Moore, P.R., Qassem, T., Xhafa, F. (2014) 'NoSQL' and electronic patient record systems: opportunities and challenges. *2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2014, 8-10 November 2014, Guangzhou, Xina: proceedings*. [S.l.]: IEEE, 2014. Pp. 300-307 Doi: http://dx.doi.org/10.1109/3PGCIC.2014.81.

NoSQL and Electronic Patient Record Systems: Opportunities and Challenges

Philip Moore
School of Information Science and Engineering
Lanzhou University
Lanzhou, China
drpmlzu@yahoo.co.uk

Tarik Qassem
Warwick Medical School
University of Warwick
Coventry, UK
T.Qassem@warwick.ac.uk

Fatos Xhafa
Technical University of Catalonia
C. Jordi Girona, 08034
Barcelona, Spain.
fatos@lsi.upc.edu

Abstract— **Research into electronic health record systems can be traced back over four decades however the penetration of records which incorporate more than simply basic information into healthcare organizations is relatively limited. There is a great demand for effective health record systems which are difficult to build with data generally stored in highly distributed systems as unstructured data with access and updating achieved over online systems. Internet application design must reflect three trends in the computing landscape: (1) growing numbers of users applications must support (along with growing user performance expectations), (2) growth in the volume and range and diversity in the data that developers accommodate, and (3) and the rise of Cloud Computing. The traditional approach to data storage has generally employed Relational Database Systems however to address the evolving paradigm interest has been shown in alternative database systems including NoSQL technologies. This paper considers the requirements of online distributed health record systems. The analysis supports the conclusion that NoSQL database systems provide a potentially useful approach to the implementation of HR systems in online applications.**

Keywords— unstructured data; database systems; NoSQL; electronic health records; online transaction processing

I. Introduction

Research into Electronic Health Record (HR) systems can be traced back over four decades however the penetration of records which incorporate more than simply basic information into healthcare organizations is relatively limited. There is a great and largely unsatisfied demand for effective HR systems by all stakeholders in healthcare provision to address a number of issues including [1]: (1) medical record movement and updating problems, (2) interoperability between different systems, (3) realizing improvements in the quality and coherence of the care process to drive process improvement, (4) automation of guidelines and care pathways, (5) assistance and facilitation of clinical research, outcomes, and management, and (6) the ability to mine the data to recognize patterns that could help in improving healthcare.

HR are however very difficult to build as data is generally stored in distributed systems and locations in a diverse range of formats as unstructured data with access and updating using *Online Transaction Processing* (OTP). These challenges are exacerbated because the current electronic data sources which include: hospital systems, laboratory systems, pharmacy systems, and physician dictation systems etc reside on multiple data stores frequently with different structures, levels of granularity, and coding systems [1].

An example of a project designed to implement a comprehensive HR is the *'National Programme for IT'* (NpfIT) [2][3] in the UK. The NpfIT project was, from its outset, an ambitious effort aimed at introducing a national HR system across NHS care providers for England. The project is distinguished by its scale, unprecedented levels of investment, complexity of systems, centrally driven delivery model, and extremely challenging timelines. The English endeavor was ultimately unsuccessful and demonstrates the scale of challenges inherent in building an effective and workable HR system [2][3]. A central feature of the NpfIT project was the use of OTP as discussed in [2][3].

Traditionally, *Relational Database Management Systems* (RDMS) have been employed to manage data. However application needs have changed dramatically and the design of Internet applications (including HR) must reflect current trends in the computing landscape: (1) the growth in the numbers of users applications must support (along with growing user performance expectations), (2) the growth in the volume and range and diversity in the data that developers accommodate, and (3) the rise of Cloud Computing (CC) (which relies on a distributed three-tier Internet architecture), and (4) possibilities for the use the data in 'Big data' applications. These trends are features of HR systems. To address these trends interest has been shown in alternative database systems including NoSQL (non-relational) technologies. While non-relational databases (including hierarchical, graph, and object-oriented databases) date back to the late 1960s NoSQL; systems are gaining traction in Internet based enterprise systems; prominent examples of such systems are developments by Google (Big Table) [4] and Amazon (DynamoBD) [5]. It has been argued that *BigTable*, *memcached*, and Amazon's *DynamoDB* have demonstrated 'proof-of-concept which has provided the motivation for many of the data stores that characterize NoSQL systems.

Why Are NoSQL Databases so Interesting? and why are they gaining traction among Internet based organizations such as Google, Amazon, and Facebook? It has been noted by Sadalage & Fowler in [6] that there are two primary reasons for considering a NoSQL database: (1) *Application development productivity* (issues in mapping data between in-memory data structures and a relational database), and (2) *Large-scale data sets* (a relational database is designed to run on a single machine, but it is usually more economic to run large data and computing loads on clusters of many smaller and cheaper machines). Database systems designed to operate on *unstructured data* are generally referred to as NoSQL; the motivation being to accommodate the diversity in the types of data and enable dynamic high user loading. Unstructured data is generally characterised by highly distributed datasets using both *horizontal* and *vertical* scaling (also termed partitioning) [7] [8] across nodes located in widely separated geographical locations [8].However, notwithstanding the potential to manage unstructured data NoSQL database systems must clear many technical and commercial obstacles if widespread take-up of the technology is to be realized [7].

