NoSQL Technologies for Real Time (Patient) Monitoring

Ciprian Dobre

University Politehnica of Bucharest, Romania, ciprian.dobre@cs.pub.ro

Fatos Xhafa
Universitat Politecnica de Catalunya, Barcelona, Spain, fatos@lsi.upc.edu

ABSTRACT
Today we witness a growing change in how public health administration thinks about medical data. We slowly moved from paper-based patient files to digitally storing medical data, in support for advanced evidence-based mining and decision support processes. With this change comes great responsibility, among which efficient storing and accessing the health status of the patient is particularly important. In this chapter, we analyze current storage technologies for storing medical data. Each time more we are witnessing a shift from traditional relational database support, to NoSQL technologies capable to offer great availability and scalability options, and back to the mixture between the SQL and NoSQL worlds, and scalable SQL databases. All these alternatives come with their own pros and cons, which we have carefully analyzed. We believe that our survey will help medical practitioners and developers of health applications take a more informed decision when designing the storage medical data support at long run.
Keywords – Storage technologies; NoSQL; medical data; scalability; availability; patient monitoring, security.

1. INTRODUCTION

The healthcare industry is generating large amounts of data, because of record keeping, compliance and regulatory requirements, and patient care (Aleksovska-Stojkovska & Loskovska, 2013). In the past, much of this data was stored on hard copy form, but today we witness a trend towards medical-related data digitalization. In 2005, only about 30 percent of office-based physicians and hospitals used even basic electronic medical records (EMRs). By 2012, the figures increased to more than 50 percent for physicians and nearly 75 percent for hospitals (Aleksovska-Stojkovska & Loskovska, 2013). More than 45 percent of US hospitals today either participate in local/regional health-information exchanges (HIEs), or are planning to do so in the near future. In US, medical data records that include information for millions of patients are today already being transferred between 80 hospitals, such that over 18,000 physicians take advantage of it (McKinsey, 2013).

Slowly, the huge pile of patient folders in the physician's cabinet is being replaced by the bits and digits stored on hard drives. Behind this move are several advantages: the potential to improve the quality of healthcare by mining records and deriving timely medical conditions, and the need to reduce costs associated with medical practice and care-giving services are among them. The digital form of medical data longly hold the promise of supporting a wide range of medical and healthcare functions, including clinical decision support, disease surveillance and population health management (Dembosky, 2012;

Feldman et al, 2012). There is no surprise that, driven by the industry and health practitioners alike, a big-data revolution in health care is well underway, partly because of vastly increased supply of health information available. Over the last decade, pharmaceutical companies alone have been aggregating years of research and development data into medical databases, while payers and providers have digitalized their patient records. The US federal and EU governments, and various other stakeholders, have been opening their vast stores of health-care knowledge, including data from clinical trials and information on patients covered under public insurance programs. In parallel, recent technological advances have made it easier to collect and analyze information from multiple sources—a major benefit in health care, since data for a single patient may come from various payers, hospitals, laboratories, and physician offices.

The increase in medical data rates being produced led, as expected, to new requirements for the solutions to efficiently and securely store them. The medical data has to be available to stakeholders being potentially in different locations (e.g., the physician might need to access the medical record of the patient, and simultaneous the record could be used in some statistical analysis, run by the hospital administration, regarding the health in a given population; when the patient goes on holiday the physicians there might also need access to his medical records). When availability and scaling are two main requirement pillars for patient data beyond traditional patient EMR records, NoSQL becomes an appealing alternative to storing medical data. For over four decades, data management typically meant relational data processing, and relational database management systems (RDBMSs) became commonplace in all medical data processing environments. But today NoSQL data management systems unleash the full power of cluster environments, and offer simpler key-value data models.

But with all the excitement around the NoSQL hype, today examples of data management implementations based on pure NoSQL databases in healthcare are still missing on large extents. Most NoSQL products analzyed in this chapter are still in beta or research pilots' stages, and largely open source, lacking in support. There are claims that medical apps are inevitably going to be extremely conservative, because people could die if the IT system fouls up (see the famous CAP theorem below). But, still, NoSQL can offer many advantages to the future, which is why we already see today more and more NoSQL Electronic Health Record systems appearing: VistA, CHCS, AHLTA, Epic, Cerner… And, on the horizon, new hybrid mechanisms designed to combine the best of both these two worlds, relational and non-relational, start appearing.

The chapter presents an analysis of current NoSQL (and scalable SQL) technologies, and their applicability to support efficient storing of medical data and patient tracking. We analyze current trends, give insights into the pros and cons of different technological choices, and present the future challenges in need to be addressed. The rest of the chapter is structured as follows. We first present an analysis of the different choices in storage technologies, for both medical data and the particular case of dementia care. We next advance towards an analysis of the different storage technologies in existence, and introduce future challenges in efficient storage of medical data. Finally, we conclude the chapter and present future work.

2. WORKING WITH MEDICAL DATA

Physicians have traditionally used their educated judgment when making treatment decisions, but in the last few years there has been a move toward evidence-based medicine, which involves systematically reviewing clinical data and making treatment decisions based on the best available information. This means that medical data itself has to be there, next to the physician, so availability and scalability starts playing bigger role with all technical health systems. Aggregating individual data sets into big-data solutions often provides the most robust evidence, since nuances in subpopulations (such as the presence of patients with gluten allergies) may be so rare that they are not readily apparent in small samples. No wonder, thus, that in ten years, eighty percent of the work people do in medicine will need to be replaced

by technology (Davis, 2012). And medicine will probably look a little different than it does today. The real breakthrough of big data will be in mashing huge volumes into what John Mattison calls "value sets" that combine our medical science knowledgebase with personal data to prioritize treatment options based on expected outcome, timing, cost, risk and impact on daily life (Vanacek, 2014).

Government-sponsored big-data initiatives in healthcare started already appearing. For example, the Italian Medicines Agency collects and analyzes today clinical data on expensive new drugs, as part of a national cost-effectiveness program (McKinsey, 2013). The results are further used to potentially re-evaluate prices (i.e., associated with medication, health procedures). Within the United States, the federal government has been encouraging the use of its healthcare data, through various policies and initiatives. These efforts, which government leaders hope will directly improve cost, quality, and the overall healthcare ecosystem, generally fall into several areas:

Legislation and incentives to promote data release and accessibility: Adequate legislation on healthcare can make it easier to access public data on patients, clinical trials, health insurance, and medical advances in the future. There are several examples of recent policy directives: the 2009 Open Government Directive, as well as the consequent actions of the Department of Health and Human Services (HHS) under the Health Data Initiative (HDI), are starting to liberate data from agencies like the Centers for Medicare and Medicaid Services (CMS), the Food and Drug Administration (FDA), and the Centers for Disease Control (CDC) (McKinsey, 2013). The Affordable Care Act (enacted in March 2010) includes a provision that authorized HHS to release data that promote transparency in the markets for healthcare and health insurance. The Health Information Technology for Economic and Clinical Health (HITECH) Act, part of the 2009 American Recovery and Reinvestment Act, authorized up to about $40 billion in incentive payments for providers to use EMRs, with the overall goal of driving adoption to 70 to 90 percent of all providers by 2019 (McKinsey, 2013). To facilitate the exchange of information, CMS created the Office of Information Products and Data Analytics to oversee its portfolio of data stores and help collaborate with the private sector. The US federal government is also sponsoring big-data initiatives at the state level. HHS, for instance, recently provided over $550 million in funding for the State Health Information Exchange Cooperative Agreement Program, which is designed to promote the creation of information exchanges. Such data clearinghouses allow clinicians to receive basic information about the treatment that a patient receives.

Data standardization and ease of use: With more data being available, the US federal government is trying to ensure that all appropriate stakeholders, including those in private industry, can access the information in standard formats. For instance, the administration's Big Data Research & Development Initiative, announced in March 2012 by the Office of Science and Technology Policy, made $200 million in funding available to support the release and usability of data stores from agencies in every branch of government (McKinsey, 2013). As another example, the HDI facilitates release of information from HHS through its HealthData.gov Web site. The portal includes federal databases with information on the quality of clinical providers, the latest medical and scientific knowledge, consumer product data, community health performance, government spending data, and many other topics. In addition to publishing information, the HDI aims to make data easier for developers to use by ensuring that they are machine-readable, downloadable, and accessible via application programming interfaces. While more will need to be done, the HDI data are already being used by a variety of new entrepreneurs, as well as existing participants in the healthcare ecosystem.

Still, there are advocates that say that the massive release of medical data to the public can be dangerous. The problem is patients are losing control over whom and when can use the data, under what circumstances. This can, on long-term, potentially backfire, as medical data is also critical data. Thus, adequate security and privacy mechanisms need to augment storage systems. Technology needs to

support adequate access policies, and the law has to protect our medical records. With more data come bigger responsibilities!

3. DATA STORAGE FOR DEMENTIA PATIENT MONITORING

When it comes to medical problems, dementia is the term used to describe various symptoms of cognitive decline such as forgetfulness[1]. People with dementia face a decline of their cognitive functions, including memory impairment and difficulty to orient in time and space. Forgetfulness is generally seen as the primary symptom of dementia, but this is not necessarily the only case for any pathology. Usually, cognitive abilities irreversibly deteriorate over time and the underlying disease is incurable. Mandell and Green (Mandell & Green, 2011) describe dementia as "a syndrome of acquired persistent intellectual impairments characterized by deterioration in at least three of the following domains: memory, language, visuospatial skills, personality or behavior, and manipulation of acquired knowledge (including executive function)".

