12/12/2014

An Investigation into Database Technologies and Metrics Leading to a Database Evaluation System.

Joe O Flaherty BSc Computing with Multimedia.

Primary Supervisor: Mr. Billy Stack.

Joe O Flaherty – T00155775

Abstract

Measurement and evaluation using the resulting metrics are omnipresent in modern society and fundamental in many ways. Common measurements are used to assess, compare and evaluate the performance of people, machines, software and services against each other or against known baselines. Measurement according to Delorme and Chatelain can “highlight strengths and weaknesses, gives an idea of the progress made over time and helps decision-makers to compare courses of action and identify the most effective mechanisms” (Delorme & Chatelain, 2011).This thesis and design document evolved from an interest in measurement and applies that interest to the area of data storage i.e. databases. Databases have become central to the operation of most companies and organisations and have recently begun a new phase in their evolution. Measuring and evaluating databases is of growing importance as a result of this evolutionary change.

Traditionally database measurement (benchmarking) has meant dealing with predominately a single type of database based on the relational model. These benchmarks are flawed when NoSQL databases are introduced. There are no industry standard benchmarks available which are backed up with comprehensive published results. Comparison systems independent of vendor interest are also notable by their absence. Development and evaluation of a system that overcomes these weaknesses is proposed.

The system should measure performance, scalability and elasticity of multiple databases, one from each database family (key-value, document, and graph, columnar and relational). The system should have the capability of measuring performance based on varying volumes of data to determine any changes in performance. What format data is stored in must also be considered, NoSQL databases are said to be aggregate oriented (Sadalage & Fowler, 2013), meaning they can store data using multiple complex structures, nested documents, super columns etc. The effect on database performance based on the data model utilised should be measured, whether the limited time frame of this project allows this remains to be seen.

Results of measurements will be displayed in graph format for each database. The system should graphically display a comparison of multiple databases, enabling interested parties make an informed decision when choosing a data storage technology for their needs.

Contents

Introduction

Databases are so common place that we have in many ways come to ignore them, it is an assumption that they will perform as required. Recently with the growth in the use of newer database technologies the assumption of performance is not so straightforward. This paper deals with data storage technologies and the need to measure and evaluate them against each other. Until relatively recently the choice of database implementation meant choosing a vendor to supply a Relational Database that used SQL as its query language. The increase in usage of NoSQL databases and polyglot persistent systems emphasise the need for performance measurements that are applicable to all data storage types. The approach taken here is to firstly give an overview of multiple database technologies and concepts highlighting specific topics of interest. Current metrics and benchmarking processes are discussed and critically evaluated, alternative metrics are proposed resulting from the evaluation. The proposed metrics and implementation of a measurement system result from the need to evaluate multiple database types and compare them in a vendor independent manner. Broad investigative and explanatory sections form the basis of the first section of the paper, followed by an outline of current benchmarks and benchmarking techniques. Resulting from this investigation an alternative is proposed, pertinent metrics identified and a methodology outlined. Investigations into current benchmarking techniques have highlighted a number of weaknesses related to measurement of multiple implementations. Transaction Processing Council (TPC) benchmarks can be very specific in nature and the only benchmark provided for non-relational systems lacks up to date results.

Yahoo provide a cloud serving benchmarking framework (YCSB) which while powerful is a technique/system not a comprehensive set of metrics. YCSB also suffers from a lack of published results that are vendor independent and is also closely coupled with Apache Hadoop and systems that utilise it. The ultimate goal of this paper and implementation is to provide a system that measures performance, scalability and elasticity of a broad selection of databases, not entire systems that include data storage elements. Furthermore the proposed system will be a ‘quick’ measurement and comparison tool enabling close to instant evaluation of a database under various scenarios, aiding in the process of choosing the most effective system for user/business needs.

Abbreviations

DBMS	– Database management system.
RDBMS	– Relational database management system.
SQL	– Structured Query Language.
API	– Application Programming Interface.
ERD	– Entity Relationship Diagram.
REST	– Representational State Transfer.

Research Questions

1. Are current database benchmarks and benchmarking tools adequate for the evaluation of multiple data storage systems?

2. Can alternative or adapted metrics be suggested?

3. Is it possible to develop a system that evaluates multiple databases and displays graphical results?

4. Can this system then be utilised as a database recommendation tool?

Databases

The most obvious value of a database is keeping large amounts of persistent data, traditionally this data it is written to a backing store – a disk or persistent memory. For most enterprise applications this backing store is a database. Databases allow more flexibility than a simple file system in storing large amounts of data in a way that makes access to small bits of that data quick and easy via an application program. (Sadalage & Fowler, 2013). The relational database model was introduced by E.F. Codd in 1970, adoption of the model was slow until System-R was introduced by IBM initially as a trial in 1977 (Chamberlain, 1981) and as a commercial product thereafter.

Since the introduction of the relational model, multiple database models were introduced such as the object oriented database, object relational database and XML database. Object oriented and object relational databases appeared as the object oriented approach to software development evolved. However, these databases never really become competitive in the marketplace (Indrawan-Santiago, 2012). The reasons could be summarized in lack of theoretical foundation and limited performance gained over the relational database as discussed by Sikha Bagui. Recently multiple new database models have emerged broadly referred to as NoSQL databases. Whether these slowly topple the relational model, integrate with it to some degree or fail is as yet unclear.

Relational Databases & RDBMS

Writing on relational database design and implementation Harrington says *"There are things about which we need to store data and these things are related to one another in a number of ways. To be considered a database, the place where data is stored must also store information about the relationships between the stored data"* (Harrington, 2009).

Relational databases are built on the relational model introduced by E.F. Codd in 1970 which is based on mathematical set theory. The relational model is based on the theory that a solution built on preconceived paths through a data structure is too restrictive. The relational model enables assigning a relation to records as required as opposed to establishing pre-defined relations when records are initially stored in a database. Queries performed on a relational database work with sets of data not with a single record at a time. (Oppel, 2004)When introduced the academic community broadly agreed with the model, practitioners had concerns about performance of relational databases in comparison to the existing network and hierarchical models (Indrawan-Santiago, 2012). Adoption of the relational model was slow until System-R a DBMS was introduced by IBM as a trial in 1977 (Chamberlain, 1981) and as a commercial product thereafter. The Database Management System (DBMS) is software provided by database vendors that provides all the functionality needed to manage and organise a database (Oppel, 2004). Relational database management systems (RDBMS) had by the early 1980s begun to dominate the database market (Indrawan-Santiago, 2012).

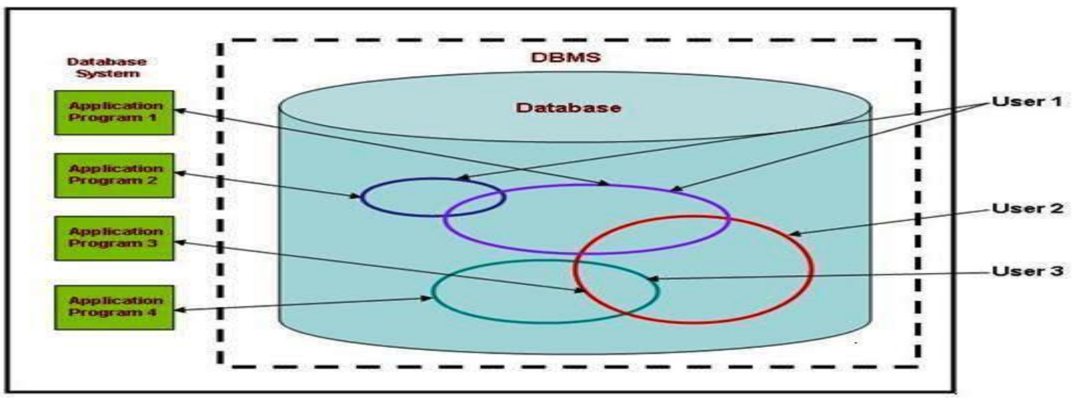


Fig 1.0: Generic high level view of a RDBMS accessible to multiple users (Harrington, 2002).

RDBMS provide many benefits the first being the ability to access persistent data. The database is more flexible than a file system in how it stores large volumes of data which enables application programs access that data quickly and relatively easily (Sadalage & Fowler, 2013). Concurrency is the second benefit offered by the RDBMS, multiple users can access and modify data simultaneously. Sadalage says "concurrency is notoriously difficult to get right with all sorts of errors that can trap even the most careful programmer". RDBMS mitigate against this by carefully controlling access to data via transactions and locks. Transactions provide the ability to alter a database but if an error occurs the operation is rolled back to the previous stable state. Thirdly RDBMS provide for shared database integration allowing multiple applications store data on a single database (Sadalage &

Fowler, 2013). RDBMS that use SQL as their query language make this integration much more efficient (Hophe & Woolf, 2011). Relational DBMS also provide a predominately standard model. There are differences between different databases but the core is essentially the same, SQL is the standard query language and transactions work in the same way (Sadalage & Fowler, 2013).

Transactions due to their importance to the RDBMS are worthy of further explanation. Oppell defines a transaction as a discrete piece of work that is either entirely processed or entirely not processed. The acronym ACID (Atomicity, Consistency, Isolation and Durability) is commonly used to describe a transactions properties.

- Atomicity: A transaction entirely succeeds or completely fails. All changes made as part of a successful transaction are preserved, any changes made by a failed transaction are undone; transaction are not partial operations. Undoing the changes made by a failed transaction is termed a rollback, the preservation of successful changes is referred to as a commit.
- Consistency: Transactions alter a database from one consistent state to another.
- Isolation: Transactions occur independently of other transactions that may occur simultaneously.
- Durability: Transactional changes persist in the database despite shutdown or system failures i.e. changes are permanent
  (Oppel, 2004).

Data stored by a relational database is represented by tables of rows and columns. Relations as tables are referred to consist of collections of data (objects) of the same type stored in rows. Data in a table is related by common keys (Primary and Foreign keys). The basis for the term relational database is the ability search for and retrieve related data from a relation.

Understanding relational databases and RDBMS requires understanding of key concepts encapsulated in Codds 12 rules:

Rule 1: All data items stored in the database, must be a value of a table cell.

Rule 2: Each unique value is logically accessible via a combination of table-name, primary key and attribute name. This combination is the exclusive means of guaranteed access.

Rule 3: NULL values in the database must be given a systematic treatment, NULL may have multiple meanings. NULL can represent missing data, unknown data and non-applicable data.

Rule 4: A structural description of the entire database must be stored in a data dictionary accessible to authorized users.

Rule 5: The database must support a language capable of data definition, data manipulation and transaction management operations. The language is the only means of accessing the database either directly or using a separate application.

Rule 6: Every view of the database, which can be theoretically updated, must also be updatable by the system.

Rule 7: The database provides high-level insert, update and delete operations, not limited to a single row. Union, intersect and minus operations must be provided for.

Rule 8: How data is physically stored is of no relevance to the system, changes to physical structure must not affect the data.

Rule 9: Logical data is independent of any user view (application), change in logical data does not imply a change in any application using it.

Rule 10: The database is independent of any application, each integrity constraint can be modified with no requirement to change the application.

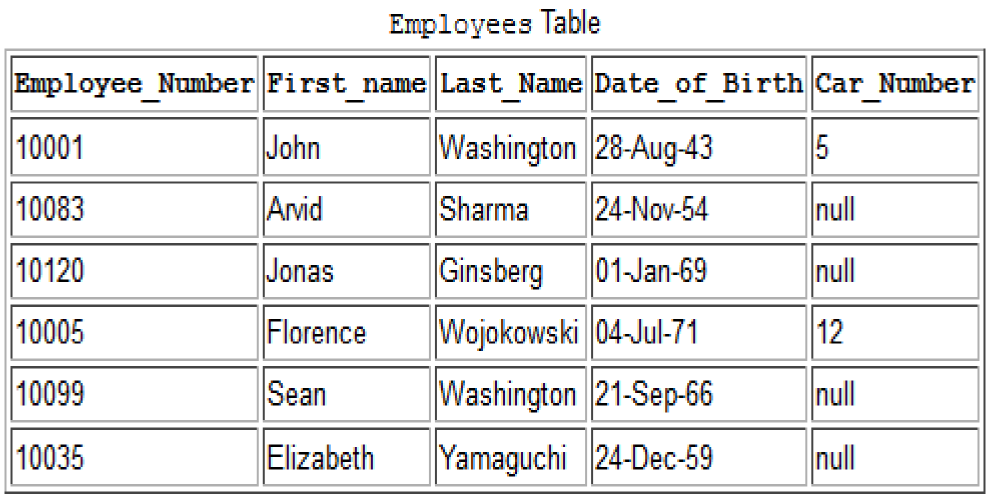
Rule 11: End users remain ignorant of how data is distributed across multiple locations. Users view the data as stored in a single location.

Rule 12: Systems that provide access to low level records cannot subvert the system bypassing security and integrity constraints

(Harrington, 2002).

There are a number of integrity rules that ensure accuracy, consistency and accessibility in the relational model. These rules impose a rigid structure to relational databases and RDBMS which is ideal for very structured data. The lack of flexibility this enforces can also be perceived as a weakness in the relational model (Stonebraker, 2010).

1. All rows in a relational table are distinct. Duplicate rows present problems in resolving which of multiple possibilities is correct.
2. Column values must not be repeating groups or arrays.
3. Null values, a null value is not equivalent to a blank or zero but indicates where data may not be available. Two null values are not equal.
4. Entity integrity, primary keys the unique values that identify individual rows in a table cannot be null. Null values in a primary key column prohibit it from being a complete identifier.



Employees Table

Employee_Number	First_name	Last_Name	Date_of_Birth	Car_Number
10001	John	Washington	28-Aug-43	5
10083	Arvid	Sharma	24-Nov-54	null
10120	Jonas	Ginsberg	01-Jan-69	null
10005	Florence	Wojokowski	04-Jul-71	12
10099	Sean	Washington	21-Sep-66	null
10035	Elizabeth	Yamaguchi	24-Dec-59	null

Fig 1.1: Employee table illustrating the structure of a relational table (Harrington, 2002).

Relational databases are designed from an Entity Relationship Diagram which models the entities of the database and the relationships between them, this is transformed into the Relational Schema.

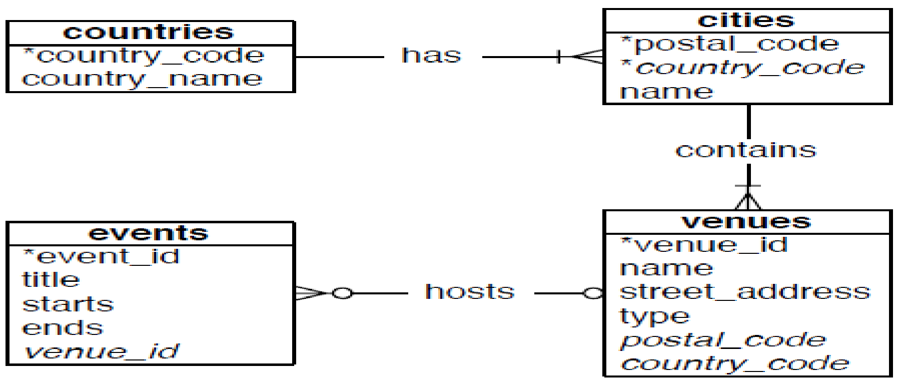


Fig 1.2: A simple ERD example (Redmond & Wilson, 2012).

The need to design a schema when you want to store data in a relational database can also be perceived as a weakness as it restricts flexibility. A relational schema defines the structure of a database, specifying tables and columns as well as the domain for each column (Sadalage & Fowler, 2013). The relational schema must be designed prior to storing data, the data you store is subsequently bound to the schema. This data to schema binding means making changes to a database, handling unstructured data difficult, to quote Sadalage "A schema puts all row of a table into a straightjacket which becomes awkward if you have different types of data in different rows".