This paper considers the requirements of distributed data in HR systems along with the relative merits of traditional and NoSQL database systems as they relate to unstructured data in HR systems using OTP. The paper is structured as follows: HR are considered with a brief introduction to some key terminology used in discussing NoSQL systems to ensure consistency in the discussion. Related research addressing RDMS and NoSQL systems is presented followed by the important approaches to enable implementation of NoSQL systems. Illustrative examples of the NoSQL approaches are set out with a scenario-based example of an HR system where the systemic requirements are set out and an analysis is provided. A discussion is presented, the paper closing with concluding observations, open research questions. The analysis provides support for the conclusion that for HR implemented using OTP unstructured data and NoSQL database systems provide a potentially useful and profitable approach to the implementation of such systems.

II. Electronic Health records

In a paper entitled: 'The Barriers to Electronic Medical Record Systems and How to Overcome Them' [1] it is argued that: *"If everyone" wants HR, and the sources of electronic patient data are so abundant, why are they so scarce?"*. The reasons are very complex and involve many factors including ethical, security, and privacy concerns [1]. However, from a computational perspective there are two principal reasons: [1]: (1) sources of [electronic] patient data "reside on many isolated 'islands' that have been very difficult to bridge", and (2) effective methods of capturing data from medical professionals based on common standards in a human and computer readable form remain elusive and have yet to be effectively achieved.

It was reported in 1997 in that there are*: "Too Many Different Separate Systems with Different Data Structures"* [1]. Additionally, the diversity in the types of care provided [which range from general medicine and emergency treatment to nursing homes] only serve to add to the inherent complexity in *"birth-to-death"* healthcare systems and the data sets that combine to create an HR. With the expansion in the range, scope, and capability of medical care pathways, the complexity identified in [1] has only increased along with the ability to create, store, and distribute such data. An additional consideration is research data gathered in laboratory experiments and medical trials which may subsequently be used in translational research [9]. Thus, it can be seen that medical record data is by its very nature inherently unstructured and as we will argue, managing such data using traditional approaches [such as a RDMS] represents a significant challenge.

A. Terminology

Prior to considering database systems with respect to HR systems it will be useful to define a number of important terms. The term NoSQL has no commonly agreed definition [6][7][8], it is however commonly accepted that the term stands for*: "Not Only SQL"* or *"Not Relational"* [7][8]. Additionally, there are many terms used when describing NoSQL systems for which the definitions vary between different NoSQL' applications and systems [8]. To ensure clarity of meaning a number of key terms are set out below with their definitions.

A ***document***: allows values to be nested documents or lists or *scalar* values with attribute names dynamically defined for each document at runtime. A document differs from a ***tuple*** in that the attributes are not defined in a global schema; thus a wider range of values are permitted. An ***extensible record***: is a hybrid between a ***tuple*** and a ***document***. Collections of attributes are defined in a schema however new attributes can be added (within an attribute collection) on a per-record basis. Attributes may be list-valued. An ***object***: is analogous to an object in high level programming languages but without the procedural methods. Values may be references or nested objects. ***Scaling*** (often termed ***partitioning***): refers to ***horizontal*** and ***vertical*** scaling. *Horizontal* scaling means the ability to distribute both the data and the load of *simple operations* over many servers, with no RAM or disk shared among the servers. *Vertical* scaling refers to a scenario where a database system utilizes many cores and/or CPUs that share RAM and disks. Systems may provide both *vertical* and *horizontal* scalability. A ***simple operation***: relates to *"key lookups", "reads",* and *"writes"* of one record or a small number of records [8]. This is in contrast to: *'complex queries'* or *'joins'*, *'readmostly access'*, or other application loads typically used in SQL systems.

III. Related Research

This section we consider documented research relating to database systems. We discuss RDMS and NoSQL database systems with consideration of unstructured data systems. In Section V a detailed discussion on approaches to implement NoSQL systems is presented with illustrative scenarios in Sections VI and VII.

A. Database Systems

A broad definition of NoSQL database systems including systems that are not relational has been used in the literature. Such systems include: (1) *Graph database systems*, (2) *Object-oriented database systems*, (3) *Distributed object-oriented systems* and (4) *Data warehousing database systems* [which provide horizontal scaling]. These systems are not considered to be relevant to the unstructured data in the context of the data and OTP used in HR systems and are therefore beyond the scope of this paper.

