaration (DTD) and XML Schema Definition (XSD). Semi-structured XML data follows the hierarchical model, a document has single root element and each subsequent element has any number of child elements. The implementation of XML has a similar feature as the virtual record in hierarchical implementations. XML elements can reference to other elements in order to avoid duplication.

Listing 1 shows the aforementioned simple schema converted to XML Schema Definition. The above XML schema uses the default W3C namespace with "xs" prefix. The "xs:ID" is similar to primary key in relational databases and "xs:IDREF" is similar to foreign keys. That way a many-to-many relationship can be defined without duplicating data.

3.3.2. JSON

JSON is an acronym for JavaScript Object Notation. Many of the new NoSQL document databases use JSON as the document notation. It was introduced as lightweight alternative to XML. The definition is done in the true spirit of the Internet as RFC 4627 [Crockford, 2006].

JSON Schema is a similar schema definition language as the XML schema [Zyp and Court, 2010]. JSON Schema defines the structure of JSON data. Listing 2 shows an example data based on the earlier paper example. Instead of showing the schema definition, Listing 2 is in the instance level. An object is delimited by curly braces and can have any number of key-value pairs. The example document has a nested structure with three levels. To describe the relationship from the organizations viewpoint, we should create two nested documents with duplicated entries. The listing, however, shows the data only from the viewpoint of papers, it is justified by the fact that we are interested in the organizations only in the context of scientific papers. This kind of discrimination happens in modelling, weak entities exist in a model only

because they are related to other entities and existence of the related entity is justified by the domain.

```
{
  "Paper" :
  {
    "title": "NoSQL - Factors Supporting the Adoption of...",
    "nrPages": 45,
    "author":
    {
      "name": "Olli Sutinen",
      "email": "olli.sutinen@uta.fi",
      "organization":
      {
        "name": "University of Tampere",
        "website": "www.uta.fi"
      }
    }
  }
}
```

Listing 2: Example JSON document

3.3.3. YAML

YAML is a recursive acronym for Yaml ain't markup language [Evans, 2001]. YAML is a superset of JSON. The main differences are YAML's ability to express recursive structures and refer to earlier anchor in the same document.

```
---
paper:
  title: NoSQL - Factors Supporting the Adoption of...
  nrPages: 45
  author:
    name: Olli Sutinen
    email: olli.sutinen@uta.fi
    organization:
      name: University of Tampere
      website: www.uta.fi
  ...
```

Listing 3: Example YAML document

The data shown in Listing 2 is directly converted to YAML in Listing 3. The hierarchy is defined by the indentation level.

```
---
organization: &id01
  name: University of Tampere
  website: www.uta.fi

person: &id11
  name: Olli Sutinen
  email: olli.sutinen@uta.fi
  organization: *id01

paper:
  title: NoSQL - Factors Supporting the Adoption of...
  nrPages: 45
  author: *id11
...
```

Listing 4: YAML document with references to earlier anchors

To understand how to refer to an earlier anchor in the document another example is shown as Listing 4 where the document structure is modified. This notation avoids duplication by referring to earlier defined items. The anchor (&) and reference (*) characters are used in the same way as primary and foreign keys in relational databases. While this saves space by avoiding duplication, the relationships aren't visible in the structure. One of the original intentions with both JSON and YAML was to make it easier for a human to write and interpret than the more verbose XML.

3.4. Unstructured Data

This category contains text documents, web pages, video and audio files. In general any identifiable structure is absent. Storing, indexing and querying unstructured data is out of scope of this study. However, search engines index all kinds of text documents to databases in structured form. Unstructured data is usually not directly usable by software systems.

4. Features and characteristics of relational databases and relational database management systems

According to Stonebraker and others [Stonebraker et al., 2007b] current major relational database vendors, both commercial and open-source have their architecture directly from IBM System R [Astrahan et al., 1976] which was the first relational database implementation. Relational databases were never designed to grow out of one server. Still relational database systems are used to support applications facing the web. The reasons why RDBMS vendors still try to sell 'one size fits all' solutions are economical. Firstly it is expensive to maintain multiple branches of code. Secondly it creates a compatibility problem because applications have to work with all the databases they will be connected to. Thirdly the salesforce gets easily confused with which solution is the best for a certain purpose. After considering the economical perspective it becomes evident that the technologically best solution is not necessarily the one that gets the biggest market share.

Baker [1992] argues that "Database research has produced a number of good results, but the relational database is not one of them". He states that Codd's relational theory simply renamed a number of existing concepts. In the 1960's applications used files with fixed-length records which were selected and merged. Baker's view is that files were renamed to relations, records renamed to rows, fields renamed to domains, and merges renamed to joins. One of Baker's arguments is performance, he states that during the 1970's and 1980's many years were wasted for optimizing relational databases to get in par with file based solutions. He was waiting for a renaissance of databases where data is accessed through pointers or hash tables. The object-oriented pointer based approach hasn't been successful yet, but access through hash tables is a key method in many new non-relational databases. This will be discussed in more detail in Section 6.

A trait of relational database management system is the lack of automated tuning aids. A skilled database administrator will outperform any automated tuning when trying to get the most out of a relational database. This issue is already discussed in Section 2. The trend is towards 'no-knobs' operations and this trait of relational databases doesn't fit to new ideas of operations-friendly applications [Hamilton, 2007].

4.1. SQL language

SQL language has been around since early 1970s. It is a declarative query language for relational databases. First commercial implementation was Oracle V2 from Relational Software, Inc. which is today known as Oracle Inc. SQL is standardized by American National Standards Institute (ANSI) and International Organization for Standardization (ISO) starting from 1986 and having several revisions since then. The latest revision has been published in 2008 as SQL:2008 [SQL, 2008]. Open-source relational databases, such as MySQL and PostgreSQL have increased the understanding of SQL among software developers.

4.1.1. Support for ad-hoc queries

Stonebraker et al. [2007a] argue that "In an OLTP world one never asks for the employees who earn more than their managers". That is a joke targeted to the textbook example of SQL queries where a recursive relationship between employees and managers is defined and the table contains person's salary. But most jokes have some truth included. If the database is a pure online transactional processing database, that kind of queries are never needed because the nature of the query is analytical. On-line analytical processing (OLAP) is a different field and the message of the joke is: use the right tool for the task. According to Stonebraker and others, traditional relational databases aren't good at OLAP applications. In an earlier paper Stonebraker et al. [2005] published analytical query performance test results where a traditional relational database was compared to newly implemented column-oriented database. The column-oriented database, c-store, was on the average 164 times faster. The development was continued commercially and the database is known as Vertica¹².

But if the single relational database is enough for the task there is no need to use separate database for analytics. By knowing only the structure of the data you it is possible to create queries that find just the needed information from the database. It might take 10 minutes to run a query but creating the SQL statement is usually fast. That power comes from the declarative nature of SQL and the expressiveness of the relational model.