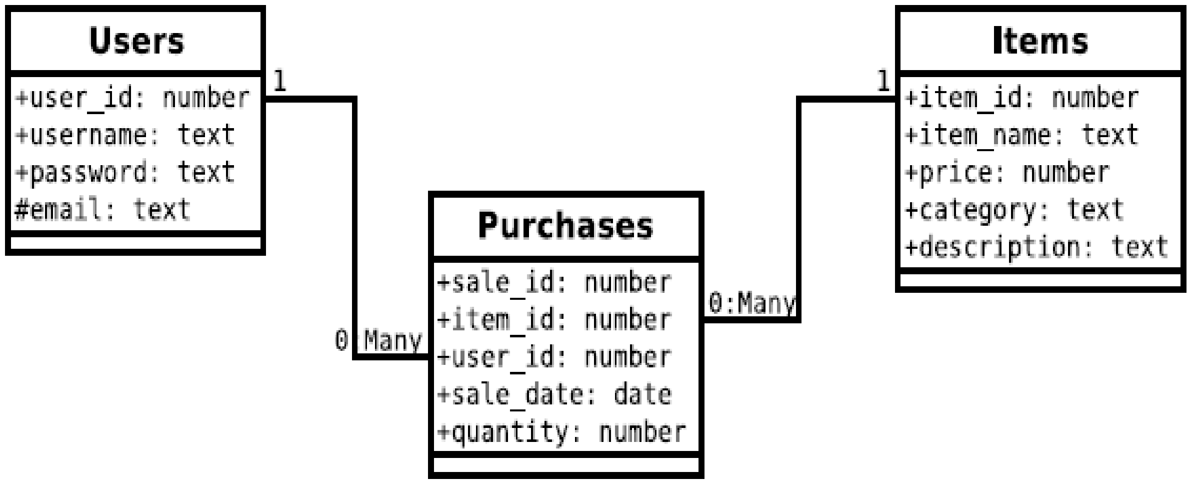


Fig 1.3: Relational Schema (Redmond & Wilson, 2012)

Normalisation is a three step process:

1. **First Normal Form (1NF)**: Groups of repeating data are removed by creating new tables for each group that can be identified by a Primary Key.

2. **Second Normal Form (2NF)**: If multiple records contain common sets of values the common values are moved to a new table. The two tables are then linked using a foreign keys.

3. **Third Normal Form (3NF)**: Fields in a table that are not dependant on the primary key are removed and placed in another table.

Essentially relational databases are very structured, rule governed and inflexible yet remain an ideal solution for the storage of structured data of any volume. They do however come with weaknesses as Redmond and Wilson explain "*If you need to scale out rather than up (multiple parallel data stores rather than a single beefy machine or cluster), you may be better served looking elsewhere. If your data requirements are too flexible to easily fit into the rigid schema requirements of a relational database or you don't need the overhead of a full database, require very high-volume reads and writes as key values, or need to store only large blobs of data, then one of the other data stores might be a better fit.*" The inherent strength of a relational database is derived from its rigidity and conformance to clearly defined rules. This strength is also the weakness of the relational model if data to be stored is unstructured and varied in type.

Structured Query Language (SQL)

Structured Query Language originally 'SEQUEL' Structured English Query Language was developed by Donald D. Chamberlin and Raymond F. Boyce in the early 1970's. SEQUEL was developed to access data in IBM's System R relational database. (Chamberlin & Boyce, 1974). HawkerSiddley a UK based aircraft manufacturer had trademarked 'SEQUEL' forcing IBM to change the name of their new language to SQL (Oppel, 2004). The American National Standards Institute (ANSI) adopted SQL as a standard in 1986 followed by the International Organization for Standardization (ISO) in 1987 (Singer, 2014). SQL has become the language of the relational database and is almost universally supported (Harrington, 2003), SQL is therefore very portable across various RDBMS products (Oppel, 2004). SQL statements are divided into 4 categories.

• Data Query Language (DQL) – These statements query a database without altering the data.
• Data Manipulation Language (DML) – These statements are used for the modification of stored data.
• Data Definition Language (DDL) – These statements create and modify database objects. DDL statements work on database tables not the data the tables contain.
• Data Control Language (DCL) – These statements are used to manage data security and to grant and revoke user privileges.

(Oppel, 2004)

NoSQL Databases

Recently a new group has emerged to challenge the dominance of the RDBMS that use SQL as their query language of choice. These new solutions are grouped together under the general term 'NoSQL' databases but as will be seen vary enormously. Ironically as Sadalage and Fowler mention the term NoSQL was originally coined as the name of an open-source relational database – StrozziNoSQL by Carlo Strozzi in the late 90's.The name refers to the fact that the Strozzi database did not use SQL as its query language and has no bearing on any of the technologies this paper will discuss. The term NoSQL as used today can traced back to a request issued on the #cassandra IRC channel by Johan Oskarsson a London based software developer. Johan wanted suggestions on a name for a meet up he was organising to discuss these new type of databases, the winning suggestion was 'NoSQL' by Eric Evans a developer at RackSpace (Sadalage & Fowler, 2013). The origins of these NoSQl databases began with the introduction by Google of BigTable in 2006 and MapReduced in 2004 (Indrawan-Santiago, 2012) . A brief description of BigTable by Chang describes it as "a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers", which is as will be shown a description that could be used to describe many current NoSQL databases. It should be noted that the term NoSQL can be interpreted in different ways namely to mean

‘No’ SQL or as Martin Fowler explains ‘Not Only’ SQL, “If we take the "not-only" interpretation, then we should write "NOSQL" rather than "NoSQL" (Fowler, 2012), this paper will interpret NoSQL to mean ‘No’ SQL.

Shashank Tiwari gives a concise overview of the new type of data storage systems that have begun to emerge, he states “The growth of user-driven content has fuelled a rapid increase in the volume and type of data that is generated, manipulated, analysed, and archived. In addition, varied newer sets of sources, including sensors, Global Positioning Systems (GPS), automated trackers and monitoring systems, are generating a lot of data. These larger volumes of data sets, often termed *big data*, are imposing newer challenges and opportunities around storage, analysis, and archival. In parallel to the fast data growth, data is also becoming increasingly semi-structured and sparse. This means the traditional data management techniques around upfront schema definition and relational references is also being questioned. The quest to solve the problems related to large-volume and semi-structured data has led to the emergence of a class of newer types of database products. This new class of database products consists of column-oriented data stores, key/value pair databases, and document databases.” Along with the slightly different graph family these are the categories that will be further discussed in this paper.

Unlike the relational model which is totally reliant on the schema, NoSQL databases are commonly schemaless. Data storage is therefore much more flexible, easier to alter and un-structured data is less difficult to deal with (Sadalage & Fowler, 2013). According to Tiwari however “Flexibility comes at a price. NoSQL alleviates the problems that RDBMS imposes and makes it easy to work with large sparse data, but in turn takes away the power of transactional integrity and flexible indexing and querying.” Han suggests that NoSQL data-stores possess the following advantages over the relational model.

1. Extremely fast read and write operations.
2. Support for mass storage.
3. Easier to expand and scale
4. Lower cost.

Aggregate Orientation

The majority of NoSQL databases have been designed to run on clusters, the exception being the graph family which uses a similar distribution model to relational databases (Sadalage & Fowler, 2013). This cluster orientation has implications for the data model employed by these databases. RDBMS employ the relational data model, a set of tables with rows representing an entity, the entity is described in single value columns. NoSQL implementations have departed from this with each utilising a different model. Key-value data-stores, the document family and the column family all share one characteristic known as ‘aggregate orientation’ (Sadalage & Fowler, 2013).

Aggregate orientation allows data storage in a more complex way than simple tables and rows, allowing for lists and records to be nested inside other records, creating a complex record. Aggregates were first described by Evans as a “collection of related objects we wish to treat as a unit” (Evans, 2003) Working with aggregates makes it easier to operate on clustered data storage solutions as aggregates support replication(copying data) and sharding(splitting data across the cluster) (Sadalage & Fowler, 2013). The primary reason for aggregate orientation is that if running on a cluster “we need to minimize how many nodes we need to query when gathering data” (Sadalage & Fowler, 2013). Aggregates give important information about which data should be treated as belonging together and therefore should be located on the same node. Aggregates further provide for a limited amount of transaction control as the aggregate is the atomic unit for updates (Sadalage & Fowler, 2013).

There are some variations in the way different NoSQL database models deal with aggregation. Key-Value stores consider the aggregate as a whole, meaning queries are performed against the complete aggregate not any part or subset of it. Document oriented databases in contrast provide for partial queries but does not act on the structure to optimize storage or retrieval. Column family implementations divide aggregates into families of columns which provide structure that improves accessibility.

Map Reduce

Developed and patented by Google map reduce allows distributed processing large clustered data sets, map reduce has been adopted by many NoSQL implementations. Map and reduce are functions used by functional programming languages, a map applies a function to each element of a list without altering the original list, a reduce function is applied the results of a map function and returns a single result. Map and reduce functions are used to process lists of data, usually vast lists, the reduce function is normally an aggregate function (count, sum, average etc.)The concept was adapted to work with collections of rows or key value pairs, a map function is applied to the collection creating a new collection to which the reduce function is applied returning a single result (Tiwari, 2011).

Sharding and Replication

Sharding is the process of distributing subsets of data across multiple servers, each server being the single source of a subset (Sadalage, 2014). The simplest description is that a single data source is cut (sharded) into smaller chunks each chunk is stored separately.

Replication copies the complete set of data across multiple servers, ALL the data is accessible in multiple locations. There are two types of replication, 'Master-Slave' replication designates one node as master, the authoritative copy of the data. The master caters for any write operations. Slave nodes synchronize with this master and predominately handle read operations only. Peer-to-peer replication in contrast permits writes to ANY node in a cluster, the nodes communicate and synchronize their individual replicas. Each type of replication comes with its own primary advantage and disadvantage. Master-slave reduces the risk of conflicting updates as writes are only permitted to one node but creates a potential single point of failure. Peer-to-peer eliminates the danger of having a single point of failure in a system as writes are not exclusively to a single server but raises the possibility of conflicting updates occurring (Sadalage, 2014).

CAP Theorem and Eventual Consistency

Proposed by Professor Eric Brewer in 2000 CAP stands for Consistency, Availability and Partition tolerance. Brewer's theorem proposes that a distributed system is unable to meet these three needs simultaneously but can meet two of them.

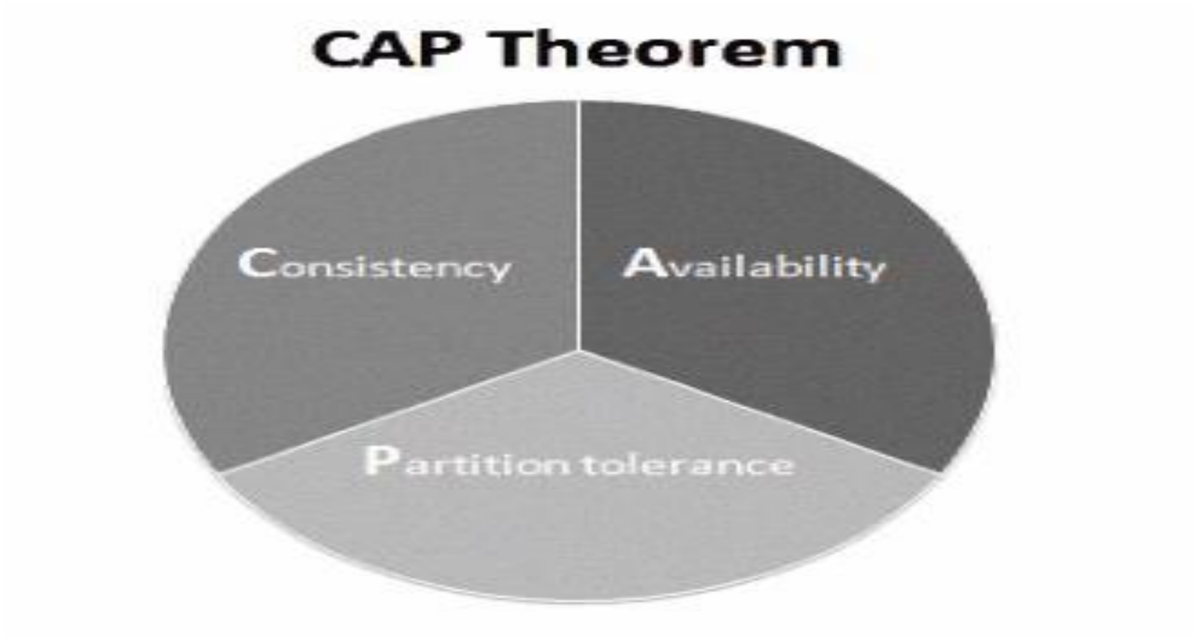
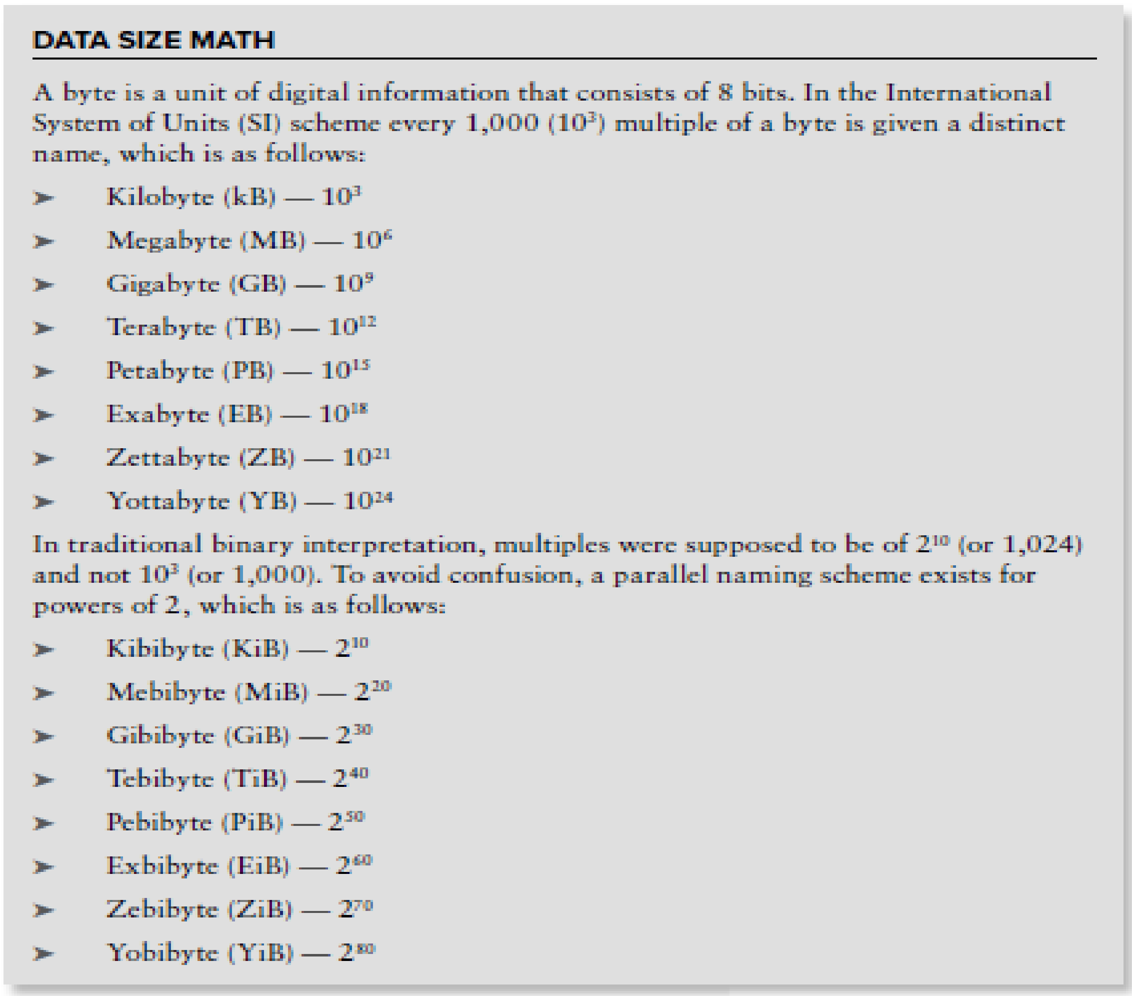


Fig 1.4: (Han, et al., 2011)

The relational model can be classified as CA, consistent and available (Han, et al., 2011). Various NoSQL implementations adhere to different combinations of CAP theory but they must be partition tolerant due to their distributed nature (Redmond & Wilson, 2012). This need for partition tolerance means that either consistency or availability must be sacrificed to some degree which can be a difficult choice. Many NoSQL databases assist in this choice by providing what is known as eventual consistency. Redmond and Wilson provide a simple explanation of eventual consistency in their book Seven Databases in Seven Weeks, "The idea behind eventual consistency is that each node is always available to serve requests. As a trade-off, data modifications are propagated in the background to other nodes. This means that at any time the system may be inconsistent, but the data is still largely accurate." Importantly eventual consistency is not inconsistency, it is simply weaker than consistency provided to RDBMS by their ACID properties (Tiwari, 2011).