Statistics show there are over 800k people with dementia in UK alone, and this will increase to over a million people by 2021 (AS, 2014). In fact, the number of people being diagnosed with dementia doubles for every 5 year age group, and the financial cost of dementia in UK alone was 22 billion pounds in 2012 (AS, 2014). The organization Alzheimer's Disease International estimates that the worldwide cost of dementia amounts to 1% of the global gross domestic product, or USD 604 billion (Wimo & Prince, 2010). Dementia incidents are higher with elderly. With estimates showing that by 2050 one third of Europe's population alone will be over 60 (Eurostat, 2012), no wonder we need to think of alternative means to provide dementia caregiving services and support an independent life even in old age.

Technology has the potential to improve and extend the quality of life of older people and people with disabilities, by helping them have more independent lives, and by improving the efficiency and effectiveness of (medical) services being provided to them. No wonder that today we witness an increasing in the interest research take on this particular topic.

The Keeping In Touch Everyday (KITE) (Robinson et al, 2009) project developed an outdoor navigation and communication system for Person with Dementia (PwDs). As the PwD carries around his mobile device, KITE provides a call button (within an app running on the smartphone) to contact the caregiver, and a navigation function (for instructions to find the way home). Additionally, the mobile device tracks the PwD's position as well as the device and alert state. All these contextual information is presented to the caregiver via a web interface. COACH (Hoey et al, 2010) is an activity assistant for the hand washing process. The system tracks visually the user's hand movement and process relevant items. The system bases its decisions on current observations and the user's level of dementia and emotions. When the PwD experiences difficulties with the task at hand, COACH presents the user with either visual or audio prompts. If prompts are ineffective and the PwD resume the task, the caregiver is called. COGKNOW (Davies et al, 2010) is an EU funded project to develop a comprehensive support system for people with mild dementia. A PwD accesses services either through a stationary touch-screen interface or a mobile device. Activities are supported through a context-aware reminder application, and video recorded instructions provided for a set of activities and simplified access to services. Context-Aware Way finder (Chang et al, 2008b) developed a navigation system for people with cognitive impairments, including dementia. The system guides their users to a destination by presenting images of the next waypoint and overlaying directions for the user's route. User and CG can call if needed. The system records the route progress and compliance. It presents this monitoring information to an authorized person.

[1] We acknowledge that dementia is not a clinical diagnosis itself, unless an underlying disease or disorder has been identified.

Still, in all these projects, it became clear that patient and context data has to be handled efficiently to cope with the dynamics and needs of dementia patients and their carers. At the center of such systems lies a data repository, in charge with storing all medical data. Traditionally, such projects use relational technologies for this – but, with time, the medical record of the patient increases quite fast. Imagine if the system needs to store timely locations of the user, to re-construct routes, and further on, on top of the data repository the caregiver would like to run some data mining algorithm to find preferred paths for the patient (e.g., for personalized medical facts the physicians might want to find out the average daily walking distance of his patient). A relational database still can handle such large data analysis. But, now, imagine that the system would like to run such analysis in near real-time, to offer personalized navigation instructions or detect when the user deviates from his typical routes (a medical sign that the patient might experience a sporadic loss of memory episode, wondering around). And multiply this with the thousands of patients probably simultaneously using the same medical navigation and alert system…. which is where alternative technologies for data storing, presented next, enter the scene…

4. FROM RELATION TO NOSQL APPROACHES

Databases can be found in most current hospital systems (e.g., HIS, RIS, PACS, Lab), smaller departmental applications, and in use by single users. There are many types of databases, as they have progressed from early flat file lists to hierarchical structures to Structured Query Language (SQL) relational tables. Starting with the relational model, several database models were subsequently introduced. The object oriented and object relational databases never really become competitive in the marketplace. XML databases were further developed to support the proliferation of XML documents. However, the adoption of native XML databases was very limited. Major relational database vendors such as Oracle, Microsoft and MySQL have included XML support for their products, but native XML databases (e.g., Tamino) have not captured much share of the database market.

Many concerns arise when dealing with healthcare data. Such data is very high in dimensionality, can be extremely complex and skewed, particularly because a single patient's record can contain hundreds of attributes. The data is both sparse and noisy.

Although relational theory has been the primary force in medical databases for many years, Object Oriented and NoSQL (which stands for "Not Only SQL") are providing interesting new possibilities. In the last few years, with the introduction of BigTable and MapReduce by Google (Chang et al, 2008a), NoSQL databases have emerged. For once, NoSQL are better adapted for storing large data over clusters of storage nodes, thus offering great scalability and fast access. The medical data in a database can generally be stored in a single physical file (as with the traditional relational model), or can be partitioned (i.e., it can be stored under multiple administrative servers, over multiple machines). The latter is known as a federated database, and allows multiple physically separated data files to appear logically as pertaining to a single database. Most of the data partitioning systems allow horizontal partitioning of data (medical records are stored on different servers, according to some key, such as the names of patients or their social IDs; the partitioning strategy in this case is called "sharding"), but there are data storing systems that allow vertical partitioning (parts of a single medical record are stored on different servers). We note that most modern relational SQL and NoSQL databases can (in different ways) partition in some form data over a cluster of storage nodes, for scalability.

A key feature of NoSQL systems is actually their "shared nothing" horizontal scaling support – they are capable to replicate and partition data over many storage servers. This allows them to support a large number of simple read/write operations per second, because generally the operations can be done on different storage nodes.

A simple load operation is traditionally called OLTP (online transaction processing). NoSQL data stores are designed to scale simple OLTP-style application loads over many servers, and to scale to thousands or millions of users doing updates as well as reads, in contrast to traditional RDBMSs and data warehouses.

In summary, a NoSQL database generally provides several key features compared to traditional relational alternatives: ability to scale horizontally simple operations (reads/writes), throughput over many (geographically distributed in some cases) data storage servers, and the ability to replicate and distribute (partition) data over many such servers.

Still, NoSQL data storage systems differ in various ways. The key property common to all of them is that they generally do not provide ACID (Atomicity, Consistency, Isolation, and Durability) transactional guarantees: this means that updates are eventually (or causally, as with newer storage technologies) propagated, but there offer limited guarantees on the consistency of reads (a read operation following a write will eventually, after a while, see the latest update over the data). The idea is that by giving up ACID constraints, one can achieve much higher performance and scalability on storing medical data (which is important when data is stored at a high velocity, in high volumes, and reliability guarantees are needed). However, NoSQL data store systems differ in how much they give up ACID constraints. Most of the data storing systems today call themselves "eventually consistent", meaning that updates are eventually propagated to all nodes, but many of them provide mechanisms for some degree of consistency, such as multi-version concurrency control (MVCC).

Proponents of NoSQL often cite Eric Brewer's CAP theorem (Gilbert & Lynch, 2002), which states that a system can have only two out of three of the following properties: 1) has strong *consistency*, 2) is always *available* for reads and writes, and 3) is able to continue operating during network *partitions*. Each of these properties is highly desirable. Strong consistency makes programming easier. Availability ensures front-end Web servers can always respond to client requests. Partition tolerance ensures the system continue operating even when datacenters cannot communicate with each other. Facing with the choice of at most two of these properties, NoSQL systems chose to sacrifice strong consistency for the other two. However, the trade-offs are complex.

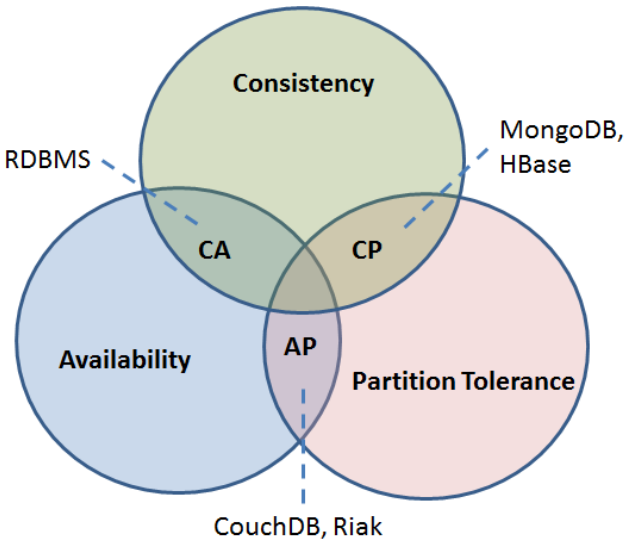

Figure 1: Examples of database technologies in relation to the CAP model (after W3Resource, 2014).

New relational DBMSs have also been introduced in the last years, to provide an alternative horizontal scaling for OLTP (compared to traditional RDBMSs). The SQL data storing systems strive to provide horizontal scalability without abandoning SQL and ACID transactions. The problem with systems such as Dynamo, Project Valdermort, and others, is that they sacrifice strong consistency for availability, guaranteeing responses with low latency, providing partition tolerance, and ensuring scalability (adding more NoSQL storage servers produces a proportional increase in the storage capacity and throughput). For this, they settle for *eventual consistency*: all writes to one datacenter will eventually appear at other datacenters. The problem with eventual consistency is that it can lead to inconsistent order of operations. For example (see Figure 2), with a medical storage system, if someone connects to the left datacenter and sets a diagnosis in the patient's medical record, and then inputs a given treatment being administrating to that treatment, then someone else connected to the right datacenter might see that first the patient receives a treatment, and as a result he developed a given condition… Therefore, eventual consistency for concurrent access data storage systems can lead to life-threatening medical decisions.