RDMS are a mature technology and the scope and nature of such systems is generally well understood with terminology that has acquired a (generally) agreed common definition [6][8]. NoSQL database systems are however immature technologies and as such the scope of such systems and the terminology used is frequently inconsistent [7][8]. To ensure clarity and consistency in comparing data models and functionality we have assumed that the NoSQL systems identified provide methods to provide a means to store scalar values. NoSQL database systems may additionally store more complex nested or reference values. All the systems discussed store sets of attribute value pairs however different data structures may be used as discussed in this paper.

B. *Relational Database Management Systems*

Traditionally, data has been stored in the form of structured data in RDMS to enable access, updating, and analysis [10]. A relational database can be viewed as a set of defragmented tables containing data in predefined categories. Each table contains one or more data categories in columns and each row contains a unique instance of data for the categories defined by the columns. Users can access, update, or create 'views' of the data based on user-defined queries created using the *Structured Query Language* (SQL) [10].

The advantages of RDMS are well known and as a mature technology there exist a large number of tools and functionality inbuilt into systems such as *Oracle* [11]. A schema definition based on relational algebra [10] centralizes and simplifies data definition, and SQL simplifies the expression of operations that span tables. In general programmers are generally conversant with SQL and it is arguably true that SQL is easier to program [given the availability of developed systems and tools] than the lower-level bespoke programmer defined commands required by 'NoSQL' systems.

Transactions [in SQL-based systems] greatly simplify coding for concurrent access and the *Atomicity, Consistency, Isolation, and Durability* (ACID) transaction model [common in the majority of RDMS] frees the developer from managing locks, out-of-date data, update collisions, and consistency.

1) *Limitations of RDMS*

In RDMS the structure of the data is predefined by a *Schema* which defines the layout of the tables and the fixed names and data types of the columns [10]. There are however important inherent and inbuilt limitations in RDMS when viewed from the perspective of highly dynamic Internet applications and systems designed to support very large numbers of concurrent users, OTP, and large diverse data sets [7][8]. Significant limitations are discussed below.

Scaling: a relational database can be scaled by running it on a single server; however, to scale beyond a the capacity of a single server the database must be distributed across multiple servers. RDMS may experience issues when implemented in a distributed system over many nodes due to difficulties in joining the tables [8]. Additionally, RDMS are not generally designed to function when data scaling is required; therefore partitioning of data is difficult [10]. Elmasari and Navathe [10] in discussing concurrency control and recovery in distributed databases observe:

> *"Failure of communication links: The system must be able to deal with one or more communication links [that connect sites]. An extreme case of this problem is that **network partitioning** may occur. This breaks up the sites into two or more partitions, where sites within each partition can communicate only with one another and not with other partitions"*

Complexity: in a RDMS data must be stored tables with decomposition to implement a 'store-once' approach in a unique record [10]. Where data will not fit conveniently into a table structure the database's structure can be complex, difficult, and exhibit significant performance issues. ***SQL***: is convenient with structured data; however, using SQL [with other types of data such as unstructured data] is difficult as it is designed to work with structured, relationally organized databases in a fixed table structure [8][10]. SQL can entail large amounts of complex code and doesn't work well with modern Internet applications and systems using OTP [10]. ***Large feature sets***: while a RDMS provides data integrity and a large feature sets. Proponents of unstructured database systems [NoSQL] argue that a large feature set may not always be required with the related cost and complexity implications [7][8].

C. *NoSQL Database Systems*

In recent years systems have been designed to provide good *horizontal* scalability for simple read/write database operations distributed over many nodes[7][8]. Many of the new systems are classified under the term NoSQL data stores. Such systems are essentially designed to manage unstructured data in applications and systems characterized by dynamic loading in online systems. NoSQL systems generally share a number of common features [7][8].

The principal benefit of NoSQL systems is the ability to scale horizontally scale and replicate /distribute data and throughput over many servers. In operation such systems use a simple call level interface or protocol (in contrast to a SQL binding) with a weaker concurrency model predicated on a *Basically Available, Soft State, Eventually consistent* (BASE) model as opposed to the ACID transaction model. The BASE approach is predicated on the assumption that updates are *eventually propagated*, but there are *limited guarantees on the consistency of reads*. In this model data returned in a search is not guaranteed to be up-to-date; however, updates are guaranteed to be propagated to all nodes eventually.

The frequently cited: *Consistency, Availability, and Partition-tolerance* (CAP) theorem [8] applies to NoSQL systems. This theorem states that: *" that a system can have only two out of three of the properties"*. As we have noted NoSQL systems generally give up *Consistency*; however, the trade-offs are generally very complex and domain specific. NoSQL systems may incorporate *Multi-version Concurrency Control* (MVCC): a concurrency control method commonly used by RDMS to enable concurrent access [8]. Efficient use of distributed indexes and RAM for data storage is a feature often found in NoSQL systems along with the ability to dynamically update data records and new attributes in OTP

NoSQL: systems may differ in terms of their functionality, the variations ranging from simple distributed hashing (as supported by the popular *memcached* open source cache) [12] to to highly scalable partitioned tables (as supported by Google's *BigTable*) [4]. ***Memcached*** [12]: a high-performance distributed memory object caching system is generic in nature and is designed to improve performance in dynamic web applications by alleviating database load. It has demonstrated that *in-memory* indexes can, in practical applications, achieve high scalability in distributing and replicating objects over multiple nodes.

