ALTERNATIVES TO RELATIONAL DATABASES IN PRECISION MEDICINE: COMPARISON OF NOSQL APPROACHES FOR BIG DATA STORAGE USING SUPERCOMPUTERS

by

Enrique Israel Velazquez

MS, University of Pittsburgh, 2011

MPH, University of Pittsburgh, 2011

MD, University of Nuevo Leon Medical School, Mexico, 2005

Submitted to the Graduate Faculty of

Department of Human Genetics

Graduate School of Public Health in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2015

UNIVERSITY OF PITTSBURGH

Graduate School of Public Health

This dissertation was presented

by

Enrique Israel Velazquez

It was defended on

June 29, 2015

and approved by

Dissertation Advisor:
Michael Barmada, Ph.D., Associate Professor, Department of Human Genetics, Graduate School of Public Health; Associate Professor, Department of Biomedical Informatics; Director, Center for Computational Genetics, Graduate School of Public Health; Associate Director, Center for Simulation and Modeling, University of Pittsburgh; Co-Director, Informatics Resource Center, Institute for Personalized Medicine, University of Pittsburgh Schools of the Health Sciences and University of Pittsburgh Medical Center (UPMC)

Committee Members:
Eleonor Feingold, Ph.D., Professor, Department of Human Genetics; Professor, Department of Biostatistics; Associate Dean for Education, Office of the Dean; Senior Associate Dean, Office of the Dean, Graduate School of Public Health, University of Pittsburgh

Harry Hochheiser, Ph.D., Assistant Professor, Department of Biomedical Informatics, School of Medicine, University of Pittsburgh

Alexandros Labrinidis, Ph.D., Associate Professor, Department of Computer Science; Co-Director, Advanced Data Management Technologies Laboratory, University of Pittsburgh; Adjunct Associate Professor, Computer Science Department, Carnegie Mellon University

Ryan Minster, Ph.D., Assistant Professor, Department of Human Genetics, Graduate School of Public Health, University of Pittsburgh

Copyright © by Enrique Israel Velazquez

2015

Michael Barmada, Ph.D.

ALTERNATIVES TO RELATIONAL DATABASES IN PRECISION MEDICINE: COMPARISON OF NOSQL APPROACHES FOR BIG DATA STORAGE USING SUPERCOMPUTERS

Enrique Israel Velazquez, PhD

University of Pittsburgh, 2015

ABSTRACT

Improvements in medical and genomic technologies have dramatically increased the production of electronic data over the last decade. As a result, data management is rapidly becoming a major determinant, and urgent challenge, for the development of Precision Medicine. Although successful data management is achievable using Relational Database Management Systems (RDBMS), exponential data growth is a significant contributor to failure scenarios. Growing amounts of data can also be observed in other sectors, such as economics and business, which, together with the previous facts, suggests that alternate database approaches (NoSQL) may soon be required for efficient storage and management of big databases. However, this hypothesis has been difficult to test in the Precision Medicine field since alternate database architectures are complex to assess and means to integrate heterogeneous electronic health records (EHR) with dynamic genomic data are not easily available.

In this dissertation, we present a novel set of experiments for identifying NoSQL database approaches that enable effective data storage and management in Precision Medicine using patients' clinical and genomic information from the cancer genome atlas (TCGA). The first experiment draws on performance and scalability from biologically meaningful queries with differing complexity and database sizes. The second experiment measures performance and

scalability in database updates without schema changes. The third experiment assesses performance and scalability in database updates with schema modifications due dynamic data. We have identified two NoSQL approach, based on Cassandra and Redis, which seems to be the ideal database management systems for our precision medicine queries in terms of performance and scalability. We present NoSQL approaches and show how they can be used to manage clinical and genomic big data. Our research is relevant to the public health since we are focusing on one of the main challenges to the development of Precision Medicine and, consequently, investigating a potential solution to the progressively increasing demands on health care.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

Finishing a Doctorate program can be a lonely process. I have been lucky enough to have the advice of many people. This section is my attempt to identify those who have made this dissertation possible.

I would like to thank, first and foremost, my wife Martha, who has encouraged me to pursue my interest and never fails to support me in any endeavor. Without her, along with my daughter Victoria and my family, I would not be the person I am today. Thanks to Vicka, Brenda and Abu for their support. I also offer thanks to my aunt Alma Elisa for her confidence in me.

Thanks to Michael Barmada, my advisor, for sticking with me to the end, even through times when health issues made work even more challenging. Despite his busy schedule, the advice Michael provided was always accurate, and his well-timed words kept me going to complete the whole Ph.D.

Thanks to my committee members -- Eleonor Feingold, Harry Hochheiser, Alexandros Labrinidis and Ryan Minster -- for their patient advising during the doctorate process. Their input on my dissertation has been invaluable.

Thanks to Nick Nystrom, Philip Blood and Bryon Gill from the Pittsburgh Supercomputing Center, Extreme Science and Engineering Discovery Environment (XSEDE), Data Exacell (DXC) pilot project and BRIDGES Pittsburgh Supercomputing Resource center at

the Carnegie Mellon University/University of Pittsburgh for their assistance and support. They provided me with the hardware and software resources to perform my big data experiments.

I would also like to thank Illyas Kamboh and all the members of his lab for giving me the opportunity to collaborate on their research projects that provided me training in the area of statistical and computational genetics.

Thanks to Ronald LaPorte, my research advisor during my first two years of the graduate school. He has been unfailingly supportive, and he championed my Ph.D candidacy even when I was no longer his student. Thanks also to Etienne Sibille for his encouragement, for always speaking well of me, and for his excellent support while my statistical genetic project.

I am grateful to Socrates Rizzo Garcia, Juan Enriquez Cabot, Julio Frenk Mora, Jesus Ancer Rodriguez and Luis Eugenio Todd Perez for their six years of excellent support and advice in the fields of Medicine, Genetics, Genomics, Public Health and Global Health. Without them this dissertation would not have been possible.

Special thanks to different laboratories that let me present my research and obtain invaluable feedback from them: Thanks to George M. Church at the Department of Human Genetics at Harvard Medical School and his laboratory members. Thanks to Daniel Salomon at the Department of Molecular and Experimental Research at the Scripps Research Institute (TSRI) campus La Jolla CA and all his laboratory members. Thanks to Andrew Su and his laboratory members, also at the TSRI. Thanks to Cecilia Lo at the Department of Developmental Biology, Children's Hospital UPMC, University of Pittsburgh and her laboratory members. Thanks to the Department of Genomic Medicine at the Georgia Regents University and all their laboratory members.

Thanks to colleagues of various institutions where I have discussed my research project: Human Longevity Institute in La Jolla CA, Illumina in La Jolla CA, The Sanger Institute at Cambridge United Kingdom, AbbVie Pharmaceutical Research in Germany, University of California in Riverside CA and Harvard School of Public Health in Boston MA.

I also offer thanks to all the friends and colleagues who simply made my life better over the course of this project: Professors, students and staff from the Department of Human Genetics, Department of Epidemiology, Multidisciplinary Master of Public Health Program and Center of Global Health at the Graduate School of Public Health, University of Pittsburgh.

1.0 INTRODUCTION

While medical databases have been created to collect related data since ancient times, anywhere from the late 1950s to the late 1970s the Digital revolution began; databases progressively evolved to digital formats, allowing for data collection at unprecedented speeds?. In fact, recently, with the governmental electronic health record initiative and the completion of the human genome sequence which made possible the beginning of Precision Medicine, medical and genomic databases have experienced a continuous, unsustainable growth. On average, 80 megabytes of data per person is added every year to individuals' electronic health records (EHR) in hospitals that have adopted this technology, and experts expect that this rate will increase as the amount of genetic data expands over time.[1] Moreover, GeneBank (NIH's genetic sequence database) - one of the most influential databases used to analyze genetic data - is doubling its information content every eighteen months.[2; 3] As a result, data management is rapidly becoming a major determinant, and urgent challenge, for the development of Precision Medicine.

Today, although successful data management is achievable using popular database systems (SQL), exponential data growth is a significant contributor to failure scenarios related to problems with complex queries on heterogeneous data, increasing database size, and frequent data updates that include database schema changes.

Growing data can also be observed in other sectors such as economics and business, suggesting that across the board alternate database approaches may be required to efficiently

store and manage big databases.[4; 5] However, this hypothesis has been difficult to test in the Precision Medicine field since alternate database architectures are complex to assess[4] and means to integrate heterogeneous electronic health records (EHR) with dynamic genomic data are not easily available.[6]

Today, few efforts have been focused on addressing the exponential growth in clinical and genomic data: only seven translational platforms from the literature that allow the management and exploration of clinical and omics data have been identiifed;[7] scientists have already explored concerns and tasks facing health information technology (HIT) developers regarding ethical, genetics and technological considerations, pointing out that HIT developers are key in the development of Precision Medicine.[8] However, different studies have just focused on clinical data, bringing interesting big data solutions. Studies have also measured the performance of data manipulation of big databases.[9] Researchers have summarized the state-of-the-art efforts in management of clinical big data, using the MapReduce programming framework and Hadoop platform to process huge volumes of clinical data.[10] However, few studies have developed methods to study alternatives (other than SQL-based options) in database management systems, such as NoSQL databases. One study did evaluate the suitability of NoSQL databases for structured clinical data by studying their performance, scalability, flexibility and extensibility, and it concluded that NoSQL systems have the potential to become a key database technology for clinical data management in the next years.[11]

In this dissertation, we identify the most suitable NoSQL databases for effective data management in Precision Medicine. In particular, we present novel experiments that focus on evaluating performance and scalability of integrated clinical and genomic databases from patients diagnosed with breast cancer.

Comparing SQL and NoSQL systems in terms of performance and scalability to effectively manage big data has always been a painstaking and subtle process, but several factors (i.e., complex queries, increasing database size and frequent data updates, including database schema changes) related to big clinical-genomic databases interact to make it even more difficult today. In the following sections, we describe these factors in detail.

1.1 THE INFORMATION AGE

The Digital revolution of the late 1950s, when analog changed over to digital technology, was succeeded by the current Information Age (also known as the Computer Age, Digital Age and New Media Age) . At this point in time, databases are rely mostly on computerization of information. Today, societies are globalized and dependent on digital databases and software to manipulate and analyze their data. However this dependence on digital formats is growing considerably, resulting in challenges to data storage capabilities. This challenge is made evident by the growth in data storage needs in the past three decades. In 1986, humankind was able to store, optimally compressed, 2.6 exabytes. This amount grew subsequently to 15.8 exabytes in 1993, 54.5 exabytes in 2000, and 295 exabytes in 2007, with an increase in globally stored information of 23% per year.[12] Thus, in these 21 years the information stored increased by 113-fold. In the US, it is estimated that by 2009, almost all US companies had stored an average of 200 terabytes (double the size of the US Wal-Mart's data warehouse in 1999).[13]

Specifically, in the health care field, electronic health records, email and social media have renovated the medical and scientific environment, facilitating information exchange.[14] This renovation is creating big data, challenging that presents challenges to the data storage,

processing, and analysis.[15] It is hardly estimated that the global size of Big Data in healthcare was approximately 150 exabytes in 2011, with an increasing yearly rate between 1.2 and 2.4 exabytes.[16] This rate is already challenging data manipulation but will urge effective data management in the coming years.

The use of big data generated in the current Information Age is extremely valuable for US health care. In 2011, a study estimated that efficiently using big data in the US healthcare industry would generate $300 billion in value every year.[13] Big data creates value in varous broad ways: unlocking significant health data by exposing transparent health information and using it on a higher frequency basis, such as registering which medications are more successful than others; collecting detailed health information on everything, from EHR to medical inventories, leading to better health care management decisions, such as those that increase the population health levels and decrease total spending levels; tailoring healthcare needs to patients, such as specific medical specialists and treatments in specific populations; and developing novel services and products such as preventive tests, diagnostic exams and treatments. However the recently generated genomic information that will soon be integrated with healthcare will exponentially increase the value of big data still more, but with a high cost: exponentially increasing the already overly-burdensome healthcare data.

1.2 EVOLUTION OF MEDICINE: PRECISION MEDICINE

It is through the creation of standards that medical practice is shaped. Since prehistoric times, humans have classified plants to treat diseases. In ancient times, Egyptians, Babylonians, Indians and Chinese standardized poisons, creams, herbs and teas to treat ailments. Ancient Greeks

developed a medical system based on the classification of human fluids to cure patients. With the birth of modern medicine, new knowledge obtained from relevant scientific events updated traditional standards in the medical practice; for example, the invention of the microscope advanced research into microorganisms, while the discovery of X-rays modernized diagnostic procedures. Important scientific events generate new knowledge to update standards that successively shape medical practice.

In the last 60 years we have observed notable scientific events in the genomics field[17]. These extend from the discovery of the structure of DNA[18], published in 1953, to the publication of the human genome project[19] (HGP) in 2003. This recently generated knowledge will eventually impact current medical practice standards, giving rise to Personalized Medicine or Precision Medicine.

Today, Personalized Medicine is related to the use of individual genetic information to prevent, diagnose and treat diseases; however, this term has evolved.[20] Some papers use Personalized Medicine to refer to the best treatment for an individual, such as those publications in 1971[21], 1990[22] and 1999[23] that, according to PUBMED, were the first times someone mentioned this term. Other papers have referenced personalized medicine, but linked it to the use of an individual's genetic information. These more recent papers, mostly published after the publication of the human genome project (HGP), introduced the current definition of Personalized Medicine; thus now it is common to find in the scientific literature the term Personalized Medicine or Precision Medicine referring to individualized prevention, diagnosis and treatment due to the use of the genetic information from each particular individual.

In the same way that the HGP impacted the definition of the term "Precision Medicine," introducing the human genome concept, this and other relevant scientific events have generated

new knowledge to not only update medical terms, but to reshape current medicine into Precision Medicine. The inclusion of genetic characteristics from each individual has been possible due to the advent of fast, accurate genotyping and whole genome sequencing technologies. Many of these resources and techniques, listed below, have contributed to our understanding of the function and organization of our genomes, lending better understanding of how changes in the sequence could impact clinical treatment.

1.2.1 Genetic Markers

A genetic marker is a specific DNA sequence or locus, often related to a recognizable trait. The identification of specific DNA sequences was possible as a result of the development of several biological molecular methods. Some of the first genetic markers described belong to a class known as Restriction Fragment Length Polymorphisms (RFLP), which are genetic variants that change the pattern of restriction fragments seen after digestion of DNA with a restriction enzyme and separation of the resulting fragments by gel electrophoresis. These markers were first described in 1974 and were used five years later in humans to study a particular sequence of DNA in the globin gene cluster. From 1974 to 1989, different genetic markers were used to detect DNA-level variation; these genetic markers were developed as a result of a new methodology that replaced the conventional hybridization-based assay methodology, namely the Polymerase Chain Reaction (PCR). During the period 1990-1993, the ongoing discovery of new genetic markers led to several advances in the molecular genetic field, such as recognition of micro-satellites or Single Sequence Repeats (SSRs) – tandem arrays of repeated sequences of 2-6 base-pairs of DNA that occur throughout genomes of all eukaryotic organisms. Microsatellite genetic markers quickly became the markers of choice among the genome mapping community

since they were common and distributed across the whole genome, and these repeated motifs offered a foothold for particular amplification using PCR. From 1994 until today many markers have improved the identification of genomic variation. New genetic markers have been identified because of the technology spillover from various genome projects. For example, today there is an array of molecular markers that have been identified due to the use of the high-throughput automated mode of DNA variants detection. Detection of DNA variants has helped further the development of Precision Medicine.[8; 9]

1.2.2 DNA microarray and gene expression studies

This tool let scientists detect DNA sequences, analyze gene expressions, determine genetic risks and identify genes that share the same expression pattern. A DNA microarray is a laboratory instrument that consists of microscopic DNA spots containing specific DNA sequences known as probes embedded in a solid surface. DNA microarrays were originally developed from an existing method -- Southern Blotting[24] -- and adapted to be used in molecular biology to detect DNA sequences. In 1982, DNA microarrays were first reported as a novel approach to analyze gene expression in a study of human colonic tumors and normal tissues[11]. The study showed one of the mains advantage of this method: the ability to measure expressions of large numbers of genes simultaneously. By 1987, DNA microarray had been used in multiple samples[25] –more than 4,000 human sequences– and four years later, it was found that DNA microarrays could assist in determining the genetic risk of colonic tissues.[26] Beyond its capacity to measure gene expression, DNA microarrays have also been found to be useful for identifying genes that shared the same expression pattern. In 1987, different genes –where their expression are modulated by interferon– were identified using a collection of DNA arrays for expression.[27] In 1995 was

reported the first use of miniaturized microarrays for gene expression[28] that two years later make possible the microarray expression analysis of a complete eukaryotic genome (Saccharomyces cerevisiae),[29] giving rise to the advent of gene expression studies.

Gene expression studies allow? the determination of the synthesis of functional gene products (proteins and RNA) and the regulation of gene expression. These studies focus on the control of gene expression that allows cells to produce essential gene products Some examples are control of the expression of insulin, which regulates blood glucose levels; the inactivation of the X chromosome in females, which prevents the expression of other genes it comprises, and control of the expression of cyclin, which influences the progression of the eukaryotic cell cycle. These studies have also generated information about gene expression related with interactions between specific molecules.[30; 31] The study of how molecular interaction affects the transcription of DNA, post-translational modification of a protein RNA splicing and translation of RNA[32; 33] opens the door to a better understanding of how genes function in the organism.

1.2.3 Genetic linkage studies

These particular studies updated the analysis of inheritance patterns in families by studying single gene disorders. These studies focus on the inheritance patterns of genetic markers, generally in large families that share a common condition. In other words, these studies are based in the increased tendency of neighborhood genes in a chromosome to be inherited together. Thus by studying genetic markers near targeted genes in a chromosome, the inheritance of those genes of interest can be determined. Genetic linkage studies were the primary method of investigation by the year 2000 and were used to detect highly penetrant genetic variants of large effects. A characteristic of these studies is that they perform a type of analysis that has been shown to be

very helpful in single gene disorders[34] such as Huntington's disease[35] (HTT gene) or cystic fibrosis[36] (CFTR gene). However, it is tough to reproduce genetic linkage studies in complex diseases;[37; 38] more recent analytic approaches have facilitated the study of this type of diseases.

1.2.4 Genome wide association studies

Genome-wide association studies (GWA or GWAS) or whole genome association studies (WGA or WGAS) have updated the information about the heritability of many diseases and traits. Through the identification of causal variants (SNPs) in genes, these studies bring information about the genetic risks for targeted diseases. GWAS focus on the differences in allele frequencies of a genetic variant between people with the phenotype of interest (i.e. specific disease) and those without that phenotype (apparently healthy individuals). GWAS consist of the comparison of genetic variants in people with a specific disease (cases) to apparently healthy individuals (controls); those genetic variants that are more highly represented in people with the targeted disease are reported to be "associated with" the disease. The first published GWAS study in 2005 found a significant allele frequency in two SNPs from patients with age-related macular degeneration compared with healthy individuals,[39] and subsequent studies got the same results but for other diseases, such as Crohn's Disease[40] (2005), myocardial infarction[41] (2005), inflammatory bowel disease[42] (2006) and type 2 diabetes[43] (2007). Moreover, when more patients and controls are added, GWAS can generate even more interesting results. For example, in 2007, a study including 14,000 cases of seven common diseases and 3,000 controls identified a previously implicated risk loci in those seven common diseases, including bipolar disorder, coronary artery disease, Crohn's disease, hypertension, rheumatoid arthritis, type 1 diabetes, and type 2 diabetes. The results of another GWAS that used over 34,000 case-control individuals

were able to explain the 80% genetic variation in type 1 diabetes, offering a list of 40 type 1 diabetes-risk associated loci.[44] Furthermore in 2010, by gathering 41,000 case-control individuals, a GWAS identified 10 significant SNPs related with autoimmune diseases such as rheumatoid arthritis, Crohn's disease, systemic lupus erythematosis, and type 1 diabetes.[45]

GWAS discoveries, when taken together, have contributed to the understanding of many disease mechanisms. For example when looking at the results of these studies in tandem, it is suggested that the CFH gene and other complement pathway genes contribute to macular degeneration,[39] the autophagy pathway involved in inflammatory bowel disease,[46-48] and cancer issues such as the 8q24 “gene desert” region associated with bladder, breast, colon, ovarian, and prostate cancers.[49] Beyond the identification of risk factors, GWAS also have offered information about the heritability of some quantitative traits and pleiotropy – genetic phenomenon where one gene influences multiple traits. In 2010, for example, a GWAS meta-analysis identified a group of ~200 loci that explain ~14% of height variation.[50] In 2007, another GWAS discovered a loci associated with multiple diseases, loci JAZF1 and TCF2 (or HNF1β), that are associated with Type 2 Diabetes and with prostate cancer;[51] TCF2 variants were protective against Type 2 Diabetes, but were a risk factor for prostate cancer. In summary, GWAS update genomic knowledge by providing novel information about the human genome; however, new technologies involving DNA sequencing have been able to perform more advanced analyses of the human genome.

1.2.5 DNA sequencing

DNA sequencing is a promising approach to studying the human genome and as such, is causing extreme optimism about the effective practice of Precision Medicine. DNA sequencing is a

technique that focuses on studying the exact order of the DNA nucleotides. Currently there are many DNA sequencing techniques, among them we find Sanger sequencing, next generation sequencing and exome sequencing.

1.2.6 Sanger sequencing

Sanger sequencing was developed in 1977 and has been widely used by the scientific community for over 38 years. It has stimulated the design of many important research projects such as the HGP, which has provided transformative information about the human genome. Sanger sequencing is a DNA sequencing technique based on the sequencing of one DNA nucleotide at a time,[52] making it a slow but precise process. Because of these qualities Sanger sequencing is considered the "gold standard" of DNA sequencing. It has been recognized as an excellent method in the context small-scale projects that require a short portion (low-throughput sequencing) of a DNA sequence. Newer methods have been designed for projects that require large amounts of DNA sequencing.

1.2.7 Next generation sequencing (NGS)

NGS accelerated the generation of human genome knowledge and the arrival of Precision Medicine by allowing researchers to achieve greater knowledge of the human genome than previously believed possible. NGS is a DNA sequencing method that produces massive amounts of accurate DNA sequence data by using a parallel sequence process – sequencing groups of DNA strands at the same time – thereby generating thousands or millions of sequences at once. Those millions of DNA sequences fragments are then rebuilt using computational methods to

generate a genomic sequence. This relatively recent method was introduced in 2005 and has been recognized as an excellent option for large-scale projects that require long portions (high-throughput sequencing) of a DNA sequence. NGS has made possible large scale projects, including metagenomics studies, which compare individual variability, disease stages and disease models.[53] NGS is highly sensitive to determining genetic information for individuals. This technique is able to show a considerable amount of genetic variation between people, including mutations and risk factors.

As part of an inherent effort to develop the practice of Precision Medicine, several medical fields have adopted the use of NGS to provide genomic data for research proposes. One of the most representative medical branches that has used NGS is the pharmaceutical field. Using NGS, different pharmaceutical labs have observed the possibility of generating genomic data for drug discovery, investigating the molecular basis of drug resistance, planning antimicrobial regimens for infectious diseases, developing vaccines, and even diagnosing diseases.[53]

1.2.8 Exome sequencing

This technique has contributed to the discovery of genetic causes of rare and complex diseases. Exome sequencing focuses on sequencing just the coding regions, or exomes, of the genome instead of every single base; in other words, it just sequences the DNA portions that are translated into a protein. Exome sequencing has become an important approach in medical genomic research since it is now known that the protein coding regions comprise around 85% of human disease-causing mutations. Also, this method has been shown to be a potential option in clinical diagnosis due to the increases in our understanding of sequence variation.[54] Exome sequencing is a relatively faster and cheaper methodology than NGS and Sanger sequencing

because exome regions just comprise 1% of the human genome. Since there is no long wait to get results, exome sequencing is the possible best option for clinical sequencing.

These DNA sequencing methods have led to different ways in which to to explore the human genome. These approaches have given rise to different genome projects such as The HGP, the first landmark project in the history of genome projects; the ENCODE project; the HapMap project; the 1000 genome project; GenBank, The International Nucleotide Sequence Database Collaboration (INSDC), and The Cancer Genome Atlas (TCGA).

1.3 THE POST-GENOMIC AGE

The Post-Genomic Age began after the sequencing of the human genome was completed in April 14th 2003. This extraordinary international scientific achievement, a result of the Human Genome Project, initiated a revolution in the genetic field and in almost all fields of knowledge. However, it also launched the production of electronic data on an unimaginable scale.[55]

1.3.1 The Human Genome Project

The Human Genome Project offered transformative information about the human genome. This large scientific effort revealed the inheritance code for all humankind, bringing hope that new ways to find cures for thousands of diseases that afflict humans around the world could be found.[56] The human genome project was an international scientific research project that sequenced the entire genome of one specific person. This thirteen-year project used the Sanger sequencing method. It was estimated that each nucleotide of the 3 billion that comprise DNA

cost ~USD $0.75 (the total cost of the HGP was ~USD $2,700 millions). Many research projects use this HGP as a human genome reference to compare to other human genomes. The HGP helped to estimate that the total number of genes in a human is 30,000 genes, which code for a variety of cellular functions. Those cellular functions are believed to play a central role in disease processes.[56]

Data volume and management challenges have increased since the Human Genome project because of the drop in cost in sequencing a human genome. While the HGP cost trillions of dollars, currently the human genome is accessible to anyone for a few thousand dollars, and it is expected that in 3 years, people will be able to get their genomes for a few hundred.[57] This accessibility of personal genomes will produce 100 gigabytes of data per person which, calculated in millions of personal genomes, would constitute hundreds of petabytes of data.[57] This means that unimaginable data volume and manipulation challenges will need to be met if millions more people want their DNA sequence to be treated with Precision Medicine.

Since the end of the Human Genome Project, many projects have emerged that are making attempts to understand the structure and functions of the human genome. Among them we find the Hapmap, 1000 genome, ENCODE and ClinVar projects. These projects keep generating data and so have saturated their databases. Other publicly available websites, such as GenBank, have experienced an exponential growth in their databases since the complete sequencing of the human genome.

1.3.2 HAPMAP Project

The International Haplotype Map (Hapmap) project revealed the variation of DNA among populations, pointing to the variants responsible for many genetic diseases. The International

HapMap Project was designed to create a catalogue of all the common variants in the human population. This project, begun a year before the HGP, was finished in 2002. The HapMap project highlighted the importance of finding DNA variants –also known as single nucleotide polymorphisms or SNPs– because of their role in making some populations more susceptible to specific diseases than others.[58]

This project has been used to determine allele risk factors, allele frequency and allele associations among neighborhood SNPs. Scientists use this information as their main resource when looking for susceptible DNA variants in different human genomes. For example if a clinician would like to know the risk alleles of a particular patient, a scientist would look for all the allele risks in the patient's genome, and those that match would explain the patients susceptibility for certain diseases. The HapMap project produced more than one million SNPs, and, in order to create an SNP map, it re-sequenced millions more SNPs. In the end, it finally compiled more than ten million SNPs, which were released in 2006 in a public database named "dbSNP." This project also allows scientists to study where mutations occur in the human genome, determine the allele risk factors in an individual, understand the impact of different SNPs in an organism, and even more surprisingly, offers the possibility of studying the human evolution.

The HapMap project has increased the amount of data being stored in the international Genetic research community. Currently, The HapMap project version Phase III Release 3 is 11.8 Gigabytes. However, since its launch, it has been downloaded over 500,000 times by researchers in more than 100 countries.[59]

1.3.3 1000 Genome Project

This project, launched in 2008, generated new knowledge in human genetics similar to the HapMap project, but at the level of the individual (not in a population base). The 1000 genome project provided an overview of human genetic variation that has contributed to the understanding of many diseases. This project consisted of sequencing at least 1000 individuals (1092 individuals) to scan 3 x 10^{12} DNA base pairs. With this project scientist and clinicians have one more base of comparison, for example, they can compare the genetic variation in patients against the genetic variation in these 1092 individuals. In other words where clinicians would like to find causal-disease variants in a specific patient, scientists could also match the genome of this specific patient with the variants found in these 1092 individuals and find the causal-disease variants that are the same between the patient and the 1092 individuals.[60; 61]

The 1000 genome project's data is continuously growing. In 2013 it was calculated to be 464 Terabytes[62] for the more than 1000 genomes. However, it is expected to sequence 2500 individuals eventually, including low coverage whole genome sequencing and exome sequencing,[63] at least doubling this size calculated two years ago.