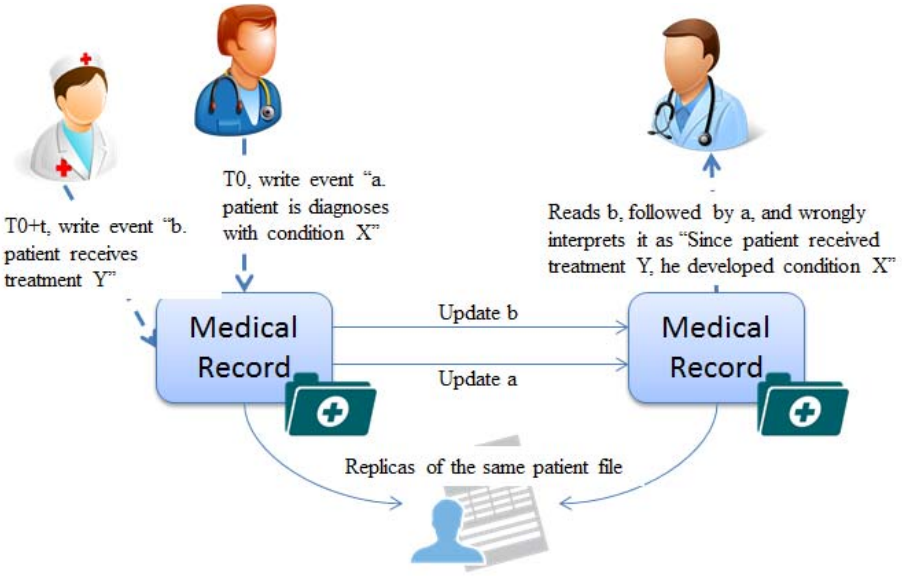

Figure 2: Example of inconsistency when updating medical records, using eventual consistency.

Fortunately, with *causal consistency* and/or transactional guarantees, as most NoSQL and modern SQL systems today provide, such problems are solved. Still, even with causal consistency, global invariants are hard to apply in medical information system, for the case when medical patient records need to be spread and shared over multiple data centers. An example of this is the following scenario: two pharmacies work on the same medical supply NoSQL database provided by a drug company (each pharmacy sees the closest datacenter). Suppose we have one unit of some hypothetical medicine, and then pharmacy A and pharmacy B concurrently sold to their customer this last unit (promising to their customers the medicine will be delivered by FedEx the very next day). Once both updates are propagated to each datacenter, it could be too late to prevent the mistake. The invariant that in all datacenters the supply account cannot go below 0 (in this case) is hard to impose using NoSQL databases. Availability dictates that operations must complete, and low latency ensures they are faster than the time it takes to communicate between geographically distributed data centers. But, fortunately, global invariants such as these are rare in Medical information systems, and, even more fortunate, alternative storage systems that provide more strict consistency guarantees can be used in this case.

Due to the characteristics of today medical data/applications, such as the heterogeneity in the data types, the need to store extremely huge/ever-increasing data size (i.e., image and video data generated by RMN and other medical equipment can overload fast storage servers), and the need to lower costs associated with storage (by using more aggregated commodity storage servers, instead of spending outrageous amounts of money on fancy storage solutions), various Medical Information Systems seek today to exploit the power of NoSQL and latest scalable SQL solutions. Medical data need to scale, need to handle replication and scalability. Data is stored at large volumes, it comes from increasingly number of sources, and high velocity can be the norm rather than the exception in medical accesses. NoSQL can provide cost-effectiveness, disaster recoverability, elasticity, manageability, and availability. Healthcare data stored in such storage system can be accessed quite fast, meaning it can be processed at incredible speed.

No wonder insurance, retail and Web companies, life sciences companies and research entities alike, are all turning to new big data technologies to enhance their results from their high performance systems by allowing more complex data ingestion and processing. But, in the Big Data era, storing huge amounts data is not the biggest challenge anymore. Companies already store huge amounts of information. Apart from storing, today researchers struggle with designing solutions to understand this amount of Big Data. Efficient parallel and concurrent algorithms and implementation techniques are needed to meet the scalability and performance requirements entailed by medical data analyses. Challenges such as scalability and resilience to failure are already being addressed at the infrastructure layer. But new Big Data problems relate to users handling too many files, and/or working with very large files. Applications need fast movement and operations on that data, not to mention support to cope with an incredible diversity of data. Big data issues also emerge from extensive data sharing, allowing multiple users and health-related stakeholders, to explore or analyze the same data set. All these demand a new movement and a new set of complementary technologies.

To give an example, Digital Imaging and Communications in Medicine (DICOM[2]) is one of the most important medical standards. The primary objectives of the standard are to achieve interoperability between medical imaging systems and to facilitate medical data exchange. The wide use of this standard in the medical domain has led to the development of various DICOM management systems: the Picture Archiving and Communication System (PACS), eDiaMoND, a grid-enabled database of mammogram images, or the ORDICOM data type in Oracle 11G (Mohamad et al, 2012). Unfortunately, such systems are even today highly expensive, IT experts dependent, weak expressiveness or/and not scalable. Particularly, in current systems the crash of a server may prevent doctors from getting the required image if it is not stored on a separate portable disk. No surprise, the disparity of data management requirements (analytical/statistical needs) has led to the development of new DBMSs other than the traditional databases. Storage systems that either relies on a column-oriented storage architecture, or on a hybrid row-column one, still suffer from the high tuple reconstruction time and/or their inability to overcome DICOM heterogeneity issue. That is due to fact that DICOM files are subject to very large-scale heterogeneity and ever-evolutive schema. Fortunately, NoSQL schemas such as the one presented in the followings, have the potential to provide an answer to such storage problems for medical data…

4.1 A Data as a Service Approach

Different from traditional patient data record, in patient data monitoring efficient access to huge volumes of data should be provided to different actors in the system. To this end, Terzo et al (2013) developed a *Data-as-a-Service* (DaaS) approach.

[2] http://medical.nema.org

DaaS is based on data virtualization to overcome limitations of state-of-the-art approaches in data technologies, according to which data is stored and accessed from repositories whose location is known and is relevant for sharing and processing. Besides limitations for the data sharing, current approaches also do not achieve to fully separate/decouple software services from data and thus impose limitations in inter-operability. Additionally, DaaS can support large communities of users that need to share, access, and process the data for collectively building knowledge from data. It also address the needs of accessing the same data from different actors (doctors, nurses, carers and stakeholders) and achieving thus different data views according to actors' access rights.

4.2 Security and Privacy Issues for NoSQL Data

Medical systems are sensitive systems in terms of security, privacy, anonymity and rights access to data.

While security solutions for RDBMSs are in place, there is a clear lack of similar mature solutions for NoSQL data storage. Research is underway for security and privacy of NoSQL data. (Xhafa et al, 2013) considers security of patient record data at Cloud using fuzzy keyword search to ensure fine-grained access control. Additionally, in (Xhafa et al, 2014) attribute-based encryption techniques are used to design cloud-based electronic health record system with attribute-based encryption, which guarantees security and privacy of medical data stored in the cloud. The techniques are exemplified in Figure 3.

4.3. Audit trials for patient data security and privacy

The state of the art using Cloud computing technologies has so far been concerned with security and privacy issues but without addressing two issues: 1) access to data is multi-user access and 2) patients have not been taken into account. With regard to the first, the multiple-access to the data by different doctors or teams of doctors is a must. For example, a doctor of a patient of dementia should have access also to routinely collected data of the patients for physiological parameters such as blood pressure, temperature, weight, etc. over time. Regarding the second, the question is whether the patient should know who is accessing the data and even to be asked to give consent to who can access his data.

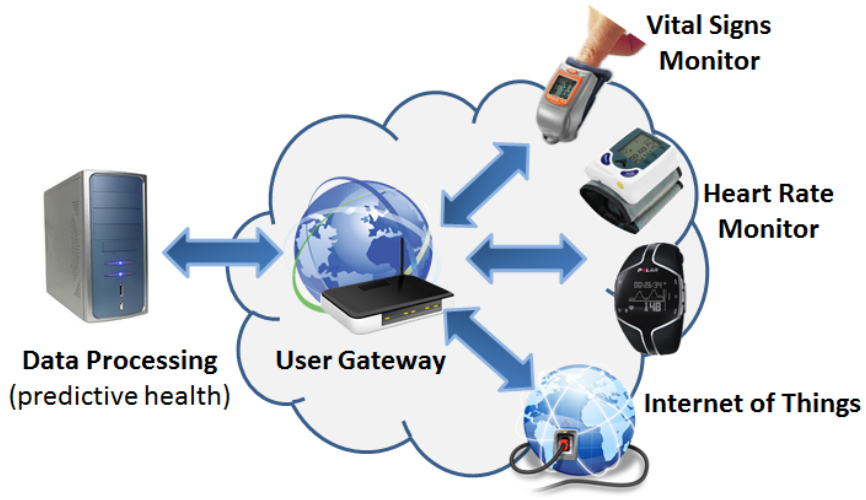

Figure 3: Techniques used in cloud-based electronic health record systems.

One approach that can satisfactorily address both issues is the audit trial, defined as (see National Information Assurance Glossary, 2014):

An audit trail (also called audit log) is a security-relevant chronological record, set of records, and/or destination and source of records that provide documentary evidence of the sequence of activities that have affected at any time a specific operation, procedure, or event.

The difference between applications of audit trials for transaction logs or in other field where are limited to the privacy/security office, with that of EPR is that such audit trial can be also available to patients (or their caregivers), who can act upon. In fact, the audit trial for EPR can be enriched also with other information such as location access, medium access, etc. to address some location-based security issues. It should be mentioned however that there is a challenge to audit trials for NoSQL EPR systems. Indeed, assuming EPR data is large and assuming active interaction with the data, the amount of audit log would be too large to be human readable and manageable. This brings us again to the need for a data cycle definition for audit trial for EPR.

5. IMPLEMENTATION OUTLOOK

In the last years a variety of NoSQL databases has been developed, mainly by practitioners and Web companies, designed to fit their specific requirements regarding scalability performance, maintenance and feature-set. Some of these databases have taken up ideas from either Amazon's Dynamo or Google's Bigtable (or a combination of both). Because of the variety of approaches and overlapping in nonfunctional requirements and their features, it can be difficult to get and maintain an overview of the non-relational database scene. There have been various approaches to classify and subsume NoSQL databases, each with different categories and subcategories. One of the mostly cited classifications, which we use as well, subsumes different NoSQL databases primarily according to their data model (Table 1).