Big Data

Prior to delving into the individual NoSQL families it is appropriate to give the reader an indication of what is meant by the term 'big data'.



DATA SIZE MATH

A byte is a unit of digital information that consists of 8 bits. In the International System of Units (SI) scheme every 1,000 (10^3) multiple of a byte is given a distinct name, which is as follows:

- Kilobyte (kB) — 10^3
- Megabyte (MB) — 10^6
- Gigabyte (GB) — 10^9
- Terabyte (TB) — 10^{12}
- Petabyte (PB) — 10^{15}
- Exabyte (EB) — 10^{18}
- Zettabyte (ZB) — 10^{21}
- Yottabyte (YB) — 10^{24}

In traditional binary interpretation, multiples were supposed to be of 2^{10} (or 1,024) and not 10^3 (or 1,000). To avoid confusion, a parallel naming scheme exists for powers of 2, which is as follows:

- Kibibyte (KiB) — 2^{10}
- Mebibyte (MiB) — 2^{20}
- Gibibyte (GiB) — 2^{30}
- Tebibyte (TiB) — 2^{40}
- Pebibyte (PiB) — 2^{50}
- Exbibyte (EiB) — 2^{60}
- Zebibyte (ZiB) — 2^{70}
- Yobibyte (YiB) — 2^{80}

Fig 1.5: (Tiwari, 2011)

There has been a relatively recent explosion of data in a variety of domains, the major challenge is how to take advantage of and manage data of this scale. Effectively exploiting this data requires scaling up and scaling out both infrastructure and standard techniques. "Our society is already data-rich, but the question remains whether or not we have the conceptual tools to handle it" (Dobre & Xhafa, 2014). Relational databases provide excellent support for structured data and will continue to model, store and access this data for the foreseeable future (Bogdan, 2013). The recent explosion in 'big data', large volumes, variety of data type and often lack of structure is for relational databases difficult because of the strict constraints they place on structure and relationships. Many NoSQL solutions are born from and are a growing area owing to this need in dealing with unstructured data (Zhao, et al., 2013).

NoSQL Database Types

Key – Value Stores

Key-Value stores are regarded as the simplest form of NoSQL to use with an API (Sadalage & Fowler, 2013). Key – Value stores are popular because they provide speedy access to data, the key is a unique value similar to the primary key in a relational database which provides easy search functionality (Tiwari, 2011). Key-Value stores pair keys to values, similar to a map or hashtable in most programming languages. Certain implementations permit complex types sets or lists for example, this is not a requirement (Redmond & Wilson, 2012). Key value databases are incredibly performant in certain scenarios but are generally not suited to complex query and aggregation needs (Redmond & Wilson, 2012).

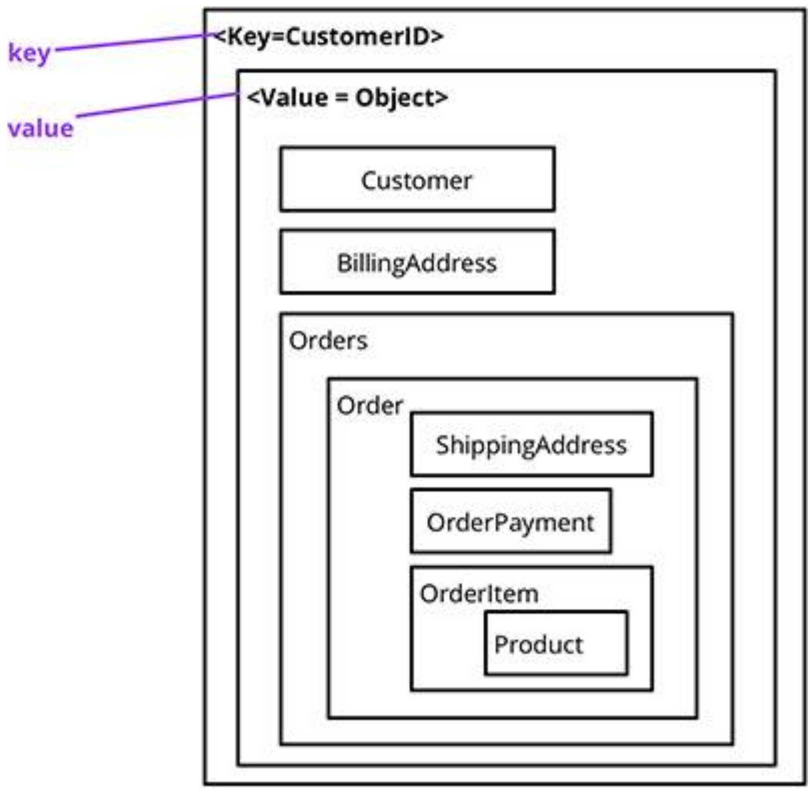


Fig 1.6: Representation of a key-value store (Sadalage, 2014).

Column Family

Column Family Stores can be referred to as column oriented stores, extensible record stores and wide columnar stores. Described by Kraska and Trushkowsky as “sparse, distributed, persistent multidimensional sorted maps” that are capable of storing arbitrary numbers of key value pairs in rows, somewhat similar to the rows of a relational database. Rows are subsequently stored in columns. These columns can be grouped into column families a feature important to data organization and partitioning (Hecht & Jablonski, 2011). Column family databases do not offer the flexibility of key values and document stores, it is possible to add columns at runtime, providing limited flexibility, but they are mostly predefined (Redmond & Wilson, 2012). Super columns consisting of multiple columns can be constructed and stored within columns families, creating a tabular representation close to that of a RDBMS. Null values and how they are dealt with are fundamentally different from RDBMS, relational databases store null values in every column of a dataset where no value is supplied. Column family data stores only store a key value pairs in one row, if and only if the dataset requires it. Termed "sparse" by Google this makes column family implementations efficient in domains involving huge volumes of data and attributes that vary in number (Hecht & Jablonski, 2011).

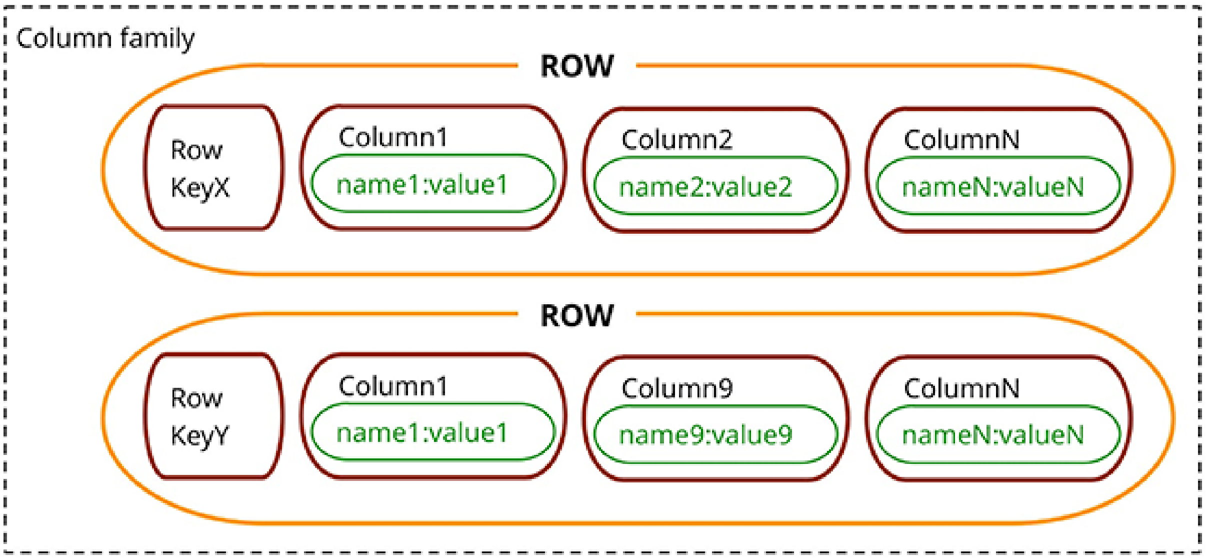
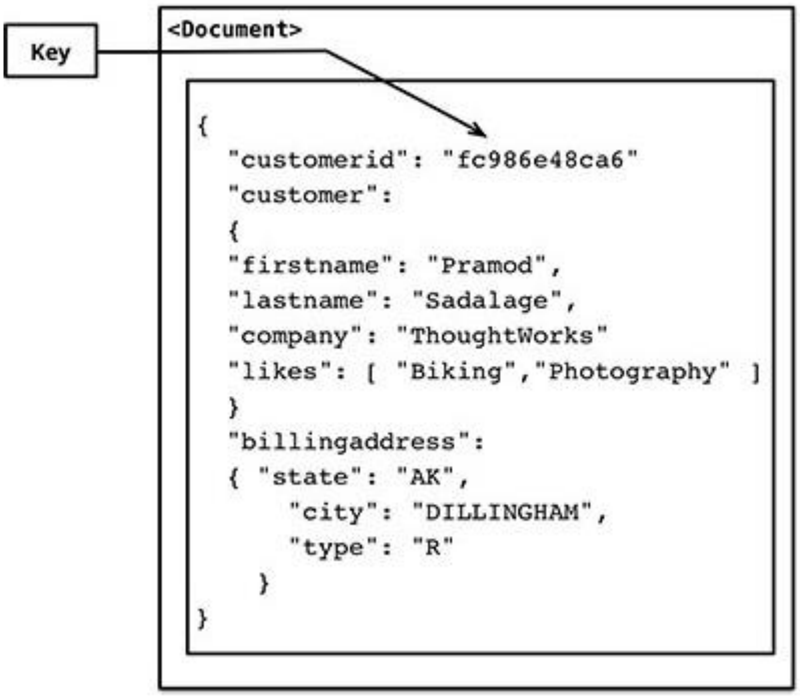


Fig 1.7: Column family database showing multiple columns associated with a row key (Sadalage, 2014).

Document Family

Document family databases store key value pairs in document format, where each key similarly to a primary key in a RDBMS must be unique. Documents also contain a unique ‘ID’ key, which explicitly identifies a document within a collection of documents. Unlike key-values stores values as well as keys can be queried which facilitates the use of complex data structures e.g. nested objects. These databases are in common with most NoSQL implementations schemaless, new documents can be stored by adding attributes to existing documents. Multi attribute lookups of documents consisting of different type key value pairs is possible, making document family systems useful for data integration and schema migrations (Hecht & Jablonski, 2011).



```
{
  "customerid": "fc986e48ca6"
  "customer":
  {
  "firstname": "Pramod",
  "lastname": "Sadalage",
  "company": "ThoughtWorks"
  "likes": [ "Biking","Photography" ]
  }
  "billingaddress":
  { "state": "AK",
      "city": "DILLINGHAM",
      "type": "R"
    }
}
```

Fig 1.8: How a document may look as stored in a document family database (Sadalage, 2014).

Graph Family

Graph databases bear a strong resemblance to common everyday structures e.g. road or rail networks. Described as ‘white-board friendly’ (Redmond & Wilson, 2012), graph databases are structures consisting of nodes, edges and properties used to represent and store data. Nodes are containers for data and edges represent the relationships between nodes.

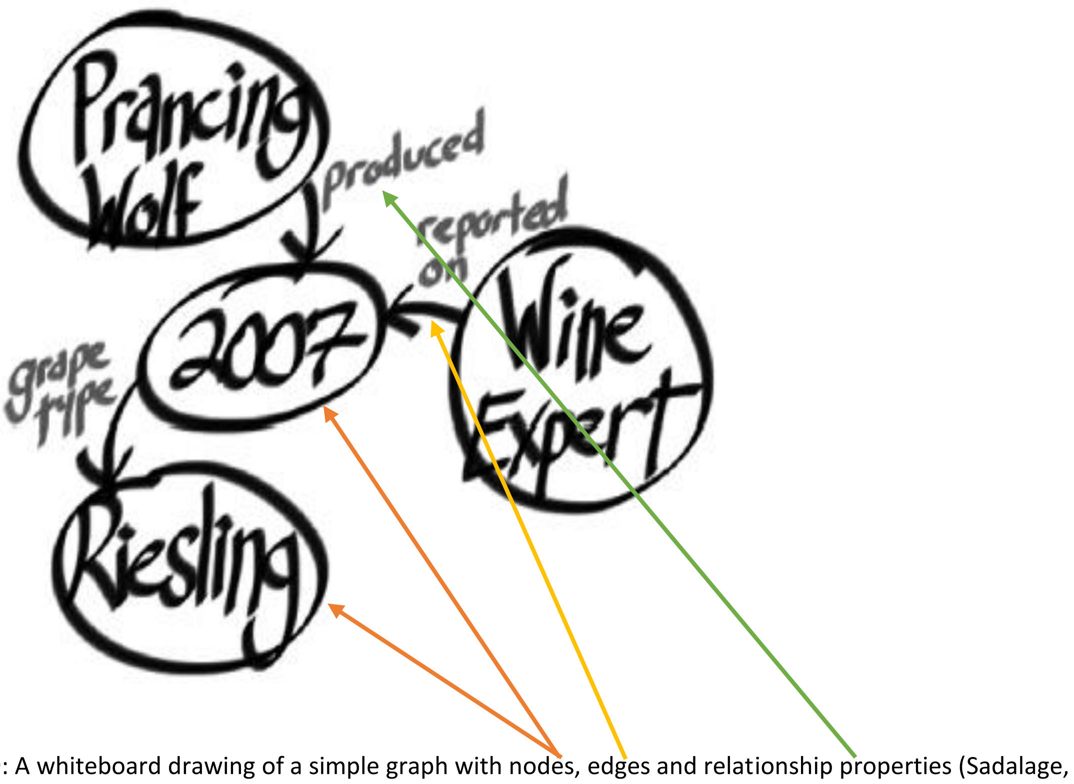


Fig 1.9: A whiteboard drawing of a simple graph with nodes, edges and relationship properties (Sadalage, 2014).

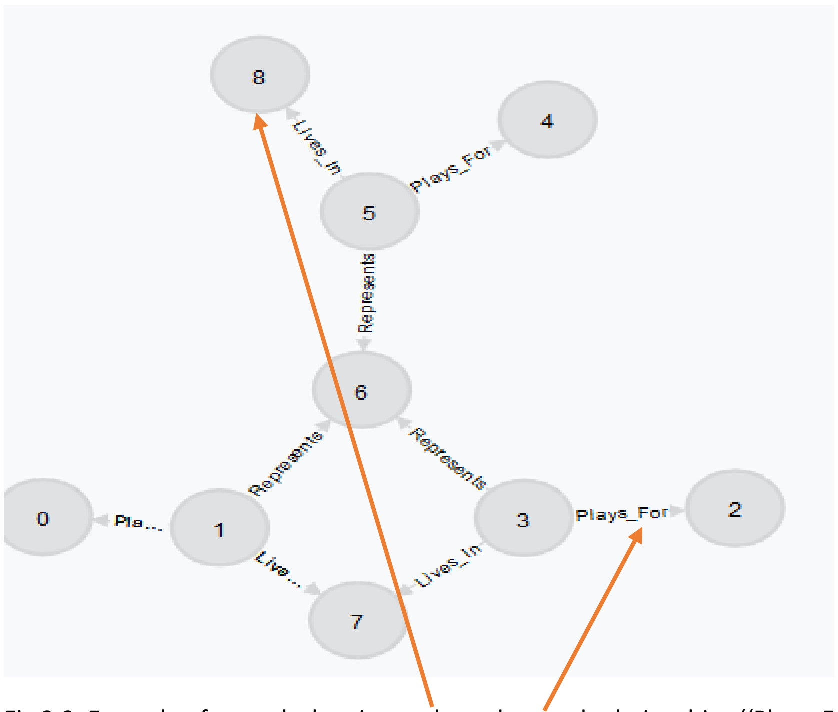


Fig 2.0: Example of a graph showing nodes, edges and relationships ('Plays_For','Lives_In'). (Authors own)

Graph databases are capable of dealing with extremely complex many to many relationships which other paradigms struggle with, making graph databases incredibly fast in dealing with associative data (Jayathilake, et al., 2012). Graph databases are very highly scalable and ad-hoc (unstructured) poses no difficulty making them very suitable for cases where relationships between data is as important as the data itself. They are heavily used in the world of social media

Polyglot Persistence

Traditionally data has been stored in a single repository, this remains a valid and successful strategy for most businesses. However there has been an increase in the type and usage of data by individual applications and within enterprises. Polyglot persistence describes the use of multiple data stores chosen, depending on the type and usage of data by the application. The concept behind polyglot persistence comes from polyglot programming – the use of multiple languages in a single project, most famously by Twitter, taking advantage of the strengths of each language (Redmond & Wilson, 2012). Polyglot persistent data storage systems utilize multiple data storage paradigms to perform various functions as part of an overall system for example a document database as the database of record, a key-value store for data population and as a cache and a graph database to handle relationships. The increase in polyglot persistent systems does not signal the demise of RDBMS systems, Sadalage argues "*All the choice provided by the rise of NoSQL databases does not mean the demise of RDBMS databases. We are entering an era of polyglot persistence, a technique that uses different data storage technologies to handle varying data storage needs.*" According to Brian F Cooper the creator of Yahoos Cloud Benchmarking System choosing the correct system is difficult and understanding performance implications is a challenge (Cooper, et al., 2010). One of the primary motivations of this paper is to ultimately create a system that can aid in the evaluation of data stores considering their potential usage in a polyglot persistent system.