¹²www.vertica.com

4.2. ACID

Traditionally all data stores strived to fulfill ACID transaction properties [Gray, 1988]. A is for atomicity. Every transaction is atomic operation meaning it either succeeds totally or fails totally. C is for consistency. After each transaction the database is in consistent state. I is for isolation. The transactions can't interfere. D is for durability, when a transaction is succeeded, all subsequent reads will see the new state.

4.3. Distributed transactions

In order to implement ACID guarantees in distributed databases, protocols have been developed to tackle the problem of coordinating the transaction in consistent manner between multiple database servers.

A two phase commit is used for distributed transactions in cluster databases. A cluster is a set of computers acting as a single system. The two phase commit consists of two phases: voting phase and decision phase. Both phases contain several steps. One node in the cluster is called the coordinator which initiates the transaction within the cluster. A happy case scenario goes as follows: (1) The coordinator sends a vote request to all nodes, (2) All nodes send their response back to the coordinator, the node can either support or oppose the suggested commit, (3) depending on the responses from other nodes, the coordinator either sends a commit message to all other nodes or if any of the other nodes opposed the commit, the coordinator sends an abort message to all nodes who are waiting for the response, i.e. voted for the commit. (4) The nodes who voted for the commit act according to the message sent by the coordinator [Bernstein et al., 1987].

Two phase commit has some weaknesses, adding nodes adds the amount of coordination messages quadratically. What may work in 10-node cluster won't be useful in 1000-node cluster. Two phase commit is vulnerable to node failures during the commit phase. Skeen and Stonebraker [1983] introduced a better protocol called three phase commit which addressed many problems by adding one more round for communication.

4.4. Scalability

As stated earlier, relational databases scale well inside a single server. The term database is widely used as a synonym of relational database as in the following example. Nygard [2007] describes the practical solution to relational database

server scaling as follows: “Database servers, for example, get very unwieldy when you try to cluster three or more redundant servers. It's better to run a beefy pair with failover.” Beefy in this context translates to lots of memory, lots of powerful processors and big and fast disks. Failover means that one of the servers is serving all requests and all data is replicated to the other server. If the first server fails, the other takes the responsibility of serving requests almost immediately. As discussed in Section 2, 'beefy' servers are expensive. This model means that a system needs at least two expensive servers but can use only one at a time because the other is waiting for a disaster before it can step in.

5. Distributed systems

The need to scale out from the single server comes from high availability requirements, need for fault-tolerance, and need for better performance. A system is distributed when it runs on at least two separate computers. Distribution is necessary but not sufficient condition to fault-tolerance.

5.1. CAP theorem

CAP, an acronym for Consistency, Availability and Partition tolerance is a term coined by Eric Brewer [2000]. The theorem states that a distributed system can't provide all three properties simultaneously. A proof of the theorem was done by Gilbert and Lynch [2002].

A storage system is consistent if after a write every requesting client is guaranteed to get the most recent value. Availability means that if a client sends a write request to the system, it will be accepted regardless of the current state of the storage system. A partition occurs when some nodes of the system can't reach the majority of nodes but still remain in running condition and clients can see them. A system can tolerate partitions if clients can't see the difference when partition happens, i.e. the system still accepts read and write requests.

The full space is useful and deciding of which properties are more important than others is a real tradeoff. Next we discuss each of the possible combinations in more detail.

- **Consistency and Availability**

Gilbert and Lynch [2002] define availability as having all non-failing nodes to respond to queries. It means that some nodes can be failing and

the system as whole still available. Brewer [2000] gives examples of this type of systems which include single-site and cluster databases. Strong consistency is provided by two phase commits. According to Stonebraker [2010] network partitions are rare and that is the case in local area networks and intra-datacenter networks. However, when the system is geographically distributed, guaranteeing continuous network connectivity is hard.

- **Availability and Partition tolerance**

A highly available database system has to give away some consistency. Having a database geographically distributed across many data centers are reported having network partitions happen. Many new non-relational databases use relaxed consistency model to achieve better availability and partition-tolerance. A truly partition-tolerant system accepts both read and write requests under a network partition.

- **Partition tolerance and Consistency**

Paxos algorithm, explained below in Section 6, fits to this space. The way network partitions is handled is that the minority partition is unavailable but majority partition can make the decision to accept a write request.

5.2. Yield and harvest

To understand characteristics of data stores it is necessary to understand the concepts of harvest and yield [Fox and Brewer, 1999]. The yield is the fraction of answered queries. In the happy scenario when all the computing power is usable and available, yield is the same as the full capacity. If one out of ten connections is dropped, yield is 90%. This is not equal to uptime although very close. If the system is down one second in peak time and one second in off-peak time, both events will reduce uptime equally but affect on yield is different. The amount of queries which won't get an answer can be several orders of magnitude bigger during peak time.

The harvest is the fraction of the complete result. Some data may be missing due to failures. If the full data set is always returned, harvest is 100%. The majority of software developers know how relational databases work and the concept of harvest is unnecessary with relational databases, 100% harvest is taken as granted. In other words relational databases can't reduce harvest, either the query returns or doesn't return but result set is always full. A more illuminating example is a search engine. Consider a situation when user wants to search for 'scalable database systems' and the indexes of those words are in

separate machines. When one of the machines is down, the query would get results from two thirds of the complete dataset. The result will most likely be still useful to the user while harvest is dropped to 66%.

6. Features and characteristics of NoSQL databases

NoSQL databases differs from relational databases in several ways. Most of the ideas found from the new NoSQL databases aren't new. For example, BerkeleyDB [Olson et al., 1999] has been around for a long time and it is a key-value database. Another example is Lotus Notes¹³ which contains a document database and has a long history. After all the combinations of ideas and particularly the implementations are new. This section describes some theoretical background and core concepts used in distributed new non-relational databases.

6.1. Eventual consistency

Given a large enough distributed system, the probability of network partitions is close to 100%. If the system has to tolerate network partitions, both availability and consistency can't be achieved. Strong consistency means that after a write operation all subsequent read operations are guaranteed to get the most recent value. Eventual consistency is a weak consistency model where all subsequent reads aren't guaranteed to get the most recent value. If no new writes are done to the particular object, the most recent value will be eventually propagated to all replicas and the database will be eventually in consistent state. The time between a write operation and the event when all replicas have most recent value is called the inconsistency window. [Vogels, 2008]

6.2. BASE

BASE is an acronym from Basically Available, Soft state and Eventual consistency [Prichett, 2008]. Basically available means that most data is available most of the time. In case of failure some data may not be available but a disaster is required to get everything down. Soft state means that database is not always up to date. In terms of harvest and yield, BASE is trying to reach 100% yield with the cost of reduced harvest while ACID always reaches for 100% harvest. In this sense BASE is the absolute opposite of ACID.