The Amazon ***DynamoDB*** [5]: is a fully managed NoSQL database service that arguably pioneered the concept of *"eventual consistency"* [7][8] as a way of realising increased availability and scalability. It is claimed [5] that the *DynamoDB* provides: *"a cost-effective method to store and retrieve any amount of data, and serve any level of request traffic"* in Internet applications and systems. ***BigTable*** [4]: a Google development, it is 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). It has demonstrated that persistent record storage can be scaled over many thousands of nodes in highly distributed systems.

A key feature of NoSQL systems is the concept of *"shared nothing"* with horizontal scaling [7][8]. This approach enables support for large numbers of concurrent simple read/write operations in OTP which is a common feature of Internet applications and systems [8]. NoSQL systems generally function on the basis of the BASE concept and (by design) do not implement ACID transactional properties and constraints; the aim is to achieve improved performance levels and scalability. However, the systems differ in the degree to which the ACID functionality is used. For example, the majority of NoSQL systems claim to be *eventually consistent* however a number of NoSQL systems provide mechanisms for some degree of consistency including MVCC) [8].

D. Unstructured Data Systems

NoSQL technologies are gaining traction among Internet companies because it offers data management capabilities that meet the needs of modern applications by providing: (a) better application development productivity through a more flexible data model, (b) greater ability to scale dynamically to support large numbers of users and greater volume and diversity of data, and (c) improved performance to satisfy expectations of users [demanding rapid response with increasingly complex data processing]. NoSQL database systems are increasingly being considered as a viable alternative to RDMS and may be considered as suitable solutions for interactive web and mobile applications.

NoSQL systems may adopt different approaches; what they have in common is that they are not relational. Their primary advantage lies in the ability to efficiently manage unstructured data [unlike RDMS] such as text files, e-mail, multimedia, and social media. Additionally, they are generally easier to work with for the many developers not familiar with the SQL. NoSQL databases can function in a distributed setting, scaling for a single database or vertically rather than by having to run it on a single more powerful server(s). Moreover, proponents argue that NoSQL databases enable improved performance; this is particularly important for applications characterised by large amounts of data in diverse types and highly dynamic concurrent user interactions using OTP.

IV. NoSQL SYSTEMS

Having briefly considered related research around database systems we now move on to consider NoSQL database systems (also termed *data stores*). Data stores will be considered under the following data models: (1) *Key-value Stores*, (2) *Document Stores*, *Extensible Record Stores*, and (4) *Scalable Relational Stores*.

All of the systems identified incorporate an administrative function [a database] with data may be stored in: a single file, a directory, or using an alternative function that defines the scope of data used by a group of applications. Each database is unique; this applies even where scaled over multiple nodes. There is no concept of a "federated database" [8] as is the case with some relational and object-oriented databases; this enables multiple databases to appear as one. A majority of the NoSQL systems support *horizontal* scaling with records stored on different servers according to a key; this is termed *"sharding"* however there are systems which provide support for *vertical* partitioning where parts of a single record are stored on different servers.

A. Key-value Stores

Key-Value Stores (KVS): store values and use indexing to locate them based on a programmer defined key. A KVS applies the most simple data model and operates on a single key-value index for all related data (an approach similar to the memcached distributed in-memory cache model). However, unlike memcached, KVS generally provide a persistence mechanism and additional functionality including: replication, versioning, locking, transactions, sorting, and/or other features. The client interface provides for inserts, deletes, and index lookups. AS for memcached, KVS systems do not use secondary indices or keys.

Space restricts a detailed discussion on the available KVS however in summary all the principal KVS systems (e.g., *Voldemort*, *Riak*, *Tokyo Cabinet*, and *enhanced memcached* systems provide support for insert, delete, and lookup operations along with scalability through key distribution over nodes. *Voldemort*, *Riak*, *Tokyo Cabinet*, and *enhanced memcached* systems can store data in RAM or on disk, with storage add-ons. Others KVS store data in RAM, and provide disk as backup, or rely on replication and recovery. Scalaris and enhanced memcached systems use synchronous replication while the others employ asynchronous replication. *Scalaris* and *Tokyo Cabinet* implement transactions, while the others do not. *Voldemort* and *Riak* use MVCC, the others use locks. Membrain and Membase are built on the popular memcached system, adding persistence, replication, and other features.

B. Document Stores

Document Stores (DS): can support more complex data than the KVS. The term DS implies that such systems can store "documents" (e.g., articles, files, etc.); in actuality a document [in such systems] can in computational terms be any kind of *pointerless object* which may be "a one file, a directory, or may use a 'mechanism' that defines the scope of some data used by a group of applications". DS impose on system developers and programmers explicit utilization of indices.