1.3.4 ENCODE Project

The Encyclopedia of DNA Elements (ENCODE) elucidates the role of genes in different cellular mechanisms. This project was designed to elicit knowledge about the structural and functional elements of the human genome. In other words, this effort is studying the structure and grammar of the DNA as an instruction book. The US National Human Genome Research Institute (NHGRI) launched this project in 2003. With results from this project, scientists are now more

confident about how sections of the genome interact with the cellular environment and how genetic regions are involved in different cellular functions. In the clinical field, if a physician is interested in identifying harmful genes and target treatments for a specific patient, scientists can match in the patient's genome the harmful genes (obtained by the ENCODE project) and study the functionality of those genes (cell cycles predisposed) to recommend a therapeutic according to those cellular pathways affected in this particular patient. As expected, the ENCODE project has shown striking initial results, consequently alerting the scientific community and the world that of the practice of Precision Medicine is not that far away. Last year, when publishing its first results, the ENCODE project revealed that 60% of the noncoding DNA region remains unknown.[64] Moreover, it mentioned that a great quantity of this functional noncoding DNA plays a role in the regulation of gene expression.[65] In other words, much of the regulatory sites from different parts of the DNA, those which regulate the expression of each gene, remain a mystery. This valuable information highlights how far we are from completely understanding human genome information in order to be able to put it into practice; in other words, their results alert us that more research about DNA functionality is needed as we are only halfway to establishing precision medicine as a practical medical field.[66; 67]

Encode is another growing big data project. This global scientific effort, which was started after the end of the Human Genome project, scaled up in 2007, generating 15 terabytes of raw data. One of the project leaders commented about the growing database : "My head explodes at the amount of data."[68] Moreover, using new-generation sequencing machines could make this data explosion even bigger.

1.3.5 GeneBank Project

GeneBank is a public database of nucleotide sequences submitted from laboratories around the world. Even though GeneBank[2] started before the Human Genome Project (in ???), its data grew exponentially in size after the human genome project was begun. Currently, it includes sequences from over 300,000 species. This database is managed by the National Center for Biotechnology Information (NCBI), in collaboration with the EMBL European Nucleotide Archive (ENA), the DNA Data Bank of Japan (DDBJ) and the International Nucleotide Sequence Database Collaboration (INSDC) among other institutes, to provide genomic sequence availability worldwide.

GeneBank is growing tremendously annually. In the last year of 2014 GeneBank had accumulated 43.6% more sequences than the previous year of 2013. Specifically, the whole genome shotgun data increased 54.7% in one year. In August 2014 the whole genomes data was $7.74x10^{11}$ sequences and all of the GeneBank sequences equalled $9.39x10^{11}$. This amount requires around 653 GB of disk storage distributed in 2093 files.[69]

1.3.6 INSDC Project

The INSDC is a global effort that has offered nucleotide sequence information publicly for 30 years. However, because, like GeneBank, this database was built before the Human Genome project, it experienced large data growth after the publication of the whole human genome.[70] The INSDC experienced a ~ 2-fold growth in terms of the number of bases in 2012. In August 2012, it accumulated 1.56 $x10^6$ sequences and 1.44 $x10^{11}$ nucleotides, which represent almost

one and a half the number of bases of the previous year. This number would occupy slightly more than 100 GB.[71]

1.3.7 The Cancer Genome Atlas (TCGA)

TCGA is a large project managed by the National Cancer Institute (NCI) and the National Human Genome Research Institute (NHGRI). This effort has been growing since its inception in 2005 using genome sequencing from tissue and bioinformatics resources. It focuses on the identification of genetic mutations involved in the growth of cancers, with the goal of improving our ability to treat cancer through elucidating the genetic etiology of the disease. This funded project was founded on tissue from 27 different tumors.[72] Tumor types were obtained from resection prior to adjunct therapy. The samples were included based on their availability, meaning the more common the tumor, the more samples of that type are accessible to the study.

The amount of information generated from tumor samples has limited the standard models for big databases. The TCGA database has generated near 2.5 PB of data.[73] Storing this volume is estimated to cost approximately US$2 million. Managing the TCGA could be also problematic because of prolonged downloading time. It would last approximately 23 days at 10 GB per second to download the whole TCGA. It is expected that the limitation of successful data storage and management in this big database could worsen even more by adding more data through new cancer samples. Thus, in order to keep this project functioning and continuing the fight against cancer, data management and storage have become a priority.

Beyond the relevance and accumulation of data from the projects above, it seems that the data management challenges began when people performed personal genome projects choosing to take advantage of genome sequencing in their health care.

1.3.8 Personal Genome Projects

In general, the human genome project has inspired new approaches to identify the genetic variation in people, such as the study of individual genetic characteristics for particular non-anonymous individuals. It is expected that this approach will eventually be realized for any individual that wants to obtain his or her personal genome. Unfortunately, while this expectation will require the study of effective data management for millions of human personal genomes may need to be sequenced and analyzed, currently, the analysis of just a single individual is already a big data challenge. Some examples of personal genome projects are described below.

1.3.8.1 Venter genome project

This project raced against the international Human Genome Project to assemble the first human genome sequence in the early 2000s. The Venter genome project highlighted the genetic variation between the genome of a particular individual and the genome from the HGP. The Venter genome project revealed novel genetic variants compared to the HGP and genes associated with traits and diseases. The Venter Genome Project consists of the DNA sequencing of J. Craig Venter (JCV). This personal genome project was published in 2007 and released fascinating results. It reported 1.2 million novel genetic variants and disclosed seven genes associated with traits and diseases like blood group and Zellweger syndrome, respectively. Moreover, it revealed that JCV was heterozygous for variants in genes related to cardiovascular diseases, such as coronary artery disease, hypertension, and myocardial infarction; correlating these cardiovascular related results with his family history, researchers found positively that JCV's familiars had antecedents of cardiovascular diseases. He also showed heterozygous for the GSTM1 gene, which is involved in detoxification and metabolism of xenobiotics, and was

correlated with his antecedents of skin cancer and other probable affectations caused by different chemicals such as antibiotics and hormones found in foods and environmental pollutants. The results of this project have inspired posterior personal genome projects.[74].

Large storage disks are used for this kind of study, which includes multiple analyses since it produces big data. This study used a total of 100 terabytes of disk storage, and requires extra storage in the network from different systems across the environment.[75]

1.3.8.2 Watson genome project

This project provides additional information about genetic variation. It consists of the diploid DNA sequence of the genome of James D. Watson. Results from this project were published one year after those of the Venter genome project. Among its fascinating results, this project shows 32 genetic variants associated with diseases such as retinitis pigmentosa and congenital nephrotic syndrome. The Watson genome project also generated information beyond the identification of SNPs; specifically, it showed small indels (insertions and deletions) and Copy Number Variation (CNV, abnormal number of copies of one or more sections of the DNA). Furthermore, it found CNVs in the olfactory receptor gene clusters, though the meaning of these first-time results is still unclear. The Dr. Watson genome required around 10 GB of data storage and was saved on two DVDs, ~4.7 GB per DVD.[76]

The Venter and Watson genome projects motivated personal genome projects beyond America as well.

1.3.8.3 African genome project

This project demonstrates the contrasting genetic variability among different human genomes. Initially the African genome project started with the DNA sequencing of an individual from the

Yoruban ethnic group in West Africa; posteriorly it included more individuals. This project, launched in 2008, attached to the HapMap project effort to collect common patterns from the human genetic variation. The African genome revealed that the Yoruban individual possessed a greater proportion of heterozygous SNPs and some homozygous SNPs associated with pharmacogenic traits such as susceptibility to cancer. Like the Watson genome, the African project detected that SNPs, indels and CNV affected different genes. For example, it recognized 2,241 genes affected by indels. By adding more African genomes from different locations the African project revealed variants in the gene (SLC24A5) related to skin color and increased melanin production; homozygote alleles linked with physiological traits such as bone mineral density (VDR allele); increased muscle power performance and sprint (ACTN3); and fixed variants for phenylthiocarbamide (PTC) tasting (these ethnic groups usually taste plants which are sometimes toxic as an inherent activity).

It is estimated that in this project an individual genome is over a gigabyte.

1.3.8.4 Asian genome projects

This project also emphasizes the genomic variation that exists among individuals. Practically speaking, there were actually two Asian genome projects: the YH genome project[77] launched in 2007, which consisted of the DNA sequence of a Han Chinese individual (donor Dr. Yang Huanming's), and the SJK genome[78] project launched in 2008, which consisted of the DNA sequence of a Korean individual (donor Dr. Seong-Jin Kim). Both genomes showed distinct genomic characteristics. The YH genome revealed different medical related outcomes; among them was found deletions in 33 genes, a heterozygous mutation in the GJB2 gene (responsible for autosomal recessive deafness), and alleles associated with tobacco addiction and Alzheimer disease (AD). Interestingly, Dr. Huanming seems to have been a heavy smoker. The SJK genome

showed distinct information as well. This project found 1,348 novel SNPs, 773 SNPs related to different diseases such as specific types of cancer, diabetes and AD; and 504 more SNPs associated with mendelian traits such as dry earwax –apparently frequent in Koreans– and drug metabolism variants. The promising results from both projects seems to have motivated the study of multiple genomes -- the YH project has already sequenced the genome of one hundred Chinese individuals.

It is estimated that each genome requires 117 GB (Calculating that 1 Gigabase is approximately 1 Gigabyte) for the YH genome project[79] and 82.73 Gb for the SJK genome project.[80]

Beyond sequencing apparently healthy individuals, another scientific approach is to relay with the study of genomics in medicine,[81] meaning to study the genome in individuals with distinct or undiagnosed disease.

1.3.9 Clinical Genome Projects

Personal genome projects have focused on the use of a genetic profile in the clinical field. The use of genomic information in clinics has stimulated the development of Clinical Genome Projects. These projects offer relevant clinical information such as structural and disease causative genes. Like the personal genome projects, they have also developed a large amount of data in sequencing and analyzing just a few patients. Four representative clinical genome projects are described below.

1.3.9.1 Charcot-Marie-Tooth neuropathy clinical sequencing case

This clinical case project identified gene variants that segregated a particular disease[82]. This project, published in 2010, involved determining the DNA sequence of a patient with Charcot-Marie-Tooth neuropathy (CMT1). Among its results, disease-causative alleles and variants that segregate the disease in the SH3TC2 gene were identified. However it genetic variants related to drug induced cholestasis, warfarin sensitivity, cocaine syndrome, erythropoietic protorphyria, Refsum disease were also found, as well as variants putting an individual at risk for many types of cancer. Other interesting findings were the identification of the ABCD1 gene responsible for X-linked adrenoleukodystrophy (though the patient did not present this disease) and mutations related with neuropathies such as carpal tunnel syndrome. Though it was one of the first documented cases of clinical sequencing, it moved the practice of personal medicine a step forward.

This project obtained 89.6 GB of sequence data from the DNA's proband.[82] Specifically, the sequencing of DNA samples obtained from the proband produced a mappable yield of 89.6 Gb of sequence data, representing an average depth of coverage of approximately 30 times per base. The data from sequential machine runs consisted of 8.3 Gb of 35-bp fragment sequence reads (one run), 30.3 Gb of 25-bp mate-pair sequence reads (two runs), and 51.0 Gb of 50-bp mate-pair sequence reads (one run).

1.3.9.2 Crohn-like disease clinical sequencing case

This project[83; 84] turned an intractable clinical case into one with successful patient recovery. Launched in 2011, this project consisted of the use of DNA sequencing for a young individual with aggressive inflammatory bowel disease –Crohn like phenotype– that showed itself to be refractory to any treatment. Clinicians asked for a DNA sequencing of this patient because it was

an intractable case that was causing disability and suffering for the patient, and, as with other clinical sequencing cases, they found unexpected results. The DNA sequencing identified a XIAP gene and loss of a signaling of NOD2 protein where it was implicated in the Crohn disease pathway. In addition the study identified a mutation in the XIAP gene that was causing a deficiency of XIAP protein function in the patient. Even more interesting, results from the DNA sequencing led to diagnosis and treatment. That it, the resulting genome data shifted the presumed diagnosis to hemophagocytic lymphohistiocytosis (HLA), which has a higher a mortality risk. Given this information, clinicians were able to tailor the treatment to this specific case, and they decided to perform a transplantation. Five months post-, the patient was stable, without any need for more transfusions and with no clinical evidence of gastro-intestinal disease. This clinical case highlights the advantages of clinical sequencing in the individualization of diagnosis and treatment.

With respect to data, this project produced around 30 MB of information[83] after sequencing around 1% of the human genome (180,000 exomes).

1.3.9.3 Hypercholesterolemia clinical sequencing case

This study clarified the causes of a particular inconclusive case of hypercholesterolemia.[85] Launched in 2010, the study sequenced the genome of an 11- month-old breast-fed girl with xantomas and very high plasma cholesterol levels and the sequence of both parents. The sequencing of these three individuals showed interesting results. The study identified 3.29 million SNPs, including 502,000 indels. Other variants identified included a compound heterozygous non-sense mutation in the ABCG5 gene of the patient's genome. The mother was found to be a carrier for a novel mutation p.Q16X and the father heterozygous for a mutation in the p.R446X reported as a cause of sitosterolemia. These results led clinicians to identify the

cause of the girl's hypercholesterolemia and enabled them to tailor a diagnosis accordingly. Through a rigorous search, scientist concluded that breast-feeding and her minimal dietary consumption of plant sterols was altering her plasma cholesterol levels. This led clinicians to opt to control the plasma cholesterol levels by using a sterol absorption inhibitor together with a low-cholesterol and low-plant sterol diet. Again this example of real personalized practice led clinicians to the best treatment for a particular patient.

Here, using whole genome sequencing, they produced ~ 3 GB of data in the proband.[85]

1.3.9.4 Dopa-responsive dystonia clinical sequencing case

This project helped to elucidate a better treatment in a specific group of patients.[86] Published in 2011, this study used DNA sequencing in 2 sets of twins with dopa-responsive dystonia (DRD) disease. After examining the sequencing results, scientist found, as expected, that identical twins share many genetic variations. From 2.50 million SNPs found in one pair of twins and 2.42 million SNPs in the other pair, each twin shared more than half, or 1.63 million of them. Focusing on the cause of disease, scientist found in the 4 patients 9,531 common SNPs, including 4,605 non-synonymous SNPs. By mapping these results, they found the SRP gene (sepiapterin reductase), a gene associated with DRD, and two mutations (p.R150G and p.K251X) compound heterozygous for the SPR gene. These results suggested that the function of the SPR gene might be the cause of DRD because this gene is involved in the synthesis of tetrahydrobiopterin (BH4) – a direct cofactor for tyrosine hydroxylase and tryptophan hydroxylase and indirect cofactor for the synthesis of serotonin. These suggestions led to an adjustment in the treatment by including supplemental therapy with adjuvant 5-hydroxy-tryptophan (5HTP) and selective serotonin reuptake inhibitors (SSRIs) to the L-Dopa therapeutic already prescribed. This adjustment compensated for the synthesis of serotonin, which resulted in

a clinical improvement, This sequencing clearly allowedclinicians to offer a better and further optimized treatment for both pairs of twins.

In all, this study produced 356.8 GB of sequence data (178.4 giga–base pairs (Gbp) of sequence data).[86]

In summary, results from clinical sequencing point to Precision Medicine as the imminent future of current medical practice; these studies highlight the advantagesPrecision Medicine with respect to providing individualized diagnosis and therapeutics through the identification of medical actionable variants using mainly DNA sequencing. However, given the amounts of date generated just by the few sequencing studies already done, to translate these clinical benefits to multiple individuals represents future challenges in data management.

1.4 EVOLUTION OF DATABASE MANAGEMENT SYSTEMS

In the 1960s and 1970s, Edgar R. Codd worked on the relational model for database management, organizing data into tables or relations of rows and columns (Figure 1). This database model brought many benefits to the computer industry. Among the benefits, this model looks for persistence of the data and it manages concurrency for transactions, as any other DBMS. The relational model led to the development and release of a default standard query language (SQL) for databases. Organizations at that time integrated this model into their applications because of its data consistency. However, even though the relational model was a very influential achievement, it showed some problems. The most obvious troubles were called impedance mismatch problems, meaning it was difficult to translate document models to a relational model (Table model). This problem led to an extra step being necessary to save a

document in a relational database: needing to assemble structures and objects in memory, often in terms of cohesive data. Thus, scientists decided to create a model with individual rows and tuples, a single useful structure in an interface that could be processed in memory and distribute in a lot of tuples. To this purpose, in the mid-90's there was a rise in object-oriented databases. Object databases would take in memory structures and save documents directly to the database disk without mapping between the two different models (documents and relational models). It was thought that this technology would replace the relational database. However, most probably because relational databases were already integrated into many commercial applications, object databases did not fulfill their potential and the relational model became the standard database management system worldwide.

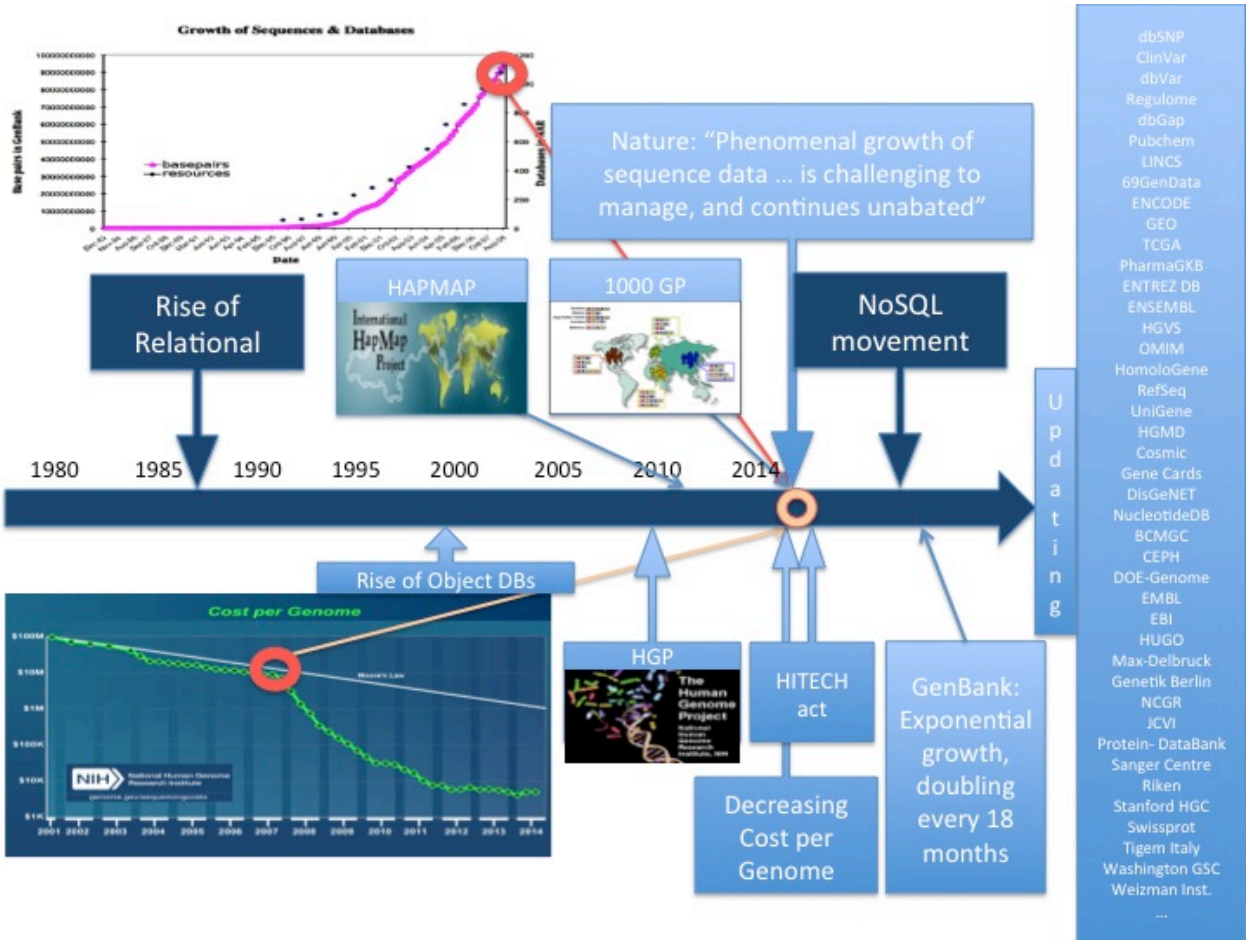

Figure 1. Timeline of database management systems and big data challenges in Precision Medicine.

After 40 years of the complete dominance of the relational database model, it was unlikely that another technology could come into the database management field. However, the rise of the Internet and websites that have enormous traffic, such as Google and Amazon, changed the rules in the production and storage of dynamic data. The saturation of traffic into web data led to scaling issues in databases that support websites. One solution was to scale up, meaning buy a bigger computer to store all the data; however, this solution presented many problems, most predominantly cost. It was very expensive to acquire a large enough computer to support huge SQL databases. SQL databases were designed to run in a single large computer and do not perform as expected with multiple computers.

Therefore, the Internet changed the standards of database management. The burgeoning traffic on websites and the SQL database problem with working on multiple computers drove the development of new models into the database field. Data managers realized that new database management systems, based on different data models, could advantage the use of multiple computers to manage data. Thus, instead of scaling up, they found another solution to database management: scaling out, that is, building database systems designed to be used across multiple computers. This also represents a less expensive option since multiple small computers are cheaper than one big machine.

A couple of organizations understood the need to effectively manage big databases and evolved the database management field. Google and Amazon developed from scratch their own database management systems that differ quite a bit from traditional SQL databases. Both novel database systems were discussed in two papers. Big Table of Google and Dynamo of Amazon were published in 2006[87] and 2007[88], respectively. These technologies became successful in commercial terms and inspired a new movement of databases called “NoSQL” (Not Only SQL).

While this term is hard to define in positive terms, it refers to all database systems that do not follow the relational model. By 2012, many NoSQL systems had appeared, inspired by one of four common data models: Document-based, Key-Value, Graph and Column, this last not necessarily different from the relational model; this will be explained in section 1.5.2.3. Among the most popular technologies considered NoSQL databases we find two inspired in the documents model: Mongodb and Couchdb; two designed from the Key-Value model: Redis and Riak; two designed from the Column model: Cassandra and Hbase; and two built from the Graph model: Neo4j and Allegro Graph.

Theoretically, NoSQL databases have the potential to manipulate heterogeneous data such as those found in Precision Medicine. With these databases, the ability to create ideal active laboratories from hospital database systems for Precision Medicine purposes, as shown in Figure 2, could be around the corner.[89]

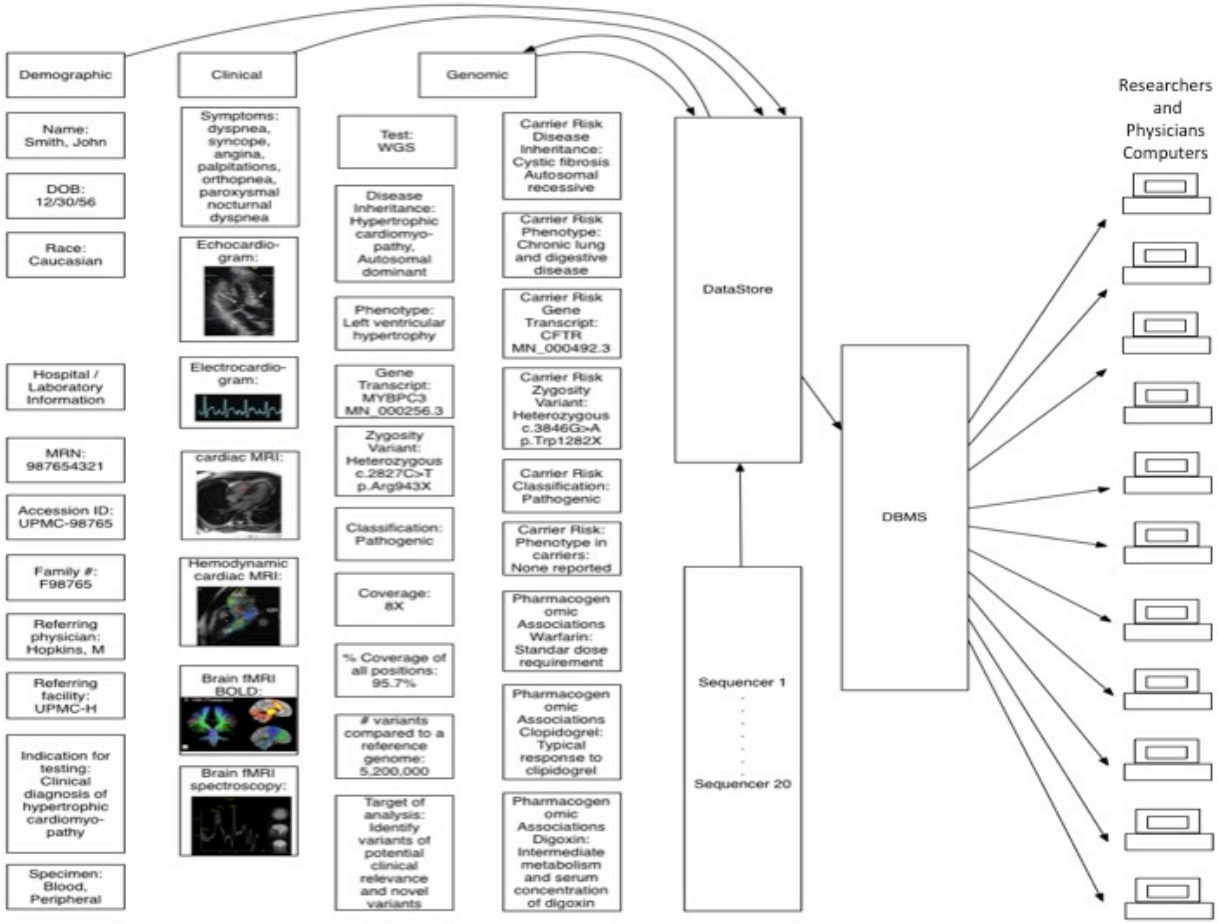

Figure 2. Example of a suitable database management system for Active Laboratories that include patients' demographic, clinical and genomic characteristics.

1.5 COMPARISON BETWEEN SQL AND NOSQL

In the past decade, not only has data volume increased tremendously, but it has also become dynamic, meaning it changes more rapidly and is more structurally varied. NoSQL databases were developed to address these data features (Table 48).[90]

It is well known that SQL databases are not the best way to manage big databases. They were designed based on a standard query language using the relational data model, which

identifies all possible answers to any request before displaying a correct solution. This aspect of the data model decreases performance when managing large datasets. Moreover, in SQL databases, the query execution times increase according to size and number of tables and columns. Also, query responses slow down as the number of joins (combining records from multiple tables) grows, a phenomenon commonly referred to as “join pain.” NoSQL databases are designed to avoid joins and consequently “join pains” as a solution to this issue.

Beyond big data production, the data structures themselves are becoming more dynamic. There is a high rate of data changing (structure) overtime. This metric rate is called velocity. A high rate of data change, together with large volumes of data, slows queries in a relational data model. Then, we have to consider data traffic, which means many requests for the same data. Large amounts of data traffic produce internal and external changes in a system so that the database also must handle time peaks. For example, in genomics it is common found that specific information become popular for data analysis. In other words, the database where this information is experienced an increased number of requests from multiple users. All this features are also handle by NoSQL databases.

NoSQL databases are also designed to manage database schema changes, even if the velocity is high such as at a peak time. They tolerate structural changes in any data element, even if these changes affect the overall structure of the data. These characteristics are commonly required for rapid-moving data such as Business Dynamics and Precision Medicine. For example, in Precision Medicine, clinical and, especially large genomic, data require frequent updates because every day there are new scientific findings, which are frequently non-predefined in a database that consequently requires schema changes. These types of features are difficult for SQL databases to address since managing high write loads and schema flexibility are very

expensive in terms of processing and operational spends, and making changes to the database after implementing these features can further increase costs.

In summary, scalability and dynamic structural change (updating process) features are key aspects in NoSQL databases that may advantage in the manipulation of big and dynamic structural data over SQL systems.

1.5.1 ACID versus BASE transactions

SQL and NoSQL databases have different data and query models (Consistency models). NoSQL databases use consistency features that support certain advantages over SQL in big data such as being able to handle huge data volume and changing schema structure. The consistency features provide a reliable environment during data transactions (single logical operations on the data).[91]

ACID (Atomicity, Consistency, Isolation, Durability) transactions are inherent features used by SQL databases that guarantee safe database transactions. These properties make transactions complete, consistent and permanent on disk.

Atomicity, as its name implies, indicates “indivisible”, meaning that transactions succeed completely or every operation is turned back. In other words, if a section of the transaction fails, all the transaction is rolled back. Thus, the transaction is protected from internal or external issues such as power failures, errors and crashes.