Table 1. Classification of non-relational databases.

Category	Examples of matching databases
Key-value Stores	Redis, Scalaris, Tokyo Tyrant, Voldemort, Riak
Document Stores	SimpleDB, CouchDB, MongoDB, Terrastore
Extensible Record Stores	Bigtable, HBase, HyperTable, Cassandra

5.1. Key-Value Stores

The simplest NoSQL data stores use a single key-value index for all the data. Such systems are known as *key-value stores*, and they generally provide a persistence mechanism and additional functionality such as: replication, versioning, locking, transactions, sorting, and others. A client interface provides operations such as inserts, deletes, and index lookups. The key here is that none of these systems offer secondary indices or keys (this is why they provide the most efficient storage support for unique-key systems, but fail for complex ones).

Project Voldemort (Sumbaly et al, 2012) is an advanced key-value store, widely known today also because of substantial contributions and support coming from LinkedIn. It can store data both in memory and in physical files. As a NoSQL data storing system, Voldemort provides multi-version concurrency control (MVCC) for updates, meaning multiple replicas can be updated asynchronously, each write operations generates a new version of the data, various clients can access concurrently various versions of the data, and it does not provides guarantees for data consistency. Voldermort can only guaranty an up-to-

date view if/when a majority of replicas are being read (this is the only guarantee, according to its designers). Voldemort also supports optimistic locking for consistent multi-record updates: if updates conflict with any other process, they can be backed out (a rudimentary construct similar to transactions). Vector clocks, constructs previously largely mediated by storage systems such as Dynamo (Cooper et al, 2010), can be used for ordering different data versions. For availability guarantees, Voldemort supports automatic sharding of data: consistent hashing is used to distribute data around a ring of nodes, and hashed data is automatically replicated to several logically-adjacent nodes. Once data partitioning is set up, all partitioning operations are transparent. Nodes can be added or removed from a database cluster, and the system adapts automatically. More, Voldemort automatically detects and recovers storage nodes that failed.

Riak (Klophaus, 2010) is a NoSQL data storage system based on both a "key-value store" and "document store" scheme. The Riak architecture is symmetric and simple. Like Voldemort, it uses consistent hashing. Storage nodes use a gossip protocol to track who is alive (not failed) and who has which data. Riak uses a map/reduce mechanism to split work over all the nodes in a cluster. Riak objects can be fetched and stored in JSON format, and thus can have multiple fields (like documents). Objects can be grouped into buckets, supporting collections (we present later in this section more on such constructs, when discussing document-based NoSQL storage systems). However, unlike pure document storage systems, Riak does not support indices on any fields except the primary key (which is why we include it in the single-key category). The non-primary fields are used just for fetch and store operations, and are part of a JSON object.

Riak supports replication of objects and sharding by applying a hash function on the primary key. Replica values can be temporarily inconsistent, but consistency is tunable by specifying how many replicas (on different nodes) must respond for a successful read, and how many must respond for a successful write. Similar to Voldemort, Riak uses a derivative of MVCC, where vector clocks are assigned when values are updated. Vector clocks can be used to determine when objects are direct descendants of each other or a common parent. And, a unique feature of Riak is that it can store "links" between objects (documents), for example to link objects for patients to the objects for their associated medical records. The link is a useful mechanism to reduce the need for secondary indices.

In Danemark, Riak is being used by the Board of eHeatlth to handle the storage of all medicine prescriptions, and provide access to several health organizations for online reads and updates (Trifork, 2014). Riak's architecture enabled the Shared Medication Record to scale up, and provided availability at an affordable price for Shared Medication Records. In 2011 the Shared Medication Record system won the Danish government digitization prize for significantly reducing medical errors and increasing patient safety.

Scalaris (Schütt et al, 2008) allows key ranges to be assigned to nodes when distributing data, rather than simply hashing to nodes (in this case, a query on a range of values will not to go to every node). Scalaris allows better load balancing, depending on key distribution, compared to other key-value stores. Replication is done synchronously (copies must be updated before the operation is complete) so data is guaranteed to be consistent. Scalaris also supports transactions with ACID properties on multiple objects. In this case, data are stored in memory, but replication and recovery from node failures provides durability of the updates. In Scalaris reads and writes must go to a majority of the replicas before the operation completes, leading to a replication strategy that requires logarithm steps for each individual read/write operations.

The **memcached** (Nishtala et al, 2013) open-source distributed in-memory indexing system, together with Membrain and Membase, include features for key-value storage such as persistence, replication, high availability, dynamic growth, backup, and provide high performances due to memory-mostly storage

support. Membase's attractive feature beyond memcached is its ability to elastically add or remove servers in a running system, moving data and dynamically redirecting requests in the meantime. Membrain, on the other hand, provides excellent tuning for flash memory. Even more advanced, **Redis** (Redis, 2014) is similar to memcached but in this case the dataset is not volatile, and values can be strings (like in memcached) but also lists, sets, and ordered sets. All data types can be manipulated with atomic operations to push/pop elements, add/remove elements, perform server side union, intersection, difference between sets, and so on. Redis supports different sorting strategies. It is very fast but at the same time persistent, because the whole dataset is taken in memory, and saved on disc asynchronously (semi persistent mode) - or alternatively, every change is written into an append-only file (fully persistent mode).

Such key-value data storage stores support operations such as insert, delete, and lookup. They provide scalability by distributing keys over multiple nodes. Voldemort and Riak, and enhanced memcached systems, can store data in RAM or on disk, with storage add-ons. The others store data in RAM, and provide disk as backup, or rely on replication and recovery such that a backup is not needed. Scalaris and enhanced memcached systems use synchronous replication, the rest use asynchronous. Scalaris also implement transactions, while the others do not. Voldemort and Riak use multi-version concurrency control (MVCC), and the others use locks. Membrain and Membase are built on the popular memcached system, adding persistence, replication, and other features. Backward compatibility with memcached gives these products an advantage. For medical data storage, key-value systems are just starting to become attractive (see the example above coming from Denmark). Their properties alone are still insufficient to preserve security-constrained imposed by sensitive information such as patient records. Thus, various architectures (Pasquier et al, 2014), combine key-value storage with traditional SQL data storage support for medical records. In this case, the key-value data storing is mostly used for fast access to the information, which in turn is preserved on the more secure technologies.

5.2. Document Stores

Document stores support more complex data than key-value stores. Such systems could store traditional documents (articles, word files, etc.), but in a more generic sense a document can be any kind of object. Unlike the key-value stores, these systems generally support secondary indexes and multiple types of documents (objects) per database, and nested documents or lists. Like other NoSQL systems, the document stores do not provide ACID transactional properties.

SimpleDB (Sciore, 2007) is part of Amazon's proprietary cloud computing offering, along with Elastic Compute Cloud (EC2) and the Simple Storage Service (S3) on which SimpleDB is based. Its model is simple: it has Select, Delete, GetAttributes, and PutAttributes operations on documents. SimpleDB is simpler than other document stores, as it does not allow nested documents. Like most of the previous key-value systems, SimpleDB supports eventual consistency, not transactional consistency, and asynchronous replication. Unlike key-value datastores, SimpleDB supports more than one grouping in one database: documents are put into domains, which support multiple indexes. Different domains may be stored on different Amazon nodes. Domain indexes are automatically updated when any document's attributes are modified. Finally, SimpleDB does not automatically partition data over servers. Some horizontal scaling can be achieved by reading any of the replicas. Writes do not scale, however, because they must go asynchronously to all copies of a domain. If customers want better scaling, they must do so manually by sharding data themselves.

CouchDB (Rascovsky et al, 2012) is an Apache project that organizes "collections" of documents similar to a SimpleDB domain. Collections comprise the only schema in CouchDB, and secondary indexes must be explicitly created on fields in collections. A document has field values that can be scalar (text, numeric, or boolean) or compound (a document or list). Indexes are implemented as B-trees, and

results of queries can be ordered or value ranges. Like SimpleDB, CouchDB achieves scalability through asynchronous replication, not through sharding. Reads can go to any server, and updates must be propagated to all the servers. Like SimpleDB, CouchDB does not guarantee consistency. Unlike SimpleDB, each client sees a self-consistent view of the database, with repeatable reads: CouchDB implements multi-version concurrency control on individual documents, with a Sequence ID that is automatically created for each version of a document. CouchDB also provides durability on system crash. All updates (documents and indexes) are flushed to disk on commit, by writing to the end of a storing file. By default, it flushes to disk data after every document update. Together with the MVCC mechanism, CouchDB's durability thus provides ACID semantics at the document level.

Today CouchDB and Membase merged to form Couchbase, and the new company plan to provide a "best of both" merge of their products, e.g. with CouchDB's richer data model as well as the speed and elastic scalability of Membase.

CouchDB was used in several Medical information systems (Rascovsky et al, 2012; Schmitt & Majchrzak, 2013; Schmitt & Majchrzak, 2012). For example, (Schmitt & Majchrzak, 2012) presents a medical information system where patient records are stored in medical documents, shared and made highly available through CouchDB. The feasibility of the storage system is proven even in case of unreliable environments. Table 2 presents an example of the mapping of the medical patient to the CouchDB's document storage schema (from Schmitt & Majchrzak, 2012).