¹³www.ibm.com/software/lotus/products/notes/

6.3. Paxos algorithm

Paxos is a consensus algorithm by Leslie Lamport [1998]. It was first published in 1998 but was criticized as being hard to understand. As response to critics Lamport described it in plain English in 2001 [Lamport, 2001]. The paxos algorithm is democratic decision making algorithm in the sense that it requires majority of decision-makers to 'vote' for the decision in order to pass the decision. In a system implementing paxos algorithm there is three roles for nodes: proposers, acceptors, and learners. Each node can have all roles or any combination, but only one node can be the proposer at any time. The proposer tries to get the majority accept a proposed value. If the majority of acceptors accept the proposed value it is a decision, and observed by the learners.

In the big picture, Paxos looks very much like the two phase commit. The proposer sends a prepare message with the proposal number n to all acceptors. When the acceptors have acknowledged their decision to accept a proposal, the proposer sends an accept message to the acceptors. Finally, the acceptors send to the proposer a message which indicates either the success or failure of the accept message. Once the majority of acceptors have accepted the value and informed the proposer, the protocol terminates. This was the happy case scenario, the details of handling conflicting proposals are lengthy to describe.

Compared to two phase commit, Paxos has some advantages under asynchronous messaging and failures. Firstly, messages are ordered so that a total ordering is possible. That makes it possible to reason which proposal to accept. Secondly majority decisions make it possible to accept new writes under node failures. This is different from the two phase commit, where write requests were committed only if every node agreed to do so. This leads to the blocking characteristics of the two phase commit, where a single failed node could lead to not accepting new writes at all.

Werner Vogels [2008] has criticized the database systems that implement paxos being unusable for the needs of Amazon because strong consistency guarantees under failures are possible only if not accepting writes.

6.4. Conflict resolution

In a system where multiple writes can happen simultaneously in many nodes, conflicting writes can happen. Distributed transactions solve this problem by not accepting new writes if there is already one in progress. If the availability

requirements are tight enough, distributed transactions are not an option. Vector clocks are a way to recognize when multiple versions of data are conflicting. Multi-version concurrency control helps if different versions of data can't be automatically merged.

6.4.1. Vector clocks

In distributed databases where multiple nodes may modify the same data, some method for data versioning and conflict detection are needed. Dynamo and Cassandra have implemented vector clocks [Lamport, 1978] for versioning and conflict detection. The concept of a vector clock is simple. Every node participating in distributed system adds a tag to all pieces of data along with a timestamp, that makes it possible to reason which version of a piece of data is latest or if the versions are in conflict.

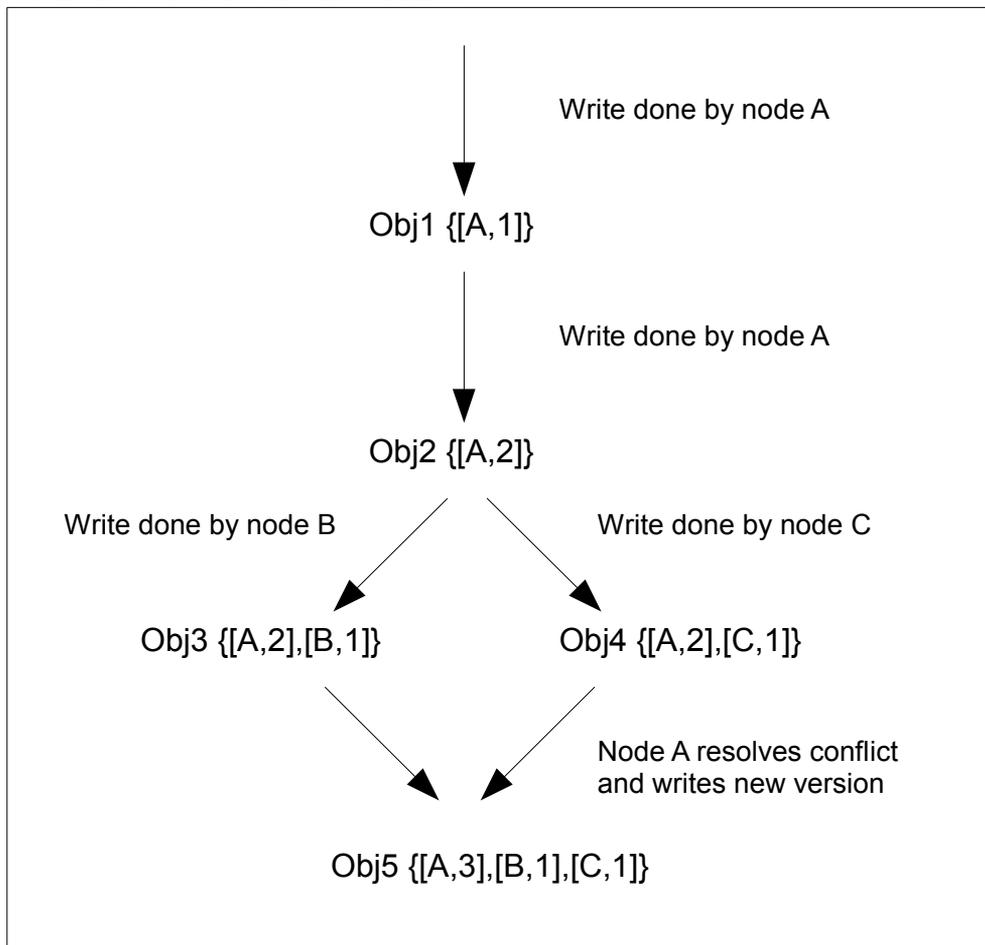


Figure 6: Example of Vector clocks

Consider the example shown in Figure 1. The term object is used here to represent a piece of data. The object has name ObjN, where N is a number describing the contents of the object. Starting from the top, a new object is written to database and node A has handled the write operation. Node A created the vector clock associated with the object and stored that with the

object. A vector clock is shown as list of pairs containing node identifier and timestamp inside curly brackets. Another write request comes and again node A handles it. The vector clock of Obj2 is direct descendant of Obj1 because all the nodes so Obj1 can be overwritten. Next thing to happen is that both Node B and Node C handle simultaneous write operations to a descendant of Obj2 creating conflicting versions. Next read operation reveals the conflicting versions and both versions are returned to client for conflict resolution. Node A handles the write and tags Obj5 as being direct descendant of both Obj3 and Obj4 so that if any other nodes have old revisions they can find that the conflict is already resolved and overwrite the old revision with the new one.

6.4.2. Multi-version concurrency control

Multi-version concurrency control means that old data is not overridden by new write operations but a completely new version is stored. It can improve performance, because read request can continue reading the old version of data while new version is being written. The opposite is to delay the write operation as long as the read operation is running. Google BigTable stores multiple versions of the same object and the user can choose how many old versions are kept. CouchDB¹⁴ takes the simple Last Write Wins approach to multi-version concurrency. It means that the database management system checks the timestamps of conflicting versions and the latest one is used as the current version.