Consistency ensures that on transaction completion, the database does not get left in an unstable state. This means that any requested data will bring the database from one valid state to another structurally sound state.

Isolation guarantees that each transaction runs one at the time. This means that the transactions do not contend with one another; in other words, they run sequentially. Isolation

provides concurrency control that is the final goal of this property, that is, it does not allow the effect of incomplete transactions to interrupt other transactions.

The durability property of SQLs warrants that the result from a transaction will be permanent, even in the event of database failures due to events such as power loss, crashes, or errors. Thus, this property defends transactions from the effects of failures by storing the results in a non-temporal memory.

NoSQL databases, on the other hand, have BASE (Basic Availability, Soft-state, Eventual consistency) transactions (Table 47). In terms of big databases, these properties seems to be more 'modern' than those from SQL databases, or, put another way, in these times of high volume data, BASE transactions seem to offer a better strategy for storage than the ACID transactions.

Basic availability ensures that the database works most of the time. In other words, the system does not guarantee availability of data but does guarantee a response to any query. However, sometimes the response will be a failure, or inconsistent data.

Soft-state means that stores do not have to be write-consistent or mutually consistent all the time. In other words, the state of the system could change over time. Moreover, in the event of updates, there may be changes going on behind the scene to the point of eventual consistency.

Eventual consistency guarantees high availability at some point in time. In other words, if the database item is not updated, that data will show eventual consistency to the last updated value. This means that the database converges or achieves replica convergence at some later point.[92]

BASE transactions exhibit limitations when compared to the ACID ones, mainly because BASE does not guarantee data consistency. However, BASE does provide data availability that

is needed in big data management. Thus, database managers and developers should be aware of BASE constraints and become familiar with NoSQL behaviors to increase their availability property to manage big databases.

1.5.2 NoSQL Approaches

There are numerous NoSQL technologies that have become popular. According to the data model in use , these technologies can be classified into four categories:, Document model, Key-Value model, Column model, and Graph model. Each category presents distinct data model characteristics, operational aspects, drivers for adoption and functional capabilities. Nonfunctional requirements may also influence managers' choice of NoSQL database category.

1.5.2.1 Document model

Document databases store data in documents, acting like an electronic bookcase. Developers can easily familiarize themselves with this category since these databases hierarchically order documents, much like web sites. Document models use portable and platform independent data formats such as JSON, BSON and XML. These formats are both human readable and suitable for machine processing. Moreover, unlike in SQL databases, where data must be mapped into tables and columns, data that are commonly stored in documents can be preserved in a state representative of their true form (document form). This feature enables more simplified big data management for data that are commonly stored in documents. Another advantage of these formats is their interoperability with other systems. Data formats from Document stores are very similar to XML schemas. In the health sector, systems such as the Health Level 7 Clinical

Document Architecture (HL7-CDA) use XML document formats to enable clinical data integration with minimal development effort. Thus, data portability is a clear advantage of Document databases.[11]

In functional terms, Document databases rely on IDs to store and retrieve information. As a result, Document databases generally use multiple indexes to facilitate data access. For example, in an Electronic Health Record (EHR), we might use indexes to represent distinct therapeutic procedures that can be offered according to the patient's genome variation. Document databases use indexes to request documents to be used in applications, such as those using decision trees for medical applications.

Performance in Document databases is enhanced by using indexes for big datasets. As in SQL databases, indexes enable better performance. Using indexes could become costly in terms of query times because of the need to maintain them, where queries should check for indexes before checking the rest of the data. However, in big databases, having to scan a few records to find pertinent data can be much less time-consuming than having to make a full search of the database. In big databases, it is important to bear in mind the importance of using indexes as wisely as possible, however, because using large amounts of irrelevant indexes in queries can make them much slower.

Document databases also offer useful operational characteristics. Document databases scale out (horizontally) by using mutually independent records at write time, with no contended state. In other words, records should not be mapped in a specific framework. In addition, data do not need to make transactions across replicas. Thus, these useful operational characteristics make document databases a strong candidate for big database management.

1.5.2.2 Key-Value model

Key-Value models are modeled after Amazon's Dynamo database. This data model works as a distributed hashmap data structure, storing and requesting values by their assigned key (domain-specific identifier). The key is related to numerous units of data on the database. Each unit of data can be transferred into several secondary storages (machines) at a single operation for continuous operation reasons.

A particularity of this database is the fault-tolerance design introduced by Amazon. Fault-tolerant systems operate properly even in the presence of one or more failures of its components. Compared to SQL databases, where a small failure can potentially bring down the whole system, NoSQL databases using this model can tolerate even severe failures. However, severe failures could decrease its quality of operation.

The fault-tolerant design is a valuable feature for big data management since it guarantees data availability in the event of frequently occurring system failures such as hardware and software problems. For example, it would be advantageous to have a clinical/genomic database with continuous operability, where the whole system is not stopped due to failures.

Key-Value databases are not complicated to use. Clients store units of data and retrieve them with their key. The hash function is designed to distribute the information across the available machines in a uniform way; no single machine stores most of the information. In addition, clients have the option to assign information to a specific machine using their keys.

Any applications wishing to store data need only know the corresponding key. Key sets are usually the most important data elements in a database. For example in a healthcare database, the key could be a Social Security number, phone number, electronic or physical address; or, in the case of a clinical/genetic database, a patient ID. In theory, a sensibly designated Key-Value

database increases the chance to not lose data when there is a logical mapping to the assigned key and low response time to retrieve information.

At this point, Key-Value models seem similar to the Document model. Both have domain specific identifiers and similarly retrieve the information requested. However, a big difference is related to the level of insight. Key-Value databases appear to be oblivious to the data contained. In other words, they are not aware about the interrelation of the information contained. This feature makes Key-value focus only on efficient storage and retrieval of data. It seems that theoretically they do not display structured stored data like JSON, BSON or XML. However, as technology keeps moving, new features in some Key-Value databases have been introduced allowing visibility of structured data, creating some overlap between the document and key-value stores. But the reality is that, in general, Key-Value databases offer data insight that is more limited than that of Document databases.

Related to Amazon's Dynamo database, Key-Value databases are designed mainly to provide high-availability, high-performance, high scalability and non-redundancy (no need of backup). For example this design could be important in genomics when the data and data model keep changing with added functionality and genomic updates; especially when it is needed evolving schema to support these changes without having database downtime. Moreover, using these databases could afford denormalizing data for performance reasons and get faster responses when writing operations. Overall, this design provides to Key-Value databases essential characteristics to the successful management of big data.

1.5.2.3 Column model

Column models come from Google's Big Table model. This data model works with columns that are inside rows from a sparsely populated table. This innovative design provides natural indexing due to the idiosyncratic classification of columns.

Columns are classified into four types of column storage according to their complexity. A name-value pair constitutes the simplest Column. The combination of different name-value pairs builds a Super Column, which gives a name to a set of columns. The combination of columns stored in a row is called a Column family. Finally, when the columns stored in rows are Super Columns they build a Super Column Family. This last type consists of a Row Key where inside the row we would find Super Columns and Columns.

Individual level rows are essential in the structure of Column databases. The function of rows in Column models is to provide a map structure to order the data. For example in a supercolumn family, regarding clinical/genetic data, the name of the patient in the Row Key would be the patient ID, a simple Column would be the DOB (with one value), and a Super Column would consist of types of personal details as the Super Column name and the values themselves as Columns. For example, nationality could be a Column name with one value and email a Column name with one value; another Super Column might have genomic information as the Super column name and the values could be chromosome, location, and function with one merged value.

Column databases provide some insight into the data. Specifically, Column stores provide more awareness about what is stored in their data than Key-Value databases, but less insight than Document databases. Rows in Column databases represent a particular broad entity; they are sort of described by the content of the columns they contain, and successively, Column

names are related entities. Thus, at some point the data is naturally ordered in a more hierarchical way than Key-Value databases; however, this order does not describe as complete their entities (Row Names) as the structural order of Document Databases. Interestingly, by turning a Column database 90 degrees, we change from being row-oriented to being column-oriented, with a whole view of entities (Row Names and Column Names).

In operational terms, Column databases are also designed for high-availability (failover), high-performance, high-scalability, and non-redundancy. In addition, because Column databases use several storage engines that enable high write loads, they can support peaks of high data traffic. . Thus, at a point in time where a big clinical/genomic database is being saturated with requests about a particular set of data, it can be expected to operate competently.

Overall, Document, Key-Value, and Column databases have the highlighted similarities and differences. All three have essential features to the successful management of big datasets.

1.5.2.4 Graph model

Graph data models are used in online database management systems. This type of model is designed to deal with highly connected data, similar to RDBMS. It is used to understand how data are connected. It uses transactional (OLTP) systems and Create, Read, Update, and Delete (CRUD) methods.

Structurally, Graph databases comprise nodes and relationships. The nodes contain properties (Key-value pairs). Relationships have a name and are connected to other relationships. Relationships are always assigned to a start and end node. In addition, Relationships can also be allocated properties.

Some Graph databases can use native graph storage to advantage for data storing and managing. Native graph storage allows the serialization of graph data into SQL or object-

oriented databases. The use of native graph storage means databases do not have to depend heavily on indexes. The Graph model by itself provides an adjacency index because one node is related to a number of nodes of interest. For example, in a clinical/genomic database, a node called Gene Function is related to a number of nodes of interest called genetic variants where, from the biological perspective, one is a consequence of the other. This feature of Graph databases allows the understanding of information that is highly related.

Graph databases largely involve related nodes of information, using node location as a functional parameter. This model can interrelate millions of nodes per second. Therefore, the performance of interrelated data is theoretically faster than if using indexes on aggregate databases (Document, Key-Value and Column databases) of highly related data.[90]

1.5.3 MapReduce programming model

Document, Key-Value, and Column databases typically require the use of an external processing infrastructure for querying big data. MapReduce, an infrastructure developed by Google and publicly available from Apache Hadoop, is a commonly used technique for retrieving information across records.

Map Reduce is a parallel functional programming model that distributes data and works on it in parallel, then clusters and aggregates results until it can identify the requested information. For example, if we want to identify how many variants (previously scientifically-identified) are related to breast cancer in Hispanic patients in a clinical/genomic database, we would extract all female patients' records and discard the non-Hispanic patients in the map phase, and then process the remaining records in the reduce function.

MapReduce operates efficiently, working with numerous machines and a fast network infrastructure. Commonly, data managers use database features to point to the specific data they are interested in and then use MapReduce on that already focused data to obtain the precise information they are looking for.

MapReduce works with aggregate databases (Document, Key-Value, and Column databases). These databases are not designed to work with highly connected data. Aggregate databases are not ideal to deal with problems that require a deep understanding about how data is connected. Graph databases deal with highly connected data. Thus, it is not expected that MapReduce could be used in Graphs models.

1.6 BIG DATA CHALLENGES FOR CLINICAL AND GENOMIC INFORMATION

As medical? technology advances, especially with the onset of Next Generation Sequencing (NGS), modern clinical and genomic information is producing exponentially bigger and more complex data that require breakthrough algorithmic and computational solutions. In 2012 a US Federal investment addressing the issue of big data generated from digital information called the "Big Data Research and Development Initiative" was announced. It is focused on improving the ability to manage large amounts of data and excerpt new knowledge. However, this initiative is just the beginning in addressing the management challenges presented by the huge volumes of digital data produced daily. Thus, future work will need to aim at making dramatic advances in data management systems, together with the development of tools and techniques needed to store, analyze, organize, access and assemble massive collections of digital information. Below are described some key challenges in big data management that need to be addressed.

1.7 DATA SIZE

Big data refers to the huge size datasets, in terms of terabytes. The size of the input set of clinical/genomic data is possibly one of the largest data sets ever acquired. As previously mentioned, mainly personal information, clinical notes, multiple laboratory results, imaging studies and, tremendously large genomic sequencing data are all needed to accurately shape the high personalized health information required for Precision Medicine. Thus, the data on a single patient easily requires tens of gigabytes to store clinical, surgical and laboratory results, tens of gigabytes to store imaging data and tens of terabytes to store multiple genomic sequences and data resulting from their analyses. For this last, for example,in a patient with cancer, it is common that each biopsy be sequenced for Whole Genome Resequencing, Targeted Resequencing,[93] DeNovo Sequencing,[94] Exomics, Gene Expression,[95] and RNA Structure Ozsolak.[96] In addition, relatively new genomic techniques that produce large amount of output such as Chromatin Immuno-precipitation Sequencing (ChIP-Seq) (66-9), Methylation Analysis,[97] Whole Transcriptome Analysis,[98] Small RNA Analysis,[99] and Metagenomics[100] are becoming increasingly popular for studying certain diseases. More recent genomic studies have also generated tons of data. Proteomics, Metabolomics and Metabonomics have created expectative in the clinical field and keep generating tons of data. Clearly, the growing amounts of data required to highly personalize health information in thousands and perhaps millions of people is definitively a big data challenge for the development of Precision Medicine.

1.8 DATA RATE

Most of the data generated in Precision Medicine comes from genomic data. Routinely, millions of sequences are generated using high-throughput genome sequencing or NGS.[101; 102] Current NGS sequencers from Solexa-Illumina[103], 454 Sequencing,[104] and Applied BioSystems-SoliD[105] produce tens of terabytes of data daily. Just this last sequencer produces in one run, approximately 7 days, more than 4 TB of data. The data generated for a single sequencer increase even more during its analysis. A single analysis, lasting one to four weeks depending on hardware infrastructure and type of analysis, requires storage of the last reference genome and of high throughput data results. This is the genomic data rate generated for a single patient; it is hard to estimate the genomic data generated by a healthcare system for millions of persons were Precision Medicine to be offered across the board. It is clear that, like data size, rhe constantly growing data rate could present a challenge to simply accessing the data.

1.9 COMPUTATIONAL COMPLEXITY

Besides storing and accessing big data, another challenge for big data is the discovery of new knowledge. Running algorithms usually generates smaller outputs than the original input of big data. However, intense computing and effective data management systems are necessary to correlate the information in various big databases. For example, a computational problem would be to correlate people of the same age, sex, and/or diagnostic with potential cancer risks in their genomes (risk alleles). Furthermore, algorithms that select a subset of input data and then

compute a query based on big samples also require effective data management systems. Thus, future work in database management is also crucial in discovering new biological knowledge.

1.10 DATA SHARING

In the scientific field, data sharing is a common practice. However, the transfer of big data is a challenge requiring improvement not only in database architecture but also in technological infrastructure. Today, using a commercially optical fiber connection data can be transferred at a rate of approximately 300 megabits per second. At this rate, if someone would like to share 2 petabytes of data, it would take almost 2 years to transfer all the information. Thus, this limitation allows just one option in sharing big data: transferring information by physically carrying out mass-storage hard disks. This data sharing challenge could represent another obstacle to Precision Medicine as well because of the need to physically transport highly personal information.

In summary, big data challenges are urgent issues for the nascent field of Precision Medicine. New technologies, hardware and software, must be generated to address these problems. Fortunately, there are potentially many avenues from commercial companies such as NoSQL technologies and compression approaches for genome data that present alternatives to current technologies to test existing hypotheses and generate new ones regarding database management systems.

1.11 ORGANIZATION OF THIS DISSERTATION

This dissertation is organized as follows:

In Chapter 2, we bring an overview of the fundamentals of performance and scalability measurement, and we outline a clinical/genomic framework for understanding different types of measurement and different types of data models. We then describe the instrumentation used in terms of hardware and software. Finally we summarize previous work in performance and scalability measurement in the context of this framework, and we summarize the limitations of existing experiments.

In Chapter 3, we give a detailed description of data management methodology. These methods include data annotation, transformation, importing and manipulation processes. These processes take highly personal information data and arrange it in four different data models. We then describe the queries that retrieve common-logical Precision Medicine information useful to select patients with similar clinical and molecular information.

In Chapter 4, we introduce experiments for measuring performance and scalability on queries of gradually increasing complexity, showing how these can be used for databases of different sizes. To jump-start this experiment, we grounded our approach in already available techniques that measure databases systems with large clinical and governmental data. We then describe the architecture of the different database models and detail the results of retrieving data using different queries in growing database sizes.

In Chapters 5 and 6, we combine the ideas in Chapter 4 to adaptively measure time responses on updating process that either requires or does not require schema changes. To motivate these experiments, we describe the importance of updating process in the Precision Medicine. We then describe scientific efforts that have aimed to address this important and

complex process in database systems. Also, we summarize different approaches already being used to solve this common problem in data management.

Finally, in Chapter 7, we summarize our approaches in this dissertation and present the conclusions we have arrived at based on our results. We then describe future work needed to enable successful data management for Precision Medicine.

2.0 BACKGROUND

Improvements in medical and genomic technologies have dramatically increased the production of electronic data over the last decade. As a result, data management is rapidly becoming a major determinant, and urgent challenge, for the development of Precision Medicine (Fig.1).

2.1 DATA STORAGE AND MANAGEMENT

One of the central problems in Precision Medicine is data storage and management. Clinical and genomic data are voluminous and complex in nature.[93] Probably this type of data is one of the most valuable but complicated types to analyze. This complication led scientists to look for alternatives database management systems that could support heterogeneous[94] data structures and query performance and offer the scalability to facilitate effective data analysis.

Since Precision Medicine is relatively new, research about data management in this area is scarce. However, the literature does contain some proposed models for medical data storage. Most of these use the relational model, which is not surprising given its predominance, even though they are not efficient or flexible enough for use with medical data. However, there are other alternatives to the relational model that look for efficient medical data storage and management. Similar to Column data, Row modeling has been suggested to arrange clinical data.[106] Such as Key-Values, a Knowledge discovery model has been proposed for management

of data according to relationships between concepts and properties.[107] Similar to Document models, it was suggested a model to relate medical data according to their contexts.[108] Most of these models need future research to realize their advantages for big medical data management.

Scientists have been looking for a better approach than the predominant but inefficient relational model to manage medical data. At this point in time, NoSQLs, with their potential to deal with large, heterogeneous and even dynamic data databases, are gaining attention. As mentioned above, NoSQL databases were originally created by web developers to revolutionize the management of real-time data, with which SQL databases had many problems managing. The use of NoSQL has become popular because, unlike SQL, NoSQL does not require strict schemas and keys.

NoSQL databases propose the use of data models different than the relational model. Specifically, NoSQL data models use mainly Key-Value, Column, Document, and Graph models.[109] These models have been successful in managing business data. Similar to medical data, Web data is large, heterogeneous and dynamic because it is frequently produced and updated. This similarity has motivated the use of NoSQL in medical data, although it is difficult to use NoSQL in the Precision Medicine field. NoSQLs present alternate database architectures that are complex to assess[4] and means to integrate heterogeneous electronic health records with dynamic genomic data are not easily available.[6]

Minimal progress has been made in determining the advantages of NoSQL approaches for clinical and genomic data management. Probably, this is because a relatively small number of software developers are familiar with NoSQL approaches; most developers are in areas other than bioinformatics (i.e., social media and web applications).[4] Another downside of NoSQL approaches is the lack of standardization among them. Each NoSQL approach works differently

from the others and has its own programming language for defining, inserting, updating and retrieving information. Thus, with these complexities, plus the multidisciplinary skills (clinical medicine, genomics and bioinformatics) required in Precision Medicine,[110] it is not surprising find few works using NoSQL approaches in Precision Medicine,[111; 112] and no works detailing how NoSQL could be used to advantage in the storage and management of clinical and genomic data to facilitate the development of Precision Medicine. However, given their capabilities, it is imperative to determine the advantages of NoSQL data model approaches (document, key-value, column and graph stores) for performance and scalability outcomes. Then we can identify the most suitable NoSQL approach (and the associated advantages of every NoSQL data model) for effective data management in Precision Medicine.

2.2 PERFORMANCE

Computer performance is mainly dependent on the computer resources available, amount of data being processed, organization of that data and algorithms used. High computer performance for a database could be partially defined as a short query time or response time to process a particular amount of data.[113]

2.2.1 Query time

Performance in database management systems can be measured by using the response time to retrieve the desired data. The performance of different systems can be compared by requesting the same information using the same database and system resources. This is a feasible way to

determine how well a system is doing a task compared to another system.[114] Faster query times are expected in database systems that efficiently manage big data.

2.3 SCALABILITY

Beyond measuring response time, performance can also be measured using other aspects such as scalability. In the database management system context, scalability is the ability of a system to adapt to 1) manage a growing volume of data, or 2) enlarge itself to handle that growth. For example, NoSQL databases are considered scalable systems due to their ability to increase total functionality by distributing a high volume of data across multiple systems. NoSQLs are capable of allocating other resources, such as hardware (nodes), to accommodate the growth of databases for more efficient management. This last is an example of scaling horizontally or scaling out. This property highlights the greater efficiency of NoSQL systems in managing Big Precision Medicine data over SQL systems, where resources must be added to a single node.

High database scalability is associated with efficient data management in cases of high data volume and supports an increasing rate of transactions per second. This computational feature means information is partitioned across the system and processed on separate database allocations. The scalability in NoSQL systems has made them popular because their power to coordinate distributed transactions among their systems. New proposals in database systems, such as Google's massively distributed Spanner technology, which has the potential to replace Google’s BigTable, are aimed at creating such new distributed data models in order to manage ever-increasing amounts of data.[115]

In database management systems, another way to measure performance in terms of scalability is to examine the size scalability of a system. Size scalability is the maximum number of nodes to which a database system can allocate information.[116] Different numbers of processors to check for performance when comparing different database systems. This other way to study performance is important to select database management systems that better handle large amount of data.

A database system with that has a shorter query time and higher scalability will perform data management more efficiently. An efficient system allows the allocation of multiple types of data, such as genomic and clinical concepts. However, the input of multiple types of information causes a database to grow exponentially and consume meaningfully computational resources such as memory and data storage. To be effective, therefore, a system must reduce the potential for problems related with administration and maintenance of databases. The development of such efficient and effective systems for Precision Medicine data could potentially accelerate the generation of new knowledge.

2.4 SIGNIFICANCE

As mentioned earlier, on average, 80 megabytes of data per person is added every year to their electronic health records (EHR) in hospitals that have adopted this technology, and experts expect that this rate will increase as genetic data expands over time.[1] Currently, more than 80 percent of hospitals in the U.S. have adopted some type of EHR technology.[117] EHR adoption by U.S. hospitals is expected to increase even more since current government requirements (Health IT initiative) specifies incentivized payments for the use of certified EHR technologies.[118; 119]

Moreover, the amount of genomic data has increased exponentially since 1995, and genomic data growth is expected to keep increasing as technological improvements continue to lower the cost of sequencing.[120] As those in the medical field focus on the production and storage of clinical and genomic data, leaders in Data Base Management Systems (DBMS) such as RDBMS experts, report possible solutions to the challenges resulting from these increases in big data. For example, a recent report from the President's Council of Advisors on Science and Technology (PCAST) highlights the use of new commercial DBMS, specifically NoSQL technologies, to address big database challenges such as those faced in Precision Medicine.[121] But, important challenges face NoSQL database approaches research. Minimum progress has been made in determining the advantages of NoSQL approaches for managing clinical and genomic data. This is because a relatively small number of software developers are familiar with NoSQL approaches; most of them are in areas (i.e., social media and web applications) other than bioinformatics.[4] Another downside to NoSQL approaches is the lack of standardization among them.

Here at the University of Pittsburgh and Carnegie Mellon University, collaborating with the Institute of Precision Medicine [122] and Pittsburgh Supercomputer Center, and having access to The Cancer Genome Atlas[123] (TCGA) through the Pittsburgh Genome Resource Repository (PGRR), we will perform parallel performance[11] and scaling[9] studies to identify "suitable database approaches" enabling effective data storage and management in Precision Medicine. The goal is to identify database approaches (and associated data structure and system requirements) for effective data storage and management, so that we can suggest the most suitable database approach (or approaches) for Precision Medicine, choose the best data structure and request ideal system resources to prevent or delay failure scenarios. Our quest for effective

data storage and management is based on the observation that growing data coincides with challenging issues in Precision Medicine.[6; 124; 125] We recognize the "dynamic nature" of clinical data, in that it consists of frequently updated, sporadic and heterogeneous data,[11] and we believe genomic data is similar in nature to clinical data. Both types of data require special attention in the design of database schema because of their distinct features.[11] These inherent features also are associated with failure scenarios; this is mainly because of the predominant use of relational database management systems (SQL). Specifically, the continuous clinical and genomic data modifications that must be made because clinical and genomic data are frequently updated,[126; 127] can potentially change the database schema. These modifications make using SQL systems difficult as they require pre-design of the exact field structures of the data(to ensure data consistency) prior to building the database[128; 129]. However, NoSQL approaches do not deal with fixed schemas,[90; 130-132] and some are designed to support concurrent transactions.[132] Theoretically, NoSQL approaches in a database used 24 hours a day make it more feasible that researchers and data managers can be simultaneously requesting and updating data without any potential failure.[132] Furthermore, NoSQL approaches appear to have promising features for clinical[11] and genomic data management because they are suited for large, heterogeneous, and dynamic data, with a high rate of missing data—which are features frequently seen in biological data. This suitability means both high performance and scalability –ability to handle a growing amount of information in a capable manner– this last making them even more feasible (or less expensive) than SQL systems for data management.[4] For instance, it is cheaper to add nodes (i.e. new computers) to a system as with a NoSQL, rather than adding resources to a single node[133; 134] (i.e. buying a powerful large machine that fits all growing clinical and genomic information)

as required by an SQL. These facts point to NoSQL as a key database technology for clinical and genomic data management as future technology (exponentially producing data) advances.

2.5 PUBLIC HEALTH RELEVANCE

Currently, with the trend of ever-increasing costs in health care and the gradual retirement of the "baby boom" generation, we are witnessing an increase in demands on the health care system.[135] The need for new strategies concerning both access and quality of healthcare.[136] Precision Medicine has been proposed by the President's Council of Advisors on Science and Technology as an option that combines improved patient outcomes with reduced costs.[135] However, the development of Precision Medicine [137] is hindered in part by the lack of tools for the efficient management of clinical & genomic data.[138; 139] Our proposal has a wider public health impact since we are focusing on one of the main challenges to the development of Precision Medicine and, consequently, addressing a possible solution to the everyday increasing demands of health care. Thus, by investigating determining which NoSQL technology is the most suited to the storage and management of clinical and genomic data, we will be contributing to the development of Precision Medicine and facilitating the introduction into the health care of a promising class of new diagnostic tests, medications and treatments, among others. These products will also contribute to the development and application of successful population-based prevention programs, which are essential for decreasing the public and global burden of disease.

2.6 INSTRUMENTATION

2.6.1 Hardware instrumentation

Four database approaches were developed, respectively, to implement the selected databases concerned in the project. The systems were built using the supercomputer components from the Data Exacell (DXC) pilot project from the National Advanced Cyberinfrastructure Ecosystem at the Pittsburgh Supercomputer Center, University of Pittsburgh and Carnegie Mellon University and a personal computer. The Data Exacell (DXC) is a pilot project funded by the NSF Data Infrastructure Building Blocks (DIBBs) award #ACI-1261721 to create, deploy, and test software and hardware building blocks to enable data analytics in scientific research. Our project used these resources because diverse data analytic requirements were necessary to motivate, test, and demonstrate the DXC's capabilities. The components of the DXC, which is shown in Figure 3, are described below:

- Crucible: innovative, disk-based near-line storage system featuring low latency, high bandwidth, and high reliability for large-scale datasets
- Blacklight: the world's largest shared-memory supercomputer, capable of running Java and applications of 1-2048 threads using up to 16TB.
- Sherlock: a unique system for hardware- and software-optimized graph analytics, using either RDF/SPARQL for productivity or threaded C++ for very broad applicability.
- Hadoop and Spark: a cluster of nodes for the Hadoop ecosystem.
- Application, Database, and Web Server Nodes: cutting-edge technologies enable the development of powerful new application architectures.

We mostly used DSXCDB, Sherlock and Blacklight to store and manipulate our data. Blacklight is an SGI UV 1000cc-NUMA shared-memory system containing 256 blades. Each blade has 2 Intel Xeon X7560 (Nehalem) eight-core processors. In total, this machine holds a total of 4096 cores, where each one has a clock rate of 2.27 GHz, supports two hardware threads and can perform 9 Gflops. In local memory terms, the two eight core processors per blade share 128 Gigabytes. In addition, 16 Terabytes of shared memory can be used in running jobs.[140] We topped our resources at 256 GB memory running queries using the DXC from a Virtual Machine (VM) that included default hosting for DBMSs such as MongoDB, Redis, Cassandra and MySQL.

We also use standard computing resources to run the four DB technologies. These resources consisted in a personal computer equipped with a 2.66 GHz Intel Core i7 CPU and an 8.00 GB 1067 MHz DDR3.

Figure 3. Software architectures developed on the DXC.