Table 2. Mapping between patient data and the document storage schema.

```
{ _id: "fs_data_medication_1293C-06_2",
        _rev: "D1C946B7",
        data: {
                medication: {
                        [ { Date: "10/08/2011",
                                Amount: "5"
                                },
                                { Date: "09/28/2011",
                                Amount: "12"
                                },
                        ] },
                }
        formdef: "fs_forms_medication_1",
        metadata: {
                changed-by: "f242-dc92",
                changed-at: "10/11/2011 – 11.23pm",
                prev-rev: "fs_data_medication_1293C-06_1"
        },
        signature: "d8e8ece39c407e515aa8997c1a1e94f1fd2a0e62"
}
```

MongoDB (Neuhaus et al, 2011) is a popular open-source document store that provides a document-oriented storage layer, B-trees indexes on collections, auto-sharding and asynchronous replication of data between servers. It supports automatic sharding by distributing documents over multiple servers. MongoDB supports dynamic queries with automatic use of indices, similar to RDBMSs. Unlike CouchDB, MongoDB does not provide MVCC on documents, but instead provides atomic operations on fields. Its indices have to be explicitly defined, and any existing indices are automatically used for query processing.

In MongoDB, data is stored in *collections*, and each collection contains *documents*. Collections and documents are loosely analogous to tables and records, respectively, found in relational databases. Each document is serialized using a binary JSON-like format (called BSON). Still, MongoDB does not require a rigid schema for documents; multiple documents in the same collection can have different structures.

MongoDB also supports a GridFS specification for large binary objects (eg., images and videos). These are stored in chunks that can be streamed back to the client for efficient delivery. MongoDB supports master-slave replication with automatic failover and recovery. Replication (and recovery) is done at the level of shards. Collections are automatically sharded via a user-defined shard key. Replication is asynchronous for higher performance, so some updates may be lost on a crash.

5.3. Extensible Record Stores

The idea of an extensible record store is motivated by Google's success with BigTable (Chang et al, 2008a). The data model is based on rows and columns, and the scalability is achieved by splitting both rows and columns over multiple nodes. Generally, rows are split (by range, rather than a hash function) across nodes through sharding on the primary key. On the other hand, columns of a table are distributed over multiple nodes by using "column groups" – a simple way for the customer to indicate which columns are best stored together.

With extensible record stores, rows are analogous to documents: they can have a variable number of attributes (fields), the attribute names must be unique, rows are grouped into collections (tables), and an individual row's attributes can be of any type.

HBase (Jin et al, 2011) is a storage system based on the Hadoop distributed file system. With HBase, all updates are placed in memory, and periodically all writes are flushed on disk. Row operations are atomic, with row-level locking and transactions. Partitioning and distribution are transparent, as there is no client-side hashing or fixed keyspace as in some NoSQL systems. HBase's B-trees allow fast range queries and sorting.

An HBase storage system comprises a set of tables. Each table contains rows and columns, much like a relational database. Each table must have an element defined as a Primary Key, and all access attempts to HBase tables must use this Primary Key. An HBase column represents an attribute of an object. For example, if the table is storing diagnostic logs from servers in an environment, where each row might be a log record, a typical column in such a table would be the timestamp of when the log record was written, or perhaps the server name where the record originated. In fact, HBase allows for many attributes to be grouped together into what are known as column families, such that the elements of a column family are all stored together. This is different from a row-oriented relational database, where all the columns of a given row are stored together. With HBase, one must predefine the table schema and specify the column families. However, it is very flexible in that new columns can be added to families at any time, making the schema flexible and therefore able to adapt to changing application requirements.

HyperTable (Jin et al, 2011) is quite similar to HBase. It uses column families that can have any number of column qualifiers, and adds timestamps on data with MVCC. It requires an underlying distributed file system such as Hadoop, and a distributed lock manager. Tables are replicated and partitioned over servers by key ranges. Updates are done in memory and later flushed to disk. HyperTable supports a number of programming language client interfaces, but natively it uses a query language named HQL.

From a user perspective, in HyperTable the data model has a database that contains tables. Each table consists of a set of rows. Each row has a primary key value and a set of columns. Each column contains a set of key value pairs commonly known as a map. A timestamp is associated with each key value pair. The only query method is a table scan. Tables are stored in primary key order, so a query easily accesses a row or group of rows by constraining on the row key. The query can specify which columns are returned, and the time range for key value pairs in each column.

The basic unit for inserting data is the key value pair, along with its row key and column. An insert will create a new row if none exist with that row key. More likely, an insert will add a new key value pair to an existing column map or have the existing value superseded if the new column key already exists in the column map.

Cassandra (Lee et al, 2013) is similar to the other extensible record stores in its data model and basic functionality. Cassandra is used by Facebook as well as other companies, so the code is reasonably mature. As a data storing system, Cassandra uses column groups, and all updates are cached in memory and then flushed to disk, and the disk representation is periodically compacted. It does partitioning and replication. Failure detection and recovery are fully automatic. However, Cassandra has a weaker concurrency model than some other systems: there is no locking mechanism, and replicas are updated asynchronously.

Cassandra adds the concept of a supercolumn, which provides another level of grouping within column groups. Databases (called keyspaces) contain column families. A column family contains either supercolumns or columns (not a mix of both). Supercolumns contain columns. As with the other systems, any row can have any combination of column values (i.e., rows are variable length and are not constrained by a table schema).

Cassandra seems to be gaining a lot of momentum, as an open source project. Apixio, a US company, uses Cassandra for serving search queries on medical records. MedProcure uses Cassandra to help provide software solutions for ordering and tacking medical supplies.

All these storing solutions took over mostly after BigTable. They are all similar, but differ in concurrency mechanisms and other features. Cassandra focuses on weak concurrency (via MVCC), while HBase and HyperTable focus on strong consistency (via locks and logging).

5.4. Scalable Relational Systems

Unlike the NoSQL data stores, relational DBMSs have a complete pre-defined schema, a SQL interface, and ACID transactions. Traditionally, RDBMSs have not achieved the scalability of the some of the previously-described data stores. Recent developments are changing things. Performance improvements were introduced in MySQL Cluster, and several new products have come out, in particular VoltDB and Clustrix, that promise to have good per-node performance, as well as scalability. It appears likely that some relational DBMSs will provide scalability comparable with NoSQL data stores, particularly when dealing with small-scope operations (operations that span many nodes, e.g. joins over many tables, do not scale well with sharding) and with small-scope transactions (transactions that span many nodes are quite inefficient, with the communication and two-phase commit overhead).

NoSQL systems avoid the performance and scalability problems, by making it difficult or impossible to perform larger-scope operations and transactions. In contrast, a scalable RDBMS does not need to preclude larger-scope operations and transactions: they simply penalize a customer for these operations if they use them. Scalable RDBMSs thus have an advantage over the NoSQL data stores, because they provide convenience of a higher-level SQL language and support for ACID properties, with a small price to pay for those when they span nodes. Scalable RDBMSs are therefore included as viable in our analysis for storage systems for medical data.

MySQL Cluster is part of the MySQL, and replaces the original InnoDB engine (in MySQL) with a distributed layer called NDB. It shards data over multiple database servers (a shared nothing architecture), and supports bi-directional geographic (i.e., location-based) replication and in-memory, as well as disk-based data writing. MySQL Cluster seems to scale to more nodes than other RDBMSs to date. Reports

state that it can reportedly run into bottlenecks after a few dozen nodes (Cattell, 2011), but in recent versions such problems are less frequently encountered. Work continues on MySQL Cluster, which is only going to make it a viable alternative for a medical data store in the future.

VoltDB (Mian, 2014) is an open-source RDBMS designed for high performance (per node) as well as scalability. The scalability and availability features are competitive with MySQL Cluster and NoSQL systems. Tables are partitioned over multiple servers, and clients can call any server. The distribution is transparent to SQL users, but the customer can choose the sharding attribute. Alternatively, selected tables can be replicated over servers, e.g. for fast access to read-mostly data. In any case, shards are replicated, so that data can be recovered in the event of a node crash. Database snapshots are also supported, continuous or scheduled.

Some features are still missing, e.g. online schema changes are currently limited. Asynchronous WAN replication and recovery are not yet mature enough. However, VoltDB has some promising features that collectively may yield an order of magnitude advantage in single-node performance. VoltDB eliminates nearly all waits in SQL execution, allowing a very efficient implementation. The system is quite fast, being designed for a database that fits in (distributed) RAM on the servers, so that the system never waits for the disk. Indexes and record structures are designed for RAM rather than disk, and the overhead of a disk cache/buffer is eliminated as well. Performance can be very poor if virtual memory overflows RAM, but the gain with good RAM capacity planning is substantial. SQL execution is single-threaded for each shard, using a shared-nothing architecture, so there is no overhead for multi-thread latching. All SQL calls are made through stored procedures, with each stored procedure being one transaction. Data is sharded to allow transactions to be executed on a single node, and then no locks are required (therefore, no waits on locks). Transaction coordination is likewise avoided. Finally, stored procedures are compiled to produce code comparable to the access level calls of NoSQL systems. They can be executed in the same order on a node and on replica node(s).

Clustrix (Mian, 2014) offers a data storage product similar to VoltDB and MySQL Cluster. However, Clustrix nodes are sold as rack-mounted appliances. The company claims scalability to hundreds of nodes, with automatic sharding and replication (with a 4:1 read/write ratio, they report 350K TPS on 20 nodes and 160M rows; see Mian, 2014). Failover is automatic, and failed node recover is automatic. They also use solid state disks for additional performance (like NoSQL databases). As with the other relational products, Clustrix supports SQL with fully-ACID transactions. Data distribution and load balancing is transparent to the application programmer. Interestingly, the company also designed their storing system to be seamlessly compatible with MySQL, supporting existing MySQL applications and front-end connectors. This could give them a big advantage in gaining adoption of proprietary hardware.