Metrics

Industry Standard Benchmarks

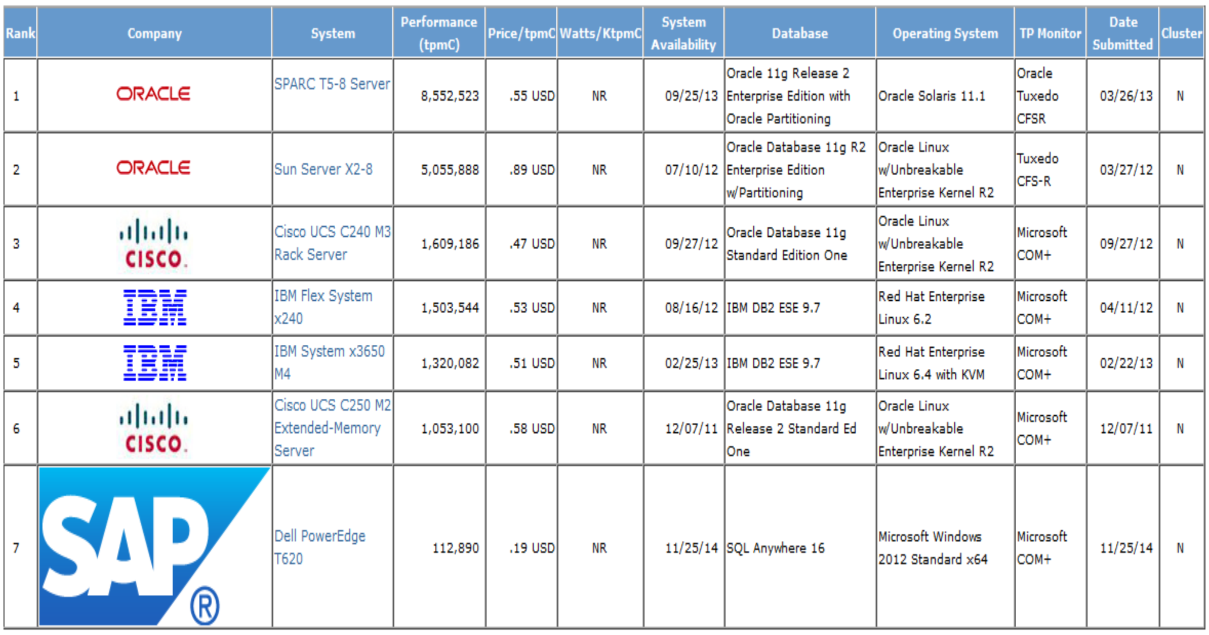
Database performance benchmarks provide standardised and important measures for the comparison of database management systems (Oracle Corporation, 2006). Benchmarking is the evaluation of a system against a known reference to ascertain the relative performance of that system. Database storage systems are integral to most organisations and businesses, deciding on the correct database with the most appropriate features is a critical decision. The increased need for polyglot persistence makes this decision even more challenging and highlights the need for benchmarking and performance comparison systems. Multiple benchmarking techniques have been developed to assist in making the correct choice.

TPC Benchmarks

The Transaction Processing Performance Council (TPC) provides a suite of benchmarks for the evaluation of transaction processing and database performance. TPC benchmarks evaluate an entire system not just the database(s) used by the system. The most relevant benchmarks are briefly outlined here and the latest available results for each benchmark shown. All TPC results displayed are from 05/12/2014.

TPC-C Benchmark

The TPC-C benchmark is an On-line Transaction Processing (OLTP) Benchmark. TPC-C utilises multiple types of concurrent transactions, a complex relational database and nine tables of differing record type and volumes of data. TPC-C simulates a multi-user environment with multiple concurrent queries of a centralized database. Performance is evaluated by monitoring all system state parameters and all system performance parameters. TPC-C is most applicable for businesses using databases to support online order processing, sales and inventory management.

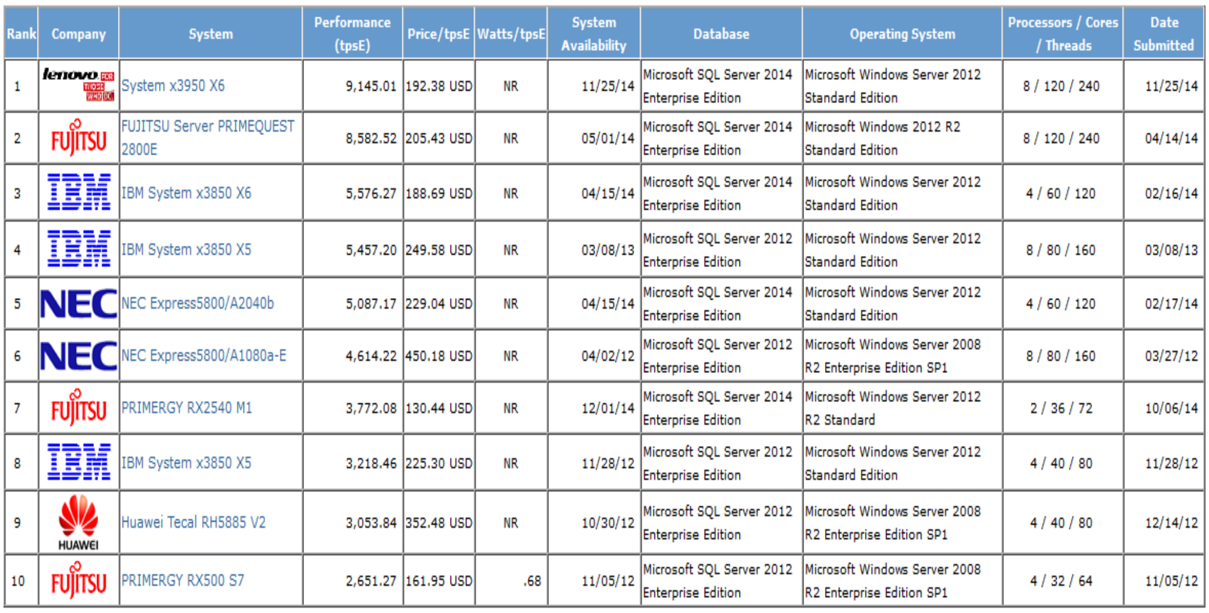


Rank	Company	System	Performance (tpmC)	Price/tpmC	Watts/KtpmC	System Availability	Database	Operating System	TP Monitor	Date Submitted	Cluster
1	ORACLE	SPARC T5-8 Server	8,552,523	.55 USD	NR	09/25/13	Oracle 11g Release 2 Enterprise Edition with Oracle Partitioning	Oracle Solaris 11.1	Oracle Tuxedo CFSR	03/26/13	N
2	ORACLE	Sun Server X2-8	5,055,888	.89 USD	NR	07/10/12	Oracle Database 11g R2 Enterprise Edition w/Partitioning	Oracle Linux w/Unbreakable Enterprise Kernel R2	Tuxedo CFS-R	03/27/12	N
3	CISCO	Cisco UCS C240 M3 Rack Server	1,609,186	.47 USD	NR	09/27/12	Oracle Database 11g Standard Edition One	Oracle Linux w/Unbreakable Enterprise Kernel R2	Microsoft COM+	09/27/12	N
4	IBM	IBM Flex System x240	1,503,544	.53 USD	NR	08/16/12	IBM DB2 ESE 9.7	Red Hat Enterprise Linux 6.2	Microsoft COM+	04/11/12	N
5	IBM	IBM System x3650 M4	1,320,082	.51 USD	NR	02/25/13	IBM DB2 ESE 9.7	Red Hat Enterprise Linux 6.4 with KVM	Microsoft COM+	02/22/13	N
6	CISCO	Cisco UCS C250 M2 Extended-Memory Server	1,053,100	.58 USD	NR	12/07/11	Oracle Database 11g Release 2 Standard Ed One	Oracle Linux w/Unbreakable Enterprise Kernel R2	Microsoft COM+	12/07/11	N
7	SAP	Dell PowerEdge T620	112,890	.19 USD	NR	11/25/14	SQL Anywhere 16	Microsoft Windows 2012 Standard x64	Microsoft COM+	11/25/14	N

(Transaction Processing Performance Council, 2014)

TPC-E Benchmark

TPC-E is also an OLTP benchmark designed specifically for the evaluation of database systems installed by brokerage (facilitates transactions of financial securities) firms. TPC-E and TPC-C benchmark are identical in setup and components with only transaction design to differentiate them. TPC-E transactions hold more relevance for brokerage firms and include account and online trading transactions.

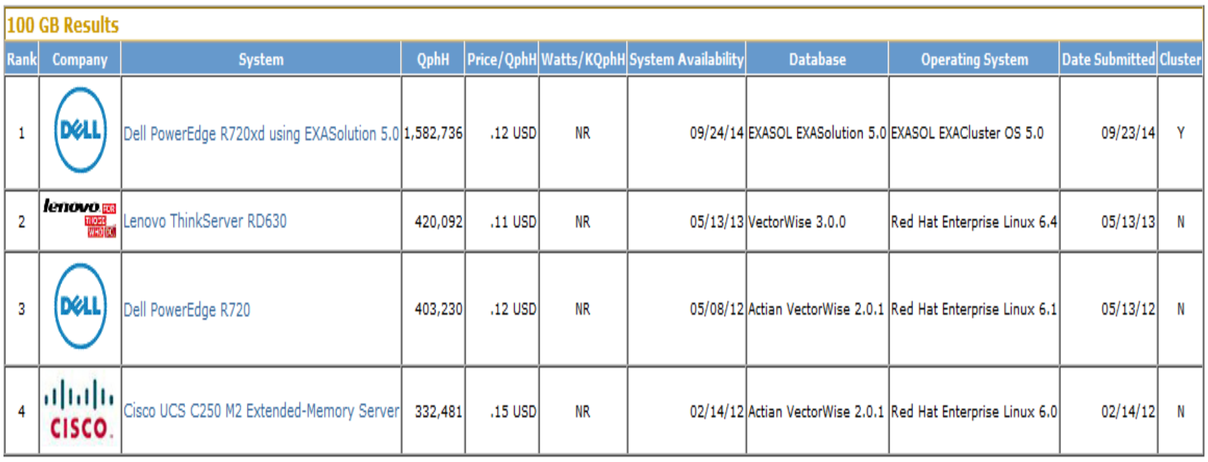


Rank	Company	System	Performance (tpsE)	Price/tpsE	Watts/tpsE	System Availability	Database	Operating System	Processors / Cores / Threads	Date Submitted
1	lenovo	System x3950 X6	9,145.01	192.38 USD	NR	11/25/14	Microsoft SQL Server 2014 Enterprise Edition	Microsoft Windows Server 2012 Standard Edition	8 / 120 / 240	11/25/14
2	FUJITSU	FUJITSU Server PRIMEQUEST 2800E	8,582.52	205.43 USD	NR	05/01/14	Microsoft SQL Server 2014 Enterprise Edition	Microsoft Windows 2012 R2 Standard Edition	8 / 120 / 240	04/14/14
3	IBM	IBM System x3850 X6	5,576.27	188.69 USD	NR	04/15/14	Microsoft SQL Server 2014 Enterprise Edition	Microsoft Windows Server 2012 Standard Edition	4 / 60 / 120	02/16/14
4	IBM	IBM System x3850 X5	5,457.20	249.58 USD	NR	03/08/13	Microsoft SQL Server 2012 Enterprise Edition	Microsoft Windows Server 2012 Standard Edition	8 / 80 / 160	03/08/13
5	NEC	NEC Express5800/A2040b	5,087.17	229.04 USD	NR	04/15/14	Microsoft SQL Server 2014 Enterprise Edition	Microsoft Windows Server 2012 Standard Edition	4 / 60 / 120	02/17/14
6	NEC	NEC Express5800/A1080a-E	4,614.22	450.18 USD	NR	04/02/12	Microsoft SQL Server 2012 Enterprise Edition	Microsoft Windows Server 2008 R2 Enterprise Edition SP1	8 / 80 / 160	03/27/12
7	FUJITSU	PRIMERGY RX2540 M1	3,772.08	130.44 USD	NR	12/01/14	Microsoft SQL Server 2014 Enterprise Edition	Microsoft Windows Server 2012 R2 Standard	2 / 36 / 72	10/06/14
8	IBM	IBM System x3850 X5	3,218.46	225.30 USD	NR	11/28/12	Microsoft SQL Server 2012 Enterprise Edition	Microsoft Windows Server 2012 Standard Edition	4 / 40 / 80	11/28/12
9	HUAWEI	Huawei Tecal RH5885 V2	3,053.84	352.48 USD	NR	10/30/12	Microsoft SQL Server 2012 Enterprise Edition	Microsoft Windows Server 2008 R2 Enterprise Edition SP1	4 / 40 / 80	12/14/12
10	FUJITSU	PRIMERGY RX500 S7	2,651.27	161.95 USD	.68	11/05/12	Microsoft SQL Server 2012 Enterprise Edition	Microsoft Windows Server 2008 R2 Enterprise Edition SP1	4 / 32 / 64	11/05/12

(Transaction Processing Performance Council, 2014)

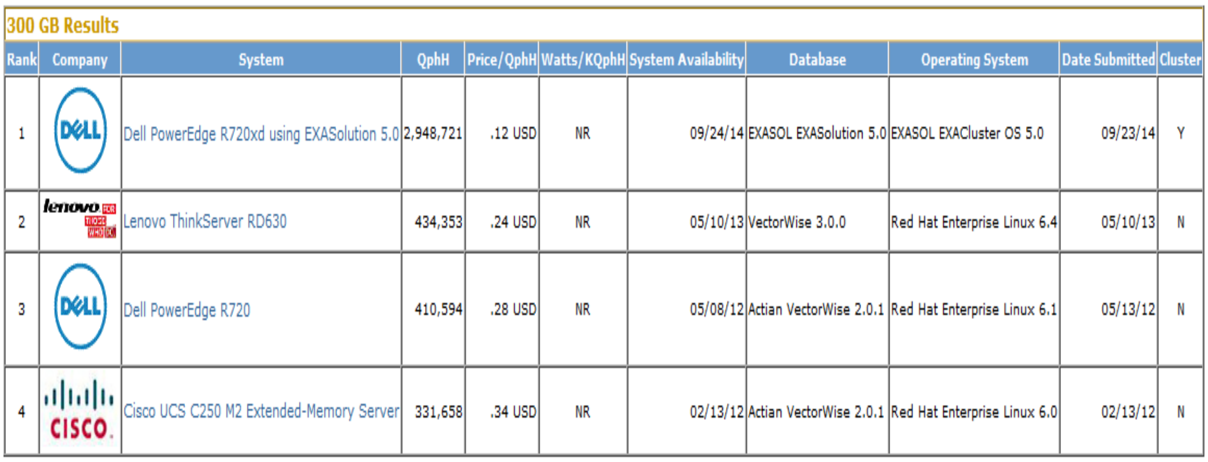
TPC-H Benchmark

The TPC-H benchmark is optimized for decision support systems and business intelligence systems. Transactions include extremely complex data-mining queries and concurrent data modification queries often involving huge amounts of data. TPC-H provides a complex composite query which is measured per hour to provide the required performance metric.