Unlike KVS, DS (generally) provide support for: secondary indexes, multiple types of documents (objects) per database, and nested documents or lists. As for other NoSQL systems DS fail to provide ACID transactional properties. DS do not use a schema except for attributes (which are simply a name, and are not pre-specified), collections (which are simply a grouping of documents), and indexes defined on collections.

There are some differences in their data models however DS are all very similar and. unlike KVS, DS (generally, but not in all cases) provides methods to query collections based on multiple attribute value constraints. DS (generally) do not provide explicit locks and have weaker concurrency and atomicity properties than traditional ACID databases. They differ in how much concurrency control they do provide. Documents can be distributed over nodes in all of the systems however scalability differs; DS can achieve scalability by reading (potentially) out-of-date replicas.

C. Extensible Record Stores

Extensible Record Stores (ERS): store extensible records that may scaled (partitioned) vertically and horizontally. The basic data model for ERS is rows and columns; the basic scalability model being to split both rows and columns over multiple nodes.

Rows are split across nodes using *sharding* on a primary key. Typically splitting is by range as opposed to the use of a hash function. This means that queries on ranges of values need not be executed on every node.

Columns [in a table] are distributed over multiple nodes using "column groups" which are a method for users to indicate which columns are best stored together.

As discussed earlier, *horizontal* and *vertical* scaling may be used simultaneously on the same table. For example, if a customer table is partitioned into three column groups then each of the three column groups is treated as a separate table for the purposes of *sharding* the rows by e.g., customer ID: the column groups for one customer may (or may not) be located on the same server.

In ERS systems, column groups must be pre-defined; this however is less of a constraint than may be initially envisaged as new attributes can be defined at any time. Rows are analogous to documents and may have a variable number of attributes (fields) however the attribute names must be unique. Rows are grouped into collections (tables) and an individual row's attributes can (with some exceptions) be of any type. The ERS systems are generally based on the *BigTable* model and notwithstanding the marked similarity there are differences in the approaches adopted to implement concurrency.

D. Scalable Relational Database Management Systems

To address the issues and challenges around scalability while attempting to retain the positive features of RDMS *Scalable Relational Database Management Systems* (SRDMS) have been developed to enable improved horizontal scaling for OTP. Unlike the other data stores discussed RDMS require a pre-defined schema, a SQL interface, and ACID transactions.

Traditionally, RDBMS have failed to achieve scalability however recent developments have targeted improved performance levels with the promise of good 'per-node' performance with improved scalability comparable with NoSQL systems. However there are a number of caveats [8], SRDMS use: (1) 'small-scope operations' (operations that span many nodes [joins over many tables, will not scale well with *sharding*], and (2) small-scope transactions (transactions spanning many nodes are generally inefficient due to the communication and two phase commit overhead).

NoSQL systems avoid the two issues identified by limiting the scope (or making it difficult) to perform large scope transactions [6][8]. In contrast, a SRDMS may implement larger scope operations and transactions however such system "simply penalize a customer for these operations" when used [7][8]. SRDMS thus have an advantage over the NoSQL data stores as they provide the convenience SQL and ACID properties; the trade-off lies in that a price is only incurred when nodes are spanned. In the correct domain a SRDMS may be seen as a viable alternative to a NoSQL database system.

V. Illustrative scenarios

Having presented an overview of the traditional RDMS and NoSQL systems. In this section we present a number of brief illustrative examples to demonstrate NoSQL database systems in action. Also we present an illustrative example of a HR scenario.

A. A Key Value Store Illustration

KVS are (generally) good solutions for simple applications where there is a single type of object where a 'look up' of objects is based on a single attribute. The simple functionality KVS make them potentially the easiest to use (particularly if conversant with *memcached*).

Consider a use-case where a web application invokes queries over a RDBMS to create a personalised login page for a user where the user's data rarely changes or it is known when changes are made as updates use the same interface. Generally in such cases there is significant latency in the query execution. It would be advantageous to store the user's tailored page as a single object in a KVS with representation in an format suitable to respond to browser requests and with indexing of these objects by, for example, a user ID. If such objects are stored persistently, then RDBMS queries can be avoided with objects reconstructed only when a user's data is updated. KVS may be used for 'lookups' based on multiple attributes by creating additional key-value indexes. However, at such a point it may be beneficial to consider using a DS approach.

B. Document Store Illustration

In an application such as in a Department of Motor Vehicles application, with vehicles and drivers [where you need to look up objects based on multiple fields such as a driver's name, registration number, vehicle, or DoB] a DS may be a sensible design choice. An important factor to consider is the desired level of concurrency guarantee; if an eventually consistent model with limited atomicity and isolation is acceptable a DS may be a good option. However, in cases where there is a systemic requirement for data be up-to-date and atomically consistent then alternative approaches must be considered as a DS may be an unsuitable system.