2.6.2 Software instrumentation

In general, the DXC components, including Blacklight, use the Linux operating system, a special version of the SuSE. As a note, the machine had Intel C, C++ and Fortran compilers and Gnu Fortran, C and C++ compilers installed. These installations allow users to run threaded, MPI and hybrid threaded and MPI programs. The machine support OpenMP programs had available UPC, Java and a list of software and DBMS packages, including MongoDB, Redis, Cassandra and MySQL.[140] The operating system of the personal computer was OSX version 10.9.5

2.6.2.1 Database Settings

We use four database management systems: Mongo DB, Cassandra, Redis and MySQL.

MongoDB

We used version 2.6.7 of this document-based open source system with its JavaScript shell for database administration and data manipulation. Mongo DB was downloaded from its official site (http://www.mongodb.org/downloads) and decompressed. We use the pre-installed default settings and the started a server ("mongod" executable file) to manipulate the system. We also used an HTTP server set up by mongod that listens on a port 1,000 higher than the main port to get some administration information about our database. The server was used for performing administrative functions and inspecting running instances. We also use a standalone MongoDB Client to be connected to the MongoDB server. Specifically, the client allowed us to connect our database to a global variable "db" that is the primary access point to the MongoDB server through the shell, allowing us to manipulate our database.

Cassandra

We used Apache Cassandra version 2.1.3 (the most stable version). It was installed from its official website (http://cassandra.apache.org/download/) and included the core server, the nodetool administration command-line interface, and a development shell (cqlsh and the old cassandra-cli). We used the default configuration found in the conf directory. We used the interactive command line interface for Cassandra (cqlsh) to execute CQL (Cassandra Query Language) to manage our databases (define schemas, insert data, execute queries).

Redis

We used this data structure server version 2.8.19, implemented as Key-Value store. Redis open source was downloaded from its official site (http://redis.io/download).

Redis was compiled using a Compiler (GCC). The test suite was enabled using Tcl 8.5. Using the default settings, we manipulated our files using Redis's command-line client (redis-cli command), a fully featured interactive client. We used redis-cli to connect to a local or remote host Redis server for performing administrative functions.

MySQL

We used version 5.6.23 of this open source relational database. MySQL was downloaded from its official website (http://dev.mysql.com/downloads/), and its default configuration was used. We managed MySQL through the MySQL client program (terminal monitor) to connect our database to a MySQL server and create/manipulate data. Specifically we used MySQL in batch mode to execute queries from files.

2.7 DATA SOURCES

The Cancer Genome Atlas (TCGA)

Our resource for clinical and genomic information was The Cancer Genome Atlas (TCGA)[123] data from the Pittsburgh Genome Resource Repository (PGRR).[123] PGRR is a framework for accessing de-identified national big data datasets relevant for Precision Medicine like TCGA.[123] PGRR allowed us to use this data easily with tools and resources provided by the Simulation and Modeling Center (SaM) and the Pittsburgh Supercomputer Center (PSC). TCGA is defined as the "comprehensive and coordinated effort to accelerate the understanding of the molecular basis of cancer through the application of genome analysis technologies, including large-scale genome sequencing".[123] The National Cancer Institute funds TCGA. From PGRR we had (as of October 24, 2014) access to 10,822 clinical and genomic records classified in 32 varieties of cancers and one group of controls. We used mainly the clinical and genomic information; for this last we used DNA and RNA sequencing. Externally, this genome sequencing was annotated using ClinVar to identify the clinically significant variants.

ClinVar

ClinVar was used to identify the clinically significant variants from TCGA patient information. ClinVar is an open source database used to annotate human genome variations according to the needs of the medical genetics community.[141] ClinVar documents the relationships among human variations and phenotypes, including observed health status. This publicly available archive was made from collections of patients with reported variants. The reported variants are related with their clinical significance using intermediate knowledge data such as observational or experimental studies. The interpreted variants are mapped to the human genome to report the

location of each variant and its associated phenotype. ClinVar can be used in many applications and bioinformatics algorithms to annotate the clinical significance of the genetic variation.[142]

Data architecture

Clinical and genomic information from TCGA patients and their corresponding ClinVar annotation were transformed into four different data architectures (Figures 98-101): Document, Key-Value, Column, and Table stores.

3.0 DATA MANAGEMENT

3.1 ANNOTATION, TRANSFORMATION, IMPORTING AND DATA MANIPULATION PROCESS

3.1.1 Database building process

To build our clinical and genomic database, we first selected patients with a diagnosis of breast cancer from TCGA that have specific demographic, clinical, surgical, pharmacological and structural and functional genomic information, these last two including Whole Genome Sequencing (WGS) and RNA sequencing (RNAseq). Because we were interested in integrating specific clinical and genomic data prior to building our database, we conducted pre-computed operations to create files that facilitated the combination of each patient's clinical and genomic information.

3.1.1.1 Pre-computed clinical files

We created each patient's clinical files by manually curating the clinical information stored in the TCGA data. Posteriorly, we selected specific data to build our pre-computed clinical files. This specific data consisted of storage data values from the following patient information: ID, gender, race, vital status, days to last follow-up, age at diagnosis, staging by Tumor-Node-Metastasis (TNM) classification to define breast cancer, anatomical site affected, surgical

procedures, drugs prescribed, and radiation types applied for treatment. An example of these files is shown in Table 1.

Table 1. Example of a pre-computed clinical file.

px	gender	race	vital_status	days_to_last_followup	age_at_diagnosis	pathologic_T	pathologic_N	pathologic_M	pathologic_stage	surgical_procedure_first
TCGA-A1-A0SB	FEMALE	WHITE	Alive	852	67	T2	N1	M0	StageIIB	ModifiedRadicalMastectomy
TCGA-A1-A0SD	FEMALE	WHITE	Alive	3153	66	T1c	N0i-	M0	StageIA	Lumpectomy
TCGA-A1-A0SE	FEMALE	BLACKORAFRICANAMERICAN	Alive	2365	36	T1	N0i-	M0	StageIA	Lumpectomy
TCGA-A2-A04Q	FEMALE	WHITE	Alive	764	53	T2	N1mi	M0	StageIIB	SimpleMastectomy

px	drug_name_1	drug_name_2	drug_name_3	drug_name_4	drug_name_5	drug_name_6	drug_name_7	drug_name_8	radiation_type_1	anatomic_treatment_site_1
TCGA-A1-A0SB	NotAvailable	Null	Null	Null	Null	Null	Null	Null	EXTERNALBEAM	RegionalSite
TCGA-A1-A0SD	Arimidex	Cytoxan	Adriamycin	Null	Null	Null	Null	Null	EXTERNALBEAM	PrimaryTumorField
TCGA-A1-A0SE	Anastrozole	Tamoxifen	Adriamycin	Cytoxan	Taxol	Null	Null	Null	EXTERNALBEAM	PrimaryTumorField
TCGA-A2-A04Q	Zometa	Tamoxifen	Herceptin	Taxol	Adriamycin	Cytoxan	Null	Null	EXTERNALBEAM	PrimaryTumorField

3.1.1.2 Genomic pre-computed files

To integrate the DNA and RNA sequencing files, we first annotated the DNA sequencing data from Variant Call Format (vcf) files using ClinVar. Then, the annotated clinically significant genomic variants were filtered from those located on chromosome 13 due to our specific queries explained in the next section. Those clinically significant variants set in chromosome 13, where BRCA2 is located, were selected and stored using their genomic location, i.e., chromosomal position. Posteriorly, to identify if the previously selected genomic variants were expressed we selected those variants that had Reads Per Kilobase per Million (RPKM) values in their RNA sequencing files. The resulting expressed variants were stored using, again, their genomic

location. Finally, the resulting structural (DNA) and functional (RNA) genomic files were merged using their patient ID. An example of these files is shown in Table 2.

Table 2. Example of a pre-computed genomic file.

PatientID	Pathogenic_Chr_Pos_1	Pathogenic_Chr_Pos_2	Pathogenic_Chr_Pos ...	Exon_1_RPKM_1	Exon_2_RPKM_2	Exon_..._RPKM_...
TCGA-A1-A0SB	13_32890572	13_32906729	...	13_32890572	Null	...
TCGA-A1-A0SD	13_32903685	13_32906729	...	Null	13_32906729	...
TCGA-A1-A0SE	13_32906729	13_32912299	...	13_32906729	13_32912299	...
TCGA-A2-A04Q	13_32890572	13_32903685	...	Null	13_32903685	...

3.1.2 Database transformation process

We transformed our original clinical and genomic database into each of the four specific data models required by our previously selected database technologies. For the Document model, our original database was transformed into JavaScript Object Notation syntax (JSON) files. For the Key-Value model, we transformed our database into dictionary approaches, lines of headers and values and saved it as as simple text files. For the Column model, our database was transformed into structured simple text files compatible with Cassandra Query Language input options. Finally, for the Relational model, we saved our data in tables where attributes represent clinical and genomic information and rows represent patient IDs. The transformation process was performed using a combination of unix scripts to successfully upload files into each database technology. The transformation process was the key step performed prior to beginning to measure query times in each selected database technology. Clinical and genomic information

from TCGA-PGRR was written in four data models: Document (JSON files), Key-Value, Column and Table stores.

3.2 DESCRIPTION OF QUERIES

In order to assess each database systems suitability for managing clinical and genomic data for Precision Medicine, we built a series of typical biomarker selection queries based on meaningful biological questions.[143] These queries varied in complexity and focused on detecting individuals with clinically relevant genetic variants for cancer prevention and early detection. In other words, with the queries, we aimed to detect individuals who had a higher number of risk factors for breast cancer and were more likely to have a successful prognosis, such as aged females with strong environmental and genetic risk factors. Age is the most important risk factor for women's breast cancer. It is estimated that more than 3% of American females over 55 years old will be diagnosed with breast cancer during the next 10 years. In addition, women who carry genetic mutations, such as mutations in the BRCA2 gene, have a higher risk of breast cancer than women who do not carry such mutations. However, these genetic mutations are estimated to account for less than 10% for all breast cancers;[144] different populations have shown different prevalence, such as in European populations.[145] Current evidence about founder mutations in these specific populations have summarized their impact in breast cancer management.[145] These type of queries could potentially impact the development of Precision Medicine in breast cancer, since they have the potential to identify people at molecular level risk to make genetic testing more affordable and cost-effective, positively influencing the healthcare system regarding attention to breast cancer.

The following Precision Medicine queries act as a filter for subgroup populations by identifying highly specific information of patients with breast cancer.[146] For example, by running a less specific query we identified a large group of individuals with similar demographic characteristics. However, by running the most specific query, we identified one patient, with ID TCGA-A2-AOEM, clinically described as a white and alive female that has survived with breast cancer for more than 30 months and listed as StageIA with a TNM classification of T1, N0i-, M0. The query also indicates that this patient underwent to conventional therapy including Lumpectomy, hormone therapy (Tamoxifen) and external beam radiotherapy. Genetically she carries a pathogenic European founder mutation identified in Swedish populations at Chromosome 13 location 32912750 that is expressed since the mutation is located in the range of the exon located at chr13:32910402-32915333, with normalized RPKM value of 0.208929385980881. Thus, these queries have the potential to filter and identify individuals with similar clinical and genetic characteristics and to identify individuals to test relevant biological questions. This filtration process could be used to compare not just individuals but groups of people to identify the efficacy of cancer therapies such as it is tested in Wang et al.[147] that query people with similar clinical and genomic information to discover gene expression patterns that affect response therapies on specific groups of people.

In order to measure the functionality of the databases approaches we created two sets of queries: “Static queries” and “Dynamic queries”.

3.2.1 Static queries

Static queries were used in experiment 1 and 2. We called them “Static queries” since they retrieve only pre-defined information stated in the first schema designed for each database implementation. The retrieved information from each static query is described below.

Query 1: Request for white female patients with more than 55 years who have survived longer than 30 months.

Query 2: Request for white female patients with more than 55 years who have survived longer than 30 months with any cancer stage I, when breast cancer can be effectively treated, with TNM values including any T, N>1 and M=0.

Query 3: Request for white female patients with more than 55 years who have survived longer than 30 months with any cancer stage I, when breast cancer can be effectively treated, with TNM values including any T, N>1 and M=0. Also including therapies in accordance with those described by the American Cancer Society (ACS) for the selected cancer stage, including: 'SurgicalResection'; Chemotheraphy: trastuzumab, pertuzumab; and 'Adjuvant hormone therapy': Tamoxifen, Toremifene, Fulvestrant.

Query 4: Request for white female patients with more than 55 years who have survived longer than 30 months with any cancer stage I, when breast cancer can be effectively treated, with TNM values including any T, N>1 and M=0. Including the following therapy: 'SurgicalResection'; Chemotheraphy: trastuzumab, pertuzumab; and adjuvant hormone therapy: Tamoxifen, Toremifene, Fulvestrant. Including clinically relevant BRCA2 variants identified as European founder mutations, as shown in Figure 4.

Query 5: Request for white female patients with more than 55 years who have survived longer than 30 months with any cancer stage I, when breast cancer can be effectively treated,

with TNM values including any T, N>1 and M=0. Including the following therapy: 'SurgicalResection'; Chemotheraphy: trastuzumab, pertuzumab; and adjuvant hormone therapy: Tamoxifen, Toremifene, Fulvestrant. Including clinically relevant BRCA2 variants identified as European founder mutations at any genetic expression level.

Figure 4. BRCA2 variants identified as European founder mutations.

3.2.2 Dynamic queries

In order to assess experiment 3, described below in section 6.2, we modified our queries to retrieve new information, non-predefined data (new header) in the initial database schema. The modification consisted of the fact that all queries also requested “Population” information. This data was obtained from the European ancestry and founder mutation study, described in section 3.1.1.2. Specifically these dynamic queries will retrieve the hypothetical Population from which each patient comes. For example, they ask if people are from one of the following populations: AshkenaziJews, Austrian, Slovenian, Italian, French, Spanish, Portuguese, Belgian, Dutch, German, Czech, Hungarian, Greek, Cypriot, Danish, Swedish, Finnish, Iceland, British, Irish, Polish, Latvian, Lithuanian, Belarusian or Russian. As a note, this information did not come

from TCGA project. We add this new information just to measure database performance and scalability toward updating processes that include schema modifications. The information retrieved from each dynamic query is described below.

Query 1: Request for white female patients with more than 55 years who have survived longer than 30 months from a specific European population.

Query 2: Request for white female patients with more than 55 years who have survived longer than 30 months, from a specific European population, with any cancer stage I, when breast cancer can be effectively treated, with TNM values including any T, N>1 and M=0.

Query 3: Request for white female patients with more than 55 years who have survived longer than 30 months, from a specific European population, with any cancer stage I, when breast cancer can be effectively treated, with TNM values including any T, N>1 and M=0. Also including therapies described by the American Cancer Society (ACS) according to the selected cancer stage, including: 'SurgicalResection'; Chemotheraphy: trastuzumab, pertuzumab; and 'Adjuvant hormone therapy': Tamoxifen, Toremifene, Fulvestrant.

Query 4: Request for white female patients with more than 55 years who have survived longer than 30 months, from a specific European population, with any cancer stage I, when breast cancer can be effectively treated, with TNM values including any T, N>1 and M=0. Including the following therapy: 'SurgicalResection'; Chemotheraphy: trastuzumab, pertuzumab; and adjuvant hormone therapy: Tamoxifen, Toremifene, Fulvestrant. Including clinically relevant BRCA2 variants identified as European founder mutations, as shown in Figure 4.

Query 5: Request for white female patients with more than 55 years who have survived longer than 30 months, from a specific European population, with any cancer stage I, when breast cancer can be effectively treated, with TNM values including any T, N>1 and M=0.

Including the following therapy: 'SurgicalResection'; Chemotheraphy: trastuzumab, pertuzumab; and adjuvant hormone therapy: Tamoxifen, Toremifene, Fulvestrant. Including clinically relevant BRCA2 variants identified as European founder mutations ay any genetic expression level.

4.0 PERFORMANCE AND SCALABILITY ON QUERIES OF GRADUALLY INCREASING COMPLEXITY AND DATABASE SIZE

4.1 INTRODUCTION

DBMS that successfully cope with characteristics of clinical and genomic data should display high performance and scalability. As discussed on §2, the burgeoning growth in the amount of clinical and genomic data that are highly heterogeneous and dynamic –constantly updated– requires database approaches suitable for successful data management.[142] Clinical and genomic data show unique characteristics that require special attention when designing a database management system. The unique characteristics are due to the current technologies that massively produce heterogeneous data. This is because recent technological advances seem to be more sensitive to generating any imaginable kind of data. For example, Next Generation sequencing machines can generate structural and functional data at an impressive rate. Thus, it appears essential to choose a recent database technology inherently designed to cope with current data challenges with respect to storage and management issues. In fact, it is more likely that newer database systems display higher performance and scalability than non-recent technologies.

However, today, clinical and genomic data are managed with a database system that has been pre-dominate for more than three decades – a relational database (SQL). As a result of its age, this technology displays many limitations with respect to manipulating recently generated

clinical and genomic data. While the relational database (SQL) is a universal approach to data management,[148] it was not designed to deal with today's current high volume of generated data, like clinical and genomic data. SQL was not built to handle large databases and presents many problems, such as the need to pre-design schema for data consistency.[148] Furthermore, SQL systems present complications to managing heterogeneous data, especially the types of data that remains unused due to the nature of the data but nonetheless still need to be stored. These drawbacks cause SQL systems to produce failures in their management of big and mercurial clinical and genomic data, resulting in inefficient storage and poor performance.

One logical way to address current database system complications is to use more recent database systems that have been designed to handle currently generated data, data similar in nature to clinical and genomic data. These recent database technologies are termed "NoSQL". NoSQL technologies are designed to cope with current generated data.[147] As described in §2, NoSQL technologies have advantages over SQL systems such as data models can typically maintain data in group order. This is just one of the many characteristics that gives them the ability to handle massively generated data. Big firms in the computational industry have achieved success using NoSQL technologies, demonstrating the value of NoSQL systems for data management in the business field.

However, as there is not one specific NoSQL technology designed to cope with the uniqueness of clinical and genomic data most effectively, we translated the use of these successful business technologies to the Precision Medicine field, which appears to have data similar to some of these business data but more heterogeneous. Thus, as a first approach we attempt to use four recent database technologies designed to cope with the characteristics of recently produced data such as clinical and genomic data.

A successful database system should display higher performance –lower query times and higher scalability –lower query times even through increasing data. To determine a suitable database approach for clinical and genomic data, three NoSQL technologies are compared in this experiment as well as one SQL approach as a reference. The NoSQL database approaches were based on Document store-MongoDB, Key-Value store-Redis, and Column store-Cassandra, and the SQL approach was the Relational Model-MySQL. The comparisons were made regarding query performance and scalability. The databases were populated with real clinical and genomic patient information obtained from the publicly available TCGA database.

In the following section we describe the details of our experiment in comparing performance and scalability on queries that gradually increase in complexity and database size. In sub-section 4.1 we briefly present the performance results from the experiment. We summarize the results in our conclusion.

4.2 EXPERIMENT 1: COMPARING PERFORMANCE AND SCALABILITY ON QUERIES OF GRADUALLY INCREASING COMPLEXITY AND DATABASE SIZE

To identify the database approaches with higher performance in making queries with different complexity and using different database sizes, we measure performance by adapting the methodology used in Lee, et al.[11] but using Precision Medicine data instead of just clinical data. To identify the fastest NoSQL approach in querying clinical and genomic data, we used standard computing resources in four databases sizes (1,000, 5,000, 10,000 and 50,000 records) and four database approaches (Mongo, Redis, Cassandra and MySQL) and measured the query times for five different types of queries of varying complexity, as described in §3.2. Each query was run 3

times to calculate the mean query time and the standard deviation. Once query times were obtained, we created tables to graph the results, facilitating comparisons among database approaches. The approach with the highest performance for each of the queries and database sizes was the database approach that had the lowest query time for each set, as described in the experiment 1 workflow shown in Figure 5. All of these calculations are described in section §4.2.1.1 Timing Performance. To identify the database approach with the highest scalability we used the query times obtained above from measuring performance, but calculating, for each database approach, the difference in query times that resulted from the largest database minus the smallest database query times, and selecting the database approach with the lowest query time. All of these calculations are described in section §4.2.1.1 Timing Scalability. In addition, to better visualize the database approach with highest scalability in the different database sizes, we display the scalability results as shown in the "Scaling the database size" experiment in Labrinidis et al.[9] Specifically, we display the query times adjusted by scale factors. These factors represent the number in which the smaller database (1000 records) is scaled or multiplied toward the different databases sizes (1,000, 5,000, 10,000, 50,000 records). We calculated the scaling factors according to the following equation:

$$y=Cx$$

Where C is the scale factor (coefficient) for x, x represents our smaller database and y represents one of our four different database sizes. As a comment, C is also called the constant of proportionality of y to x.

Thus, our four different scaling factors are as follows:

- 1, which corresponds to 1,000 patients records
- 5, which corresponds to 5,000 patients records

- 10, which corresponds to 10,000 patients records
- 50, which corresponds to 50,000 patients records

This experiment 1 was repeated, but using supercomputing resources and even larger databases. These databases consisted of 1,000, 5,000, 10,000, 50,000, 100,000, 1 million, 10 millions and 100 millions records, respectively. We used the same four database technologies: MongoDB, Redis, Cassandra and MySQL. Performance and Scalability results are annotated in Tables and Graphs, and described in section §4.2.1.1 Timing Performance and Timing Scalability, respectively.

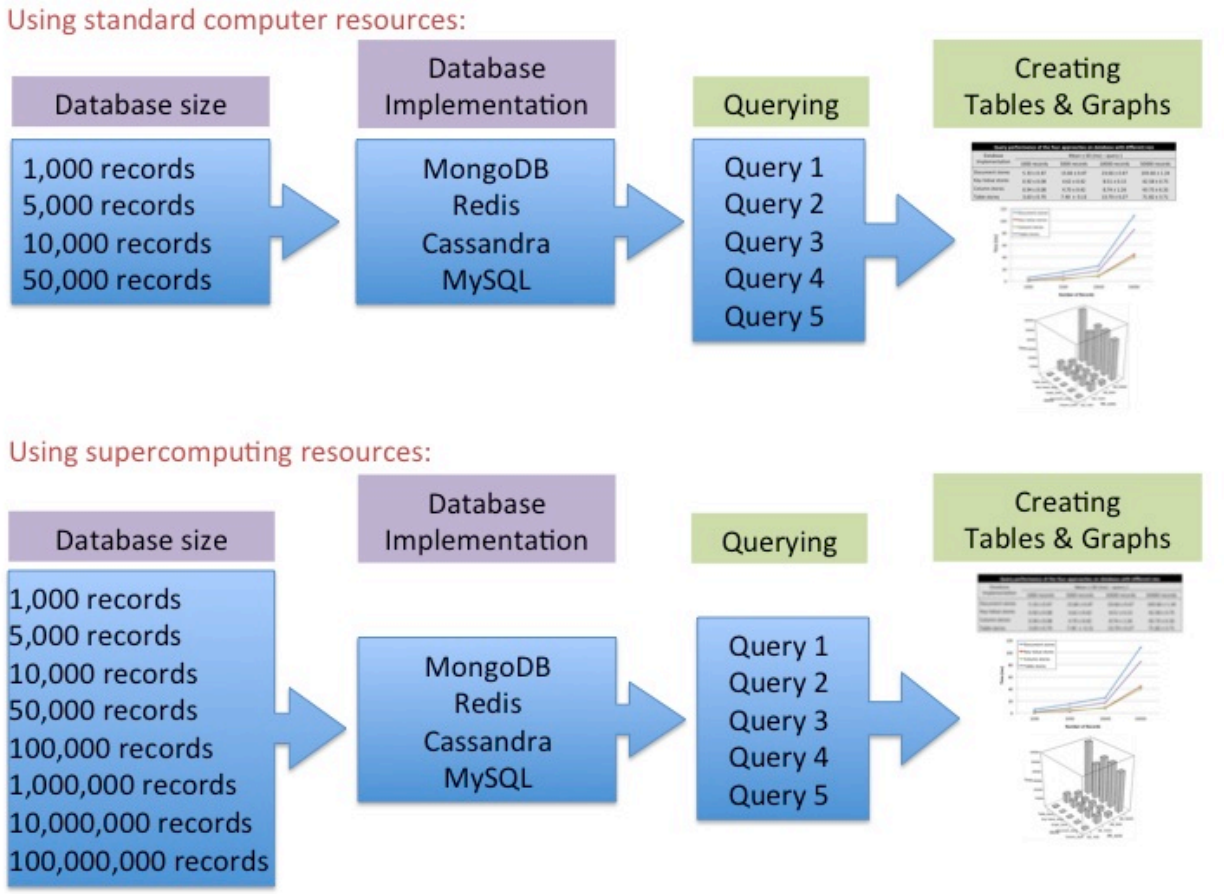

Figure 5. Workflow of Experiment 1.

4.2.1 Experimental results

The query time for each of the four DBMS was evaluated by making five different queries, as described above, of gradually increasing complexity using databases of different sizes, as shown in Figure 5. The variation in query time in the database updating process was also studied. For each of the four database approaches, the time taken to make the queries of varying complexity, as described above, was measured with databases containing 1000, 5000, 10,000 and 50,000 patients' records. In addition, using the same methodology but with supercomputing resources the query time of the four DBMS was measured using the same and even larger databases, ones with 100,000, 1,000,000, 10,000,000 and 100,000,000 patient's records, respectively, as shown in Figure 5. For each of the database sizes mentioned above, each query was made 3 times in order to calculate the average query time and the standard deviation (SD). The query times are given in Tables 3-17.

4.2.1.1 Query time results for queries of varying complexity in databases of different sizes

Timing Performance

The timing performances for the simplest query are given in Table 3. For query 1 in the DB size of 1,000 records, Cassandra and Redis showed almost the same query times. In fact, Cassandra and Redis obtained results more than 5 and 7 times faster than MySQL and MongoDB, respectively. In the 5,000-record database, again Cassandra and Redis showed very similar query times. In this DB size, Cassandra and Redis were almost 2 and 3 times faster than MySQL and MongoDB, respectively. In the 10,000-record database, Cassandra and Redis again had almost the same query times, and both technologies were again almost 2 and 3 times faster than MySQL

and MongoDB, respectively. Finally, in the 50,000-record database, Cassandra and Redis had similar query times. In this DB size, Cassandra and Redis were two and more than two times faster than MySQL and MongoDB, respectively.

The timing performances for query 2 are given in Table 4. For query 2 in the 1000-record DB, Cassandra and Redis had similar times, Cassandra and Redis were more than two and seven times faster than MySQL and MongoDB, respectively. In the 5,000-record DB, the results were somewhat different than those previous. Cassandra, Redis and MySQL showed similar results. For this size DB, these three technologies were around 3 times faster than MongoDB. In the 10,000-records DB, the same three technologies -- Cassandra, Redis and MySQL -- again had similar results. They were around two times faster than MongoDB. In the 50,000-record DB, Cassandra and Redis were the faster technologies. Both technologies were slightly faster than MySQL and around 2 times faster than MongoDB.

The timing performances for query 3 are given in Table 5. For query 3, in the DB size of 1,000 records Cassandra and Redis showed the smallest query times. They were almost 3 and 87 times faster than MySQL and MongoDB, respectively. In the 5,000 record DB, Cassandra and Redis were the fastest technologies. They were slightly faster than MySQL and almost 19 times faster than MongoDB. In the 10,000 record DB, Cassandra and Redis returned the smallest query times. Again, their results were slightly faster than those of MySQL and more than 11 times faster than those of MongoDB. Finally, in the 50,000-record DB, Cassandra store was the fastest technology. This technology was slightly faster than the Redis and MySQL, and almost 5 times faster than MongoDB.

The timing performances for query 4 are given in Table 6. For query 4, in the DB of 1,000 records, Cassandra had the lowest query time. It was almost 2 times faster than Redis and

MySQL and more than 129 times faster than MongoDB. In the 5,000-recordDB, Cassandra was the fastest technology. It was slightly faster than the MySQL and Redis, and more than 27 times faster than MongoDB. In the 10,000-record DB, Cassandra had the smallest query time. This was almost two times faster than the MySQL and Redis and almost 17 times faster than MongoDB. In the 50,000-record DB, again, Cassandra was the fastest technology. Its query time was almost 2 times faster than MySQL and Redis and 5.39 times faster than MongoDB.