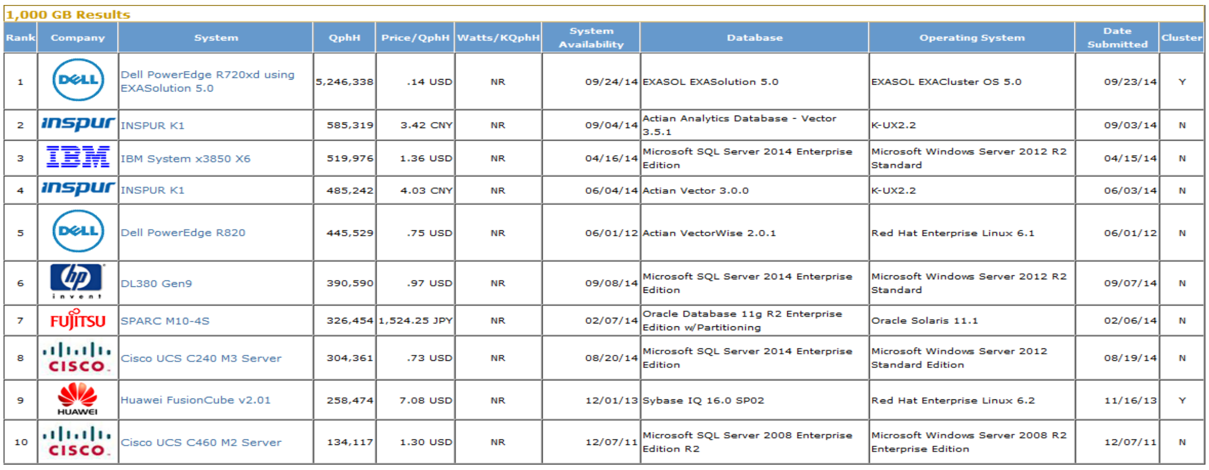
100 GB Results

Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operating System	Date Submitted	Cluster
1	DELL	Dell PowerEdge R720xd using EXASolution 5.0	1,582,736	.12 USD	NR	09/24/14	EXASOL EXASolution 5.0	EXASOL EXACluster OS 5.0	09/23/14	Y
2	lenovo	Lenovo ThinkServer RD630	420,092	.11 USD	NR	05/13/13	VectorWise 3.0.0	Red Hat Enterprise Linux 6.4	05/13/13	N
3	DELL	Dell PowerEdge R720	403,230	.12 USD	NR	05/08/12	Actian VectorWise 2.0.1	Red Hat Enterprise Linux 6.1	05/13/12	N
4	CISCO	Cisco UCS C250 M2 Extended-Memory Server	332,481	.15 USD	NR	02/14/12	Actian VectorWise 2.0.1	Red Hat Enterprise Linux 6.0	02/14/12	N



300 GB Results

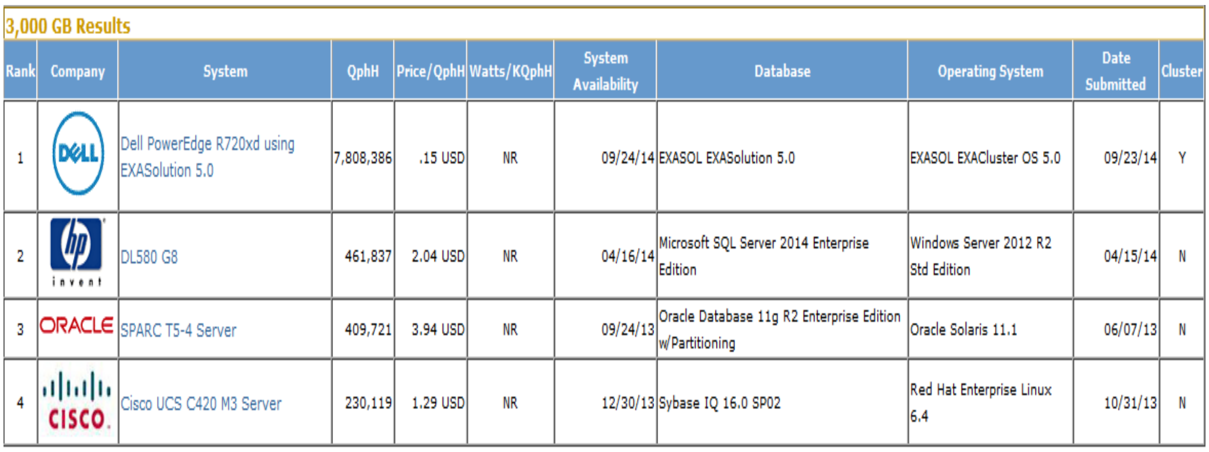
Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operating System	Date Submitted	Cluster
1	DELL	Dell PowerEdge R720xd using EXASolution 5.0	2,948,721	.12 USD	NR	09/24/14	EXASOL EXASolution 5.0	EXASOL EXACluster OS 5.0	09/23/14	Y
2	lenovo	Lenovo ThinkServer RD630	434,353	.24 USD	NR	05/10/13	VectorWise 3.0.0	Red Hat Enterprise Linux 6.4	05/10/13	N
3	DELL	Dell PowerEdge R720	410,594	.28 USD	NR	05/08/12	Actian VectorWise 2.0.1	Red Hat Enterprise Linux 6.1	05/13/12	N
4	CISCO	Cisco UCS C250 M2 Extended-Memory Server	331,658	.34 USD	NR	02/13/12	Actian VectorWise 2.0.1	Red Hat Enterprise Linux 6.0	02/13/12	N



1,000 GB Results

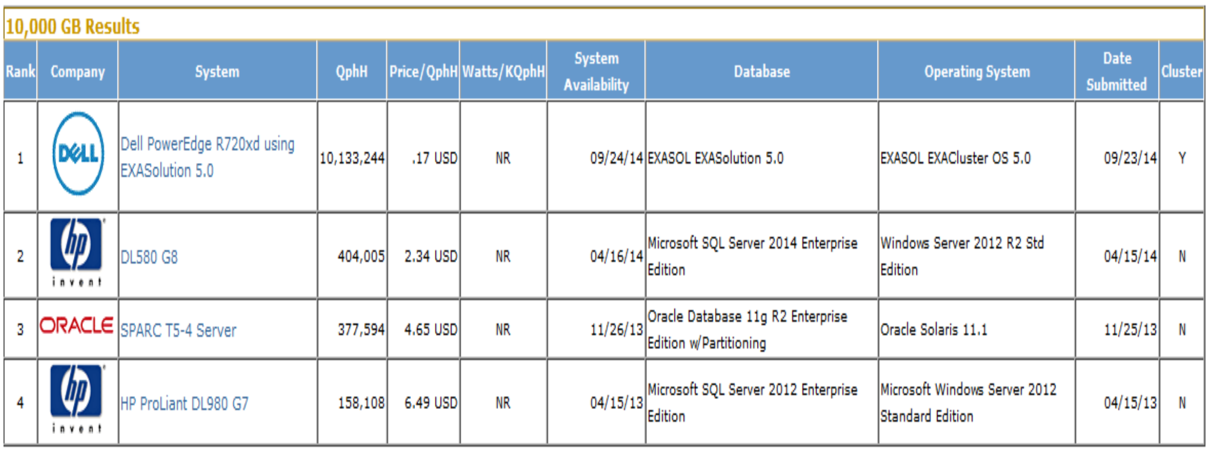
Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operating System	Date Submitted	Cluster
1	DELL	Dell PowerEdge R720xd using EXASolution 5.0	5,246,338	.14 USD	NR	09/24/14	EXASOL EXASolution 5.0	EXASOL EXACluster OS 5.0	09/23/14	Y
2	inspur	INSPUR K1	585,319	3.42 CNY	NR	09/04/14	Actian Analytics Database - Vector 3.5.1	K-UX2.2	09/03/14	N
3	IBM	IBM System x3850 X6	519,976	1.36 USD	NR	04/16/14	Microsoft SQL Server 2014 Enterprise Edition	Microsoft Windows Server 2012 R2 Standard	04/15/14	N
4	inspur	INSPUR K1	485,242	4.03 CNY	NR	06/04/14	Actian Vector 3.0.0	K-UX2.2	06/03/14	N
5	DELL	Dell PowerEdge R820	445,529	.75 USD	NR	06/01/12	Actian VectorWise 2.0.1	Red Hat Enterprise Linux 6.1	06/01/12	N
6	hp invent	DL380 Gen9	390,590	.97 USD	NR	09/08/14	Microsoft SQL Server 2014 Enterprise Edition	Microsoft Windows Server 2012 R2 Standard	09/07/14	N
7	FUJITSU	SPARC M10-4S	326,454	1,524.25 JPY	NR	02/07/14	Oracle Database 11g R2 Enterprise Edition w/Partitioning	Oracle Solaris 11.1	02/06/14	N
8	CISCO	Cisco UCS C240 M3 Server	304,361	.73 USD	NR	08/20/14	Microsoft SQL Server 2014 Enterprise Edition	Microsoft Windows Server 2012 Standard Edition	08/19/14	N
9	HUAWEI	Huawei FusionCube v2.01	258,474	7.08 USD	NR	12/01/13	Sybase IQ 16.0 SP02	Red Hat Enterprise Linux 6.2	11/16/13	Y
10	CISCO	Cisco UCS C460 M2 Server	134,117	1.30 USD	NR	12/07/11	Microsoft SQL Server 2008 Enterprise Edition R2	Microsoft Windows Server 2008 R2 Enterprise Edition	12/07/11	N

(Transaction Processing Performance Council, 2014)



3,000 GB Results

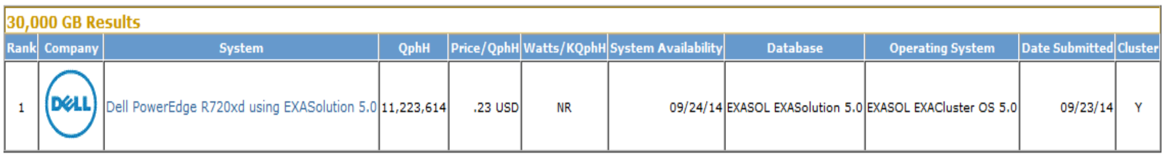
Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operating System	Date Submitted	Cluster
1	DELL	Dell PowerEdge R720xd using EXASolution 5.0	7,808,386	.15 USD	NR	09/24/14	EXASOL EXASolution 5.0	EXASOL EXACluster OS 5.0	09/23/14	Y
2	hp invent	DL580 G8	461,837	2.04 USD	NR	04/16/14	Microsoft SQL Server 2014 Enterprise Edition	Windows Server 2012 R2 Std Edition	04/15/14	N
3	ORACLE	SPARC T5-4 Server	409,721	3.94 USD	NR	09/24/13	Oracle Database 11g R2 Enterprise Edition w/Partitioning	Oracle Solaris 11.1	06/07/13	N
4	CISCO	Cisco UCS C420 M3 Server	230,119	1.29 USD	NR	12/30/13	Sybase IQ 16.0 SP02	Red Hat Enterprise Linux 6.4	10/31/13	N



10,000 GB Results

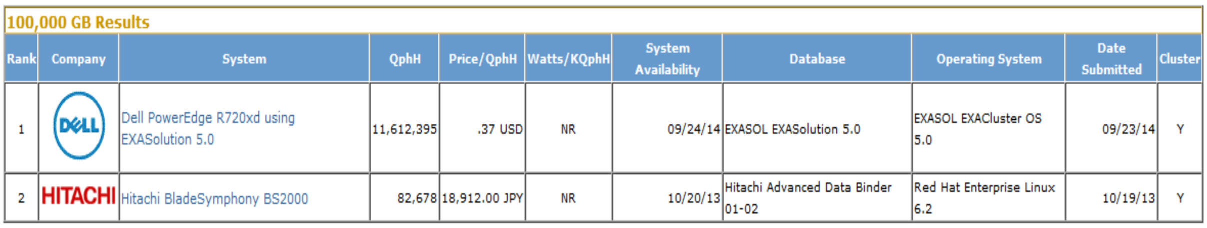
Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operating System	Date Submitted	Cluster
1	DELL	Dell PowerEdge R720xd using EXASolution 5.0	10,133,244	.17 USD	NR	09/24/14	EXASOL EXASolution 5.0	EXASOL EXACluster OS 5.0	09/23/14	Y
2	hp invent	DL580 G8	404,005	2.34 USD	NR	04/16/14	Microsoft SQL Server 2014 Enterprise Edition	Windows Server 2012 R2 Std Edition	04/15/14	N
3	ORACLE	SPARC T5-4 Server	377,594	4.65 USD	NR	11/26/13	Oracle Database 11g R2 Enterprise Edition w/Partitioning	Oracle Solaris 11.1	11/25/13	N
4	hp invent	HP ProLiant DL980 G7	158,108	6.49 USD	NR	04/15/13	Microsoft SQL Server 2012 Enterprise Edition	Microsoft Windows Server 2012 Standard Edition	04/15/13	N

(Transaction Processing Performance Council, 2014)



30,000 GB Results

Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operating System	Date Submitted	Cluster
1	DELL	Dell PowerEdge R720xd using EXASolution 5.0	11,223,614	.23 USD	NR	09/24/14	EXASOL EXASolution 5.0	EXASOL EXACluster OS 5.0	09/23/14	Y



100,000 GB Results

Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operating System	Date Submitted	Cluster
1	DELL	Dell PowerEdge R720xd using EXASolution 5.0	11,612,395	.37 USD	NR	09/24/14	EXASOL EXASolution 5.0	EXASOL EXACluster OS 5.0	09/23/14	Y
2	HITACHI	Hitachi BladeSymphony BS2000	82,678	18,912.00 JPY	NR	10/20/13	Hitachi Advanced Data Binder 01-02	Red Hat Enterprise Linux 6.2	10/19/13	Y

(Transaction Processing Performance Council, 2014)

TPCx-HS Benchmark

TPC has recognised that 'big-data' technologies have become hugely important in modern enterprises. TPC Express Benchmark HS (TPCx-HS) has been developed to objectively measure hardware, operating system and Apache Hadoop File System API compatible software distributions. TPCx-HS aims to provide verifiable performance, price v's performance and availability metrics for such systems. TPCx-HS simulates a continuously available system (24 hours per day, 7 days per week). The simulated system is relatively simple and is claimed to generate results relevant to any ‘big data’ system, TPCx-HS stresses both hardware and software through use of a heavy workload. Theoretically TPCx-HS can evaluate multiple system topologies and implementations while remaining vendor independent. Unfortunately no results of TPCx-HS benchmarking tests have been published to date.

Yahoo Cloud Services Benchmark (YCSB)

In 2010 a group of researchers from Yahoo began to investigate NoSQL systems, from their research a framework developed to assess the performance of NoSQL databases. The framework subsequently named the Yahoo Cloud Services Benchmark provides standard scenarios and workloads as part of a cloud service testing client (Barata, et al., 2014).

The cloud service testing client is comprised of two parts a workload generator and a set of scenarios (workloads) that perform reads, write and update operations.

YCSB predefines these workloads as:

- Workload A: Update heavy a combination of 50% reads and 50% updates.
- Workload B: Read heavy, 95% reads and 5% of updates.
- Workload C: Read only, self-explanatory 100% read.
- Workload D: Read latest, 95% reads and 5% inserts.
- Workload E: Short ranges, 95% scans and 5% inserts.
- Workload F: Read-modify-write, the client reads a record, modifies it and writes the modifications.

(Barata, et al., 2014).

YCSB runs from the command line and is capable of creating an arbitrary number of threads that will query the system under test. It measures throughput in operations per second and records the latency in performing these operations. The YCSB benchmark builds tiers that evaluate performance, scalability, elasticity, availability, and replication.

Wisconsin Benchmark

Developed in 1981 the Wisconsin benchmark tests the performance of the major components of relational databases. The benchmark was designed in a way that made the addition of new queries to relations very straightforward (DeWitt, n.d.). The Wisconsin benchmark was superseded by TPC BM A and holds no further relevance for this paper.

Research Question 1

Are current database benchmarks and benchmarking tools adequate for evaluation of multiple data storage systems?

What is noticeable about the most popular benchmarking systems is that they are in general heavyweight, there are no vendor independent ‘quick’ comparison/benchmarking systems on the market. Furthermore these benchmarks predominately test both the database element of a system and the system itself, benchmarks that are independent of commercial interests which evaluate just the database are extremely rare. Research has also highlighted the lack of a single multiple database benchmark, where NoSQL and at least some relational models can be measured and compared to each other using a standard data source and standard methodology. Examination of the most up to date TPC results available show that most systems benchmarked have RDBMS at their core, the only TPC benchmark (TPCx-HS) developed with NoSQL and ‘big data’ in mind has to date no published results. Considering a company or business that wishes to view benchmark results for multiple database types TPC comes with severe limitations. YCSB is best describes as a tool/framework used to evaluate databases as opposed to a specific set of metrics, research has shown that sourcing comprehensive and up to date results from YCSB usage is difficult and very vendor dependant i.e. results can be portrayed to suit particular products. Based on these findings it is suggested that current benchmarks and benchmarking tools are inadequate for evaluation of multiple data storage systems.