C. Extensible Record Store Illustration

Use cases where ERS form a sensible design choice are similar to those for DS: e.g., multiple types of objects with lookups based on any field. However, ERS projects are generally aimed at higher throughput and may provide stronger concurrency guarantees albeit at the cost of slightly more complexity than in DS systems.

For example, in cases where customer information is stored and you wish to scale the data both horizontally and vertically it may be useful to: cluster customers by country and separate rarely-changed 'cor' customer information in one place and locate frequently-updated customer information on a different node (to improve performance). While this approach to scaling may be achieved by programmer implemented scaling onto a DS by creating multiple collections for multiple dimensions, the scaling is most easily achieved with an ERS system.

D. Scalable RDBMS Illustration

Consider the DS example quoted above and further consider a more complex requirement specification where a query interface for law enforcement that can interactively search for attributes is required. In such a use-case ACID transactions would be useful in a database being updated from multiple locations.

Additionally, the definition of a common relational schema and administration tools can be highly beneficial in complex project development with multiple developers. These advantages are dependent on a RDMS being scalable to met the requirements specification. Additionally, there may be issues for RDMS where applications require updates or joins that span many nodes; the transaction co-ordination and data movement in such cases is prohibitive. However, NoSQL systems also (generally) fail to achieve transactions or query joins across nodes.

VI. Electronic Health Records scenario

This section considers HR in the light of the illustrative examples of for the different data store classifications discussed in this paper. Consider the National Health Service (NHS) in the UK [a typical extremely large HR system in global terms]. In examining and treating patients the NHS collects, stores, and processes huge volumes of data frequently in an unstructured format. Patient data in the form of HR may be accessed generally in OTP in computerized systems using with both online (Internet) and networked (internet) systems.

It has been reported in [3] that the NHS currently employs more than 1.7m people with just under half being clinically qualified. There are approximately: (1) 40.000 general practitioners (GP), 380,000 nurses, 19,000 ambulance and paramedic staff , and 106,000 *Hospital and Community Health Service* medical and dental staff [?]; in addition medical staff there are ancillary and administrative staff.

The NHS in England caters for in excess of 53 million patients [3] (importantly, it is worth noting that an individual can be both a patient and a member of staff simultaneously). In terms of access to the system [by both staff and patients] the NHS system on average deals with in excess of 1 million patients every 36 hours [3].The vast amounts of data involved in this operation and the inherent complexity in both the data and the access afforded to staff and patients confirms the challenges faced. In practice to ensure data security and manage the ethical issues inherent in EPR there is a clear need to implement widely controlled and variable access based on defined access rights and permissions.

Additionally, there are complications inherent in adaptation in the presentation and visualisation [of the patient health data] in an appropriate *Human Computer Interface* (HCI) on differing devices including: workstations, notebook computers, tablets, and mobile phones [17]. The range of demands and requirements only serves to emphasize the challenges in developing and implementing a system providing 24/7 OTP access to HR for all stakeholders while addressing a highly dynamic system load and implementing effective data security protocols. The scale of the challenges faced in implementing an effective HR system is clear. Moreover, EPR provide unique opportunity not only to deliver optimal healthcare, but also to optimize use of resources to achieve the best therapeutics outcome. That would have a huge impact on maximizing the cost-effectiveness of care pathways.

A. Treatment Options

Initially consideration must be given to the treatment options available to patients. Patients are examined and treated in multiple locations including for example: (1) a GP practice by a doctor or practice nurse, (2) outpatient clinics, hospital(s) with many departments including *Accident and Emergency* departments, outpatient clinics, and admissions for intensive treatment. In addition to the treatment options there are additional outpatient treatments such as dentistry, chiropody, and chiropractor services. We must also consider long-term support services such as nursing homes where e.g., patients with including Alzheimer's disease are treated and cared for.

When viewed from a patient data storage perspective each of the treatment options may be delivered at a different location (treatment centre) and further complexity is introduced when patients are treated under a different, *Primary Health Trust* (PHT) which is a local organization [in the UK] which manages healthcare provision in, for example, a town or city area. Additionally, there are instances where treatment s carried out using private healthcare.

As alluded to in this paper such complexity in the range of services provided in diverse locations with multiple data structures, database systems (including legacy systems), and data input identifies the challenges faced in creating a comprehensive HR system with OTP. Such complexity poses severe challenges. In a British Computer Society report on the NpfIT project *'The Way Forward for NHS Health Informatics'* [2] the argument is made that: "health informatics need radical improvement" and the observation s made that:

"Meeting the challenges of delivering healthcare in the 21st century requires much improved IT-enabled business systems based on a new IT infrastructure, irrespective of whether care is delivered through the NHS or otherwise, and irrespective of the method used to fund the healthcare system"

It is observed in [2] that there is a need to build on current systems already in place and to: "facilitate the sharing of knowledge, information and workflows across care communities and to include patients and their carers". It is clear that such an informatics infrastructure is critical to the successful implementation of HR systems in the implementation of Government healthcare policy. Developing effective HR systems in a within a Government policy framework is extremely challenging in a complex adaptive system such as the English NHS.