6.5. Distribution and replication

Distribution and replication are the core techniques for scalability and fault tolerance. Proper distribution of load is a way to increase performance. When combined with replication it also increases fault-tolerance. Sharding means the partitioning of data among multiple computers. Consistent hashing is a method to distribute load and data.

6.5.1. Sharding

“Shards are secret ingredients of web-scale sauce, they just work” [xtrnormal, 2010].

Sharding is horizontal partitioning of data. One shard is one partition which doesn't have to be located in single instance of the database server. Traditional partitions have to be located within a single database server or schema.

¹⁴couchdb.apache.org

Sharding can be combined with replicating data which is seldom updated. Depending on the nature of the data, in some cases most reads and writes can be done within one shard. That obviously leads to better performance.

6.5.2. Consistent hashing

Consistent hashing was first developed to address caching problem in the web [Karger et al., 1997]. The concept is pretty simple and easily understood when visualized. Available hash values can be thought as a circle, the lowest value succeeding immediately the highest value as illustrated in Figure 7. Both nodes and object keys are hashed using the same hash function. To find the node where a certain object is stored, calculate the hash value of the object key and look for a node starting from the hash value and continuing clockwise in the circle until a node value is found. In Figure 7 (i) Obj 1 is stored to Node B while the other objects are stored to Node A. When Node C is added (ii), only values succeeding the added node value are reassigned. Objects are not moved between already existing nodes.

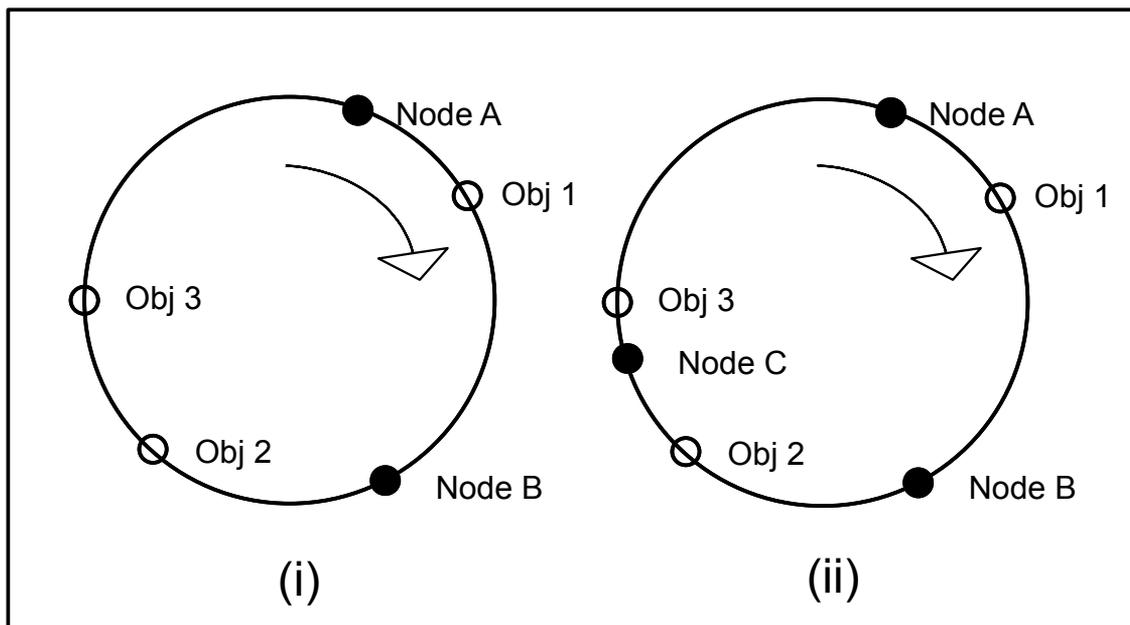


Figure 7: Consistent hashing visualized

6.6. Setting fault-tolerance and optimizing for read or write operations

For fault-tolerance, distributed databases usually write all data on more than one node. Fault-tolerance increases as the number of nodes where data is replicated increases. The downside is decreased performance. The replication setting can be described as tuple (N, W, R) where N is the number of nodes where all data is eventually replicated, W is the number of nodes who have to acknowledge the writes before returning to requesting client, and R is the number of nodes who have to return before a read request is returned to client. A read-optimized system has $R=1$ and $W=N$ where $N>1$. That means the decreased performance affects only write operations because all the replicas are updated before the request gets response. A write-optimized system has the opposite setup $W=1$ and $R=N$. The most common setup among Dynamo users is $N=3, R=2, W=2$ [DeCandia et al., 2007]. Variations exists, some systems have data center-aware or rack-aware nodes who can make sure data is replicated to a node in a different data center or rack to improve fault-tolerance.

6.7. Map/Reduce querying

One of the main reasons for new non-relational databases is the amount of data exceeding the limits of single computer and the cost of sophisticated commercial relational database systems which somehow bypass the limits of single computer. Querying must be a parallel operation in order to get the result in reasonably short time. Dean and Ghemawat [2004] introduced Google MapReduce, a framework for application developers to do conceptually simple computations with large datasets without worrying of failures, distribution of data and parallelization of computing.

Map/reduce querying consists of two functions, a map function and a reduce function, which are applied in phases. In the first phase the map function is applied to all objects in the whole dataset to find interesting objects. In the second phase the reduce function is applied to each of these intermediate result objects to get a single result value or set. The advantage is that these functions can be run in parallel. According to Dean and Ghemawat [2004] they usually run Map/reduce computations in clusters with 2000 computers. The input data is distributed using Google File System [Ghemawat et al., 2003] among cluster nodes where the computations happen. Output files are also in the Google File System where they can be found by the client application.

Many non-relational databases have implemented map/reduce querying as part of the database including CouchDB, Riak¹⁵ and MongoDB¹⁶. In addition, BigTable [Chang et al., 2008] can be used as input data source to Google MapReduce as well as output data storage. In key-value stores there is usually no query methods, only accessing with key. Map/reduce makes it possible to run arbitrary queries even when the data model itself doesn't support querying.

6.8. Shared nothing architecture

Shared nothing architecture is a way to think of both computer hardware architecture [Stonebraker, 1986] and computer software architecture [Armstrong, 2007]. A set of computers can have shared disks which can be used to share data. That makes the computers vulnerable to failures in other computers within the set. Also the shared resource becomes a possible single point of failure which takes down the whole system. In a shared nothing architecture the computers have to communicate by passing messages. The same is true for software, shared nothing means no shared resources within a computer or across a network of computers.