The timing performances for the more complex query are given in Table 7. For query 5, in the DB of 1,000 records, Cassandra was the fastest technology. It was 2 times faster than Redis, more than 3 times faster than MySQL, and almost 150 times faster than MongoDB. In the 5,000-record DB, Cassandra showed the lowest query time. It was around 2 times faster than the Redis and MySQL, and more than 31 times faster than MongoDB. In the 10,000-record DB, Cassandra was the fastest technology. It was more than 2 times faster than the Redis and MySQL, and almost 19 times faster than MongoDB. Finally, in the 50,000-record DB, Cassandra had the smallest query time. It was almost 2 times faster than MySQL, more than two times faster than Redis, and almost 6 times faster than MongoDB.

The query times for the larger databases are given in Tables 8-12. We studied the same four DBMS technologies but using supercomputing resources, making the five varied complex queries in the same and even larger databases (100,000, 1,000,000, 10,000,000 and 100,000,000 records).

The timing performances for the simplest query are given in Table 8. For query 1 in the database size of 1,000 records, Cassandra and Redis showed almost the same query times. In fact, Cassandra and Redis obtained results more than 8 and 13 times faster than MySQL and MongoDB, respectively. In the 5,000-record database, again Cassandra and Redis were the

fastest technologies. In this database size, Cassandra and Redis were more than 6 and 8 times faster than MySQL and MongoDB, respectively. In the 10,000-record database, Cassandra and Redis again had almost the same query times, and both technologies were more than 5 and 8 times faster than MySQL and MongoDB, respectively. Finally, in the 50,000-record database, Cassandra and Redis had similar query times, respectively. In this database size, Cassandra and Redis were more than 5 and 8 times faster than MySQL and MongoDB, respectively. In the 100,000 records, Cassandra and Redis were the fastest technologies. In fact, they were more than 4 and 6 times faster than MySQL and MongoDB, respectively. In the 1-million records database, again Cassandra and Redis were the fastest technologies. For this database size, they were more than 7 and 8 times faster than MySQL and MongoDB. In the 10 million-records, Cassandra and Redis again were the fastest technologies. Both technologies were again more than 7 times faster than MySQL and MongoDB. In 100 million records, Cassandra and Redis were the fastest technologies. In fact, they were more than 7 and 23 times faster than MySQL and MongoDB, respectively.

The timing performances for query 2 are given in Table 9. For this query in the database size of 1,000 records, Cassandra and Redis showed almost the same query times. In fact, Cassandra and Redis obtained results more than 3 and 16 times faster than MySQL and MongoDB, respectively. In the 5,000-record database, again Cassandra and Redis were the fastest technologies. In this database size, Cassandra and Redis were more than 2 and 5 times faster than MySQL and MongoDB, respectively. In the 10,000-record database, Cassandra and Redis again had almost the same query times, and both technologies were more than 2 and 5 times faster than MySQL and MongoDB, respectively. Finally, in the 50,000-record database, Cassandra and Redis had similar query times, respectively. In this database size, Cassandra and

Redis were more than 2 and 4 times faster than MySQL and MongoDB, respectively. In 100,000 records, Cassandra and Redis were the fastest technologies. In fact, they were slightly faster than MySQL and more than 3 times faster than MongoDB, respectively. In the 1-million records database, again Cassandra and Redis were the fastest technologies. For this database size, they were more than 3 and 4 times faster than MySQL and MongoDB. In 10 million-records, Cassandra and Redis again were the fastest technologies. Both technologies were again more than 3 and 4 times faster than MySQL and MongoDB. In 100 million records, Cassandra and Redis were the fastest technologies. In fact, they were more than 3 and 10 times faster than MySQL and MongoDB, respectively.

The timing performances for query 3 are given in Table 10. For this query in the database size of 1,000 records, Cassandra and Redis showed almost the same query times. In fact, Cassandra and Redis obtained results more than 2 and 510 times faster than MySQL and MongoDB, respectively. In the 5,000-record database, again Cassandra and Redis were the fastest technologies. In this database size, Cassandra and Redis were slightly faster than MySQL and more than 104 times faster than MongoDB, respectively. In the 10,000-record database, Cassandra and Redis again had almost the same query times, and both technologies were slightly faster than MySQL and more than 52 times faster than MongoDB, respectively. Finally, in the 50,000-record database, Cassandra and Redis had similar query times, respectively. In this database size, Cassandra and Redis were slightly faster than MySQL and more than 14 times faster than MongoDB, respectively. In 100,000 records, Cassandra and Redis were the fastest technologies. In fact, they were slightly faster than MySQL and more than 8 times faster than MongoDB, respectively. In the 1-million records database, again Cassandra and Redis were the fastest technologies. For this database size, they were more than 2 and 5 times faster than

MySQL and MongoDB. In 10 million-records, Cassandra and Redis again were the fastest technologies. Both technologies were again more than 2 and 4 times faster than MySQL and MongoDB. In 100 million records, Cassandra and Redis were the fastest technologies. In fact, they were more than 2 and 8 times faster than MySQL and MongoDB, respectively.

The timing performances for query 4 are given in Table 11. For this query in the database size of 1,000 records, Cassandra and Redis showed almost the same query times. In fact, Cassandra and Redis obtained results slightly faster than MySQL and 318 times faster than MongoDB. In the 5,000-record database, again Cassandra and Redis were the fastest technologies. In this database size, Cassandra and Redis were slightly faster than MySQL and more than 108 times faster than MongoDB. In the 10,000-record database, Cassandra and Redis again had almost the same query times, and both technologies were slightly faster than MySQL and more than 52 times faster than MongoDB. Finally, in the 50,000-record database, Cassandra and Redis had similar query times. In this database size, Cassandra and Redis were slightly faster than MySQL and more than 12 times faster than MongoDB. In 100,000 records, Cassandra and Redis were the fastest technologies. In fact, they were slightly faster than MySQL and more than 6 times faster than MongoDB. In the 1-million records database, again Cassandra and Redis were the fastest technologies. For this database size, they were slightly faster than MySQL and 3 times faster than MongoDB. In 10 million-records, Cassandra and Redis again were the fastest technologies. Both technologies were again slightly higher than MySQL and 6 times faster than MongoDB. In 100 million records, Cassandra and Redis were the fastest technologies. In fact, they were slightly faster than MySQL and more than 4 times faster than MongoDB.

The timing performances for query 5 are given in Table 12. For this query in the database size of 1,000 records, Cassandra and Redis showed almost the same query times. In fact,

Cassandra and Redis obtained results slightly faster than MySQL and 521 times faster than MongoDB. In the 5,000-record database, again Cassandra and Redis were the fastest technologies. In this database size, Cassandra and Redis were slightly faster than MySQL and more than 106 times faster than MongoDB. In the 10,000-record database, Cassandra and Redis again had almost the same query times, and both technologies were slightly faster than MySQL and more than 54 times faster than MongoDB. Finally, in the 50,000-record database, Cassandra, Redis and MySQL had similar query times. In this database size, they were more than 11 times faster than MongoDB. In 100,000 records, MySQL were the fastest technology. In fact, it was slightly faster than Cassandra and Redis and more than 6 times faster than MongoDB. In the 1-million records database, Cassandra and Redis were the fastest technologies. For this database size, they were slightly faster than MySQL and 2 times faster than MongoDB. In 10 million-records, Cassandra and Redis again were the fastest technologies. Both technologies were again slightly higher than MySQL and MongoDB. In 100 million records, Cassandra and Redis were the fastest technologies. In fact, they were slightly faster than MySQL and more than 3 times faster than MongoDB.

Table 3. Query performance of experiment 1: query 1.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 1			
	1000 records	5000 records	10000 records	50000 records
MongoDB	5.33 ± 0.47	13.66 ± 0.47	23.66 ± 0.47	103.66 ± 1.24
Redis	0.92 ± 0.08	4.62 ± 0.42	8.51 ± 0.15	42.58 ± 0.75
Cassandra	0.94 ± 0.08	4.70 ± 0.42	8.74 ± 1.26	43.73 ± 6.32
MySQL	3.63 ± 0.70	7.40 ± 0.13	13.70 ± 0.27	71.82 ± 3.71

Table 4. Query performance of experiment 1: query 2.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 2			
	1000 records	5000 records	10000 records	50000 records
MongoDB	11 ± 0	21.33 ± 0.47	34.66 ± 0.47	144.33 ± 0.47
Redis	1.33 ± 0.18	6.67 ± 0.90	11.81 ± 0.19	59.05 ± 0.96
Cassandra	1.52 ± 0.08	7.64 ± 0.41	13.54 ± 0.25	67.71 ± 1.27
MySQL	3.57 ± 0.76	7.73 ± 0.04	15.18 ± 0.01	73.93 ± 0.20

Table 5. Query performance of experiment 1: query 3.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 3			
	1000 records	5000 records	10000 records	50000 records
MongoDB	157 ± 1.41	170.66 ± 2.04	186.33 ± 0.81	348.66 ± 0.94
Redis	1.80 ± 0.24	9.04 ± 1.20	16.19 ± 0.12	80.98 ± 0.62
Cassandra	1.53 ± 0.04	7.66 ± 0.24	15.06 ± 0.50	75.31 ± 2.53
MySQL	5.15 ± 0.19	10.46 ± 0.16	21.63 ± 1.18	97.78 ± 1.61

Table 6. Query performance of experiment 1: query 4.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 4			
	1000 records	5000 records	10000 records	50000 records
MongoDB	243.66 ± 0.47	260.66 ± 0.47	280.33 ± 0.47	448.66 ± 0.47
Redis	3.03 ± 0.24	15.17 ± 1.23	28.37 ± 0.43	141.86 ± 2.18
Cassandra	1.88 ± 0.28	9.41 ± 1.44	16.64 ± 0.36	83.2 ± 1.80
MySQL	3.54 ± 0.89	14.36 ± 0.56	26.62 ± 0.11	133.13 ± 7.19

Table 7. Query performance of experiment 1: query 5.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 5			
	1000 records	5000 records	10000 records	50000 records
MongoDB	301.33 ± 1.69	317 ± 0.81	336 ± 0.47	508 ± 0.81
Redis	4.17 ± 0.18	20.89 ± 0.91	40.72 ± 0.47	203.61 ± 2.37
Cassandra	2.01 ± 0.36	10.09 ± 1.84	17.75 ± 0.45	88.76 ± 2.27
MySQL	7.15 ± 2.31	19.76 ± 1.46	38.60 ± 2.65	162.65 ± 3

Table 8. Query performance of experiment 1 in larger databases: query 1.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 1			
	1,000 records	5,000 records	1,0000 records	50,000 records
MongoDB	2 ± 0	7 ± 0	13.66 ± 0.47	71.66 ± 1.24
Redis	0.16 ± 0.002	0.84 ± 0.005	1.70± 0.01	8.57 ± 0.09
Cassandra	0.15 ± 0.001	0.59 ± 0.02	1.65 ± 0.01	8.31 ± 0.05
MySQL	1.39 ± .003	5.08 ± 0.11	9.79 ± 0.10	48.50 ± 0.26
	100,000 records	1,000,000 records	10,000,000 records	100,000,000 records
MongoDB	139.8 ± 0.97	1460.8 ± 144.96	15830 ± 275.01	456720 ± 0.31
Redis	20.52 ± 0.10	179.08 ± 0.98	1984.02 ± 64.28	19493.38 ± 379.01
Cassandra	20.36 ± 0.15	173.19 ± 0.74	1977.52 ± 56.39	19381.29 ± 150.38
MySQL	96.38 ± 0.52	1336.81 ± 3.44	14168.82 ± 467.92	142600 ± 0.78

Table 9. Query performance of experiment 1 in larger databases: query 2.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 2			
	1,000 records	5,000 records	1,0000 records	50,000 records
MongoDB	6.33 ± 0.47	12 ± 0	22 ± 0	92.33 ± 0.47
Redis	0.40 ± 0.002	2.06 ± 0.04	4.16 ± 0.07	21.08 ± 0.16
Cassandra	0.39 ± 0.02	2.02 ± 0.03	4.04 ± 0.02	20.15 ± 0.06
MySQL	1.42 ± 0.01	5.25 ± 0.02	9.93 ± 0.05	49.54 ± 0.11
	100,000 records	1,000,000 records	10,000,000 records	100,000,000 records
MongoDB	195.5 ± 3.04	1943.6 ± 4.12	21751.9 ± 198.34	485420 ± 2.42
Redis	51.97 ± 0.34	426.54 ± 6.36	4389.67 ±77.62	45066.84 ± 681.53
Cassandra	50.41 ± 0.18	419.80 ± 7.92	4245.53 ± 12.39	44972.37 ± 398.27
MySQL	96.74 ± 0.42	1325.02 ± 2.26	13667.24 ± 14.44	140220 ± 0.33

Table 10. Query performance of experiment 1 in larger databases: query 3.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 3			
	1,000 records	5,000 records	1,0000 records	50,000 records
MongoDB	347 ± 3.55	349 ± 6.37	350.66 ± 14.19	478 ± 14.85
Redis	0.68 ± 0.003	3.33 ± 0.01	6.71 ± 0.09	34.04 ± 0.26
Cassandra	0.65 ± 0.01	3.31 ± 0.01	6.55 ± 0.16	32.94 ± 0.73
MySQL	1.52 ± 0.002	6.27 ± 0.006	12.64 ± 0.05	60.25 ± 0.18
	100,000 records	1,000,000 records	10,000,000 records	100,000,000 records
MongoDB	618.2 ± 24.49	3459 ± 22.85	34211.8 ± 207.30	586010 ± 0.10
Redis	70.66 ± 0.07	686.52 ± 8.60	6901.39 ± 90.29	70502.90 ± 567.23
Cassandra	70.49 ± 0.56	673.19 ± 11.73	6898.69 ± 330.39	70438.83 ± 391.94
MySQL	120.06 ± 0.57	1576.91 ± 30.99	15926.95 ± 51.91	162960 ± 0.23

Table 11. Query performance of experiment 1 in larger databases: query 4.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 4			
	1,000 records	5,000 records	1,0000 records	50,000 records
MongoDB	628.33 ± 2.49	642.66 ± 3.39	633.33 ± 9.56	780.33 ± 8.80
Redis	1.97 ± 0.004	5.94 ± 0.05	12.13 ± 0.25	60.05 ± 1.11
Cassandra	1.89 ± 0.03	5.38 ± 0.01	11.15 ± 0.09	60.01 ± 0.55
MySQL	2.76 ± 0.01	8.75 ± 0.02	16.38 ± 0.09	76.35 ± 0.38
	100,000 records	1,000,000 records	10,000,000 records	100,000,000 records
MongoDB	925.2 ± 7.88	3783.1 ± 18.14	35005.6 ± 195.54	590500 ± 0.14
Redis	135.40 ± 1.12	1218.50 ± 14.46	12766.23 ± 201.82	129661.37 ± 427
Cassandra	133.46 ± 0.24	1136.63 ± 31.82	12597.44 ± 99.71	120711.53 ± 184
MySQL	154.61 ± 0.36	1895.05 ± 7.42	19628.45 ± 489.1	196020 ± 0.08

Table 12. Query performance of experiment 1 in larger databases: query 5.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 5			
	1,000 records	5,000 records	1,0000 records	50,000 records
MongoDB	965.66 ± 3.39	1000.66 ± 0	1026.33 ± 0	1116.66 ± 0
Redis	1.85 ± 0.007	9.36 ± 0.07	18.73 ± 0.06	93.36 ± 0.14
Cassandra	1.79 ± 0.03	8.77 ± 0.01	17.15 ± 0.01	91.29 ± 0.07
MySQL	3.75 ± 0.02	11.22 ± 0.02	20.69 ± 0.07	95.62 ± 0.01
	100,000 records	1,000,000 records	10,000,000 records	100,000,000 records
MongoDB	1320 ± 32.64	4270.6 ± 103.86	36715.2 ± 974.60	610520 ± 0.09
Redis	201.18 ± 1.22	1992.25 ± 4.76	18780.90 ± 43.54	196953.04 ± 547.7
Cassandra	200.53 ± 0.09	1959.52 ± 8.35	18598.02 ± 59.42	195722.15 ± 397.1
MySQL	190.36 ± 0.58	2237.75 ± 4.64	22983.58 ± 26.57	232160 ±0.55

Timing Scalability

The scalability of the database approaches, allowing more patient records to be processed in a smaller amount of time as expected, is described per query number in Figures 6-15.

As shown in Figures 6 and 7, in query 1, Cassandra and Redis took the least time to scale from 1,000 to 50,000 records. Specifically, in around 42 ms, they scaled to the largest database, which is almost 2 times faster than the second fastest technology. As shown in Figures 8 and 9, in query 2, Cassandra and Redis too the least time to scale to the largest database. Specifically, it took 57.72ms to scale from 1,000 to 50,000 records, which was slightly faster than Cassandra, which needed 66.19ms. As shown in Figures 10 and 11, in query 3, Cassandra was the fastest technology to scale to the largest number of records. It needed 73.78ms to scale to 50,000 records, making it slightly faster than Redis at 79.18ms. As shown in Figures 12 and 13, in query 4, Cassandra store was the fastest technology to reach the largest database; it took 81.32ms, slightly higher than MySQL and Redis, which took 129.59ms and 138.83ms, respectively. Finally, as shown in Figures 14 and 15, in query 5, Cassandra store was again the fastest technology to scale from 10,000 to 50,000 records. It took 86.75ms to scale to the largest database, making it slightly faster than Redis and MySQL, which took 199.44ms and 154.85ms.

As shown in Figures 16 and 17, in query 1, Cassandra and Redis took the least time to scale from 1,000 to 100,000,000 records. Specifically they took 19493.22 ms and 19381.14 ms, to scale to the largest database. As shown in Figures 18 and 19, in query 2, Cassandra and Redis took the least time to scale to the largest database. Specifically, they took 45066.44 and 44971.98 ms, respectively, to scale from 1,000 to 100,000,000 records. As shown in Figures 20 and 21, in query 3, Cassandra and Redis were the fastest technology to scale to the largest number of records. They needed 70502.22 and 70438.18 ms to scale from 1,000 to 100,000,000 records. As

shown in Figures 22 and 23, in query 4, Cassandra and Redis were the fastest technology to reach the largest database; they took 129659.4 and 120709.64 ms, respectively. Finally, as shown in Figure 24 and 25, in query 5, Cassandra and Redis were again the fastest technology to scale from 1,000 to 100,000,000 records. They took 196951.19 and 195720.36 ms, respectively, to scale to the largest database.

4.2.1.2 Summary

Using standard computing resources in four DBMS, the query speed for the MongoDB was slower for all queries and database sizes as expected. Figures 6-15 reveal that, for the simple queries, the performances of Cassandra, Redis and MySQL were similar when the size of the database was small but results diverged significantly as the number of records increased to 10,000 records and beyond, where the query speed of Cassandra and Redis became highlighted. Similar results were obtained by using supercomputing resources (Figures 16-25), Cassandra and Redis, showed the lowest query times in most of the queries tested, showing high performance in larger databases, specially those that had 100 million patients. These two technologies also display high scalability from 1,000 to 100 million patients, requiring less query time than the other two technologies.

The performance of Cassandra and Redis was significantly faster than MongoDB for large databases. The timing among these three NoSQL approaches was similar for more complex queries; using standard computing resources, the query time of Cassandra and Redis for the 50,000-record database and the most complex query was at least 2 times faster than MongoDB (Document NoSQL approach) and almost 2 times faster than MySQL (SQL approach). Comparably, using supercomputing resources, the query time of Cassandra and Redis for the 100 million-record database and the most complex query was more than 3 times faster than the

MongoDB and slightly faster than MySQL, even though both, Cassandra and Redis had significantly more keys to process because of its default design. This is probably because recent database technologies (NoSQL approaches) have been developed to handle a higher number of keys to filter and find specific information from large amounts of data.

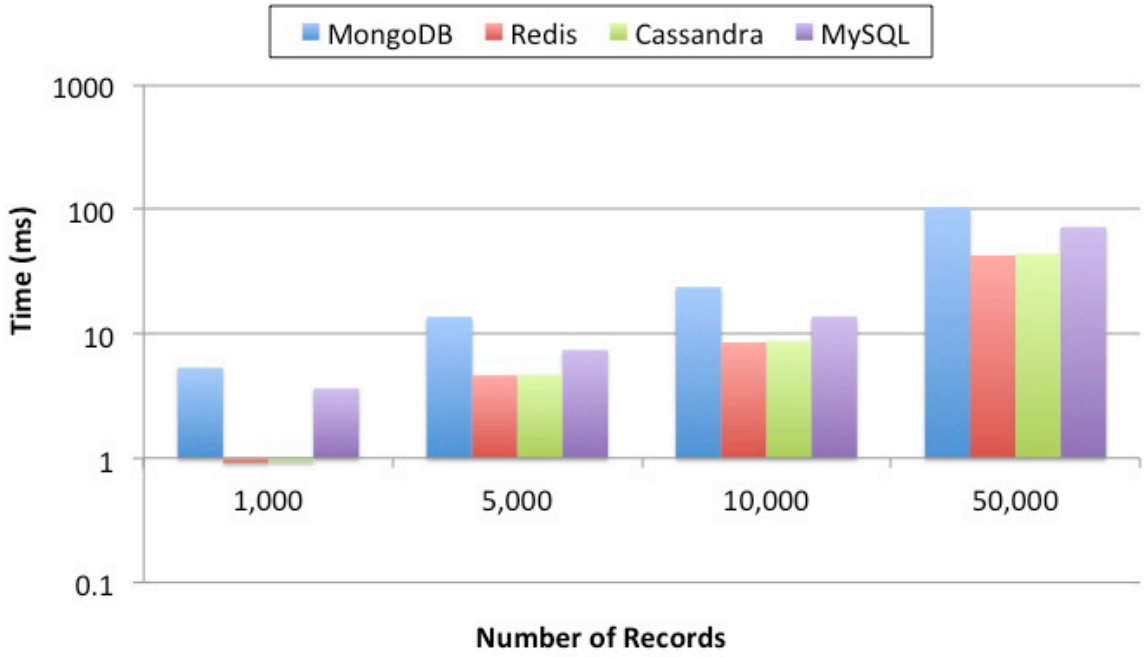

Figure 6. Variation in query time (base-10 log scale) of experiment 1: query 1.

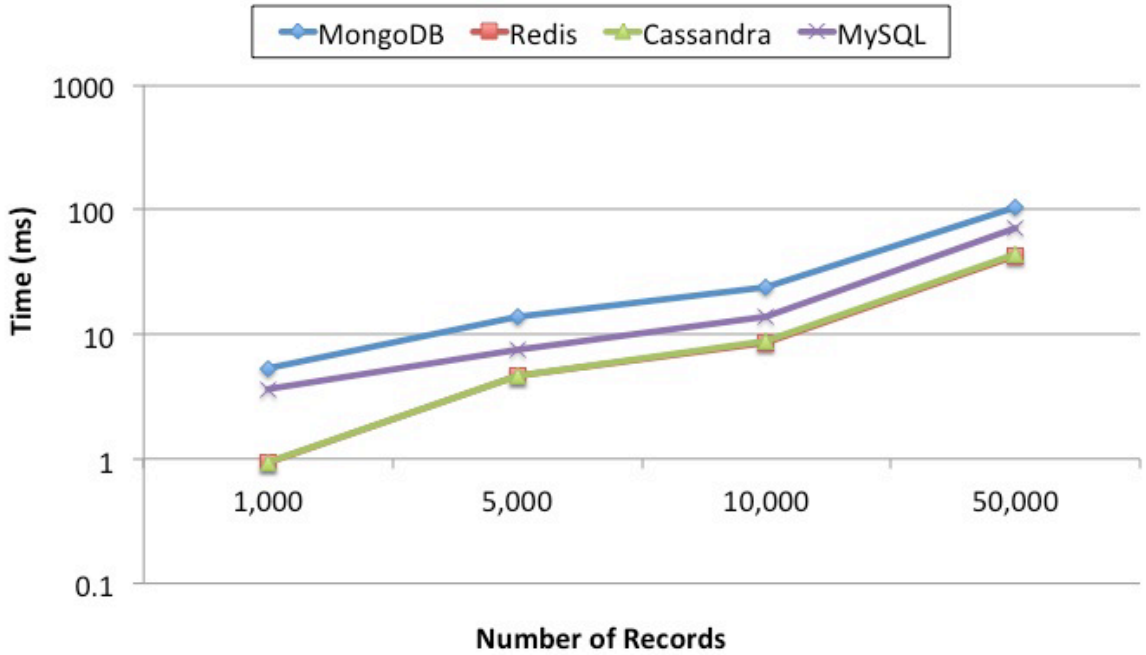

Figure 7. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 1: query 1.

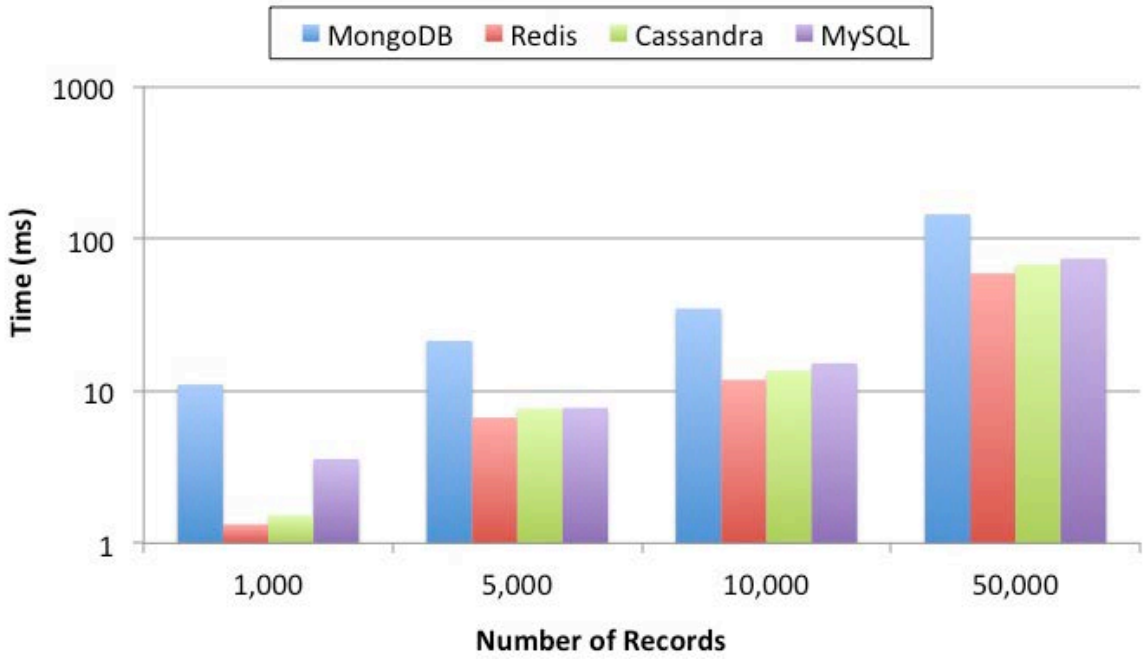

Figure 8. Variation in query time (base-10 log scale) of experiment 1: query 2.

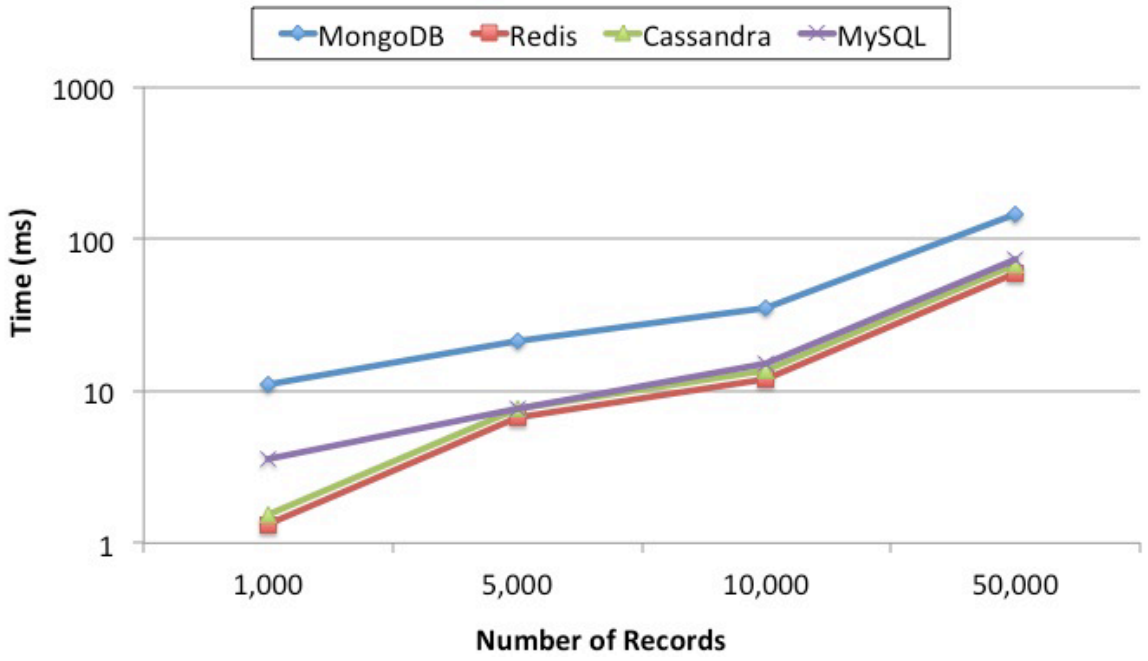

Figure 9. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 1: query 2.