B. *Health Record System Requirements*

There are many systemic requirements in a national HR system which can be summarised as follows: (1) technical requirements including: dynamic OTP using both Internet (online) and internet (Intranet (network) online systems, (2) data structures and database design, (3) implementation of HR using Cloud-Based solutions, and (4) adaptation in respect of date access and presentation based on defined access rights and permissions. There is an important constraint in that: time critical patient data (e.g., patient observations data must be consistent, current, accurate, and immediately available). However there are data (e.g., historical health records data) that will never change and form background information which, while essential as a health record, may not be immediately required or available.

As we have alluded to, essential considerations for HR are security, privacy, legal, and regulatory requirements including ethical and informed consent, patient choice, and GP procurement options. Managing acceptance of HR by both medical professionals, staff, and importantly the public (patients) forms a critical element in the successful and promotion of HR systems where OTP using Cloud-Based Systems (CBS) are developed. Additionally, there are moves by the UK Government to provide 24/7 online access for all patients to their HR [2][3]. This has not received universal approval on security and privacy grounds [3] however the trend towards provision of this service to patients represents a significant policy decision which has implications for both the design and implementation of HR systems and the related data structures and database management systems.

C. *Implementation of System Requirements*

We have considered database systems including NoSQL systems and their features along with HR systems. In considering HR systems the highly complex distributed structure that combines to create a system designed to provide 'birth to death' treatment for patients has been discussed. This structure presents severe challenges for the designers and developers of effective and comprehensive There is a clear requirement to implement a Internet and internet based system and the dynamic loading characteristics of an HR system point to a CBS as a potentially good option. Space restricts a discussion on the relative benefits obtained using CBS (a detailed exposition can be found in [13]) however in summary, a CBS implemented using a *Hybrid Cloud solution* (a combination of a *Private Cloud* and *Public Cloud*) offers the ability to address security and critical data constraints while leveraging the processing capabilities of a public cloud infrastructure to address dynamic processing needs.

Moving on to consider the database design, as with many computing technologies, the selection of a suitable database system is highly domain specific. The prioritization of features and the approach to scaling will be different depending on the application and system needs. In considering HR systems the range and diversity of patient data points to the unstructured nature of such data. The volume of data managed, the multiple locations where data is captured and stored, and the highly dynamic OTP and concurrent access loadings arguably make traditional technologies, such as RDMS, difficult to implement effectively where systems are characterized by unstructured data. It is for such cases that NoSQL systems can achieve scalability (horizontal and vertical) with good performance characteristics.

VII. Discussion

We have considered database systems for structured and unstructured data and a general observation is that while RDMS are mature and well understood technology NoSQL systems are immature and are clearly less well supported, indeed most NoSQL are open source [7][8]. There is a vigorous and ongoing debate on the topic SQL (relational) versus NoSQL [7]. NoSQL databases provide a potentially more effective method of integrating and querying various sources of data with a better query performance; also, scalability can be achieved to the cost of less standardisation. NoSQL has fewer advantages in the storing and processing linked data due to the structure of linked data, however, there are a number of solutions being proposed for NoSQL systems for storing, querying and exposing RDF data [18].

A controversial topic within this debate is *scalability*. In actuality, neither method represents a universally good solution as systems and applications are domain specific requiring domain specific design to meet the diverse requirement of domains of interest. While RDMS can be used in a federated way, efficient SQL query to the distributed databases and scalability become the bottlenecks [18].

The debate around the use of RDMS can be summarised as follows: RDMS are mature technologies, where RDMS performance and scalability matches that of a NoSQL system (with the added convenience of transactions and SQL) a RDMS represents a sensible option. RDMS have been shown to handle specific application loads for: ‘read-only’ or ‘read-mostly’ data warehousing, OTP on multi-core multi-disk CPUs, in-memory databases, distributed databases, and now horizontally scaled databases [8].

The argument for NoSQL can be summarised as follows. It has yet to be demonstrated that RDMS can realise scaling comparable with NoSQL database systems. Where the system requirements require a ‘lookup’ of objects based on a single key, then a KVS approach is generally an appropriate solution. For a DS, the level of complexity can be domain specific. There are applications that require a flexible schema with each object [in a collection] having different attributes. While some RDBMS allow efficient “packing” [8] of *tuples* with missing attributes others support the addition of new attributes at runtime (this however is not common). A RDBMS enables multi-node multi-table) operations while NoSQL systems make them impossible or alternatively make for increased programmer involvement.

From a design perspective, it is generally recognised that attempting to impose an inappropriate solution onto a problem generally results in a poorly performing solution [14]. Thus it is clear that all of the SQL (relational) and NoSQL approaches have their place when appropriately used. Focussing on HR systems our analysis provides support for the conclusion that for distributed systems with large data sets made up of unstructured data a ‘NoSQL’ database solution may provide a potentially useful and profitable approach to the implementation of HR for online systems where OTP forms an important feature.