In the era of relational databases the database was seen as universal storage for all types of data. The popular Ruby on Rails¹⁷ web framework is designed to be full-stack web framework from the database abstraction layer to javascript code intended to run in the client browser. Scaling a website done with Ruby on Rails is trivial as long as you don't have to touch the database, just add more application servers and route requests somehow evenly between them. When the point is reached where the database is running in full capacity things start to get really hard. The shared nothing was true in all other layers but the database layer.

Pat Helland [2007] suggests that a scalable system should consist of two layers, a scale-agnostic upper layer and scale-aware lower layer. In case of Ruby on Rails, the application servers running Ruby application is the scale-agnostic layer, the database is in the lower layer. Helland's suggestion makes sense, it is easier to find developers if the developers don't have to understand all the details of parallel programming.

¹⁵www.basho.com/Riak.html

¹⁶www.mongodb.org

¹⁷www.rubyonrails.org

The Google MapReduce framework somehow implements Helland's vision. It makes it easy to application developers to write the map and reduce functions while making the heavy lifting in the background.

7. Notable implementations and business needs behind them

As Scott Adams [2010] wrote to his blog: "Ideas are worthless. Execution is everything". Trading consistency for availability is criticized being awful engineering decision [Stonebraker, 2010]. However, there is usually no point of making judgements based on pure theory or mere ideas. The implementation matters more than the ideas behind it. Google, Amazon and Facebook are solving business problems with their data store implementations. Dan North [2010] stated that users are not interested in features, they need capabilities. Having a data store which guarantees consistency is a nice feature but if it doesn't offer the capability of serving business needs all the superior features are worthless.

7.1. BigTable by Google

Google BigTable was the implementation which started the boom of new data stores. According to Chang et al. [2008] BigTable is used in more than sixty Google products and projects including Google Analytics, Google Finance, Orkut, Personalized Search and Google Earth.

7.1.1. Data model

The data model in BigTable is a "sparse, distributed, persistent multidimensional sorted map" [Chang et al., 2008]. A map is a collection of keys and associated values. The data structure can be accessed by key. Other names for this data structure in different programming languages are dictionary, hash table or associative array. There is some new concepts defined in the paper regarding the data model. A *table* actually can be thought as a relational database table. The table contains *column families* and *rows*. Each range of rows is partitioned, and a partition is called a *tablet*. Tablet is the unit of distribution and load balancing. To get the issue even more complicated, each chunk of data contains a row key, a column key and a timestamp. That means there might be configurable number of versions of a single piece of data with different timestamp keys. The values are not interpreted. Querying is by keys only, data can be accessed by column family key or by row key. To select anything by value, it must be done with MapReduce.

The data is stored in lexicographic order by row keys. Having data always in sorted order and combining it with the fact that data can be queried only by key makes distribution easier. A tablet contains all data lexicographically between two keys. Queries of short row ranges typically requires communication with only small number of nodes. Chang et al. [2008] use domain names in reverse order as example of keys. When searching for something about university of Tampere website the keys are `fi.uta.www`, `fi.uta.cs.www`, `fi.uta.mail`, etc. It makes the data about sub-domains being close to each other and makes perfect sense for search engine usage.

```
{
  "Webtable" : {
    "fi.uta.www" : {
      "contents" : {
        "" : {
          12 : "<html> mistake </html>",
          25 : "<html> corrected </html>"
        }
      },
      "anchors" : {
        "www.cs.uta.fi" : {
          13 : "University of Tampere"
        }
      }
    },
    "fi.uta.cs.www" : {
      "contents" : {
        ...
      },
      "anchors" : {
        ...
      }
    }
  }
}
```

Listing 5: BigTable data model visualized as JSON document

JSON can be used to visualize the data model. If we continue using the same example set by Chang et al., the data might look similar to Listing 5. The lowest level of the hierarchy contains the row keys, in this case the domain names in reverse order. The next level is column families. The listing has two column families defined, “contents” and “anchors”. Each column family may contain any number of columns. In the listing the “contents” column family contains only one column using an empty string (“”) as the column key. That indicates there is no need for more dimensions in this column family. All the columns may have multiple versions available identified by timestamps. The listing has two examples of “contents:” column, with timestamps 12 and 25. The “anchors” column family contains urls of web pages that link to the page defined by the row key. In the example, a link from www.cs.uta.fi pointing to www.uta.fi is saved with column key “anchors:www.cs.uta.fi” and containing the link text “University of Tampere”.

BigTable involves the following features.

- **Scalability**

BigTable supports horizontal scalability. When adding servers to the system the total throughput increases by over a factor of 100 when number of servers is increased from 1 to 500. The performance increase is not linear, per-server throughput experiences a significant drop when going from 1 to 50 servers.

- **Availability**

Chang et al [2008] report that BigTable has achieved its availability requirements. There is a number reported that tells us the availability in percentage of time, on average the unavailability has been 0.0047% due to lock service problems. That means 99.53% availability.

- **Fault-tolerance**

Distribution is achieved using Google File System [Ghemawat et al. 2003], a proprietary distributed file system developed by Google and used only inside Google. BigTable relies on Chubby [Burrows 2006] which is a lock service running on five replicas and communicating using paxos algorithm. It requires majority of nodes to be up and able to communicate with each other for the lock service to be available.

7.2. Dynamo by Amazon

Amazon.com implemented their own key-value store called Dynamo [DeCandia et al., 2007]. The need is to store and retrieve data by primary-key only, no need for ad-hoc queries. RDBMS solutions need highly-skilled personnel for its operation which adds costs. Also replication options for relational database management systems usually choose consistency over availability. Requirements are 'always writeable', no hierarchical name spaces or complex relational schema. Built for latency sensitive applications where 99,9 percent of requests have to be completed within several hundreds of milliseconds. Dynamo is being used inside Amazon by variety of services including the best seller lists, shopping carts, customer preferences, session management, sales rank, and product catalogue.

While the system itself is a key-value store, a single node may use MySQL or BerkeleyDB for persistent storage. So a relational database management system can be part of a non-relational data storage solution. The data model is very limited, accessing is based on key only, no querying based on values.

Dynamo involves the following features.

- **Scalability**

A requirement when the system was designed was to be able to scale out at one node at a time with minimal impact on both system operators and the system itself. The request routing solution is a limit to the number of nodes in the system. All nodes have the full routing information and maintaining that in consistent state across all the machines becomes impractical in some point. DeCandia et al. [2007] didn't provide numbers describing the throughput when adding new nodes.

- **Availability**

One of the requirements was 'always writeable' that means the availability for read operations are not higher than availability for write operations. Dynamo has weakened consistency to achieve high availability. In practice 99.95% of the data never has a conflict according to Amazon's numbers [DeCandia et al., 2007]. At the same time the proportion of successful responses have been 99.9995%. That's a bit better than the reported BigTable number and generally good achievement. In the literature 'five nines' i.e. 99.999% is considered to be a good availability number.