ScaleBase (Mian, 2014) takes a novel approach, seeking to achieve the horizontal scaling with a layer entirely on top of MySQL, instead of modifying MySQL. The Database Load Balancer, the product of the ScaleBase company, is a proxy server that sits in front of the actual database and in this case, the tool breaks a monolithic relational database into chunks and spreads it out across multiple physical servers (the sharding part). The Database Load Balancer looks exactly like a MySQL or Oracle database would at the network level to any application. But it shards the database across multiple nodes, and does so automatically. The database proxy then accepts SQL commands and depending on what those commands are, it either runs the query against the appropriate subset of the database or across all the shards at once. You do not have to change one line of your application code, but you may have to work out a different license with your database vendor.

Implementing sharding as a layer on top of MySQL introduces a problem, as transactions do not span MySQL databases. ScaleBase provides an option for distributed transaction coordination, but the higher performance option provides ACID transactions only within a single shard/server.

NimbusDB (Mian, 2014) is another new relational system. It uses MVCC and distributed object based storage. SQL is the access language, with a row-oriented query optimizer and AVL tree indexes. MVCC provides transaction isolation without the need for locks, allowing large scale parallel processing. Data is horizontally segmented row-by-row into distributed objects, allowing multi-site, dynamic distribution.

Google has recently created a layer on top of BigTable called **Megastore**. Megastore adds functionality that brings BigTable closer to a (scalable) relational DBMS in many ways: transactions that span nodes, a database schema defined in a SQL-like language, and hierarchical paths that allow some limited join capability.

In theory, RDBMSs such as the ones presented in this section should be able to deliver scalability as long as applications avoid cross-node operations. If this proves true in practice, the simplicity of SQL and ACID transactions would give them an advantage over NoSQL for most applications.

6. USE CASES FOR MEDICAL STORAGE

Not one of these data stores is the best candidate for any Medical Data use case. A user's prioritization of features will be different depending on the medical application, as will the type of scalability required. With all the storing systems available, we are still far from understanding all the complexities behind processing large amounts of data. For example, data analytics scientists today generally work on one unit of analysis at a time. Scientists working on human genome use statistics tailored for their purpose – which is relatively easy to organize in the storing system. But, in the future, scientists working on human genome data may improve their analysis if they could take all publications on Medline and analyze it in conjunction with the human genome data. However, this requires combining several technologies and data formats, which might put an unusual pressure on the data scheme for accessing those data.

Recent projects such as Megastore have the potential to encourage scientists to put their data into the Cloud, where maybe other end-users might have access as well. Google had used previously the tool (under the name Dremel, lately BigQuery) internally for years before releasing a form of it in their generally available service. Megastore is hosted on Google's infrastructure, and its main advantage is simplicity. We will see in the years to come if instruments such as this represent the best combination to reconcile two worlds: NoSQL vs. scalable RDBMs.

6.1. Key-value Store Example

Key-value stores are generally good solutions for applications working with only one type of object (and all lookup operations are based on that one attribute). The simple functionality of key-value stores may make them the simplest to use. As an example, keys could represent social security numbers of health insurance customers, and values are their medical claim details, or their personal medical records. For medical records, a health information management system (e.g., Electronic Medical Records or Health Information Systems) today relies on many RDBMS queries to create a tailored personalized medical history of the patient (Zikos, 2014): each medical condition is generally stored separately from the patient personal data, and separated from other tables for signs and symptoms, risk factors, therapies, tests, studies and trials, supplements, and many others. Suppose it takes several seconds to execute those queries, and the user's data is rarely changed (after all, how many times you are diagnosed with a new disease?), or you know when it changes. Then you might want to store the user's medical records as a single object in a key-value store, represented in a manner that is efficient to send in response to requests, and index these objects by user ID (or, the social security number). If you store these objects persistently,

then you may be able to avoid many RDBMS queries, reconstructing the objects only when a user's data is updated.

Even in the case of an application like Facebook, where a user's home page changes based on updates made by the user as well as updates made by others, it may be possible to execute RDBMS queries just once when the user logs in, and for the rest of that session show only the changes made by that user (not by other users). Then, a simple key-value store could still be used as a relational database cache.

One could use key-value stores to do lookups based on multiple attributes, by creating additional key-value indexes that you maintain yourself. However, at that point it is better maybe to move directly to a document store. And, in the example above, outsourcing key-value pairs to a public cloud might create significant privacy and security risks (not to mention that even law forbids sometimes outsourcing in the cloud sensitive information such as social security numbers and human-rights sensitive data).

6.2 Document Store Example

Document Stores are more natural selections for dealing with medical health record storage. Electronic medical records include: patient records, research reports, laboratory reports, hospitals records, MRIs and Cat Scans, X-Rays & more. A healthcare facility or hospital information management application for document/record store would be one with multiple different kinds of objects, where you need to look up objects based on multiple fields (say, a patient's name, social security number, medical procedure, or birth date).

An important factor to consider is what level of concurrency guarantees needed. If one can tolerate an "eventually consistent" model with limited atomicity and isolation, a document store is the right choice. That might be the case for a medical application, where it would be quite unlikely for two hospital offices to be updating the same patient's record at the same time (after all, can the same patient be admitted in two different hospitals in the same time?). But when data need to be up-to-date and atomically consistent, e.g. if one wants to lock out logins to the patient's data after three incorrect attempts, then it is better, again, to consider alternatives, or to use a mechanism such as quorum-read to get the latest data.

6.3 Extensible Record Store Example

The use cases for extensible record stores are similar to those for document stores: multiple kinds of objects, with lookups based on any field. However, the extensible record store projects are generally aimed at higher throughput, and may provide stronger concurrency guarantees, at the cost of slightly more complexity than with document stores.

Suppose you are storing patient information, and you want to partition your data both horizontally and vertically. You might want to cluster patients by country (after all, we live in a globalized world), so that you can efficiently search all of the patients in one country. But you might also want to separate the rarely-changed "core" patient information such as patient address and email address in one place, and put certain frequently-updated patient information (such as prescribed medical recipe the patient is taking, or the history of medical treatments he received) in a different place, to improve performance. Although one could do this kind of horizontal/vertical partitioning on top of a document store by creating multiple collections for multiple dimensions, such partitioning is most easily achieved with an extensible record store such as HBase or HyperTable.

6.4 Scalable RDBMS Example

The advantages of relational DBMSs are well-known. When the medical application requires many tables with different types of health data, a relational schema definition centralizes and simplifies the data definition, and SQL greatly simplifies the expression of operations that span tables. Many programmers are already familiar with SQL, and many would argue that the use of SQL is simpler than the lower-level commands provided by NoSQL systems. Transactions greatly simplify coding concurrent access. ACID semantics free the developer from dealing with locks, out-of-date data, update collisions, and consistency. And many more tools are currently available for relational DBMSs, for report generation, forms, and so on.

As a good example for relational, imagine a more complex healthcare application, perhaps with a query interface for physicians to interactively search on patient's history of illnesses, by different fields such as known history for injuries, treatments, or medical reported problems at the patient's head, liver, heart, kidney, etc., and/or using constraints and/or aggregate data based on for the street residence for patients, and sex (sometimes, doctors need data about potential confinement areas and known medical problems to understand better the health problem; e.g., if the patient lives next to a kindergarten where supposedly multiple cases of measles were reported in the last month, than they might prioritize tests for this condition for a patient experiencing related symptoms). ACID transactions could also prove valuable for a database being updated from many locations (from the hospitals, as well from the pharmacy), and the aforementioned tools would be valuable as well. The definition of a common relational schema and administration tools can also be invaluable on a project with many programmers.

These advantages are dependent, of course, on a relational DBMS scaling to meet an application needs. Recently-reported benchmarks on VoltDB, Clustrix, and the latest version of MySQL Cluster suggest that scalability of relational DBMSs is greatly improving. Again, this assumes that an application does not demand updates or joins that span many nodes; the transaction coordination and data movement for that would be prohibitive. However, the NoSQL systems generally do not offer the possibility of transactions or query joins across nodes...

6.5. Hybrid approaches

Probably the most natural solution to store medical data is to use a combination of scalable SQL and NoSQL data. Medical centers can generate tens of gigabytes of image data a day, and analytic systems further need to access them efficiently (Sobhy et al, 2012; Byczkowska-Lipinska & Wosiak, 2013). Furthermore, for medical diagnosis, medical records must be related to other patient data. While the argument still continues whether to use a SQL or a NoSQL approach, we see an emergence of NewSQL database systems. NewSQL are relational databases with faster read-write performance similar to NoSQL databases and guarantee ACID property similar to SQL databases. With the power of NewSQL databases, medical data mining becomes an extensive explored topic today, and more faster and responsive medical applications powered by analytical engines is on the horizon.

As case study, the system for rheumatologic medical data analysis presented in (Byczkowska-Lipinska & Wosiak, 2013), for example, is designed to diagnose juvenile idiopathic arthritis with children. The authors combine for this Oracle NoSQL Database with Oracle Database 11g (a NoSQL with SQL database). In this case, the data obtained in the diagnosis of a patient include: personal information, laboratory tests, imaging studies as DICOM files, related descriptions and the results of additional studies gained by specialist consultation (usually rehabilitation and ophthalmology).

There are several requirements for data acquisition and managing in the designed system (which typically apply to many other medical imaging systems):

- for each medical institutions, all the treated patients' data should be stored in the system (data storage guarantees),
- the access to patient data is permitted to healthcare professionals (physicians, nurses, laboratory technicians), call-center staff and patients themselves (user-centric policies),
- the system should include the ability to store sensitive data, but also the specific data required at the institution, depending on its policies (non-uniform data access, different administrative security policies in place),
- the conclusions should be drawn using all the data, including medical imaging metadata (consistency),
- the access to patient data and the authorization process should be as soon as possible (efficient access, availability).