VIII. Conclusions and Future work

In considering traditional RDMS and NoSQL non-relational database systems in summary each of the database systems considered has its place in the domain specific environments that characterise applications and systems.

The research to date has investigated the potential of NoSQL systems for HR systems; future work will investigate in greater detail the individual data store approaches including KVS, DS, ERS, and SRDMS (in single or hybrid systems) and programming languages such as the *Erlang* programming language [15] to address the programming requirements identified for NoSQL systems to realise an effective HR system. *Erlang* is a declarative language for programming concurrent and distributed systems [16] and has been proposed as a solution to three issues in the development of “highly concurrent, distributed “soft real-time systems”: (1) effective development of software, (2) tolerance to hardware failures resulting in software errors, and (3) the ability to update software on the fly (i.e., without stopping execution). *Erlang* has been successfully used in diverse applications including: distributed databases, financial systems, and chat servers and represent a potentially useful approach to develop NoSQL database systems.

Given the high profile failures in implementing a national HR system with OTP the scale the challenges faced are not underestimated (the NpfIT project identified in the introduction serves as an alarm call) however there are exciting opportunities for the creation of an effective HR systems implemented using NoSQL solutions.

References

[1] C. J. McDonald, “The Barriers to Electronic Medical Record Systems and How to Overcome Them”, *Journal of the American Medical Informatics Association*, Vol 4(3), May / Jun 1997, 1997.

[2] BMJ, “Implementation and adoption of nationwide electronic health records in secondary care in England: final qualitative results from prospective national evaluation in “early adopter” hospitals”, *BMJ*, 2011;343:d6054 doi: 10.1136/bmj.d6054, 2014.

[3] NHS, *“The NHS in England”*, [online], Available from: http://www.nhs.uk/NHSEngland/thenhs/about/Pages/overview.aspx, Accesed: June, 2014.

[4] Chang. F. *et al*, *“ Bigtable: A Distributed Storage System for Structured Data”*, [online], Available from: http://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf, Accessed: June, 2014.

[5] Amazon Web Services, *“Amazon DynamoDB”*, [online], Available from: http://aws.amazon.com/dynamodb, Accessed: June, 2014.

[6] J. P. Sadalage and M. Fowler, *“NoSQL distilled: a brief guide to the emerging world of polyglot persistence*”. Pearson Education, 2012.

[7] N. Leavitt, “Will NoSQL Databases Live Up to Their Promises”, *Computer*, February, 2010, 2010.

[8] R. Cattell, “Scalable SQL and NoSQL Dara Stores”, *SIGMOD Record*, Vol 39(4), December 2010, 2010.

[9] R. Ashford, P. Moore, B. Hu, M. Jackson and J. Wan, “Translational Research and Context in Health Monitoring Systems, In *Proc of the Fourth International Conference on Complex, Intelligent and Software Intensive Systems* (CISIS 2010). Krakow, Poland, February 15-18, pp 81-86, 2010.

[10] R. Elmasari and S. B. Navathe, *“Fundamentals of Database Systems”*, Benjamin/Cummings, 1994.

[11] Oracle, *“Oracle”*, [online], Available from: http://www.oracle.com/index.html, 2014.

[12] Memcached, *“What is Memcached?”*, [online], Available from: http://memcached.org/, Accessed: June, 2014.

[13] P. Moore. and M. Sharma, “Enhanced Patient Management in a Hospital Setting”, *IT CoNvergence PRActice* (INPRA), vol 1(3), pp 1-21, [online], Available from: http://isyou.info/inpra/papers/inpra-v1n3-01.pdf., 2013.

[14] P. Moore. and H. V. Pham, “Personalization and Rule Strategies in Data Intensive Intelligent Context-Aware Systems”, *The Knowledge Engineering Review*, Cambridge University Press, UK (to appear), 2014.

[15] F. Cesarini. and S. Thompson, *“Erlang Programming”*, O’Reilly, Cambridge, UK, 2009.

[16] J. Armstrong, *“Programming Erlang: Software for a Concurrent World”*, Pragmatic Programmers, LLC, 2013.

[17] A. Gentile, A. Santangelo, S. Sorce and S. Vitabile, S. “Novel human-to-human interactions from the evolution of HCI”, In *Proc of the International Conference on Complex, Intelligent and Software Intensive Systems* (CISIS), pp. 600-605, 2011.

[18] O. Curé, R. Hecht, C. Le Duc and M. Lamolle. “Data integration over NoSQL stores using access path based mappings”. In *Proc of the 22nd international conference on Database and expert systems applications*, Volume Part I (DEXA'11), Abdelkader Hameurlain, Stephen W. Liddle, Klaus-Dieter Schewe, and Xiaofang Zhou (Eds.), Vol. Part I. Springer-Verlag, Berlin, Heidelberg, pp. 481-495, 2011.