- **Fault-tolerance**

Dynamo uses consistent hashing for distribution. Each node has many tokens in the consistent hashing scheme to achieve better load balance. That means there is both virtual and physical nodes in the hash ring. Replication is implemented so that a coordinator, which is the node communicating with client, makes sure the data is replicated along configured number of physical nodes. According to DeCandia and others [2007] Dynamo users in Amazon usually have data replicated to three nodes while read and write operations are returned to client when two nodes have responded. Vector clocks are used to determine whether data is conflicting or not. In case of conflicting values, both versions are returned to the requesting client.

7.3. Cassandra by Facebook

Facebook developed Cassandra [Lakshman and Malik, 2010] to power the inbox search feature in the largest social network platform in the world. Both BigTable and Dynamo has influenced the design of Cassandra. Cassandra has a BigTable -like data model on a Dynamo -like distribution model.

Cassandra derives data model from BigTable and adds a new concept to it. *Super columns* are columns that contain another map containing keys and values. That makes the data structure one level deeper if the user decides to use super columns. A column family can contain either only columns or only super columns but not both.

Cassandra has been released as open source under Apache license and is now a top level Apache project¹⁸. In inbox search the challenge was about storing reverse indices of messages that the users of Facebook send to each other via Facebook network. Cassandra involves the following features.

- **Scalability**

Cassandra has a scalability model very similar to Dynamo. When a new node is added to the system, it is assigned a place in the consistent hashing scheme such that it can take some work from a heavily loaded node. Cassandra is designed for linear scalability and references to linear scalability with Cassandra are found from the web but unfortunately any numbers weren't given.

- **Availability**

¹⁸cassandra.apache.org

While published numbers weren't available, Cassandra is said to be highly-available. The availability depends on consistency settings. As mentioned earlier the users can set how many nodes have to acknowledge the write before the write request is returned to client. Reducing consistency increases availability.

- **Fault-tolerance**

Distribution is done using consistent hashing. The replication model is slightly different in Cassandra than in Dynamo. Dynamo uses virtual nodes to balance the load between physical nodes. Cassandra doesn't use the concept of virtual nodes at all. Instead Cassandra analyzes load information and moves the lightly loaded nodes to get balanced data and load distribution.

7.4. Summary

Scalability, availability and fault-tolerance requirements of BigTable, Dynamo and Cassandra are achieved according to the authors of respective papers. The Table 1 is a summarized view of the solutions to these challenges.

Table 1: Comparison of the implementations

	BigTable	Dynamo	Cassandra
Scalability	Horizontal scalability. Production deployment more than 500 nodes.	Horizontal scalability.	Horizontal scalability. Production deployment to 150 node cluster
Availability	Relies on lock service which implements paxos algorithm.	'Always writeable' availability. The balance between consistency and availability can be decided by the user.	Availability is increased using 'rack aware' and 'datacenter aware' replication strategies. Each node is identical.
Fault-tolerance	Data distribution done by Google File System.	Consistent hashing distribution with virtual nodes. Number of nodes for data replication can be decided by the user.	Consistent hashing distribution.

8. Discussion

The driving force for developing new data stores seems to be the need for scaling services. Brewer [2001] points out that during an excessive load the system has to do graceful degradation. If all the subsystems of a web site, for example Amazon.com, share the same hardware resources it might be wise to stop showing recommendations, product ranking or user history and use the computing power to keep up the basic services like shopping cart and product catalog.

Some very large scale services such as Google web search, Amazon.com or Facebook must have systems running on multiple continents to keep the system running. When considering the business model of Facebook which makes its billions of dollars by showing ads to its 500 million users it becomes obvious that a loss of users will affect profits. Google does kind of the same, shows ads to people using the search engine. Amazon actually sells something directly to its users. Google has told that if the latency for showing search results increases by 500 milliseconds it affects revenue by 20%. Amazon had done tests where they delayed the response by 100 millisecond intervals and each increase lowered sales by 1% [Hamilton, 2009].

At some point even the most powerful computer won't be able to keep up with the growing load. Scaling up by adding new computers should be able to be done without excessive modifications to the system. According to Joe Armstrong [2007], that can be achieved by doing things in the first place in the same way as if the components have no shared resources but have to communicate by message-passing. Relational database management systems are meant to be efficient in disk usage, so duplication is minimized by heavy normalization. That was exactly the reason why shared memory concurrency has been used in concurrent programming, it was more efficient than message-passing concurrency and computer hardware was expensive. In contrast, new non-relational databases usually have self-contained data. Comparing new non-relational database management systems to traditional relational database management systems seems conceptually very much the same as message-passing concurrency compared to shared memory concurrency.

Stonebraker and others [2007a] found that a popular commercial RDBMS used two thirds of its processing time creating logs for recovery and other purposes. The backup window has shrunk to zero. The difference in thinking between traditional RDBMS world and NoSQL world is that a failure is not an exception but a thing that will happen in the real life. When a server crashes in the RDBMS world a restore will be necessary and depending on preparedness to a failure it will take more or less but always some time. There is simply no time to do the recovery manually. That being said there is no sense writing persistent recovery logs which is a performance bottleneck of traditional systems. If either an established business which is expecting growth or a startup which is dreaming for a boom of sales develops a new service, it would be bad judgement not to take scalability needs into account. At the time when

the problems surface it would be too late to prevent a drop in client trust if the service is down because of technology scaling problems.

A counter-example of the above paragraph would be the MySQL cluster. It is based on shared nothing architecture and no single point of failure exists. Nodes can be added to the cluster without downtime and it is designed to five nines availability. However, the previously mentioned limitations apply to it. The MySQL cluster uses the two phase commit protocol to ensure consistency. The weaknesses of two phase commit are already discussed. The maximum number of data nodes is fifty. That is a lot less than the number of nodes Cassandra, BigTable or Dynamo can use.

People are used to web services and many business are depending on services being available in the web. It is not rare to have a startup which is getting all its income by selling software as a service. When something has to be up all the time and user base is increasing, failures will happen. The system needs to be designed to handle failure as normal business rather than as a crisis. Based on business requirements the system should be designed to either reduce harvest or yield or both.

9. Conclusion

Distributed systems, such as web services, can't offer simultaneously strong consistency and high availability. The choice between availability and consistency is not merely a technical one, it is also a business choice. Because the environment, access patterns and technology is changed new solutions for scalable, fault-tolerant and highly-available data stores is needed. Many projects and products are in the marketplace to respond to these needs.

Relational databases are not even close to being pushed out from the marketplace. But for many applications, new non-relational data stores are a good choice. Popular systems will get more load and eventually vertical scaling reaches the limit of a single server. New data stores makes it easier to scale incrementally one server at a time as need emerge.