Such requirements show that a relational database is a better candidate, since the overall data consistency and availability. On the other hand NoSQL solutions offer greater opportunities for distributed storage and rapid access to selected data. Moreover they are a very good choice for multimedia storage (see above). For these reasons, authors in (Byczkowska-Lipinska & Wosiak, 2013) conclude that is better to build a hybrid database. As such, most of the medical data could be stored in the centralized relational database, while nodes of NoSQL storage could mainly be used for storing multimedia content. The approach enables loading data into the database, and still conduct quick search on the basic attributes of the data, especially using the patient's identifier. The advanced search is performed using a relational database that offers mature indexing techniques to speed up this search process. Data of big volumes (i.e., multimedia files is accessed fast using a query processor depending on attributes specified using what authors call QBF query (Query By Feature).

7. LESSSONS LEARNED, FUTURE RESEARCH AND CHALLENGES

Many developers are today willing to abandon globally-ACID transactions in order to gain scalability, availability, and other advantages. The popularity of NoSQL systems has already demonstrated this. Customers tolerate airline over-booking, and orders that are rejected when items in an online shopping cart are sold out before the order is finalized. The world is not globally consistent. The simplicity, flexibility, and scalability of NoSQL data stores is appealing for Web sites with millions of read/write users and relatively simple data schemas. Even with improved relational scalability, NoSQL systems maintain advantages for some applications. No wonder that in the last years more medical systems migrate to NoSQL.

New relational DBMSs will most likely also take a significant share of the scalable data storage market. If transactions and queries are generally limited to single nodes, these systems should be able to scale. Where the desire for SQL or ACID transactions is important, these systems will be the preferred choice. Many of the scalable data stores are new and have not yet achieved the robustness, functionality, and maturity of database products that have been around for a decade or more. Early adopters have already seen Web site outages with scalable data store failures, and many large sites continue to rely on their own solution, by sharding with existing RDBMS products. However, some of these new systems will mature quickly, given the great deal of energy directed at them.

There will be major consolidation among the systems we described. One or two systems will likely become the leaders in each of the categories. It seems unlikely that the market and open source community will be able to support the sheer number of products existing today.

Nevertheless, SQL (relational) versus NoSQL scalability is a controversial topic. The argument for relational over NoSQL goes something like this: If new relational systems can do everything that a NoSQL system can, with analogous performance and scalability, and with the convenience of transactions

and SQL, why would you choose a NoSQL system? Relational DBMSs have taken and retained majority market share over other competitors already for more than 30 years. Successful relational DBMSs were built to handle other specific application loads in the past: read-only or read-mostly data warehousing, OLTP on multi-core multi-disk CPUs, in-memory databases, distributed databases, and now horizontally scaled databases. While we don't see "one size fits all" in the SQL products themselves, we do see a common interface with SQL, transactions, and relational schema that give advantages in training, continuity, and data interchange.

The counter-argument for NoSQL is the lack of mature benchmarks showing that RDBMSs cannot achieve scaling comparable with NoSQL systems like Google's BigTable. (Cattell, 2011) mentions benchmarks demonstrating that MySQL Cluster fails to adequately support even more than a dozen of storage nodes. If you only require a lookup of objects based on a single key, then a key-value store is adequate and probably easier to understand than a relational DBMS. Likewise for a document store on a simple application: you only pay the learning curve for the level of complexity you require. Some applications require a flexible schema, allowing each object in a collection to have different attributes. While some RDBMSs allow efficient "packing" of tuples with missing attributes, and some allow adding new attributes at runtime, this is uncommon. A relational DBMS makes "expensive" (multimode multi-table) operations "too easy". NoSQL systems make them impossible or obviously expensive for programmers. Finally, while RDBMSs have maintained majority market share over the years, other products have established smaller but non-trivial markets in areas where there is a need for particular capabilities, e.g. indexed objects with products like BerkeleyDB, or graph-following operations with object-oriented DBMSs.

Both sides of this argument have merit.

What we managed to see if that NoSQL databases face several challenges that stop them from being perfect candidates for storing medical data in health systems.

Overhead and complexity: Because NoSQL databases do not work with SQL, they require manual query programming, which can be fast for simple tasks but time-consuming for others. In addition, complex query programming for the databases can be difficult.

Reliability: Relational databases natively support ACID, while NoSQL databases do not. NoSQL databases thus do not natively offer the degree of reliability that ACID provides. If users want NoSQL databases to apply ACID restraints to a data set, additional programming is needed.

Consistency: Because NoSQL databases do not natively support ACID transactions, they also can compromise consistency, unless manual support is provided. Not providing consistency enables better performance and scalability but is a problem for certain types of applications and transactions, such as those involved in banking.

Unfamiliarity with the technology: Most organizations are unfamiliar with NoSQL databases, and thus may not feel knowledgeable enough to choose one or even to determine that the approach might be better for their purposes.

Limited ecostructure: Unlike commercial relational databases, many open source NoSQL applications do not yet come with customer support or management tools.

NoSQL databases will probably be used largely for working with unstructured data in ways that require scalability. During the next years, users will most likely adopt NoSQL databases primarily for

specialized projects, such as those that are distributed, that involve large amounts of data, or that must scale. After that, broader adoption could occur.

Thus, most likely NoSQL databases will not replace relational databases, but instead will become a better option for certain types of projects. There will be a growing realization that the relational databases in use today are often good tools but that other tools have their place as well. And, it is more natural that in the end most Health Systems will rely on hybrid approaches, where large and simpler data (multimedia files in general) will be stored in NoSQL, but the business logic of the actual applications will make a lot of use of relational and scalable SQL alternatives.

8. CONCLUSIONS

The healthcare industry today is already generating large amounts of data. We witness a trend towards medical-related data digitization. Medical data records that include patient information are already being transferred between hospitals, such that concurrently a large number of physicians can take advantage of it. Of course, with this increase in medical data rates being produced, new requirements for the solutions to store them appeared. The medical data has to be available to end-users potentially being in different locations (e.g., the physician might need to access the medical record of the patient, and simultaneous the record could be used in some statistical analysis, run by the hospital administration, regarding the health in a given population; when the patient goes on holiday the physicians there might also need access to his medical records). When availability and scaling are two main requirement pillars, NoSQL becomes an appealing alternative to storing medical data. For over four decades, data management typically meant relational data processing, and relational database management systems (RDBMSs) became commonplace in all medical data processing environments. But today NoSQL data management systems unleash the full power of cluster environments, and offer simpler key-value data models.

But with all the excitement around the NoSQL hype, today examples of data management implementations based on pure NoSQL databases in healthcare are still missing on large extents. Most NoSQL products presented in this chapter are still in beta or research pilots, and largely open source, lacking in support. Some say that medical apps are inevitably going to be extremely conservative, because people could die if the IT system fouls up (see the famous CAP theorem). But, still, NoSQL can offer many advantages to the future, which is why we already see today more and more NoSQL Electronic Health Record systems appearing. And, on the horizon, new hybrid mechanisms designed to combine the best of these two worlds, relational and non-relational, start appearing.

The chapter presented an analysis of current NoSQL (and scalable SQL) technologies, and their applicability to support efficient storing of medical data and patient tracking. We analyzed current trends, give insights into the prons and cons of different technological choices, and presented the future challenges in need to be addressed. We presented an analysis of the different choices in storage technologies, for both medical data and the particular case of dementia care. We studies different storage technologies in existence, and introduce future challenges in efficient storage of medical data. Although we focused primarily on NoSQL solutions, we have also highlighted technologies that in the future will most likely combine scalable SQL into what is today known as NewSQL.

ACKNOWLEDGMENT

This work has been supported by Research Project COMMAS TIN2013-46181-C2-1-R Computational Models and Methods for Massive Structured Data. The research is also partially supported by national project MobiWay, Project PN-II-PT-PCCA-2013-4-0321.

REFERENCES

Aleksovska-Stojkovska, L., & S. Loskovska (2013). *Data Mining in Clinical Decision Support Systems*. Recent Progress in Data Engineering and Internet Technology. pp. 287-293, Springer Berlin Heidelberg.

AS (2014). Alzheimer's Society. *Statistics*. Retrieved from http://bit.ly/1l0dWRi [June 2014].

Byczkowska-Lipinska, L., & A. Wosiak (2013). *Multimedia NoSQL database solutions in the medical imaging data analysis*. Report Institute of Information Technology, Lodz University of Technology. Retrieved from http://bit.ly/1qlB3vx [June 2014].

Cattell, R. (2011). Scalable SQL and NoSQL data stores. *ACM SIGMOD Record* 39.4 (2011): 12-27.

Chang, F., J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, & R. E. Gruber. (2008a). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems* (TOCS) 26(2), 4.

Chang, Y.-J., S.-K. Tsai, & T.-Y. Wang (2008b). A context aware handheld wayfinding system for individuals with cognitive impairments. In *Proceedings of the 10th international ACM SIGACCESS conference on Computers and accessibility* (pp. 27-34). Halifax, Nova Scotia, Canada, ACM.

Cooper, B. F., A. Silberstein, E. Tam, R. Ramakrishnan, & R. Sears. (2010). Benchmarking cloud serving systems with YCSB. In Proceedings of the 1st ACM symposium on Cloud computing (pp. 143-154), Indianapolis, USA, ACM.

Davies, R., C. D. Nugent, & M. Donnelly (2010). *Prototyping cognitive prosthetics for people with dementia*. *Supporting People with Dementia Using Pervasive Health Technologies*. Springer London, pp. 145-163.

Davis, L. (2012). *Technology will replace 80 percent of docs*. The Health Cre Blog, Retrieved from http://bit.ly/1jCLLZ7 [June 2014].

Dembosky, A. (2012). *Data Prescription for Better Healthcare*. *Financial Times*, December 12, 2012, Retrieved from http://on.ft.com/1s0f416. [June 2014].