Research Question 2

Can alternative or adapted metrics be suggested?

As previously stated Han suggests that NoSQL data-stores possess the following advantages over the relational model.

1. Extremely fast read and write operations.
2. Support for mass storage.
3. Easier to expand and scale
4. Lower cost.

Excluding lower cost because most NoSQL databases are open source and implicitly lower cost these suggestions are the basis of what needs to be evaluated.

What to Measure

Prior to discussing what metrics will be evaluated, the databases to be evaluated must be specified. Giving as broad a range of results as possible is important, to this end one implementation from each family is chosen.

Key-Value store – Redis.

Column family – Cassandra.

Document family – MongoDB.

Graph family – Neo4J.

Traditional RDBMS – MySQL.

While the system is under development it is envisaged that data will be stored in the most basic format possible i.e. not using complex data structures. This basic format will ease portability across the multiple databases under evaluation.

Throughput – records retrieved (read) or saved (written) across a fixed time interval – is the most common measure of database performance (Oracle Corporation, 2006). Performance and comparison of performance therefore is the most basic metric that must be measured. Performance implies speed thus the speed of insert/save and retrieve/find queries will be evaluated using a standard data set.

Scalability and elasticity are features of NoSQL database systems that must be measured, these features are often confused. Scalability is a 'time-free' concept that deals with heavier workloads by adding more computing power to a system in a linear manner. Scalability does not capture the time taken to reach the performance levels required (Sakr & Liu, 2013). Elasticity in contrast is extremely time oriented and dynamic, resources can be added to a running system, elasticity is dependent on speed of response to changed workloads, as described by Sakr and Liu, "*a truly elastic database system must be able to dynamically and rapidly allocate and de-allocate computing resources to a running system to handle changing workloads in a graceful manner without significantly interrupting the system performance".* Scalability and its impact on performance particularly relating to the NoSQL implementations is extremely important. Running the databases on a cluster, will changes in the cluster effect the performance of the database? Elasticity will be measured in line with two existing metrics, speed-up and scale-up. Speed-up is the time required for performance to stabilize once dynamic allocation of additional resources is complete. Scale-up is a measure of gains in throughput resulting from the dynamic allocation of additional resources, it is representative of the deviation from linear scalability (Sakr & Liu, 2013). The key difference is that scalability measures deviations in performance when new resources are added or removed statically e.g. the system in not running when new servers are added, elasticity adds or removes existing resources while the system is active e.g. increase the availability of existing servers to a system based on increased workload or data volume.

What data to use

It is proposed that the set of data used in evaluations will be a data dump of Wikipedia (English version) available at http://dumps.wikimedia.org/enwiki/20140203/. The primary motivations for using this source are its relatively manageable size (circa 10,000,000 articles, ~28 GB), its free availability and the inclusion of comprehensive instructions on how to import the entire contents to a number of database types running on a Linux system http://meta.wikimedia.org/wiki/Data_dumps/Import_examples. The data dump consists of every unique article on Wikipedia, individual articles will be stored using an integer value as id/key. Using a combination of id/key and a value (article) fits very well with the various data models chosen for initial evaluation. The alternative to using id/key and a unique Wikipedia article as a value is to use an integer as id/key and a BLOB (Binary Large Object) that models an entity, for example a student with name, student number and course identification number. Deciding on how data will be stored and tested will be finalised during the early stages of implementation. Using Wikipedia data dumps will smooth the process of measuring scalability and elasticity, dumps of varying volumes are available and will be used to simulate the need for more or less storage and increasing, decreasing volume of queries.

Methodology

Research Methodology

The methodology applied to the research section of this paper follows a pattern of explanation, findings and alternatives where necessary. Databases form the largest research section, it was decided to explain in detail relational databases, the rules and structures that govern them in this section. Without clearly explaining these rules it was felt that the reader would not get a sense of the rigidity of RDBMS, understanding this rigidity is vital in the context of comparing RDBMS to NoSQL systems. SQL its originations and evolution are briefly discussed followed by an overview of the various NoSQL database types, the brevity of these explanations serves to illustrate their simplicity in comparison with RDBMS. The most pertinent additional features that NoSQL databases offer are explained with a lengthier discussion on polyglot systems. One of the primary motivations for conducting this research is the emergence of multiple database (polyglot) systems as mentioned in this section. Current benchmarking techniques and their suggested weaknesses constitute the next chapter of the research document, again brief explanations are given, faults detailed as a conclusion and alternatives proposed. Faults and weaknesses of current industry standard benchmarking techniques identified by the research in conjunction with the emergence of multiple database types are the justification for development of a database evaluation system.

Design Methodology

Source Control

The source control system chosen for this project is Git using an account created on BitBucket with SourceTree utilised as the GUI of choice on development machine. Primarily Git/Bitbucket have been chosen because of their distributed nature, meaning the files and folders contained in a Git repository hosted by BitBucket are accessible from multiple locations. Use of a distributed system such as Git also provides a huge element of security and safety from a data redundancy perspective. SourceTree as a GUI reduces the need to interact with Git via the command line and subsequently learning the full range of Git commands. Previous experience using Git via command line also highlighted a number of inherently unsafe commands that can inadvertently be entered resulting in catastrophic data loss, this is to be avoided at all costs. All documents will be placed under source control as well as any code required by the project.

Sprint Monitoring

Sprint tasks will be managed and monitored using Trello a free software tool that implements a ‘kanban’ board. Trello https://trello.com/ provides excellent visualisation of time and how tasks relate to time and should prove invaluable from a time management perspective. Burndown charts are an excellent means of monitoring progress towards a target along with velocity and trajectory of work. BurndownforTrello https://www.burndownfortrello.com/# is a free burndown chart tool that integrates with Trello and will also be used to monitor sprints. Both tools have been chosen as a result of previous experience using them and their free availability.

Web Services

The system under development will have a number of components and tools that require communication across a network. Provision of a RESTful web service is the most logical and straightforward way to implement this communication. Many of the databases proposed for usage by the system support REST as a communication medium as standard, greatly reducing overall complexity.

Placeholder.

Repository Pattern

Fowler states that a repository, "encapsulates the set of objects persisted in a data store and the operations performed over them, providing a more object-oriented view of the persistence layer". Repository further supports the objective of achieving abstraction of the data mapping layers away from the domain logic (Fowler, 2011). What the repository specifically supplies is 'persistence ignorance' it provides a clear separation between the data layer and the rest of the application. Multiple repositories are possible within an application each dealing with a specific context, in this instance a different database. The application calls methods of the required repository to save and find objects that are persisted by the data layer (Fowler, 2011). Best practice when using the repository pattern is to code against an interface, this supports multiple implementations, eases testing and any changes to implementation. Mocks or stubs are created when test removing the need to implement actual database access. The application under test works with the interface, ignorant of how persistence is handled.

Aspect Oriented Programming

Placeholder.

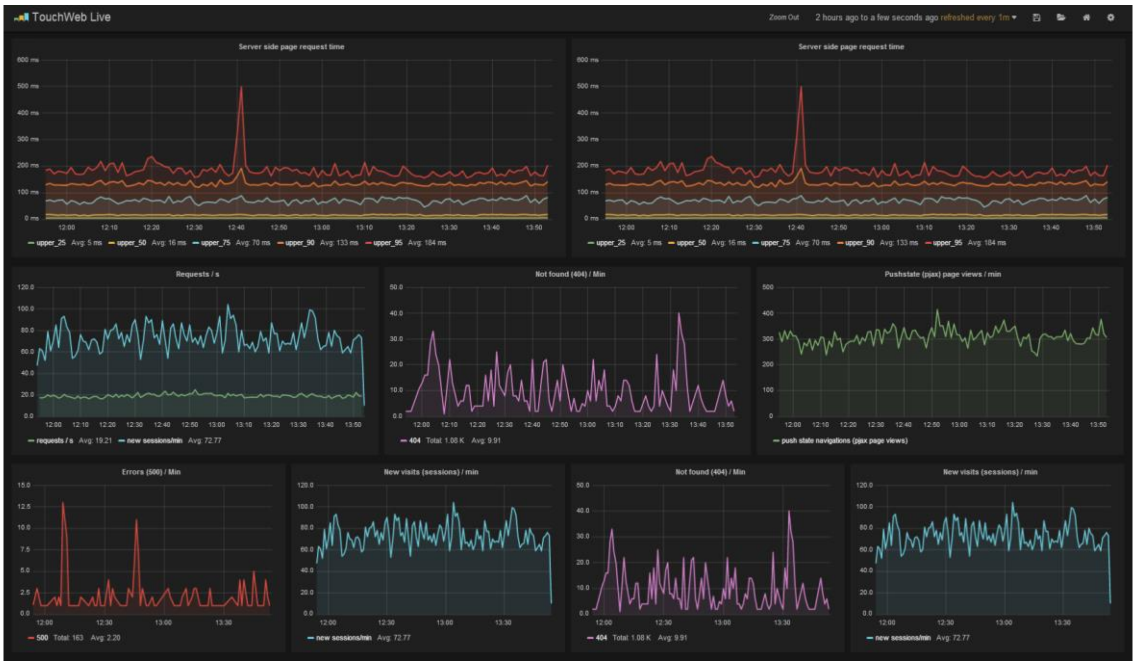
Software tools

Graphing tools

To-Do

Graphite

Grafana

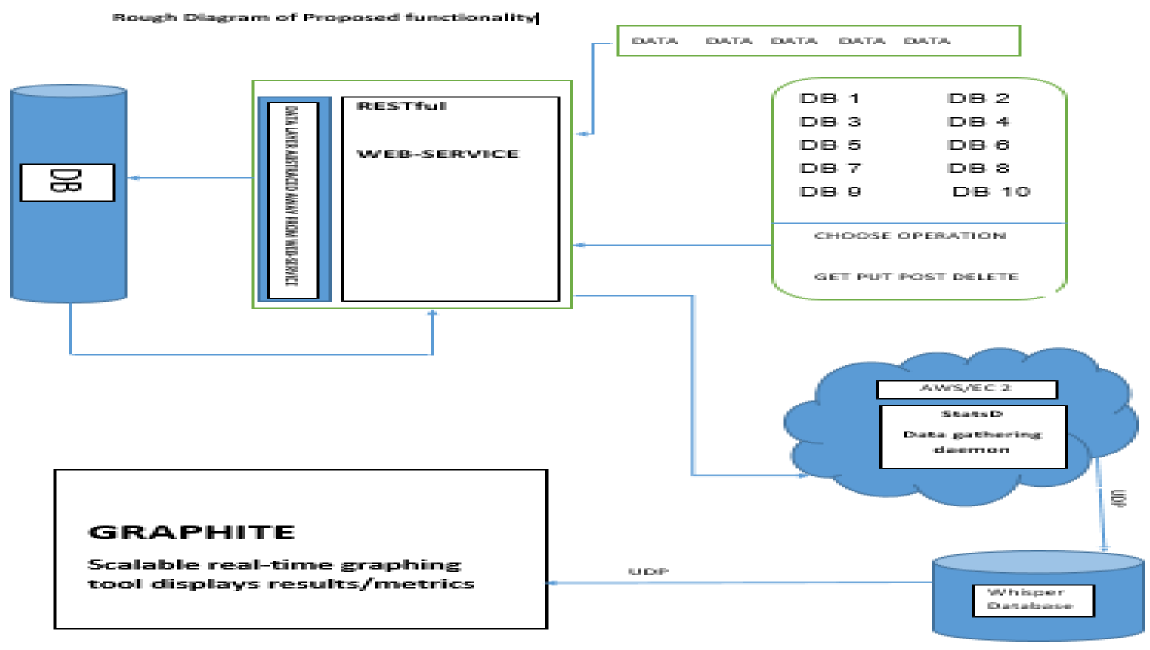

TouchWeb Live
Zoom Out
2 hours ago to a few seconds ago refreshed every 1m
Server side page request time
upper_25 Avg: 5 ms
upper_50 Avg: 16 ms
upper_75 Avg: 70 ms
upper_90 Avg: 133 ms
upper_95 Avg: 184 ms
Server side page request time
upper_25 Avg: 5 ms
upper_50 Avg: 16 ms
upper_75 Avg: 70 ms
upper_90 Avg: 133 ms
upper_95 Avg: 184 ms
Requests / s
requests / s Avg: 19.21
new sessions/min Avg: 72.77
Not found (404) / Min
404 Total: 1.08 K Avg: 9.91
Pushstate (pjax) page views / min
push state navigations (pjax page views)
Errors (500) / Min
500 Total: 163 Avg: 2.20
New visits (sessions) / min
new sessions/min Avg: 72.77
Not found (404) / Min
404 Total: 1.08 K Avg: 9.91
New visits (sessions) / min
new sessions/min Avg: 72.77

(Ödegaard, 2013).

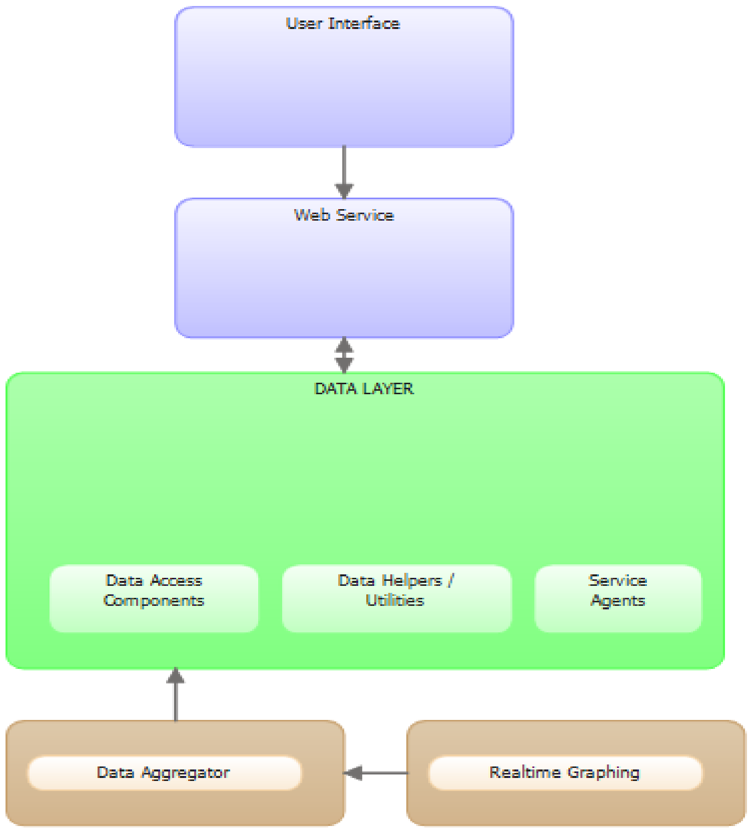
Data aggregation tools

Placeholder.

Design



The first conceptual designs of the proposed system it looks at the system from an extremely high level.