References

- [Adams, 2010] Adams, S. 2010. The Value of Ideas, blog entry http://www.dilbert.com/blog/entry/the_value_of_ideas
- [Agrawal et al., 2009] Agrawal, R., Ailamaki, A., Bernstein, P.A., Brewer, E.A., Carey, M.J., Chaudhuri, S., Doan, A., Florescu, D., Franklin, M.J., Garcia-Molina, H., Gehrke, J., Gruenwald, L., Haas, L.M., Halevy, A.Y., Hellerstein, J.M., Ioannidis, Y.E., Korth, H.F., Kossmann, D., Madden, S., Magoulas, R., Beng Ooi, C., O'Reilly, T., Ramakrishnan, R., Sarawagi, S., Stonebraker, M., Szalay, A.S., and Weikum, G. 2009. The Claremont report on database research. *Commun. ACM* 52, 6 (June 2009), 56-65.
- [Astrahan et al., 1976] Astrahan, M.M., Blasgen, M.W., Chamberlin, D.D., Eswaran, K.P., Gray, J.N., Griffiths, P.P., King, W.F., Lorie, R.A., McJones, P.R., Mehl, J.W., Putzolu, G.R., Traiger, I.L., Wade, B.W. and Watson, V. 1976. System R: relational approach to database management. *ACM Trans. Database Syst.* 1, 2 (June 1976), 97-137.
- [Armstrong, 2007] Armstrong, J., *Programming Erlang: Software for a Concurrent World*, The Pragmatic Programmers, Raleigh, NC, 2007
- [Baker, 1992] Baker, H.G., 1992. Relational Databases considered harmful (relative to object-oriented databases.) *ACM Forum. Comm. of the ACM* 35, 4 (April 1992), 16,18.
- [Bernstein et al., 1987] Bernstein, P., Hadzilacos, V., and Goodman, N. *Concurrency Control and Recovery in Database Systems*, Chapter 7, Addison Wesley Publishing Company 1987
- [Birman, 2010] Birman, K. 2010. History of the Virtual Synchrony Replication Model. Appears in *Replication: Theory and Practice*. B. Charron-Bost, F. Pedone, A. Schiper (Eds) Springer Verlag, 2010. *Replication, LNCS 5959*, pp. 91-120, 2010.
- [Brewer, 2000] Brewer, E.A., Towards robust distributed systems (abstract). In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing (Portland, Oregon, United States, July 16-19, 2000)*. PODC '00. ACM, New York, NY, 7.

- [Brewer, 2001] Brewer, E. Lessons from Giant-Scale Services. *Internet Computing*, IEEE (2001) vol. 5 (4) pp. 46 – 55
- [Brewer, 2004] Brewer, E. Combining Systems and Databases: A Search Engine Retrospective, in *Readings in Database Systems*, M. Stonebraker and J. Hellerstein, Eds., 4 ed, 2004.
- [Burrows, 2006] Burrows, M. The Chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation* (Seattle, Washington, November 06-08, 2006). *Operating Systems Design and Implementation*. USENIX Association, Berkeley, CA, 335-350.
- [Chamberlin and Boyce, 1974] Chamberlin, D.D. and Boyce, R.F. SEQUEL: A Structured English Query Language. *Proceedings of the 1974 ACM SIGFIDET Workshop on Data Description, Access and Control* (1974): 249–264.
- [Chang et al., 2008] Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.* 26, 2 (Jun. 2008), 1-26.
- [Chen, 1976] Chen, P. The entity-relationship model – towards a unified view of data. *ACM Transactions on Database Systems*, 1 (1976) 9-36.
- [Codd, 1970] Codd, E. F. A relational model of data for large shared data banks. *Commun. ACM* 13, 6 (Jun. 1970), 377-387.
- [Crockford, 2006] Crockford, D., The application/json Media Type for JavaScript Object Notation (JSON), IETF, RFC 4627, July 2006
- [Dean and Ghemawat, 2004] Dean, J. and Ghemawat, S. MapReduce: simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6* (San Francisco, CA, December 06-08, 2004). *Operating Systems Design and Implementation*. USENIX Association, Berkeley, CA, 10-10.

- [DeCandia et al., 2007] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. Dynamo: Amazon's highly available key-value store. In Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles (Stevenson, Washington, USA, October 14-17, 2007). SOSP '07. ACM, New York, NY, 205-220.
- [Edlich, 2009] Edlich, S. 2009. NoSQL archive page, <http://nosql-database.org/>
- [Elmasri and Navathe, 2000] Elmasri R. and Navathe S.B. Fundamentals of Database Systems. 3rd edition, Addison Wesley, 2000.
- [Evans, 2001] Evans, C. YAML Draft 0.1, Yahoo! Tech groups: sml-dev. <http://tech.groups.yahoo.com/group/sml-dev/message/4710>
- [Evans, 2009] Evans, E. NOSQL 2009, blog entry, http://blog.sym-link.com/2009/05/12/nosql_2009.html
- [Fidge, 1988] Fidge, C. J. Timestamps in message-passing systems that preserve the partial ordering. In K. Raymond, editor, Proceedings of the 11th Australian Computer Science Conference (ACSC'88), pages 56-66, February 3-5 1988.
- [Fox and Brewer, 1999] Fox, A. and Brewer, E. A. Harvest, Yield, and Scalable Tolerant Systems. In Proceedings of the Seventh Workshop on Hot Topics in Operating Systems (March 28-30, 1999). HOTOS. IEEE Computer Society, Washington, DC, 174.
- [Ghemawat et al., 2003] Ghemawat, S., Gobioff, H., and Leung, S. The Google file system. SIGOPS Oper. Syst. Rev. 37, 5 (October 2003), 29-43.
- [Gilbert and Lynch, 2002] Gilbert, S. and Lynch, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33, 2 (Jun. 2002), 51-59.
- [Gray, 1988] Gray, J. 1988. The transaction concept: virtues and limitations. In Readings in Database Systems Morgan Kaufmann Publishers, San Francisco, CA, 140-150.

- [Hamilton, 2007] Hamilton, J. 2007. On designing and deploying internet-scale services. In Proceedings of the 21st Conference on Large installation System Administration Conference (Dallas, November 11-16, 2007). P. Anderson, Ed. USENIX Association, Berkeley, CA, 1-12.
- [Hamilton, 2009] Hamilton, J. 2009. The Cost of Latency, blog entry, <http://perspectives.mvdirona.com/2009/10/31/TheCostOfLatency.aspx>
- [Helland, 2007] Helland, P. Life beyond Distributed Transactions: an Apostate's Opinion. 3rd Biennial Conference on Innovative DataSystems Research (CIDR) January 7-10 2007, Asilomar, CA.
- [Junkkari, 2007] Junkkari, M. A concept-oriented data modeling and query language approach to next generation information systems. Department of Computer Sciences, University of Tampere, 2007-2. University of Tampere.
- [Kotler and Keller, 2008] Kotler, P. and Keller, K. Marketing management. Prentice Hall, 13th edition, 2008.
- [Karger et al., 1997] Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., Lewin, D. Consistent hashing and random trees. Proceedings of the twenty-ninth annual ACM symposium on Theory of computing (1997). 654-663. ACM Press New York, NY, USA
- [Lakshman and Malik, 2010] Lakshman, A. and Malik, P. Cassandra: a decentralized structured storage system. SIGOPS Oper. Syst. Rev. 44, 2 (Apr. 2010), 35-40.
- [Lakshman, 2008] Lakshman, A. Cassandra – A structured storage system on a P2P Network, blog entry, http://www.facebook.com/note.php?note_id=24413138919&id=9445547199&index=9
- [Lamport, 1978] Lamport, L. Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21, 7 (Jul. 1978), 558-565.
- [Lamport et al., 1982] Lamport, L., Shostak, R., and Pease, M. The Byzantine Generals Problem. ACM Trans. Program. Lang. Syst. 4, 3 (Jul. 1982), 382-401.