Eurostat (2012). *Active ageing and solidarity between generations. A statistical portrait of the European Union*. Retrieved from http://bit.ly/1oD3AN8 [June 2014].

Feldman, B., E. M. Martin, & T. Skotnes (2012). *Big Data in Healthcare Hype and Hope*. October 2012. Retrieved from http://bit.ly/1oN2Weg [June 2014].

Gilbert, S., & N. Lynch (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2), 51-59.

Hoey, J., P. Poupart, A. V. Bertoldi, T. Craig, C. Boutilier, & A. Mihailidis. (2010). Automated handwashing assistance for persons with dementia using video and a partially observable markov decision process. *Computer Vision and Image Understanding* 114(5), 503-519.

Jin, Y., T. Deyu, & Z. Xianrong (2011). Research on the distributed electronic medical records storage model. In *Proceedings of 2011 International Symposium on IT in Medicine and Education* (ITME, pp. 288-292), Vol. 2., Guangzhou, China, IEEE.

Klophaus, R. (2010). Riak Core: building distributed applications without shared state. In *Proceedings of ACM SIGPLAN Commercial Users of Functional Programming* (pp. 14). Baltimore, MD, USA, ACM.

Lee, K. K.-Y., W.-C. Tang, & K.-S. Choi (2013). Alternatives to relational database: Comparison of NoSQL and XML approaches for clinical data storage. *Computer methods and programs in biomedicine*, 110.1 (2013), 99-109.

Mandell, A., & R. C. Green (2011). *Alzheimer's Disease*, Wiley-Blackwell, Oxford, UK, pp. 1-91.

McKinsey (2013). McKinsey&Co. *Center for US Health System Reform Business Technology Office*. The 'big data' revolution in healthcare. Retrieved from http://goo.gl/r8NQGo [June 2014].

Mian, M. A. (2014). *Healthcare Big Data Exploration in Real-Time*. PhD Thesis. University of Washington, USA.

Mohamad, B., L. d'Orazio, & L. Gruenwald (2012). Towards a hybrid row-column database for a cloud-based medical data management system. In *Proceedings of the 1st International Workshop on Cloud Intelligence* (pp. 2). Istanbul, Turkey, ACM.

Neuhaus, C., R. Wierschke, M. von Löwis, & A. Polze. (2011). Secure cloud-based medical data exchange. *Lecture Notes in Informatics (LNI) Proceedings* (pp. 192).

Nishtala, R., H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, & V. Venkataramani. (2013). Scaling Memcache at Facebook. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation* (pp. 385-398). USENIX Association, Seattle, WA, USA.

Pasquier, T. F.J.-M., B. Shand, & J. M. Bacon (2013). *Information Flow Control for a Medical Records Web Portal*. University of Cambridge, Retrieved from http://bit.ly/1jCLO7r [June 2014].

Rascovsky, S. J., J. A. Delgado, A. Sanz, V. D. Calvo, & G. Castrillón. (2012). Informatics in Radiology: Use of CouchDB for Document-based Storage of DICOM Objects. *Radiographics* 32.3 (2012), 913-927.

Redis (2014). *Redis project*. Retrieved from http://redis.io/ [June 2014].

Robinson, L., K. Brittain, S. Lindsay, D. Jackson, & P. Olivier. (2009). Keeping In Touch Everyday (KITE) project: developing assistive technologies with people with dementia and their carers to promote independence. *International Psychogeriatrics*, 21(03), 494-502.

Schmitt, O., & T. A. Majchrzak (2012). Using Document-Based Databases for Medical Information Systems in Unreliable Environments. In *Proceedings of the 9th International ISCRAM Conference*, Retrieved from http://bit.ly/VEO9JE [June 2014].

Schmitt, O., & T. A. Majchrzak (2013). Document-Based Databases for Medical Information Systems and Crisis Management. *International Journal of Information Systems for Crisis Response and Management* (IJISCRAM) 5.3 (2013), 63-80.

Schütt, T., F. Schintke, & A. Reinefeld (2008). Scalaris: reliable transactional p2p key/value store. In *Proceedings of the 7th ACM SIGPLAN workshop on ERLANG* (pp. 41-48). Victoria, BC, Canada, ACM.

Sciore, E. (2007). SimpleDB: a simple java-based multiuser syst for teaching database internals. *ACM SIGCSE Bulletin.* Vol. 39. No. 1., ACM, pp. 561-565.

Sobhy, D., Y. El-Sonbaty, & M. Abou Elnasr (2012). MedCloud: Healthcare cloud computing system. In *Proceedings of 2012 International Conference for Internet Technology And Secured Transactions* (pp. 161-166), London, UK, IEEE.

Sumbaly, R., J. Kreps, L. Gao, A. Feinberg, C. Soman, & S. Shah. (2012). Serving large-scale batch computed data with project voldemort. In *Proceedings of the 10th USENIX conference on File and Storage Technologies* (pp. 18-18). USENIX Association, San Jose, CA, USA.

Terzo, O., P. Ruiu, E. Bucci, & F. Xhafa (2013). Data as a Service (DaaS) for Sharing and Processing of Large Data Collections in the Cloud. In Proceedings of 7th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS, pp. 475-480), Taichung, Taiwan.

Trifork (2014). *Shared Medicine Record*. Retrieved from http://bit.ly/1pr2vIL [June 2014].

Vanacek, J. (2014). *How will Big Data remake medicine?* Forbes, 2014. Retrieved from http://onforb.es/1i4XwGo [June 2014].

W3Resource (2014). NoSQL. Retrieved from http://bit.ly/1noR555 [June 2014].

Wimo, A., & M. Prince (2010). *World alzheimer report*. Retrieved from http://bit.ly/1li3WaK [June 2014].

Xhafa, F., J. Li, G. Zhao, J. Li, X. Chen, & D. S. Wong (2014). Designing cloud-based electronic health record system with attribute-based encryption. *Multimedia Tools and Applications*, Springer,1-8.

Xhafa, F., J. Wang, X. Chen, J.K. Liu, & J. Li (2013). An Efficient PHR Service System Supporting Fuzzy Keyword Search and Fine-grained Access Control, *Soft Computing*, Springer, 1-8.

Zikos, D. (2014) Database Systems I, class CSE 5330, University of Texas Arlington, Chapter 5: Databases in Healthcare, Retrieved from http://bit.ly/UTT6yh [June 2014].

Key Terms and Definitions

A **database** is an organized collection of data. The data are typically organized to model aspects of reality in a way that supports processes requiring information.

Database management systems (DBMSs) are computer software applications that interact with the user, other applications, and the database itself to capture and analyze data. A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases. Well-known DBMSs include MySQL, PostgreSQL, Microsoft SQL Server, Oracle, SAP and IBM DB2. Sometimes a DBMS is loosely referred to as a "database".

NoSQL is a class of database management systems (DBMS) that do not follow all of the rules of a relational DBMS and cannot use traditional SQL to query data. The translates to "Not Only SQL," as this type of database is not generally a replacement but, rather, a complementary addition to RDBMSs and

SQL. NoSQL-based systems are typically used in very large databases, which are particularly prone to performance problems caused by the limitations of SQL and the relational model of databases. Some notable implementations of NoSQL are Facebook's Cassandra database, Google's BigTable and Amazon's SimpleDB and Dynamo.

NewSQL is a type of database language that incorporates and builds on the concepts and principles of Structured Query Language (SQL) and NoSQL languages. By combining the reliability of SQL with the speed and performance of NoSQL, NewSQL provides improved functionality and services.

Healthcare industry is an aggregation of sectors within the economic system that provides goods and services to treat patients with curative, preventive, rehabilitative, and palliative care. The modern health care industry is divided into many sectors and depends on interdisciplinary teams of trained professionals and paraprofessionals to meet health needs of individuals and populations.

A **medical datum** is any single observation of a patient - for example, a temperature reading, a red-blood-cell count, a past history of rubella, or a blood-pressure reading. As this last example shows, it is a matter of perspective whether a single observation is in fact more than one datum. A blood pressure of 120/80 might well be recorded as a single data point in a setting where knowledge that a patient's blood pressure is normal is all that matters. If the difference between diastolic (while the heart cavities are beginning to fill) and systolic (while they are contracting) blood pressure is important for decision making or for analysis, however, the blood-pressure reading is best viewed as two pieces of information (systolic pressure = 120 mm Hg, diastolic pressure = 80 mm Hg). Human beings can glance at a written blood-pressure value and easily make the transition between its unitary view as a single data point and the decomposed information about systolic and diastolic pressure. Such dual views can be much more difficult for computers, however, unless they are specifically allowed for in the design of the method for data storage and analysis. The notion of a data model for computer-stored medical data accordingly becomes an important issue in the design of medical data system.

Dementia is not typically a specific disease, but rather is a term that describes a group of symptoms affecting thinking and social abilities severely enough to interfere with daily functioning. Many causes of dementia symptoms exist. Alzheimer's disease is the most common cause of a progressive dementia. Memory loss generally occurs in dementia. However, memory loss alone doesn't mean you have dementia. Dementia indicates problems with at least two brain functions, such as memory loss and impaired judgment or language, and the inability to perform some daily activities such as paying bills or becoming lost driving.

Key-value stores are among the simplest form of database management systems. They can only store pairs of keys and values, as well as retrieve values when a key is known. These simple systems are normally not adequate for complex applications. On the other hand, it is exactly this simplicity that makes such systems attractive in certain circumstances. For example resource-efficient key-value stores are often applied in embedded systems or as high performance in-process databases.

Document stores support more complex data than key-value stores. Such systems could store traditional documents (articles, word files, etc.), but in a more generic sense a document can be any kind of object. Unlike the key-value stores, these systems generally support secondary indexes and multiple types of documents (objects) per database, and nested documents or lists. Like other NoSQL systems, the document stores do not provide ACID transactional properties.