Risk Analysis

This project comes with a number of risks that range from medium to potentially catastrophic

Risk	Alternative Action
Choosing the correct programming language -The learning curve required to reach the necessary level of competence to implement the project. Python was the initial choice but research has shown a lack of knowledge available in the required areas.	Ample online resources should mitigate against this risk. Switch development language to something with more resources and support available. Use C# as the main development language using MonoDevelop IDE to allow development on a Linux distribution. There are myriad C# tutorials and source of support available.
Difficulty in gaining an understanding of the required programming patterns and concepts.	Again ample online resources available to mitigate against this. Complete as many tutorials as possible before commencement of the implementation phase.
Learning the effective use of the Repository Pattern.	Utilize as many online resources as is feasible. Begin working with simple examples and implementations and steadily build up the required level of knowledge.
Implementation of a Repository Pattern capable of coping with multiple databases and/or multiple data-types and sources.	Research as comprehensively as possible. Action taken to reduce the impact of the previous risk is also applicable here.
Aspect Oriented Programming – what is it and how is it used effectively.	Tutorials and online references available which should reduce the impact of this risk.
Difficulty in writing a suitable web service due to lack of familiarity.	Tutorials available on line. Some work done with web services as part of another course will help reduce this risk significantly.
Working on a Linux based system, working with a Linux server, communicating between server and client (s).	Continuous use of Linux based systems to attain a level of familiarity and confidence. Server created early in the process to allow

	ample scope for difficulties and experimentation.
Access to data of a sufficient volume and variety.	Use of a smaller subset of data to implement and test theories and metrics on a smaller scale. Extrapolate results and calculate for a larger data volume. If results are consistent for multiple smaller data set then consistency can be assumed for larger volumes.
Sufficient storage capacity.	Particularly important for scalability and elasticity measurements. Starting with a base set of data of approximately 28GB is storage capacity available to run multiple nodes and clusters. Cloud based storage is available and should cope with requirements but potentially adds a layer of complexity.
Ability to manipulate multiple databases from a single application	Trial and error in combination with repositories. Incremental development, begin with a single familiar database performing a single operation. Add a new database with each increment.
Unfamiliar tools and technologies. Installation of certain required tools is challenging.	Some but not comprehensive online resources and tutorial available. Use contacts in industry that have experience with installation of the required packages. Some reduction in risk as familiarity with Linux increases.
Time is extremely limited.	Develop iterative development plan to reduce this risk. Incremental development

	should allow for a reduction in scope if required thus alleviating time issues.
Finances – Access to Amazon AWS is required. Concerns about the ability of the free tier to meet requirements.	Constant monitoring of usage. Avoid Elastic IP addresses which have been shown to add significant hidden cost.
Scope – There is a danger that the scope of the project is too large and will be potentially un-manageable.	Use the iterative development plan to eliminate the possibility of scope creep.
Overall project complexity.	Correct research, planning and design should reduce complexity to manageable proportions.

High Level Features

High level features are specified according to the MoSCoW technique rating features as one of Must have, Should have, Could have, Won't have.

Must Have Features

Implementation must provide at the most basic level at least 1 CRUD operation that can be performed and subsequently measured against multiple databases with aggregated results represented visually.

The application must provide a means of measuring database operations.

The application must be able to display metric results graphically.

The application must have a simple user interface.

The application must provide a RESTful Web Service.

The application must provide operations that run on at least 2 databases.

The application must provide at least 1 database operation per database.

The application must have 1 Relational Database implementation – MySQL

The application must have 1 NoSQL database implementation.

Should Have Features

This application should provide full CRUD functionality on each database that supports the operations.

This application should have 1 database from each previously identified implementation type – 1 columnar database, 1 document database, 1 key – value store, 1 graph data store.

This application should provide as close as possible to real time graphical results of metric calculations.

This application should have an interactive user interface including secure log-in and authentication.

Could Have

The application could have the functionality to reproduce graphical results as part of a spreadsheet report.

Won't Have

The application won't have more than 1 database from each family.

User Stories

Must have features

The application must provide a means of measuring database operations.

The application must be able to display metric results graphically.

The application must have a simple user interface.

The application must provide a RESTful Web Service.

The application must provide operations that run on at least 2 databases.

The application must provide at least 1 database operation per database.

The application will use at a minimum 1 Relational Database implementation – MySQL

The application will use at least 1 NoSQL database implementation.

Should have features

This application should provide full CRUD functionality on each database that supports the operations.

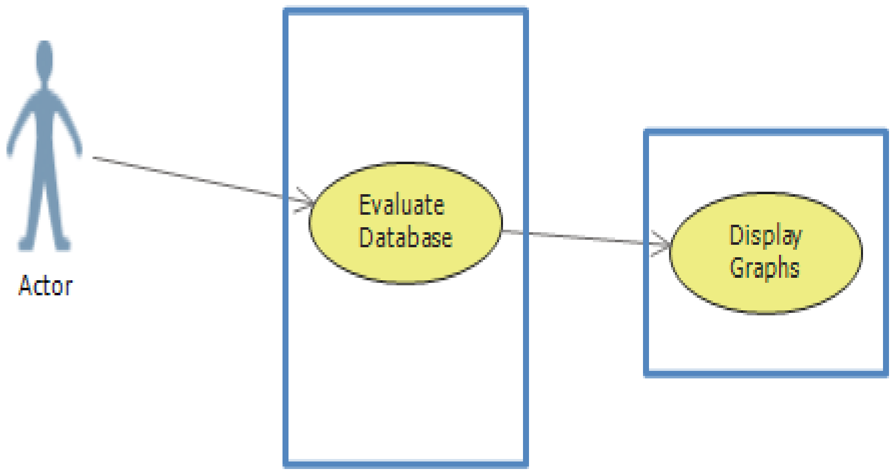
This application should have 1 database from each previously identified implementation type – 1 columnar database, 1 document database, 1 key – value store, 1 graph data store.

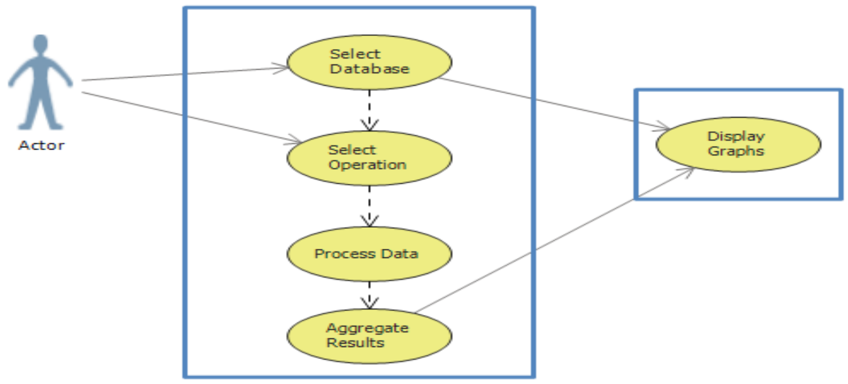
This application should provide as close as possible to real time graphical results of metric calculations.

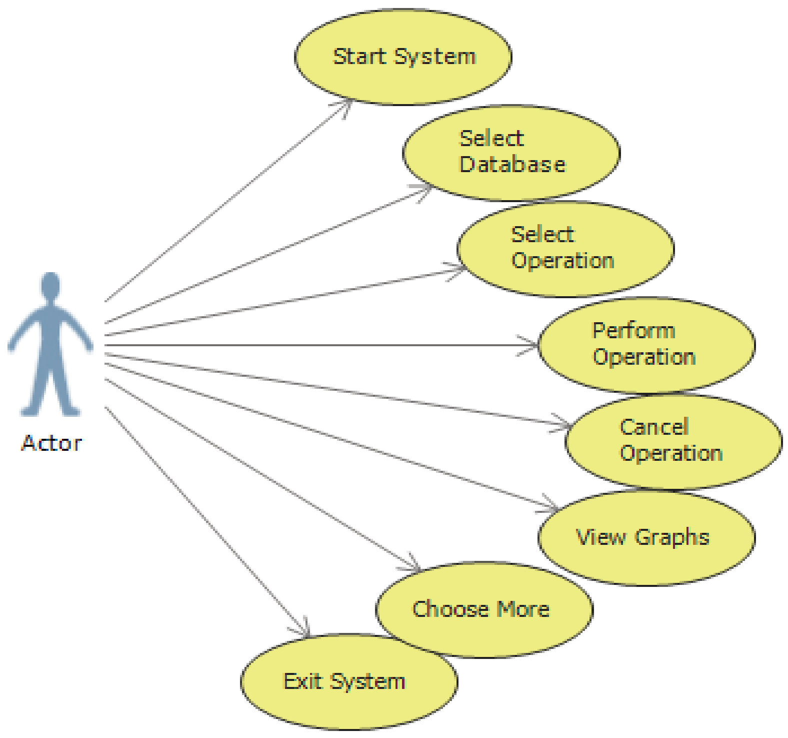
Use Case Diagrams

Two use case diagrams can represent the entire process of the application.

As a user want I to evaluate a database(s) and view the results as a graph or graphs.







Use case diagram representing the most basic functionality of the system.

Use Cases

All use case are prioritised based on the MoSCoW method and categorised as follows.

1 = Must have.

2 = Should have.

3 = Could have.

4 = Won't have.

Note: This is NOT an exhaustive listing.

Name	**Start Application**
Priority	**1**
Description	As a user I want to start the application
Acceptance Criteria	The application starts and the user is presented with the user interface

Name	Select Database
Priority	1
Description	As a user I must be able to select a database
Acceptance Criteria	The user interface displays a selectable list of databases. The user selects a database – the user interface displays a selectable list of operations.

Name	Select Operation
Priority	1
Description	As a user I must be able to select an operation.
Acceptance Criteria	The user interface displays a selectable list of operations. The user selects an operation, confirmation of database and operation is requested. The user selects yes Start button is activated.

Name	Perform Operation
Priority	1
Description	As a user I want to perform selected operation.
Acceptance Criteria	The user clicks the start button. The system begins execution of selected operation. Confirmation message displayed to user – “in progress”. Cancel button is visible in the User Interface.

Name	Cancel Operation
Priority	1
Description	As a user I want to cancel the current operation.
Acceptance Criteria	The user clicks the cancel button. Execution ceases immediately. Confirmation message displayed to user – “process cancelled”. User menu is displayed.

Name	View Graphs
Priority	1
Description	As a user I want to view graphical results of the operation.

Acceptance Criteria	When operation has completed a graph or graphs of the results are displayed to the user. The user is presented with the option of saving selected graphs.

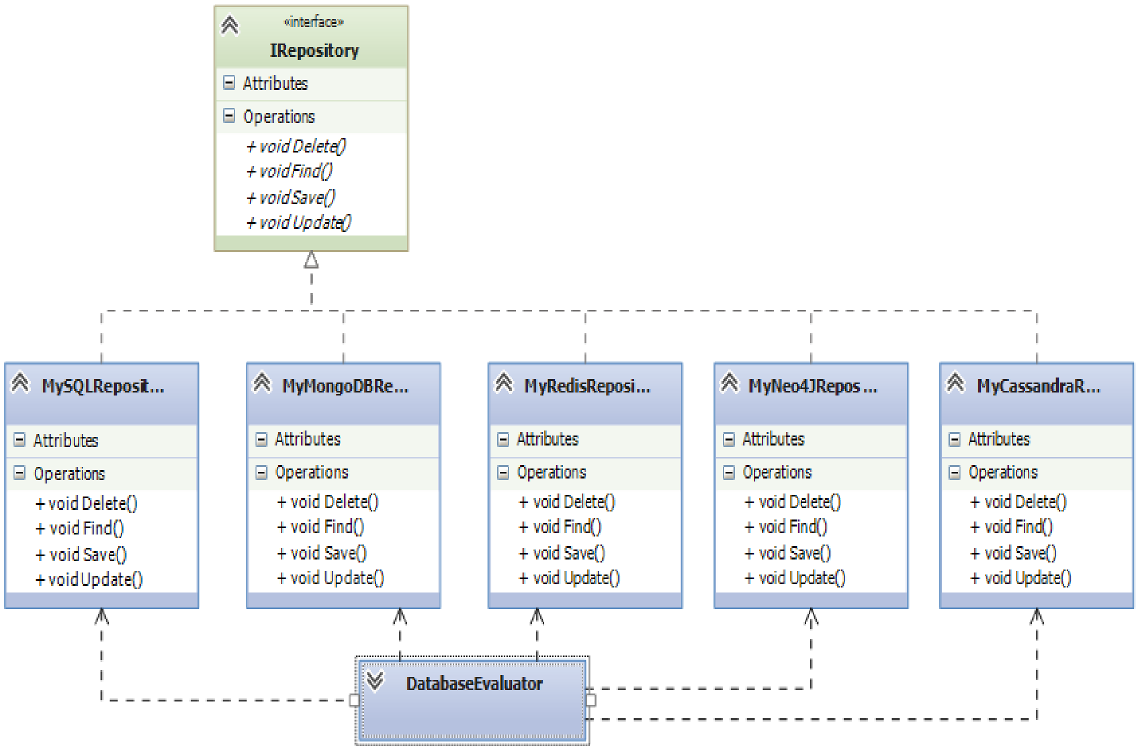
Name	**Choose More**
Priority	**1**
Description	As a user I want to perform further operations.
Acceptance Criteria	The user interface displays continue and exit options. The user selects continue. The user interface returns to offer a database selection.

Name	**Exit System**
Priority	**1**
Description	As a user I want to exit the application
Acceptance Criteria	The user clicks the exit button. The system displays a confirmation dialog. The user selects yes option. The system shuts down.

Name	**Secure Log-in**
Priority	**2**
Description	I as a user want to access the system.
Acceptance Criteria	The system has successfully started. Username field is visible. Password field is visible. User enters username and password. Selects enter. If username and password are valid welcome message is displayed. User menu is displayed and active. If username and/or password are invalid error message is displayed. Re-entry requested.

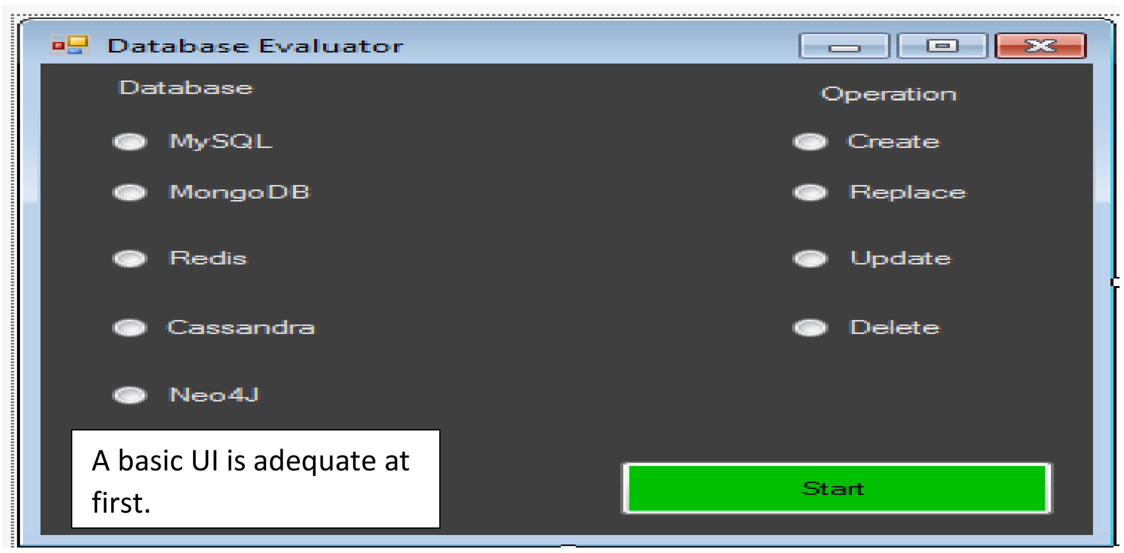
Class Diagrams

High level class diagram that represents the earliest basis for a system. The diagram is not complete and will be added to as development progresses.



User Interface Prototype

The user interface required by the application is minimal, this prototype displays the minimum requirements.



A basic UI is adequate at first.

Implementation

Implementation will take an agile approach, with the process divided into a number of 'sprints' with achievable goals outlined per sprint. Taking into consideration that design is presently a work in progress the following section should be regarded as a draft and is subject to revision. All sprints are subject to change and will evolve over time, backlog items will be included when process has been formalized.

Sprint Process

Implementation is divided into sprints, each sprint will be two weeks in duration giving a total of eight sprints. Note that all sprints outlined in this section are aspirational at present and subject to change.

It has been agreed that time is allocated to a sprint as follows:

1 week = 12hrs (4hrs lab work + 8hrs project work).

Each sprint = 2 weeks in duration.

Activities per sprint should include where applicable:

Research.

More design detail.

Unit testing -Any unit tests must be designed before the code.

Coding – adhere to recognised coding standards.

Run code through a static analyser tool.

Review Sprint and update plan and backlog items.

Sprints as planned focus on implementation and technical tasks, documentation will also be updated, re-written and added to over the course of implementation. Tasks associated with documentation are more difficult to schedule and will be undertaken on an ad hoc basis and are therefore excluded from the sprint plan. Sprint 8 is exceptional as it includes documentation and tidy up tasks, also the finish date for sprint 8 has not been established at this point.