- [Lamport, 1998] Lamport, L. The part-time parliament. ACM Transactions on Computer Systems, 16(2):133–169, May 1998.
- [Lamport, 2001] Lamport, L. 2001. Paxos Made Simple. ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001) 51-58.
- [Mattern, 2010] Mattern, F. 2010. Virtual Time and Global States of Distributed Systems. In: Cosnard M. et al. (Ed.): Proc. Workshop on Parallel and Distributed Algorithms. pp. 215-226, North-Holland / Elsevier, 1989 (Reprinted in: Z. Yang, T.A. Marsland (Eds.), "Global States and Time in Distributed Systems", IEEE, 1994, pp. 123-133.)
- [McJones, 1997] McJones , P. (editor). The 1995 SQL Reunion: People, Project, and Politics. August 20, 1997 (2nd edition).
- [Moore, 1965] Moore, G.E. Cramming more components onto integrated circuits. Electronics, volume 38, number 8 (19 April 1965)
- [North, 2010] North, D. Simplicity, The Way of the Unusual Architect. Presentation in QCon London, Nov 17 2010, <http://www.infoq.com/presentations/Simplicity-Architect>
- [Nygard, 2007] Nygard, T.M. Release It! Design and Deploy Production-Ready Software, The Pragmatic Programmers, Raleigh, NC, 2007
- [Olson et al., 1999] Olson, M.A., Bostic, K., and Seltzer, M. Berkeley DB, Proc. FREENIX Track, USENIX Annual Tech. Conf. (Jun. 1999)
- [O'Neil et al., 1996] O'Neil, P., Cheng, E., Gawlick, D., and O'Neil, E. 1996. The log-structured merge-tree (LSM-tree). Acta Inf. 33, 4 (Jun. 1996), 351-385.
- [Popescu, 2010] Popescu, A. 2010. Powered by NoSQL, blog entry, <http://nosql.mypopescu.com/kb/powered-by-nosql>
- [Pritchett, 2008] Pritchett, D. 2008. BASE: An Acid Alternative. Queue 6, 3 (May. 2008), 48-55.

- [Rodriguez and Neubauer, 2010] Rodriguez, M and Neubauer, P. 2010. The Graph Traversal Pattern. AT&T and NeoTechnology Technical Report, April 2010. CoRR abs/1004.1001 2010.
- [Rumbaugh et al., 1999] Rumbaugh, J., Booch, G., and Jacobson, I. The Unified modeling Language Reference Manual, Addison-Wesley Professional, 2004
- [Sears and Brewer, 2006] Sears, R. and Brewer, E. 2006. Stasis: flexible transactional storage. In Proceedings of the 7th Symposium on Operating Systems Design and Implementation (Seattle, Washington, November 06-08, 2006). Operating Systems Design and Implementation. USENIX Association, Berkeley, CA, 29-44.
- [Skeen and Stonebraker, 1983] Skeen, D., Stonebraker, M. "A Formal Model of Crash Recovery in a Distributed System". IEEE Transactions on Software Engineering 9 (3) (May 1983), 219–228.
- [SQL, 2008] SQL:2008 standard, ISO/IEC 9075(1-4,9-11,13,14), 2008
- [Stonebraker, 1986] Stonebraker, M., The case for shared nothing. Database Engineering Bulletin 9, 1 (Mar. 1986), 4-9.
- [Stonebraker and Cetintemel, 2005] Stonebraker, M. and Cetintemel, U. One Size Fits All: An Idea whose Time has Come and Gone. In Proc. ICDE, 2005.
- [Stonebraker et al., 2005] Stonebraker, M., Abadi, D.J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., O'Neil, P., Rasin, A., Tran, N., and Zdonik, S. C-store: a column-oriented DBMS. In Proceedings of the 31st international conference on Very large data bases (2005). VLDB Endowment 553-564.
- [Stonebraker et al., 2007a] Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hachem, N., and Helland, P. The end of an architectural era: (it's time for a complete rewrite). In Proceedings of the 33rd international Conference on Very Large Data Bases (Vienna, Austria, September 23-27, 2007). Very Large Data Bases. VLDB Endowment, 1150-1160.

- [Stonebraker et al., 2007b] Stonebraker, M., Bear, C., Cetintemel, U., Cherniack, M., Ge, T., Hachem, N., Harizopoulos, S., Lifter, J., Rogers, J. and Zdonik, S. One Size Fits All? - Part 2: Benchmarking Results. In Proc. CIDR, 2007.
- [Stonebraker, 2010] Stonebraker, M. 2010. Clarifications on the CAP Theorem and Data-Related Errors, blog entry, <http://voltdb.com/blog/clarifications-cap-theorem-and-data-related-errors>
- [Sutter, 2005] Sutter, H. The Free Lunch Is Over, A Fundamental Turn Toward Concurrency in Software. Dr. Dobbs's Journal, 30(3), March 2005
- [Ullman, 1988] Ullman, J.D. Principles of database and knowledgebase systems, Computer Science Press, Rockville, MD, 1988
- [Vogels, 2008] Vogels, W. 2008. tweet. <http://twitter.com/Werner/statuses/1008722501>
- [Vogels, 2009] Vogels, W. Eventually consistent. Commun. ACM 52, 1 (Jan. 2009), 40-44.
- [W3C, 2008] World Wide Web Consortium, Extensible Markup Language (XML) 1.0 (Fifth Edition), XML specification, <http://www.w3.org/TR/2008/REC-xml-20081126/>
- [Whitehorn, 2009] Whitehorn, M. 2009. In the spin of SSDs on database servers, blog entry, http://www.theregister.co.uk/2009/08/24/whitehorn_ssds_servers/
- [xtranormal, 2010] A parody video of NoSQL databases, <http://www.xtranormal.com/watch/6995033/>
- [Zyp and Court, 2010] Zyp, K. and Court, G., A JSON Media Type for Describing the Structure and Meaning of JSON Documents, IETF Internet-Draft, 2010, <http://tools.ietf.org/html/draft-zyp-json-schema-03>