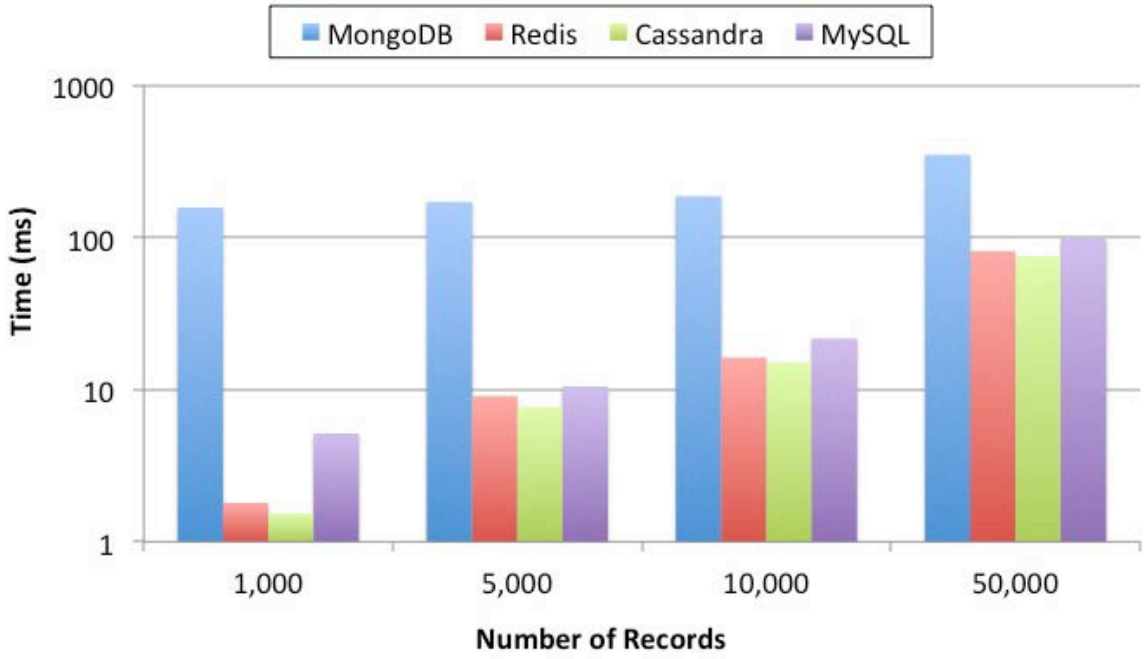

Figure 10. Variation in query time (base-10 log scale) of experiment 1: query 3.

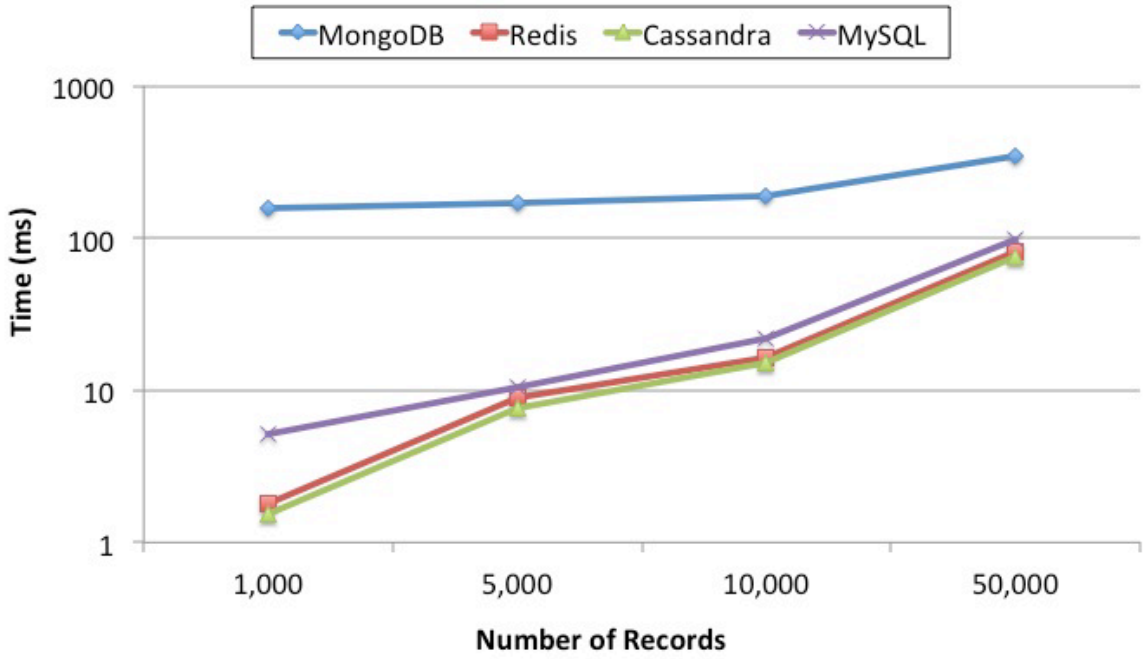

Figure 11. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 1: query 3.

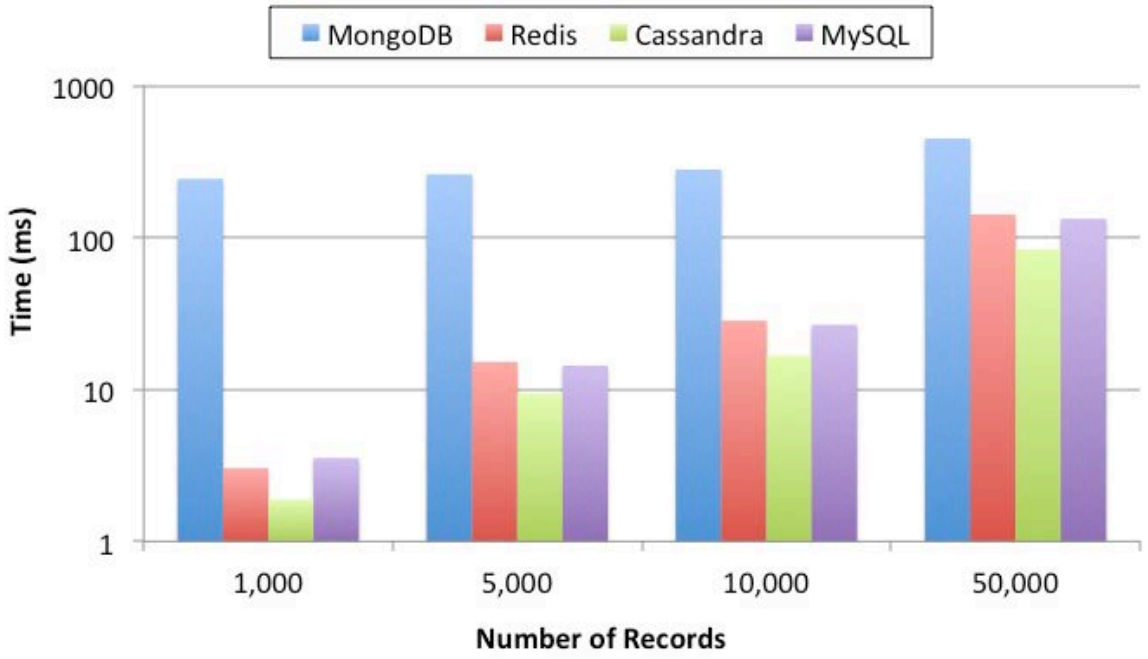

Figure 12. Variation in query time (base-10 log scale) of experiment 1: query 4.

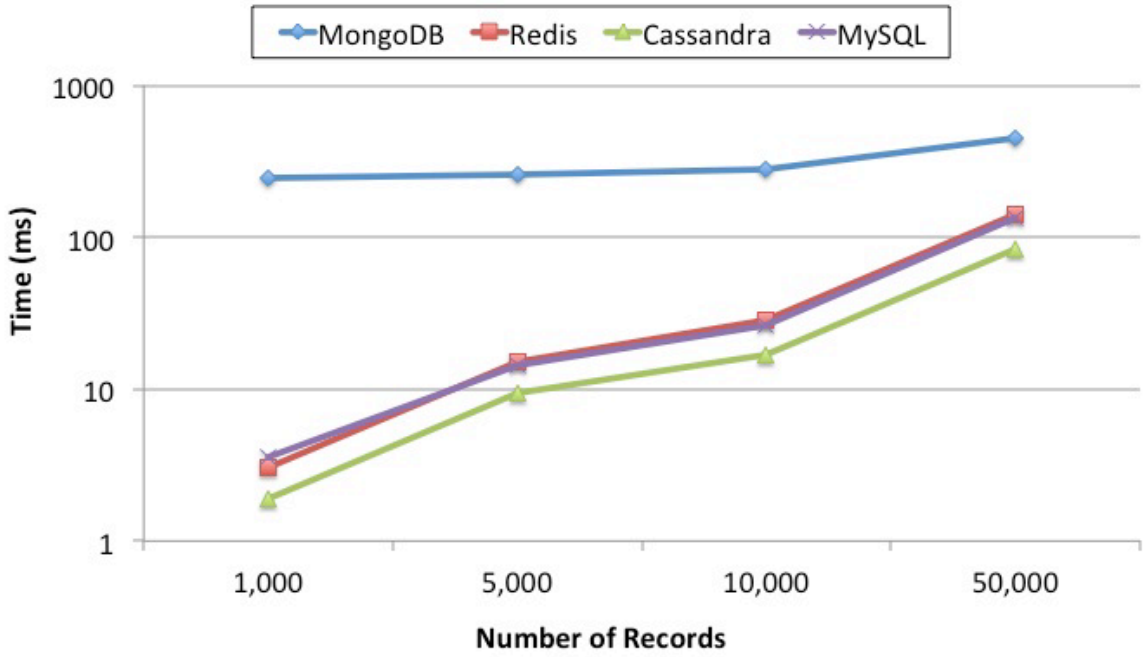

Figure 13. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 1: query 4.

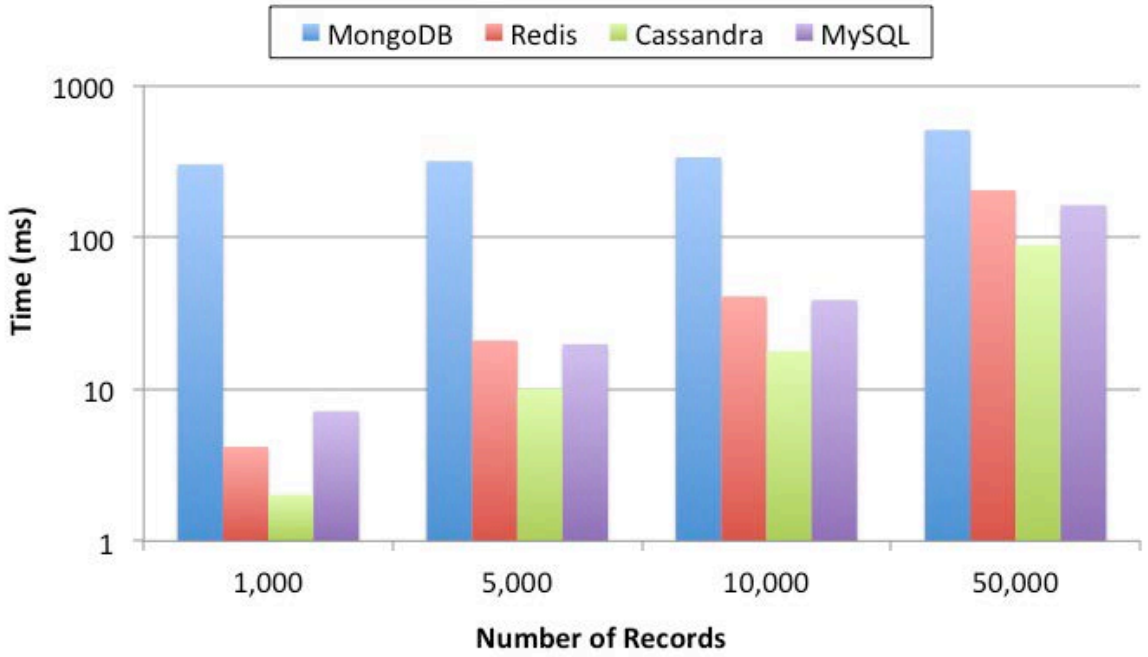

Figure 14. Variation in query time (base-10 log scale) of experiment 1: query 5.

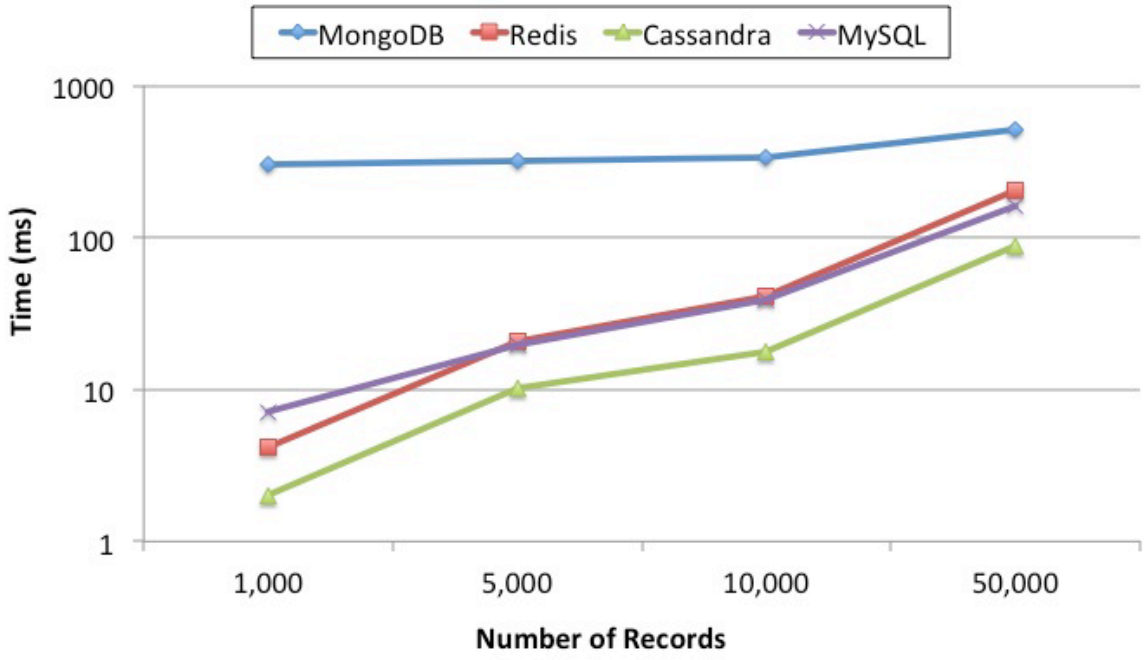

Figure 15. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 1: query 5.

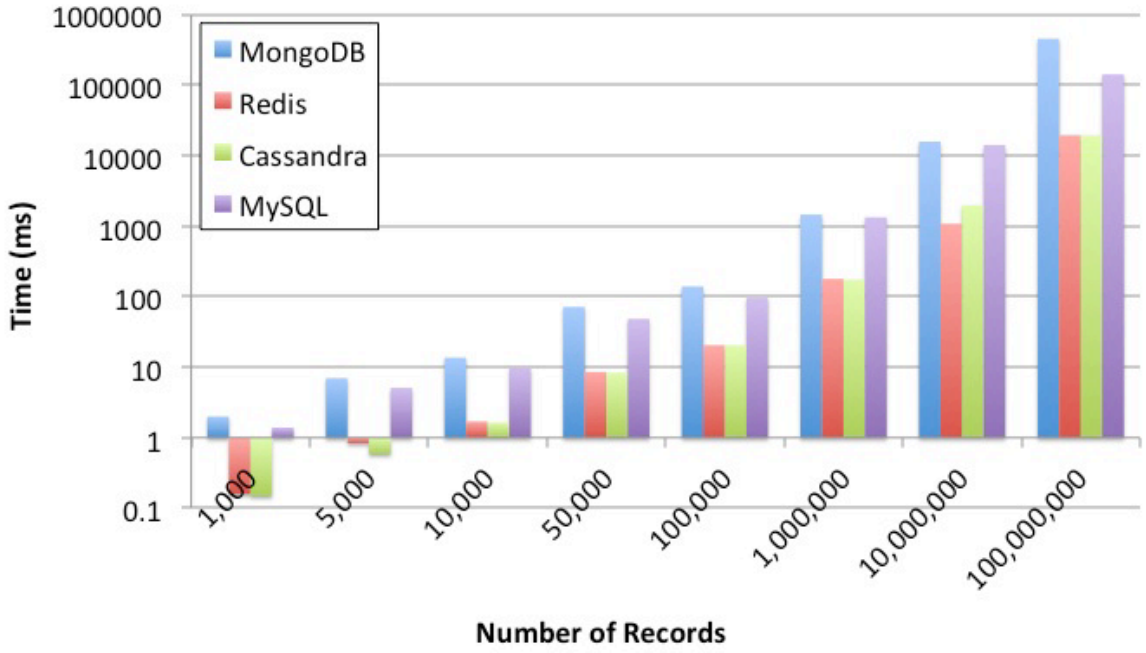

Figure 16. Variation in query time (base-10 log scale) of experiment 1 in larger databases: query 1.

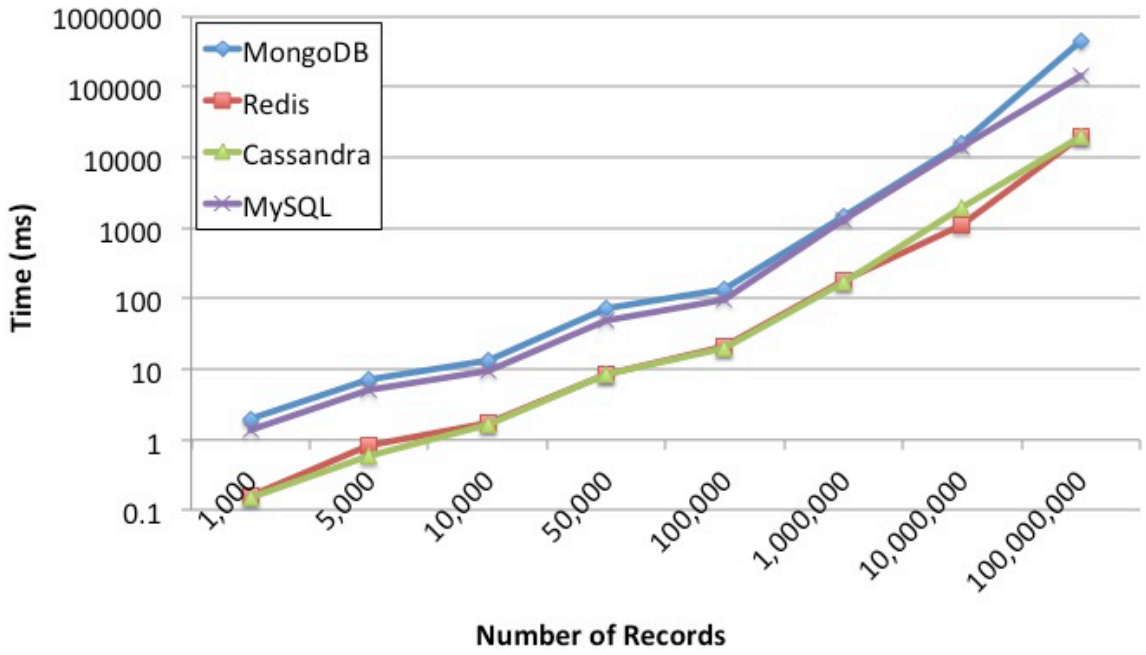

Figure 17. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 1 in larger databases: query 1.

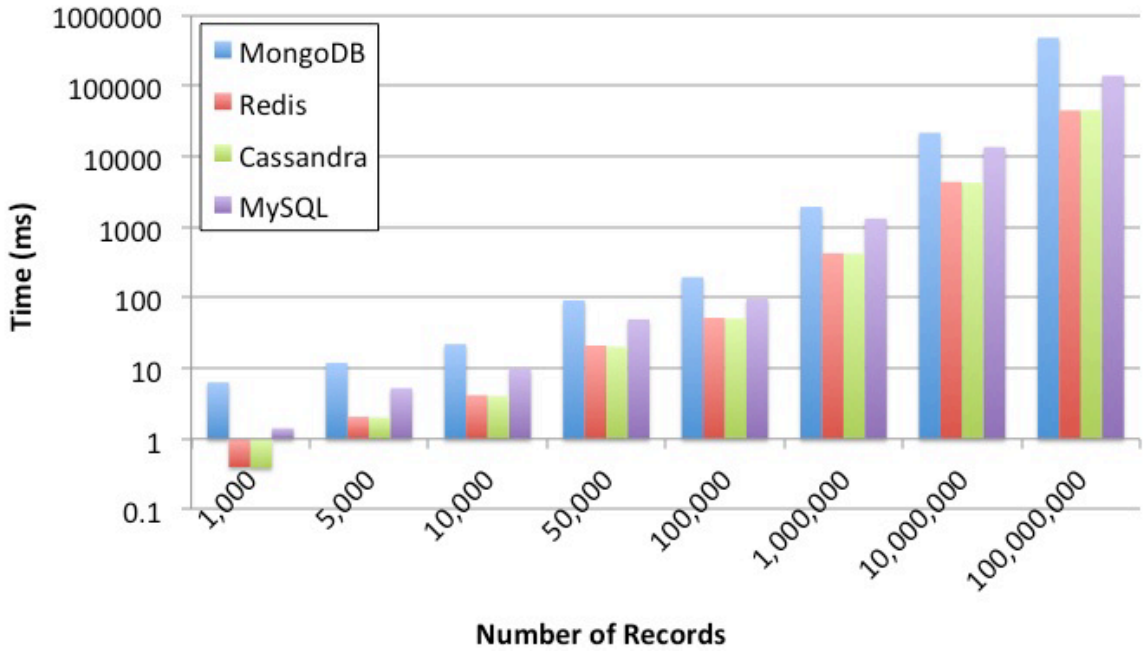

Figure 18. Variation in query time (base-10 log scale) of experiment 1 in larger databases: query 2.

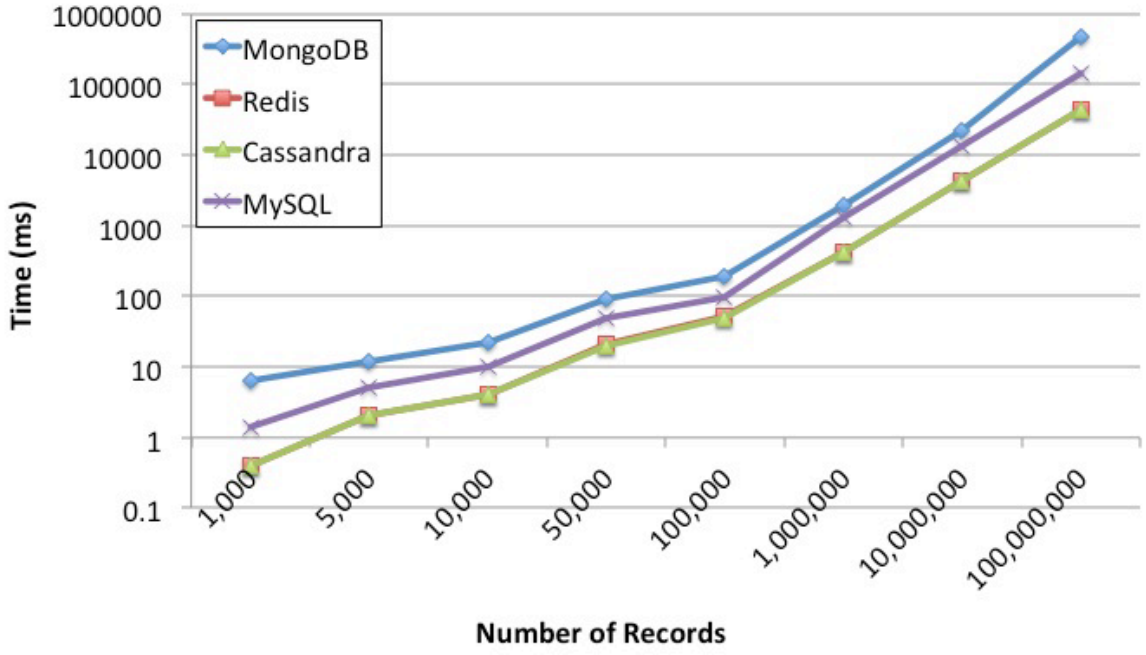

Figure 19. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 1 in larger databases: query 2.

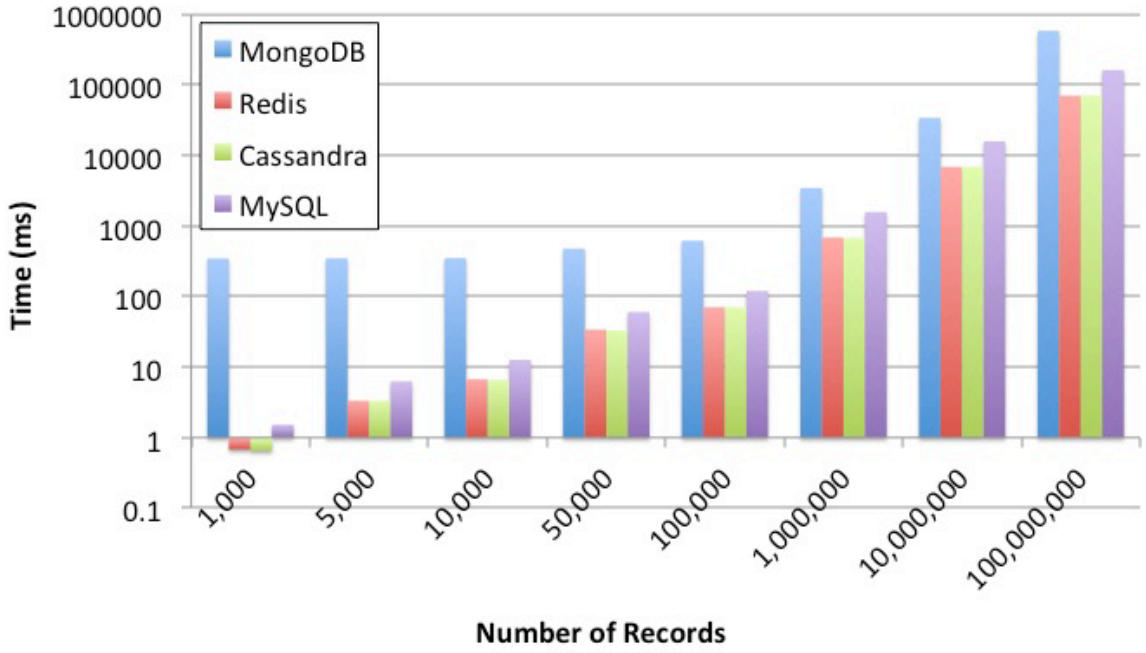

Figure 20. Variation in query time (base-10 log scale) of experiment 1 in larger databases: query 3.