Sprint Schedule

Sprint Number	Start Date	Finish Date
1	01/12/2014	14/12/2014
2	19/01/2015	01/02/2015
3	02/02/2015	15/02/2015
4	16/02/2015	01/03/2015
5	02/03/2015	15/03/2015
6	16/03/2015	29/03/2015
7	30/03/2015	12/04/2015
8	13/04/2015	TBC

Sprint Plan

Sprint 1:

This first sprint aims to complete multiple 'housekeeping', installation and software set-up tasks prior to beginning full development of the application. This sprint also aims to see a number of tutorials undertaken and one measurement operation successfully completed.

Sprint Number	Start Date	Finish Date
1	01/12/2014	14/12/2014

Task Number	Details	Status
1	Create Ubuntu client and server machines using VMWare Workstation	Complete
2	Install MonoDevelopIDE and MongoDB on client machine	Complete
3	Install Graphite and StatsD on server machine	Complete
4	Tutorials on Graphite and StatsD available at https://www.digitalocean.com/community	Complete More available
5	Test Graphite and StatsD using tutorial. Display basic graphs.	Complete
6	Create draft non-functioning UI	Complete
7	Write entity class for testing purposes using MonoDevelopIDE	Complete
8	Write application that connects to MongoDB saves an instance of entity class to a collection and times the operation	Complete

Sprint 2:

Sprint Number	Start Date	Finish Date
2	19/01/2015	01/02/2015

Task Number	Details	Status
1	Update UI prototype to include functionality	N/A
2	Implement timed Find and Delete operations against MongoDB	N/A

3	Send time metrics from test application to Graphite via StatsD over UDP	N/A
4	Display graphs of metrics on Graphite web interface	N/A
5	Take tutorial on writing a Web Service	N/A
6	Write and test a basic Web Service	N/A
7	All code fully tested	N/A
8	Download Wikipedia data dump	N/A
9	Implement Save operation using Wikipedia data dump, varying the volume of data.	N/A

Sprint 3:

Sprint Number	Start Date	Finish Date
3	02/02/2015	15/02/2015

Task Number	Details	Status
1	Implement and time Update against MongoDB	N/A
2	Alter test application to interact with MongoDB via a RESTful Web Service	N/A
3	Time operations completed using Web Service	N/A
4	Send time metrics to Graphite via StatsD	N/A
5	All code fully tested	N/A
6	Tutorials on Repository Pattern	N/A
7	Write test application using repository pattern	N/A
8	Write IRepository interface	N/A

Sprint 4:

Sprint Number	Start Date	Finish Date
4	16/02/2015	01/03/2015

Task Number	Details	Status
1	Fully functioning UI	N/A
2	Re-factor test application to implement Repository Pattern	N/A
3	Write more comprehensive entity class e.g. Student with name, student number, course number.	N/A
4	Code fully tested	N/A
5	Review work to date	N/A
6	Create schema for MySQL database	N/A
7	Install Redis,Neo4J,Cassandra on virtual machine(s)	N/A

Sprint 5:

Sprint Number	Start Date	Finish Date
5	02/03/2015	15/03/2015

Task Number	Details	Status
1	Implement and time CRUD operations on a 2nd database MySQL/Redis	N/A
2	All repositories classes written. Implemented where applicable.	N/A
2	UI fully functional	N/A
3	All code tested	N/A
4	Create Master-Slave replicated cluster for MongoDB (needed to measure scaling and elasticity)	N/A
5	Tutorials on Aspect Oriented Programming	N/A
6	Re-engineer application to follow AOP principles	N/A
7	Working prototype ready for demonstration purposes	N/A
8	Finalise what metrics will be included in finished system	N/A

Sprint 6:

Sprint Number	Start Date	Finish Date
6	16/03/2015	29/03/2015

Task Number	Details	Status
1	Completed Web Service. All calls to databases and communication with StatsD via Web Service.	N/A
2	Perform a comparison between 2 databases using finalised metrics and generate results graph(s)	N/A
3	Add 3rd and 4th database to application	N/A
4	All code tested	N/A
5	All functionality tested	N/A
6	Demo application prepared for supervisors	N/A
7	Implement any suggested improvements	N/A

Sprint 7:

Sprint Number	Start Date	Finish Date
7	30/03/2015	12/04/2015

Task Number	Details	Status
1	Introduce final database to application	N/A
2	Include security features in UI – login/password	N/A
3	All code tested	N/A
4	All functionality tested	N/A
5	Comparison of all databases available and metrics generated	N/A
6	Review	N/A

Sprint 8:

Sprint Number	Start Date	Finish Date
8	13/04/2015	TBC

Task Number	Details	Status
1	Review and tidy up	N/A
2	Finalize documentation	N/A
3	Design and prepare presentation	N/A
4	All documentation proof read and peer reviewed	N/A

Test Plan

Place holder.

To – Do.

References

Bagui, S., 2003. Achievements and Weaknesses of Object - Oriented Databases. *Journal of Object Technology,* 2(4), pp. 29 - 41.

Barahmand, S. & Ghandeharizadeh, 2013. *BG:A Benchmark to Evaluate Interactive Social Networking Actions,* Los Angeles: University of Southern California.

Barata, M., Bernardino, J. & Furtado, P., 2014. *YCSB and TPC-H: Big Data and Decision Support Benchmarks.* Anchorage, AK, IEEE, pp. 800 - 801.

Bogdan, G. T., 2013. Challenges for the NoSQL Systems: Directions for Further Research and Development. *International Journal of Sustainable Economies Management(IJSEM),* 2(1), pp. 55-64.

Brooks, J., 2011. Does NoSQL Matter to Your Company?. *eweek,* 28(15), pp. 28-29.

Chamberlain, D. D. e. a., 1981. A History and Evaluation of System R. *Communications of the ACM,* 24(10), pp. 632 - 646.

Chamberlin, D. D. & Boyce, R. F., 1974. *SEQUEL: A Structured English Query Language.* s.l., Association for Computing Machinery, pp. 249 - 264.

Chang Fay, D. J. S. W. C. D. A. M. T. A. R. E., 2006. *Bigtable: A Distributed Storage System for Structured Data.* Seattle, Proc. of OSDI '06.

Cooper, B. F. et al., 2010. *Benchmarking Cloud Serving Systems with YCSB,* Santa Clara, CA: ACM Association for Computer Machinery.

Delorme, P. & Chatelain, O., 2011. *The Role and Use of Performance Measurement Indicators,* Brussels: European Commission, EuropeAid.

DeWitt, D. J., n.d. *The Wisconsin Benchmark: Past, Present, and Future,* Minneapolis: Universtiy of Wisconsin.

Dobre, C. & Xhafa, F., 2014. Parallel Programming Paradigms and Frameworks in Big Data Era.. *International Journal of Parallel Programming,* 42(5), pp. 710 - 738.

Evans, E., 2003. *Domain-Driven Design.* 1st ed. Upper Saddle River, New Jersey: Prentice Hall.

Fowler, M., 2011. *Patterns of Enterprise Application Architecture.* 17th ed. Crawfordsville, Indiana: Addison-Wesley.

Fowler, M., 2012. *NoSQLDefinition.* [Online]
Available at: http://martinfowler.com/bliki/NosqlDefinition.html
[Accessed 9 October 2014].

Han, J., Haihong, E., Le, G. & Du, J., 2011. *Survey on NoSQL database.* Beijing, Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on, pp. 363 - 366.

Harrington, J. L., 2002. *Relational Database Design Clearly Explained.* 2nd ed. San Francisco, CA: Morgan Kaufmann.

Harrington, J. L., 2003. *SQL Clearly Explained.* 2nd ed. San Francisco: Morgan Kaufmann.

Harrington, J. L., 2009. *Relational Database Design and Implementation.* 3rd ed. Burlington, MA: Morgan Kaufmann.

Hecht, R. & Jablonski, S., 2011. *NoSQL Evaluation A Use Case Oriented Survey.* Hong Kong, IEEE, pp. 336 - 341.

Hophe, G. & Woolf, B., 2011. *Enterprise Integration Patterns.* 15 ed. Westford, Massachusetts: Pearson Education Inc.

Indrawan-Santiago, M., 2012. *Database Research: Are we at a crossroads? Reflections on NoSQL.* Melbourne, IEEE, pp. 45 -51.

Jayathilake, D. et al., 2012. *A study into the capabilities of NoSQL databases in handling a highly heterogeneous tree.* Beijing, IEEE, pp. 106 - 111.

Kraska, T. & Trushkowsky, B., 2013. The New Database Architectures. *Internet Computing, IEEE,* 17(3), pp. 72,75.

Ödegaard, T., 2013. *Grafana.* [Online]
Available at: http://grafana.org/
[Accessed 9 December 2014].

Oppel, A., 2004. *Databases Demystified.* San Francisco: San Francisco, CA: McGraw-Hill Osborne Media.

Oracle Corporation, 2006. *Oracle Berkeley DB: Performance Metrics and Benchmarks.* Redwood Shores, CA: Oracle Corporation.

Paul, S., n.d. s.l.: s.n.

Redmond, E. & Wilson, J. R., 2012. *Seven Databases in Seven Weeks - A Guide to Modern Databases and the NoSQL Movement.* 1st ed. Dallas , Texas: The Pragmatic Bookshelf.

Sadalage, P., 2014. *NoSQL Databases: An Overview.* [Online]
Available at: http://www.thoughtworks.com/insights/blog/nosql-databases-overview
[Accessed 24 November 2014].

Sadalage, P. J. & Fowler, M., 2013. *NoSQL Distilled A Brief Guide to the Emerging World of Polyglot Persistence.* Crawfordsville, Indiana: Pearson Education Inc.

Sakr, S. & Liu, A., 2013. *Is Your Cloud-Hosted Database Truly Elastic?.* Santa Clara , CA, IEEE, pp. 444 - 447.

Singer, B. P., 2014. *Introduction to Databases: Structured Query Language.* Brussels: Dept of Computer Science, Vrije Universiteit Brussel.

Stonebraker, M., 2010. SQL Databases v. NoSQL Databases. *Communications of the ACM,* 53(4), pp. 10 - 11.

Tiwari, S., 2011. *Professional NoSQL.* 1st ed. Indianapolis: John Wiley & Sons Inc..

Transaction Processing Performance Council, 2014. *Transaction Processing Performance Council.* [Online]
Available at: http://www.tpc.org/information/results.asp
[Accessed 27 November 2014].

University, C. D., 2011. *learnline.cdu.edu.au/units/databaseconcepts.* [Online]
Available at:

http://learnline.cdu.edu.au/units/databaseconcepts/module2/normalisation.html [Accessed 23 October 2014].

Zhao, G., Huang, W., Liang, S. & Yong, T., 2013. *Modeling MongoDB with Relational Model.* Xi'an, IEEE, pp. 115 - 121.

Bibliography

Bagui, S., 2003. Achievements and Weaknesses of Object - Oriented Databases. *Journal of Object Technology,* 2(4), pp. 29 - 41.

Barahmand, S. & Ghandeharizadeh, 2013. *BG:A Benchmark to Evaluate Interactive Social Networking Actions,* Los Angeles: University of Southern California.

Barata, M., Bernardino, J. & Furtado, P., 2014. *YCSB and TPC-H: Big Data and Decision Support Benchmarks.* Anchorage, AK, IEEE, pp. 800 - 801.

Bogdan, G. T., 2013. Challenges for the NoSQL Systems: Directions for Further Research and Development. *International Journal of Sustainable Economies Management(IJSEM),* 2(1), pp. 55-64.

Brooks, J., 2011. Does NoSQL Matter to Your Company?. *eweek,* 28(15), pp. 28-29.

Chamberlain, D. D. e. a., 1981. A History and Evaluation of System R. *Communications of the ACM,* 24(10), pp. 632 - 646.

Chamberlin, D. D. & Boyce, R. F., 1974. *SEQUEL: A Structured English Query Language.* s.l., Association for Computing Machinery, pp. 249 - 264.

Chang Fay, D. J. S. W. C. D. A. M. T. A. R. E., 2006. *Bigtable: A Distributed Storage System for Structured Data.* Seattle, Proc. of OSDI '06.

Cooper, B. F. et al., 2010. *Benchmarking Cloud Serving Systems with YCSB,* Santa Clara, CA: ACM Association for Computer Machinery.

Delorme, P. & Chatelain, O., 2011. *The Role and Use of Performance Measurement Indicators,* Brussels: European Commission, EuropeAid.

DeWitt, D. J., n.d. *The Wisconsin Benchmark: Past, Present, and Future,* Minneapolis: Universtiy of Wisconsin.

Dobre, C. & Xhafa, F., 2014. Parallel Programming Paradigms and Frameworks in Big Data Era.. *International Journal of Parallel Programming,* 42(5), pp. 710 - 738.

Evans, E., 2003. *Domain-Driven Design.* 1st ed. Upper Saddle River, New Jersey: Prentice Hall.

Fowler, M., 2011. *Patterns of Enterprise Application Architecture.* 17th ed. Crawfordsville, Indiana: Addison-Wesley.

Fowler, M., 2012. *NoSQLDefinition.* [Online]
Available at: http://martinfowler.com/bliki/NosqlDefinition.html
[Accessed 9 October 2014].

Han, J., Haihong, E., Le, G. & Du, J., 2011. *Survey on NoSQL database.* Beijing, Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on, pp. 363 - 366.

Harrington, J. L., 2002. *Relational Database Design Clearly Explained.* 2nd ed. San Francisco, CA: Morgan Kaufmann.

Harrington, J. L., 2003. *SQL Clearly Explained.* 2nd ed. San Francisco: Morgan Kaufmann.

Harrington, J. L., 2009. *Relational Database Design and Implementation.* 3rd ed. Burlington, MA: Morgan Kaufmann.

Hecht, R. & Jablonski, S., 2011. *NoSQL Evaluation A Use Case Oriented Survey.* Hong Kong, IEEE, pp. 336 - 341.

Hophe, G. & Woolf, B., 2011. *Enterprise Integration Patterns.* 15 ed. Westford, Massachusetts: Pearson Education Inc.

Indrawan-Santiago, M., 2012. *Database Research: Are we at a crossroads? Reflections on NoSQL.* Melbourne, IEEE, pp. 45 -51.

Jayathilake, D. et al., 2012. *A study into the capabilities of NoSQL databases in handling a highly heterogeneous tree.* Beijing, IEEE, pp. 106 - 111.

Kraska, T. & Trushkowsky, B., 2013. The New Database Architectures. *Internet Computing, IEEE,* 17(3), pp. 72,75.

Ödegaard, T., 2013. *Grafana.* [Online]
Available at: http://grafana.org/
[Accessed 9 December 2014].

Oppel, A., 2004. *Databases Demystified.* San Francisco: San Francisco, CA: McGraw-Hill Osborne Media.

Oracle Corporation, 2006. *Oracle Berkeley DB: Performance Metrics and Benchmarks.* Redwood Shores, CA: Oracle Corporation.

Paul, S., n.d. s.l.: s.n.

Redmond, E. & Wilson, J. R., 2012. *Seven Databases in Seven Weeks - A Guide to Modern Databases and the NoSQL Movement.* 1st ed. Dallas , Texas: The Pragmatic Bookshelf.

Sadalage, P., 2014. *NoSQL Databases: An Overview.* [Online]
Available at: http://www.thoughtworks.com/insights/blog/nosql-databases-overview
[Accessed 24 November 2014].

Sadalage, P. J. & Fowler, M., 2013. *NoSQL Distilled A Brief Guide to the Emerging World of Polyglot Persistence.* Crawfordsville, Indiana: Pearson Education Inc.

Sakr, S. & Liu, A., 2013. *Is Your Cloud-Hosted Database Truly Elastic?.* Santa Clara , CA, IEEE, pp. 444 - 447.

Singer, B. P., 2014. *Introduction to Databases: Structured Query Language.* Brussels: Dept of Computer Science, Vrije Universiteit Brussel.

Stonebraker, M., 2010. SQL Databases v. NoSQL Databases. *Communications of the ACM,* 53(4), pp. 10 - 11.

Tiwari, S., 2011. *Professional NoSQL.* 1st ed. Indianapolis: John Wiley & Sons Inc..

Transaction Processing Performance Council, 2014. *Transaction Processing Performance Council.* [Online]
Available at: http://www.tpc.org/information/results.asp
[Accessed 27 November 2014].

University, C. D., 2011. *learnline.cdu.edu.au/units/databaseconcepts.* [Online]
Available at:
http://learnline.cdu.edu.au/units/databaseconcepts/module2/normalisation.html
[Accessed 23 October 2014].

Zhao, G., Huang, W., Liang, S. & Yong, T., 2013. *Modeling MongoDB with Relational Model.* Xi'an, IEEE, pp. 115 - 121.