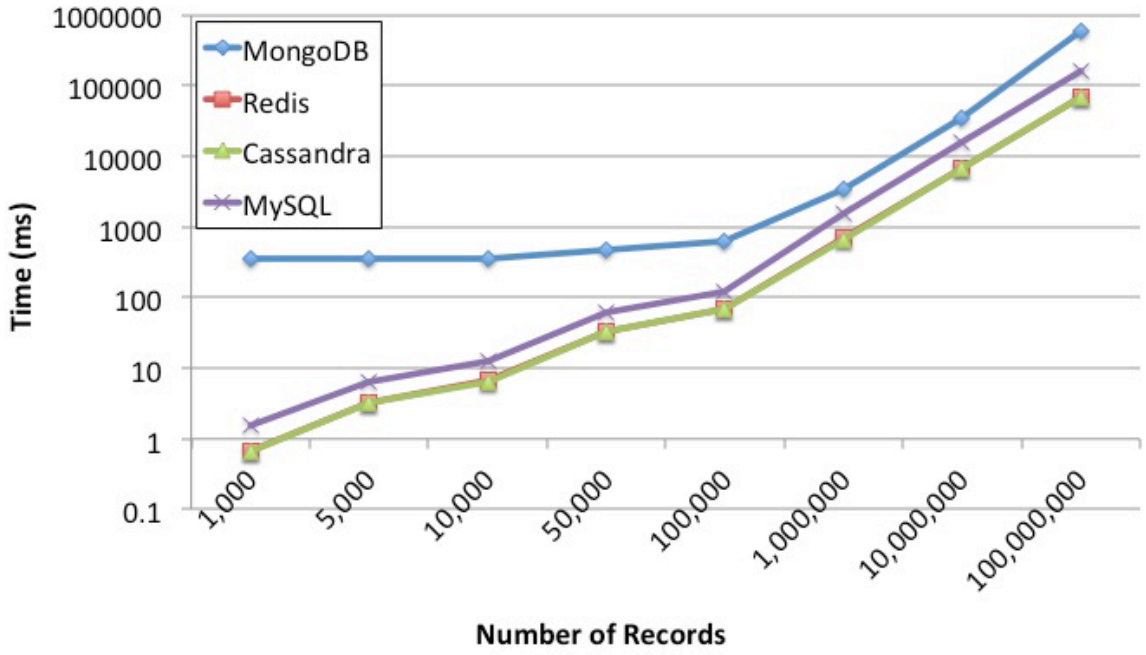

Figure 21. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 1 in larger databases: query 3.

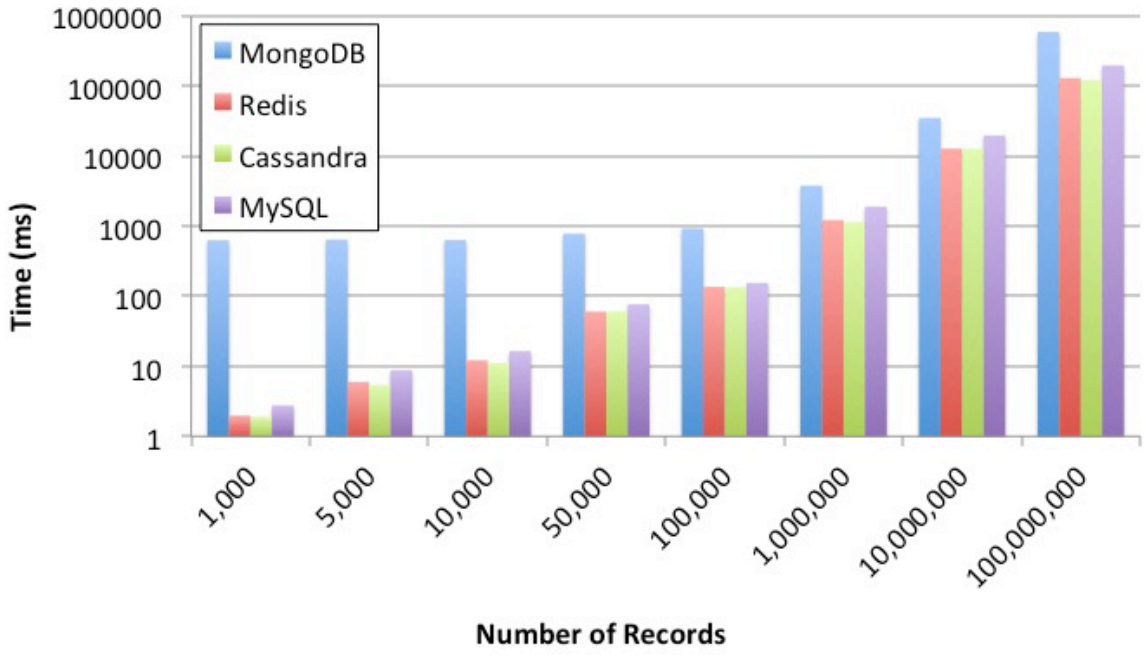

Figure 22. Variation in query time (base-10 log scale) of experiment 1 in larger databases: query 4.

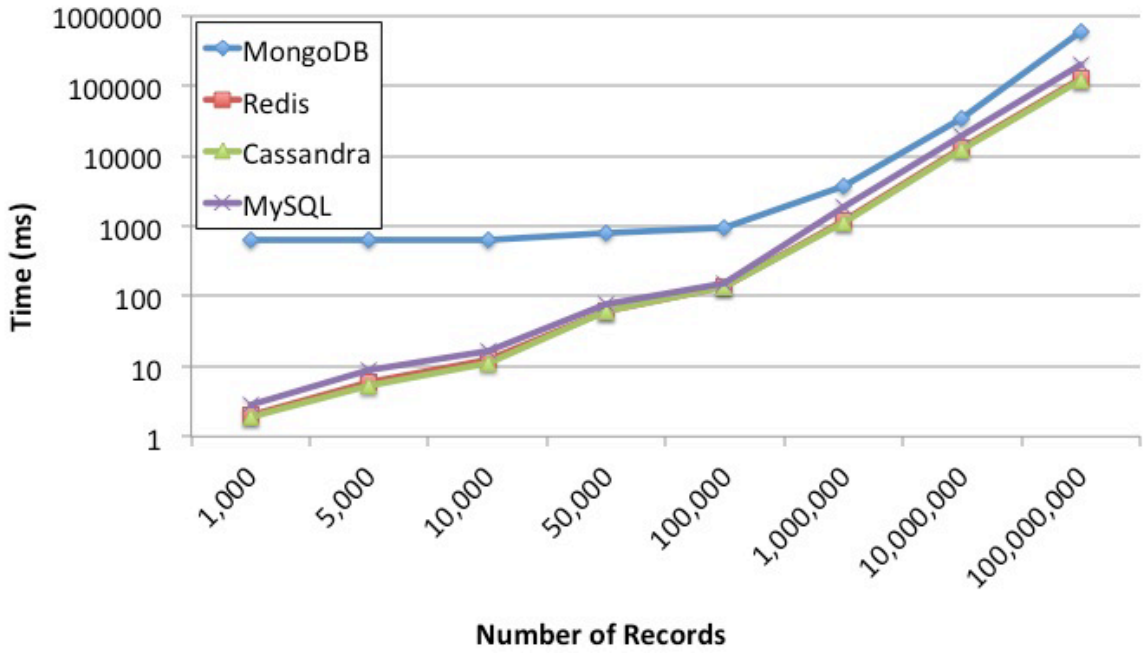

Figure 23. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 1 in larger databases: query 4.

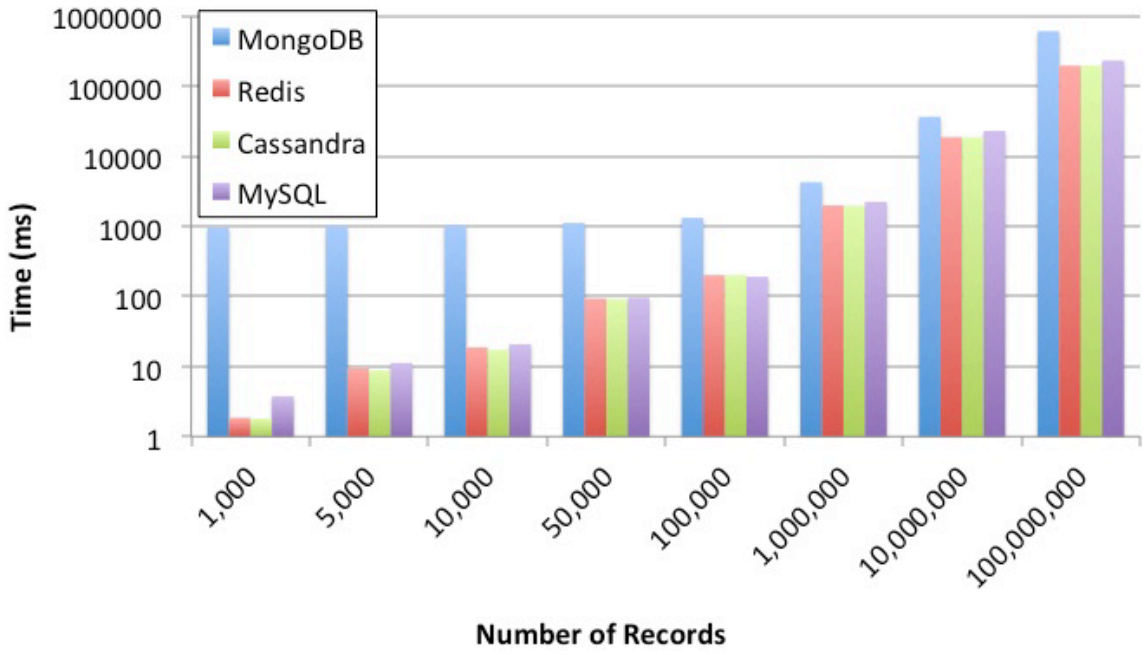

Figure 24. Variation in query time (base-10 log scale) of experiment 1 in larger databases: query 5.

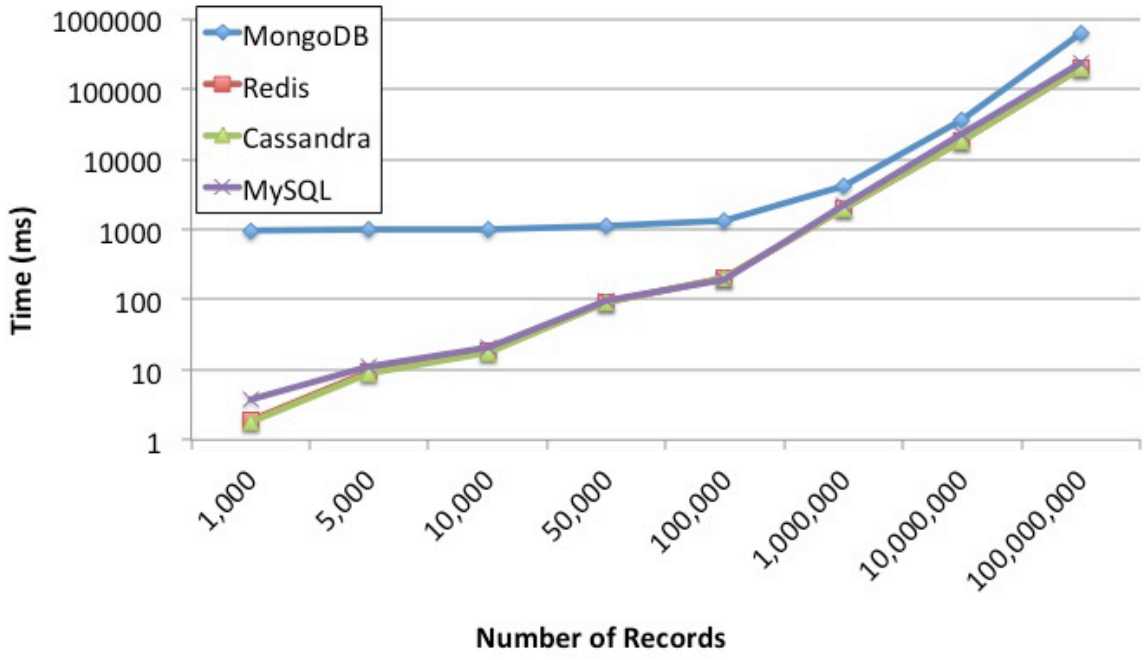

Figure 25. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 1 in larger databases: query 5.

5.0 SCALABILITY ON UPDATING WITHOUT SCHEMA CHANGES

5.1 INTRODUCTION

Precision Medicine is characterized by its continuous updating of information. Clinical and genomic data are one such type of information. As genomic analyses are based on traditional public databases that are changing every time new knowledge is produced, an efficient database management system for this type of data should be able to handle update processes easily. NoSQL technologies are characterized by their design to easily handle volatile data and frequently updated information. One way to study these features and how well NoSQL technologies handle update processes is to design experiments where we can keep database sizes constant, but vary update processes, and measure their query performance and scalability.

In the following experiment, we study the scalability of our previously selected database approaches to identify the one that can best adapt itself to the challenges in managing clinical and genomic data.

In the following sub-sections we describe the details of our experiment comparing performance and scalability using our previous queries with varied complexity and an updating process that does not modify the database schema. In sub-section 5.2.1 we briefly present the results of the experiment for performance and scalability. Also, we describe the resulted scalability followed by a summary with our conclusions.

5.2 EXPERIMENT 2: COMPARING SCALABILITY ON UPDATING WITHOUT SCHEMA CHANGES

To identify the database approaches with higher scalability in database updates (modifications without changing the schema), we adapted the experiment used in Labrinidis et al. “Scaling the maintenance workload.” In uploading new data, the databases were updated in 10% of the records, with all records corresponding to the same headers or already predefined information. Updates consisted of specific modifications of the database (updating demographic, diagnostic, treatment and genomic patient’s information). The updating process modified the database size. However, it did not modify the number of records of each pre-designed database, since we were not adding new patients, just modifying the information of patients who were already included. We ran each query in a manner similar to the last experiment, but using the updated databases. Once query times were obtained, we created tables that laid out the results, thereby facilitating comparisons among database approaches. To identify the approaches with higher performance in the updated database for each of the queries and databases of each size, we selected the database approach that resulted in the lowest query time for each set, as described in workflow experiment 2 shown in Figure 26. All of these calculations are described in section §5.2.1.1 Timing Performance. To identify the database approach with highest scalability in the updated database, we used the query times obtained above from measuring performance, but calculating, for each database approach, the difference in query times for the 50,000-record database minus query times from the 1,000-record database, selecting the database approach with the lowest resulting query time. All these calculations are described in section §5.2.1.1 Timing Scalability.

Experiment 2 was repeated but in even larger databases. These databases consisted of 1,000, 5,000, 10,000, 50,000, 100,000, 1 million, 10 millions and 100 millions records. For

experiment 2, we used the same four database technologies: Document-MongoDB, Key-Value-Redis, Column-Cassandra and Table-MySQL. Performance and Scalability results are annotated in Tables and Graphs, and described in section §5.2.1.1, Timing Performance and Timing Scalability, respectively.

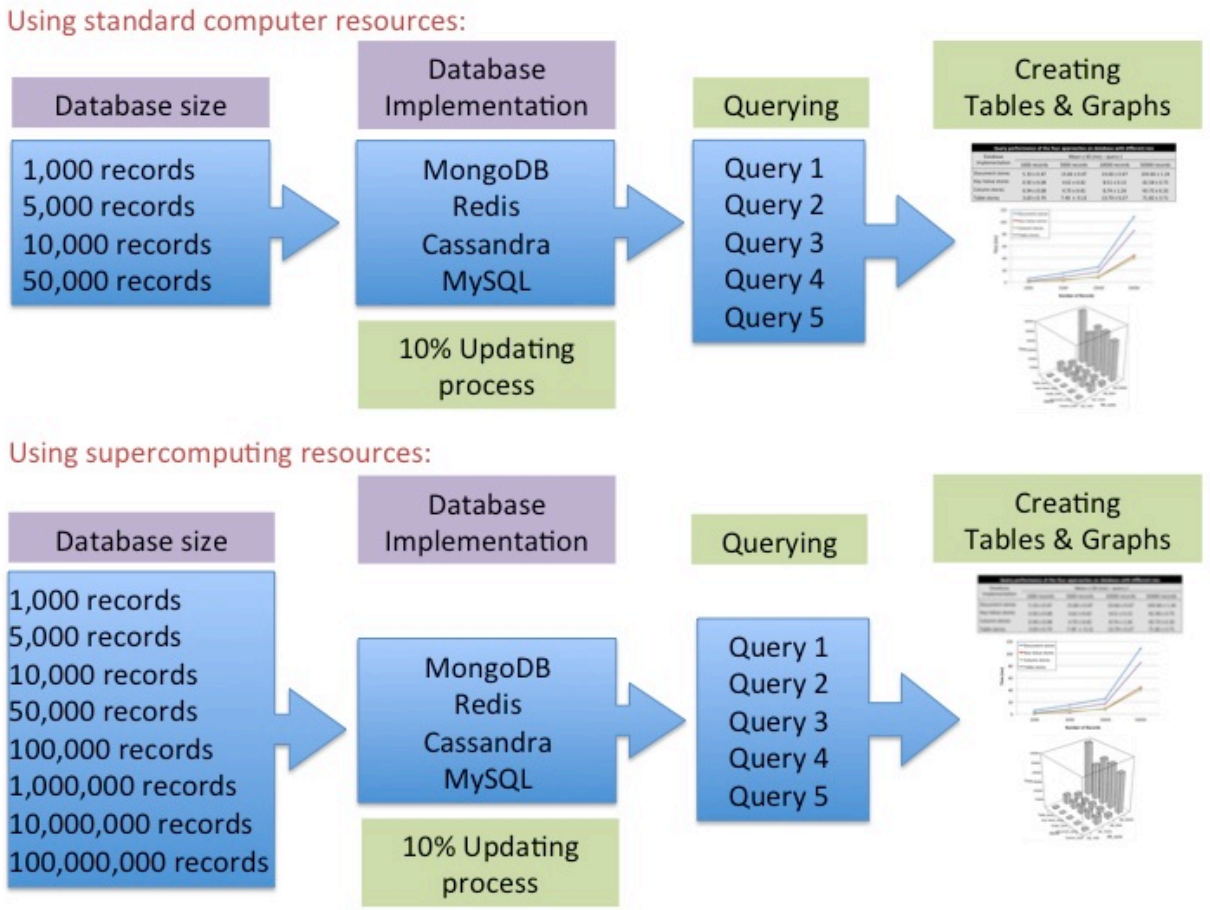

Figure 26. Workflow of Experiment 2.

5.2.1 Experimental results

Query times for the four DBMS were evaluated by making five different queries with varied complexity of the updated databases of different sizes, as shown in Figure 26. The variation in query time in the database updating process was also studied. For each of the four database

approaches, the time taken to make the queries with varying complexity, as described above, was measured in databases containing 1000, 5000, 10,000 and 50,000 patient records. In addition, using the same methodology but supercomputing resources, the query times of the four DBMS were measured for the same and even larger databases, with 100,000, 1,000,000 and 10,000,000 and 100,000,000 patient records, as shown in Figure 26. Finally, for each of the database sizes mentioned above, each query was made 3 times to calculate the average query time and the standard deviation (SD). The query times are given in Tables 13-22.

5.2.1.1 Query time results for queries of varying complexity in databases of different sizes

Timing Performance

The timing performances of the simplest query are given in Table 13. For query 1 in the 1,000-record DB, the Cassandra and Redis offered the smallest query times. They were almost two times faster than MySQL and more than six times faster than MongoDB. In the 5,000-record DB, Cassandra and Redis were the fastest technologies. Both technologies were around 2 times faster than MySQL and around 3 to 4 times faster than MongoDB. In the 10,000-record DB, the Cassandra and Redis offered the smallest query time. They were around two times faster than the MySQL and MongoDB. Finally, in the 50,000-record DB, the Cassandra was fastest technology. It was slightly faster than Redis and almost 2 times faster than MySQL and almost 3 times faster than MongoDB.

The timing performances for query 2 are given in Table 14. For query 2, in the 1,000-record database, Cassandra, Redis and MySQL came back with similar query times. They were around 7 to 9 times faster than MongoDB. In the 5,000-record DB, Redis had the smallest query time. It was slightly faster than the Cassandra and MySQL, and almost 4 times faster than

MongoDB. In the 10,000-record DB, Cassandra and Redis were the faster technologies. They were slightly faster than MySQL and around three times faster than MongoDB. In the 50,000-record DB, the Cassandra and Redis were fastest technologies. They were slightly faster than MySQL and more than two times faster than MySQL.

The timing performances for query 3 are given in Table 15. For query 3, in the 1,000-record DB, Cassandra and Redis resulted in the smallest query times. They were slightly faster than MySQL and 96 to 107 times faster than MongoDB. In the 5,000-record DB, the Cassandra and Redis were the fastest technologies. They were slightly faster than MySQL and around 21 to 23 times faster than MongoDB. In the 10,000-record DB, Cassandra and Redis were the fastest technologies. They were slightly faster than MySQL and around 12 times faster than MongoDB. In the 50,000-record DB, Cassandra offered the fastest time. It was slightly faster than the Redis and MySQL and more than 4 times faster than MongoDB.

The timing performances for query 4 are given in Table 16. For query 4, in the 1,000-record DB, Cassandra was the fastest technology. It was slightly faster than the Redis and MySQL, and 134 times faster than MongoDB. In the 5,000-record DB, Cassandra had the smallest query time. It was slightly faster than Redis, almost 2 times faster than MySQL and almost 29 times faster than MongoDB. In the 10,000-record DB, Cassandra resulted in the smallest query time. It was slightly faster than Redis and MySQL and more than 17 times faster than MongoDB. In the 50,000-record DB, Cassandra was the fastest technology. It was almost 2 times faster than Redis and MySQL and more than 5 times faster than MongoDB.

The timing performances for query 5 are given in Table 17. For query 5, in the 1,000-record DB, Cassandra had the smallest query time. It was almost 2 times faster than Redis and MySQL and more than 147 times faster than MongoDB. In the 5,000-record DB, Cassandra was

the fastest technology. It was slightly faster than the Redis and MySQL and more than 24 times faster than MongoDB. In the 10,000-record DB, Cassandra resulted in the smallest query time. It was more than two times faster than MySQL and Redis and more than 19 times faster than MongoDB. In the 50,000-record DB, Cassandra was the fastest technology. It was around two times faster than MySQL and Redis and more than 5 times faster than MongoDB.

The timing performances for the simplest query are given in Table 18. For query 1 in the database size of 1,000 records, Cassandra and Redis showed almost the same query times. In fact, Cassandra and Redis obtained results more than 7 and 16 times faster than MySQL and MongoDB, respectively. In the 5,000-record database, again Cassandra and Redis were the fastest technologies. In this database size, Cassandra and Redis were more than 5 and 8 times faster than MySQL and MongoDB, respectively. In the 10,000-record database, Cassandra and Redis again had almost the same query times, and both technologies were more than 5 and 7 times faster than MySQL and MongoDB, respectively. In the 50,000-record database, Cassandra and Redis had similar query times, respectively. In this database size, Cassandra and Redis were more than 5 and 8 times faster than MySQL and MongoDB, respectively. In the 100,000 records, Cassandra and Redis were the fastest technologies. In fact, they were more than 5 and 7 times faster than MySQL and MongoDB, respectively. In the 1-million records database, again Cassandra and Redis were the fastest technologies. For this database size, they were more than 6 and 7 times faster than MySQL and MongoDB. In the 10 million-records, Cassandra and Redis again were the fastest technologies. Both technologies were again more than 8 and 6 times faster than MySQL and MongoDB. In 100 million records, Cassandra and Redis were the fastest technologies. In fact, they were more than 5 and 19 times faster than MySQL and MongoDB, respectively.

The timing performances for query 2 are given in Table 19. For this query in the database size of 1,000 records, Cassandra and Redis showed almost the same query times. In fact, Cassandra and Redis obtained results more than 3 and 16 times faster than MySQL and MongoDB, respectively. In the 5,000-record database, again Cassandra and Redis were the fastest technologies. In this database size, Cassandra and Redis were more than 2 and 5 times faster than MySQL and MongoDB, respectively. In the 10,000-record database, Cassandra and Redis again had almost the same query times, and both technologies were more than 2 and 4 times faster than MySQL and MongoDB, respectively.-In the 50,000-record database, Cassandra and Redis had similar query times, respectively. In this database size, Cassandra and Redis were more than 2 and 4 times faster than MySQ and MongoDB, respectively. In 100,000 records, Cassandra and Redis were the fastest technologies. In fact, they were slightly faster than MySQL and more than 3 times faster than MongoDB, respectively. In the 1-million records database, again Cassandra and Redis were the fastest technologies. For this database size, they were more than 2 and 4 times faster than MySQL and MongoDB. In 10 million-records, Cassandra and Redis again were the fastest technologies. Both technologies were again more than 6 and 10 times faster than MySQL and MongoDB. In 100 million records, Cassandra and Redis were the fastest technologies. In fact, they were more than 2 and 9 times faster than MySQL and MongoDB, respectively.

The timing performances for query 3 are given in Table 20. For this query in the database size of 1,000 records, Cassandra and Redis showed almost the same query times. In fact, Cassandra and Redis obtained results more than 2 and 543 times faster than MySQL and MongoDB, respectively. In the 5,000-record database, again Cassandra and Redis were the fastest technologies. In this database size, Cassandra and Redis were slightly faster than MySQL

and more than 105 times faster than MongoDB, respectively. In the 10,000-record database, Cassandra and Redis again had almost the same query times, and both technologies were slightly faster than MySQL and more than 55 times faster than MongoDB, respectively. Finally, in the 50,000-record database, Cassandra and Redis had similar query times, respectively. In this database size, Cassandra and Redis were slightly faster than MySQL and more than 14 times faster than MongoDB, respectively. In 100,000 records, Cassandra and Redis were the fastest technologies. In fact, they were slightly faster than MySQL and more than 7 times faster than MongoDB, respectively. In the 1-million records database, again Cassandra and Redis were the fastest technologies. For this database size, they were more than 2 and 5 times faster than MySQL and MongoDB. In 10 million-records, Cassandra and Redis again were the fastest technologies. Both technologies were again more than 2 and 5 times faster than MySQL and MongoDB. In 100 million records, Cassandra and Redis were the fastest technologies. In fact, they were slightly faster than MySQL and 6 times faster than MongoDB.

The timing performances for query 4 are given in Table 21. For this query in the database size of 1,000 records, Cassandra and Redis showed almost the same query times. In fact, Cassandra and Redis obtained results 2 and 532 times faster than MySQL and MongoDB. In the 5,000-record database, again Cassandra and Redis were the fastest technologies. In this database size, Cassandra and Redis were slightly faster than MySQL and more than 116 times faster than MongoDB. In the 10,000-record database, Cassandra and Redis again had almost the same query times, and both technologies were slightly faster than MySQL and more than 51 times faster than MongoDB. Finally, in the 50,000-record database, Cassandra and Redis had similar query times. In this database size, Cassandra and Redis were slightly faster than MySQL and more than 12 times faster than MongoDB. In 100,000 records, Cassandra and Redis were the fastest

technologies. In fact, they were slightly faster than MySQL and more than 6 times faster than MongoDB. In the 1-million records database, again Cassandra and Redis were the fastest technologies. For this database size, they were slightly faster than MySQL and 3 times faster than MongoDB. In 10 million-records, Cassandra and Redis again were the fastest technologies. Both technologies were again slightly higher than MySQL and 3 times faster than MongoDB. In 100 million records, Cassandra and Redis were the fastest technologies. In fact, they were slightly faster than MySQL and more than 4 times faster than MongoDB.

The timing performances for query 5 are given in Table 22. For this query in the database size of 1,000 records, Cassandra and Redis showed almost the same query times. In fact, Cassandra and Redis obtained results 2 and 524 times faster than MySQL and MongoDB. In the 5,000-record database, again Cassandra and Redis were the fastest technologies. In this database size, Cassandra and Redis were slightly faster than MySQL and more than 110 times faster than MongoDB. In the 10,000-record database, Cassandra and Redis again had almost the same query times, and both technologies were slightly faster than MySQL and more than 57 times faster than MongoDB. Finally, in the 50,000-record database, Cassandra, Redis and MySQL had similar query times. In this database size, they were more than 12 times faster than MongoDB. In 100,000 records, MySQL were the fastest technology. In fact, it was slightly faster than Cassandra and Redis and more than 6 times faster than MongoDB. In the 1-million records database, Cassandra and Redis were the fastest technologies. For this database size, they were slightly faster than MySQL and 2 times faster than MongoDB. In 10 million-records, Cassandra and Redis again were the fastest technologies. Both technologies were again slightly higher than MySQL and 2 times faster than MongoDB. In 100 million records, Cassandra and Redis were

the fastest technologies. In fact, they were slightly faster than MySQL and more than 2 times faster than MongoDB.

Table 13. Query performance of experiment 2: query 1.

Query performance of four DBMS using updated (10% total) databases with different size				
Database Implementation	Mean ± SD (ms) – query 1			
	1000 records	5000 records	10000 records	50000 records
MongoDB	5.66 ± 0.47	15 ± 0	25.33 ± 0.47	108.33 ± 0.94
Redis	0.85 ± 0.04	3.35 ± 1.10	8.75 ± 0.25	43.76 ± 1.26
Cassandra	0.92 ± 0.04	4.64 ± 0.23	7.93 ± 1.45	39.69 ± 7.29
MySQL	1.63 ± 0.05	8.67 ± 0.18	16.96 ± 0.16	84.85 ± 2.68

Table 14. Query performance of experiment 2: query 2.

Query performance of four DBMS using updated (10% total) databases with different size				
Database Implementation	Mean ± SD (ms) – query 2			
	1000 records	5000 records	10000 records	50000 records
MongoDB	11.66 ± 0.47	22.66 ± 0.47	37 ± 0	146.33 ± 0.94
Redis	1.18 ± 0.06	5.93 ± 0.32	12.26 ± 0.37	61.33 ± 1.86
Cassandra	1.60 ± 0.16	8.03 ± 0.81	13.60 ± 0.45	68.03 ± 2.29
MySQL	1.60 ± 0.20	8.26 ± 0.02	16.14 ± 0.06	81.01 ± 1.48

Table 15. Query performance of experiment 2: query 3.

Query performance of four DBMS using updated (10% total) databases with different size				
Database Implementation	Mean ± SD (ms) – query 3			
	1000 records	5000 records	10000 records	50000 records
MongoDB	164 ± 0.81	179.66 ± 1.24	195.66 ± 1.24	362.66 ± 2.05
Redis	1.71 ± 0.05	8.57 ± 0.27	16.62 ± 0.33	83.1 ± 1.69
Cassandra	1.53 ± 0.008	7.69 ± 0.04	15.64 ± 0.22	73.21 ± 8.11
MySQL	1.89 ± 0.40	10.89 ± 0.08	20.84 ± 0.04	107.37 ± 3.87

Table 16. Query performance of experiment 2: query 4.

Query performance of four DBMS using updated (10% total) databases with different size				
Database Implementation	Mean ± SD (ms) – query 4			
	1000 records	5000 records	10000 records	50000 records
MongoDB	246.33 ± 0.47	263 ± 0.81	285.33 ± 0.47	454.33 ± 1.69
Redis	2.87 ± 0.02	14.38 ± 0.11	28.45 ± 0.31	142.26 ± 1.56
Cassandra	1.83 ± 0.25	9.17 ± 1.25	16.62 ± 0.38	83.1 ± 1.92
MySQL	2.79 ± 0.73	17.25 ± 0.09	28.94 ± 1.15	141.40 ± 4.08

Table 17. Query performance of experiment 2: query 5.

Query performance of four DBMS using updated (10% total) databases with different size				
Database Implementation	Mean ± SD (ms) – query 5			
	1000 records	5000 records	10000 records	50000 records
MongoDB	305 ± 0.81	324.33 ± 0.47	340.66 ± 0.47	512 ± 1.63
Redis	4.02 ± 0.04	20.11 ± 0.22	40.61 ± 0.37	203.08 ± 1.86
Cassandra	2.07 ± 0.40	13.21 ± 0.11	17.75 ± 0.35	88.78 ± 1.77
MySQL	4.04 ± 0.86	18.18 ± 0.07	37.71 ± 1.53	171 ± 3

Table 18. Query performance of experiment 2 in larger databases: query 1.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 1			
	1,000 records	5,000 records	1,0000 records	50,000 records
MongoDB	3 ± 0	8.33 ± 0.47	15 ± 0	85 ± 2.44
Redis	0.18 ± 0.002	0.95 ± 0.01	1.90 ± 0.01	9.61 ± 0.15
Cassandra	0.17 ± 0.02	0.92 ± 0.01	1.89 ± 0.01	9.59 ± 0.03
MySQL	1.42 ± 0.008	5.23 ± 0.03	10.14 ± 0.006	49.58 ± 0.07
	100,000 records	1,000,000 records	10,000,000 records	100,000,000 records
MongoDB	142.2 ± 1.24	1468.9 ± 148.32	16474.8 ± 1288.67	460510 ± 0.15
Redis	19.59 ± 0.44	200.18 ± 2.55	2034.22 ± 57.44	23345.42 ± 1593.8
Cassandra	18.74 ± 0.51	198.03 ± 1.90	1998.6 ± 73.59	23200.85 ± 355.45
MySQL	98.24 ± 0.29	1347.71 ± 4.50	13874.83 ± 49.47	139350 ± 0.92

Table 19. Query performance of experiment 2 in larger databases: query 2.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 2			
	1,000 records	5,000 records	1,0000 records	50,000 records
MongoDB	7.66 ± 0.47	13.33 ± 0.47	23.33 ± 0.47	103.66 ± 1.69
Redis	0.47 ± 0.01	2.36 ± 0.10	4.82 ± 0.16	24.17 ± 0.90
Cassandra	0.44 ± 0.03	2.15 ± 0.03	4.38 ± 0.01	23.39 ± 0.01
MySQL	1.58 ± 0.008	5.36 ± 0.02	10.14 ± 0.02	50.51 ± 0.25
	100,000 records	1,000,000 records	10,000,000 records	100,000,000 records
MongoDB	206.4 ± 1.2	2123.5 ± 73.30	22669.9 ± 209.95	498500 ± 0.42
Redis	52.17 ± 1.74	497.64 ± 10.45	2249.63 ± 48.75	50530.21 ± 995.34
Cassandra	50.83 ± 2.86	490.53 ± 6.08	2238 ± 95.84	50397.37 ± 54.15
MySQL	99.97 ± 0.18	1349.61 ± 1.36	13822.16 ± 35.23	139860 ± 0.38

Table 20. Query performance of experiment 2 in larger databases: query 3.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 3			
	1,000 records	5,000 records	1,0000 records	50,000 records
MongoDB	391 ± 5.79	387.66 ± 4.02	411.33 ± 2.62	549 ± 2.94
Redis	0.72 ± 0.02	3.66 ± 0.12	7.36 ± 0.30	38.59 ± 0.74
Cassandra	0.70 ± 0.01	3.45 ± 0.04	7.15 ± 0.01	38.45 ± 0.03
MySQL	1.78 ± 0.01	6.36 ± 0.07	13.64 ± 0.01	61.21 ± 0.09
	100,000 records	1,000,000 records	10,000,000 records	100,000,000 records
MongoDB	661.9 ± 31.04	4436.2 ± 67.22	41485.7 ± 158.45	587530 ± 0.11
Redis	83.11 ± 1.94	790.03 ± 7.79	8018.96 ± 104.33	87189.46 ± 1444
Cassandra	80.84 ± 0.31	769.61 ± 10.06	7986.40 ± 30.17	85350.29 ± 15.82
MySQL	122.83 ± 0.29	1580.04 ± 2.31	16227.49 ± 14.71	162690 ± 0.63

Table 21. Query performance of experiment 2 in larger databases: query 4.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 4			
	1,000 records	5,000 records	1,0000 records	50,000 records
MongoDB	654.66 ± 3.85	719 ± 3.26	656.33 ± 2.86	814.33 ± 17.91
Redis	1.23 ± 0.01	6.29 ± 0.06	12.66 ± 0.14	63.30 ± 0.66
Cassandra	1.19 ± 0.01	6.15 ± 0.03	11.39 ± 0.01	61.73 ± 0.03
MySQL	2.85 ± 0.03	8.82 ± 0.01	17.49 ± 0.03	77.21 ± 0.05
	100,000 records	1,000,000 records	10,000,000 records	100,000,000 records
MongoDB	968.4 ± 25.48	5067.4 ± 171.78	42951.8 ± 306.76	593340 ± 0.009
Redis	140.33 ± 1.63	1283.73 ± 10.52	12940.75 ± 60.49	139245.12 ± 603.1
Cassandra	138.91 ± 0.36	1268.50 ± 13.01	12593.47 ± 26.15	135740.15 ± 139.5
MySQL	156.81 ± 0.23	1917.89 ± 1.49	19524.18 ± 16.38	196450 ± 0.44

Table 22. Query performance of experiment 2 in larger databases: query 5.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 5			
	1,000 records	5,000 records	1,0000 records	50,000 records
MongoDB	986.66 ± 4.78	1040 ± 14.76	1089.66 ± 2.05	1173.66 ± 9.39
Redis	1.88 ± 0.01	9.45 ± 0.01	18.89 ± 0.04	96.53 ± 0.92
Cassandra	1.71 ± 0.03	8.37 ± 0.01	18.09 ± 0.01	95.37 ± 0.03
MySQL	3.86 ± 0.02	12.22 ± 0.02	21.72 ± 0.04	96.91 ± 0.04
	100,000 records	1,000,000 records	10,000,000 records	100,000,000 records
MongoDB	1358.7 ± 29.58	5980.3 ± 336.44	45958.5 ± 91.39	613400 ± 0.04
Redis	195.27 ± 1.52	2124.91 ± 21.63	19465.82 ± 400.83	218099.58 ± 6529
Cassandra	189.25 ± 0.79	2002.29 ± 17.35	19371.65 ± 59.17	209359.40 ± 172.4
MySQL	192.56 ± 0.33	2268.38 ± 3.10	23096.30 ± 30.38	232790 ± 1.35

Timing Scalability

The scalability of the database approaches, allowing more patient records to be examined the in least amount of time, are described per query number in Figures 27-36.

As shown in Figures 27 and 28, in query 1, Cassandra had the lowest query times to scale from 1,000 to 50,000 records. It scaled to the largest database in 38.77ms, slightly faster than Redis, which needed 42.91ms, and more than 2 times faster than the MySQL. As shown in Figures 29 and 30, in query 2, Redis took the least time to scale to the largest database, being slightly faster than Cassandra. As shown in Figures 31 and 32, in query 3, Cassandra was the technology to scale to the largest number of records the fastest, being slightly faster than Redis. As shown in Figure 33 and 34, in query 4, Cassandra was the fastest technology; it took 81.27ms, a slightly lower time than the MySQL AND Redis. Finally, as shown in Figure 35 and 36, in query 5, scaling from 10,000 to 50,000 records, Cassandra again was the fastest technology. It took 86.71ms to scale to the largest database, half the time than Redis and MySQL that took 199.06ms and 166.96ms respectively.

As shown in Figures 37 and 38, in query 1, Cassandra and Redis took the least time to scale from 1,000 to 100,000,000 records. Specifically they took 23345.24 and 23200.68 ms, to scale to the largest database. As shown in Figures 39 and 40, in query 2, Cassandra and Redis took the least time to scale to the largest database. Specifically, they took 50529.74 and 50396.93 ms, respectively, to scale from 1,000 to 100,000,000 records. As shown in Figures 41 and 42, in query 3, Cassandra and Redis were the fastest technology to scale to the largest number of records. They needed 87188.74 and 85349.59 ms to scale from 1,000 to 100,000,000 records. As shown in Figure 43 and 44, in query 4, Cassandra and Redis were the fastest technology to reach the largest database; they took 139243.89 and 125738.96 ms, respectively. Finally, as shown in

Figure 45 and 46, in query 5, Cassandra and Redis were again the fastest technology to scale from 1,000 to 100,000,000 records. They took 218097.7 and 209357.69 ms, respectively, to scale to the largest database.

5.2.1.2 Summary

In this experiment, MongoDB also resulted in larger query times, as expected for all queries and database sizes. As shown in Figures 27-36, and similar to the last experiment, for the simple queries, the performances of Cassandra, Redis and MySQL were similar when the size of the database was small, but performances diverged significantly as the number of records was increased to 10,000 records and beyond.

Similar to the previous experiment, the performance of Cassandra and Redis was significantly better than MongoDB (the other NoSQL approach) for large number of records. The timing was notoriously lower for more complex queries and large database than the other technologies; using standard computer resources, the query time of Cassandra for the largest database and most complex query was more than 2 times faster than the other NoSQL approaches and almost 2 times faster than the SQL approach as well. By using supercomputing resources (Figures 37-46), the query time of Cassandra and Redis for the 100 million-record database and the most complex query was almost 3 times faster than MongoDB and slightly faster than MySQL, even though both, Cassandra and Redis had significantly more keys to process because of its default architecture. This is probably because recent NoSQL approaches have been designed to handle higher update process and a large number of keys to filter and find specific information from a large amount of data.

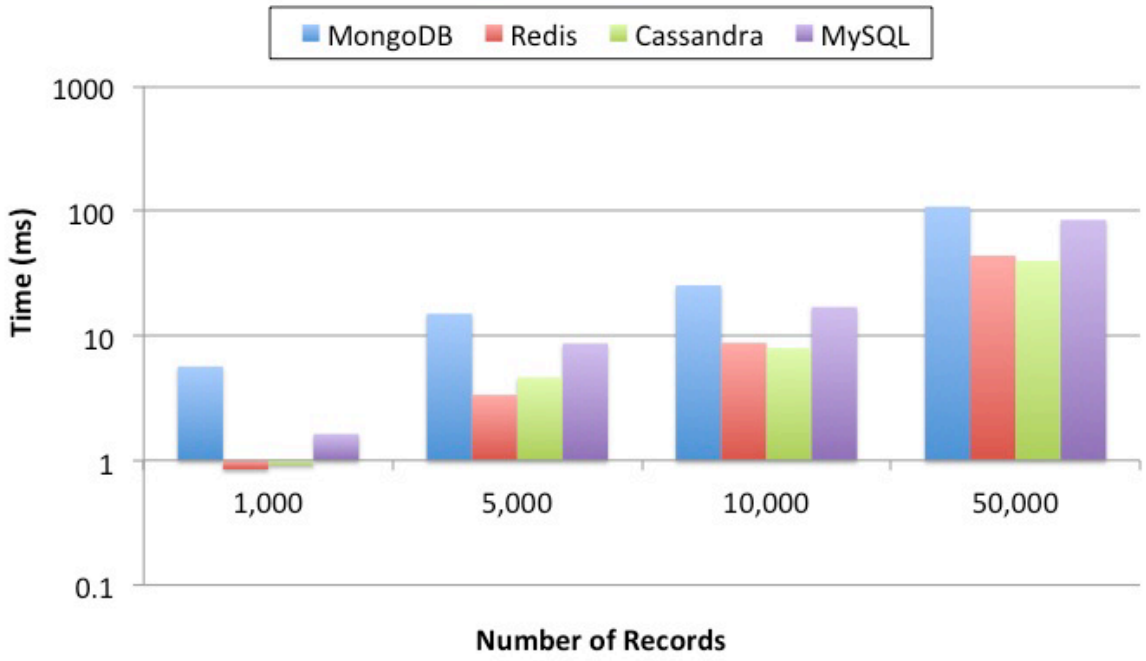

Figure 27. Variation in query time (base-10 log scale) of experiment 2: query 1.

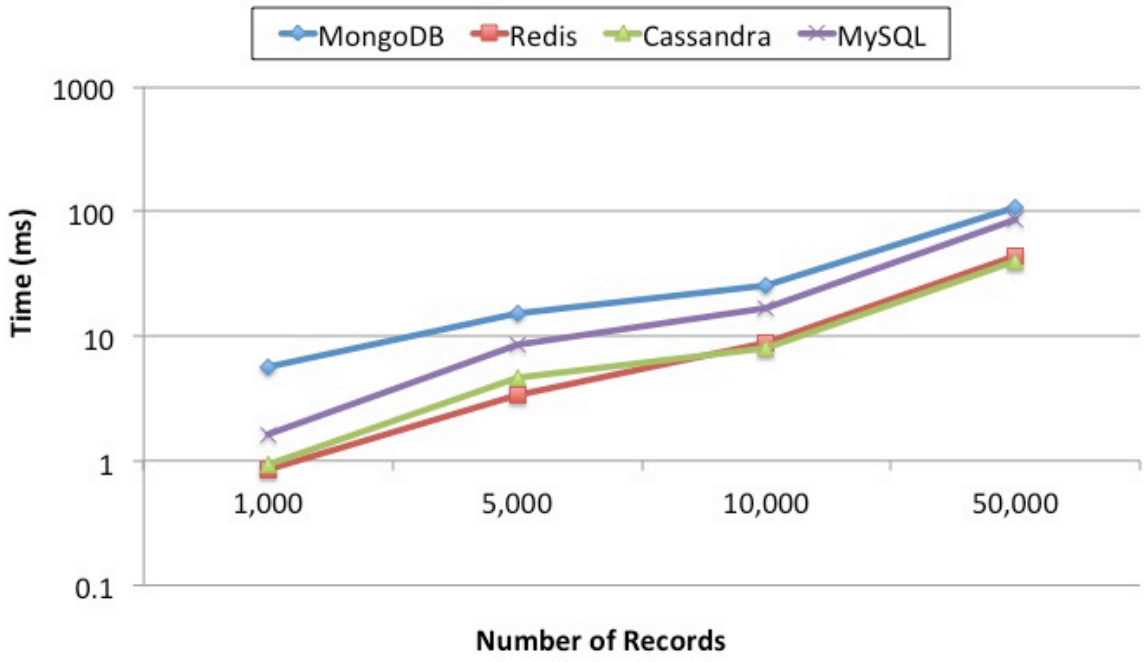

Figure 28. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 2: query 1.

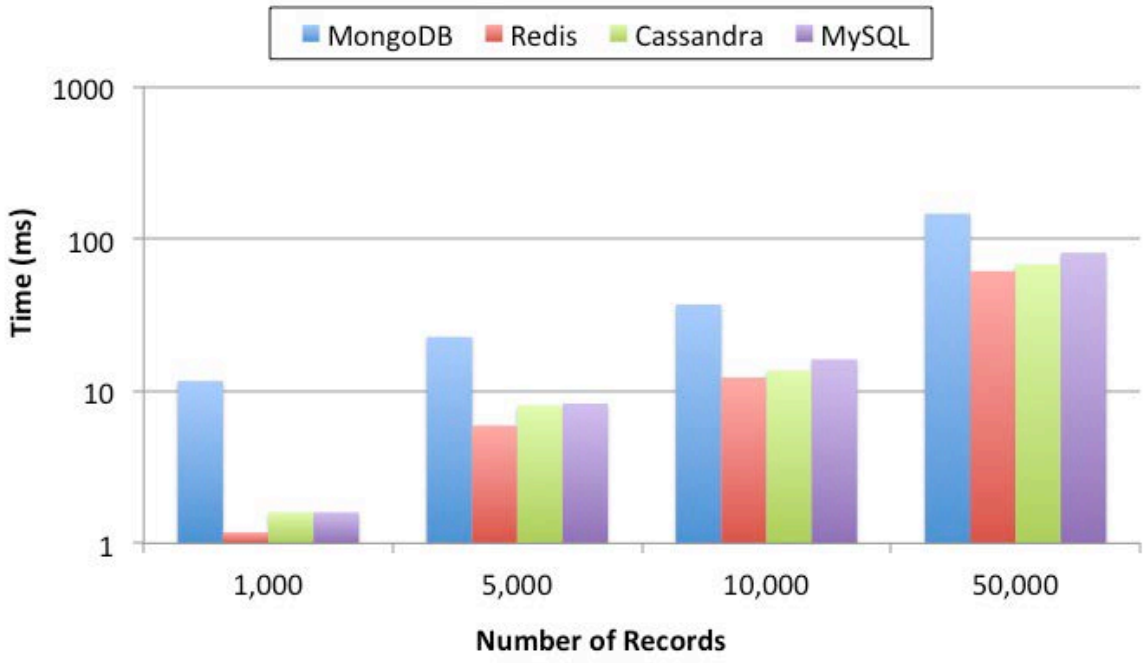

Figure 29. Variation in query time (base-10 log scale) of experiment 2: query 2.

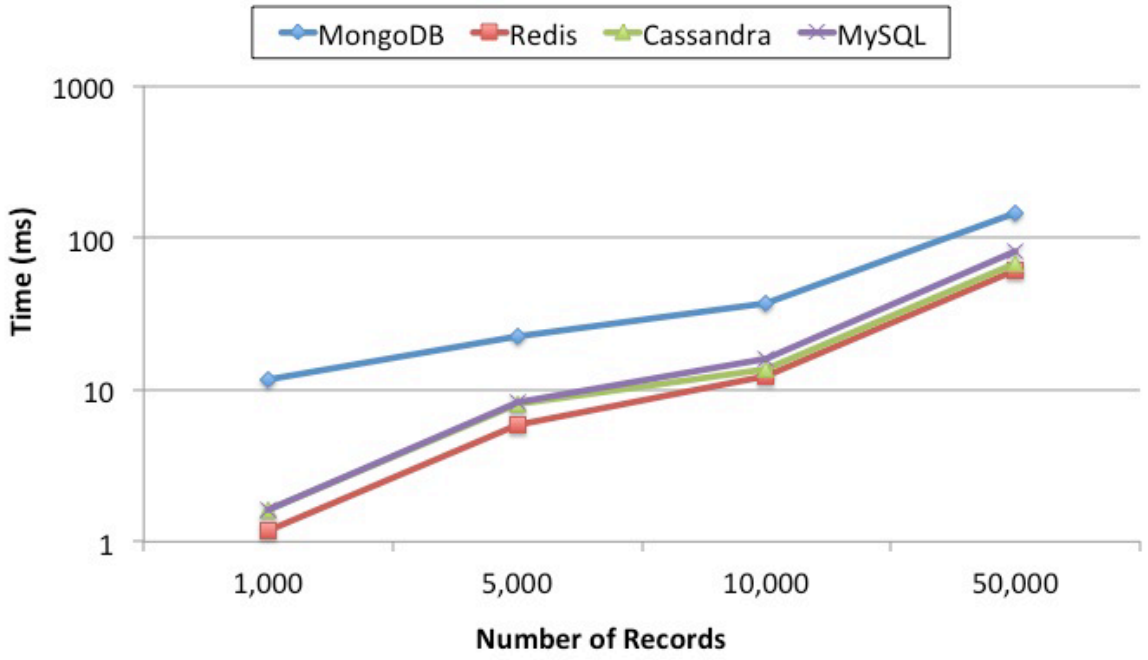

Figure 30. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 2: query 2.

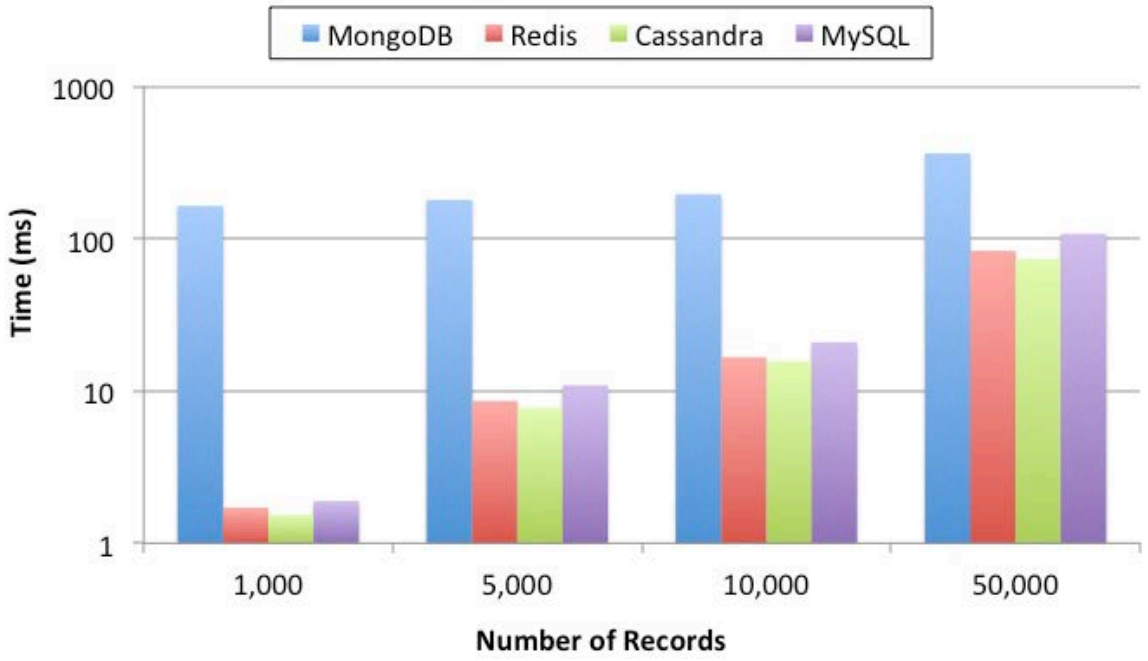

Figure 31. Variation in query time (base-10 log scale) of experiment 2: query 3.

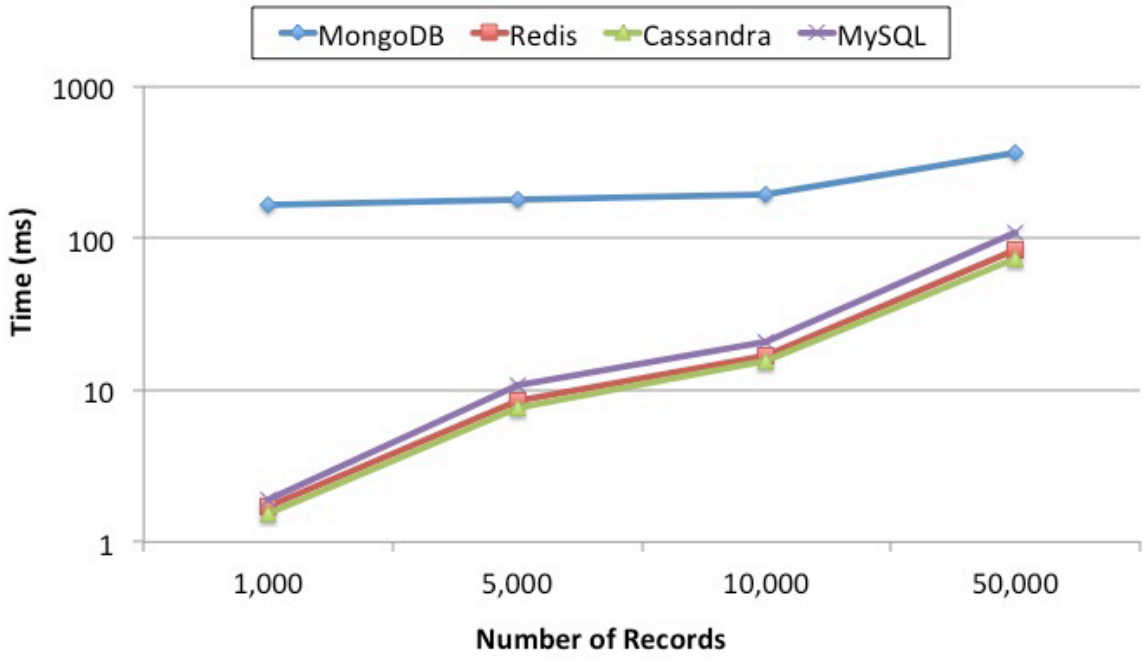

Figure 32. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 2: query 3.

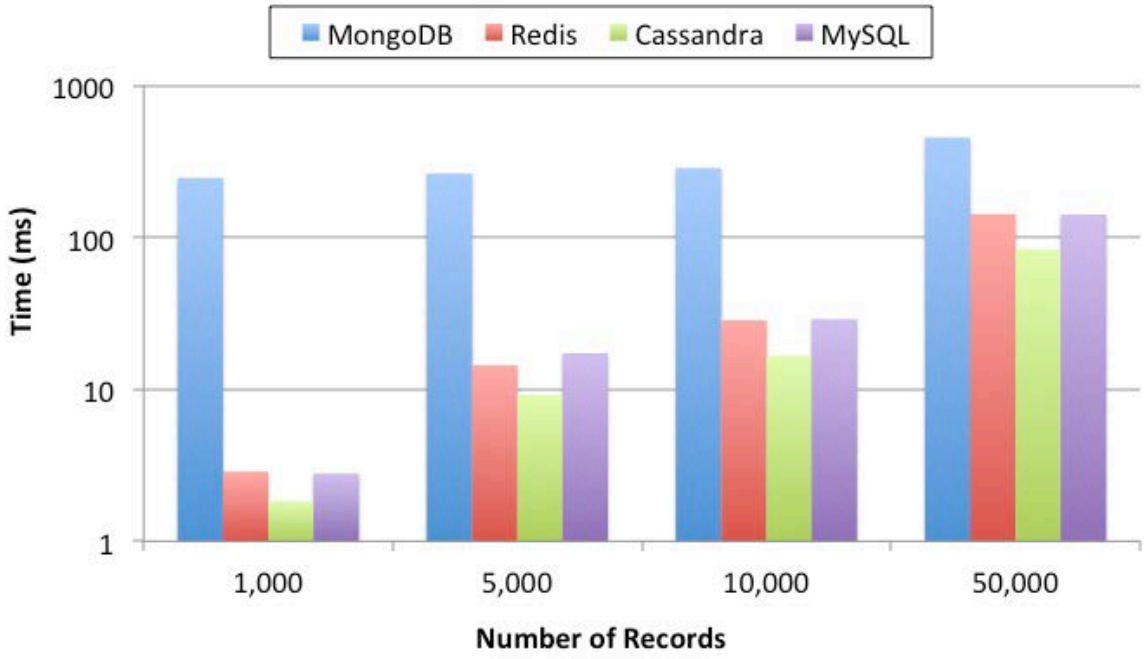

Figure 33. Variation in query time (base-10 log scale) of experiment 2: query 4.

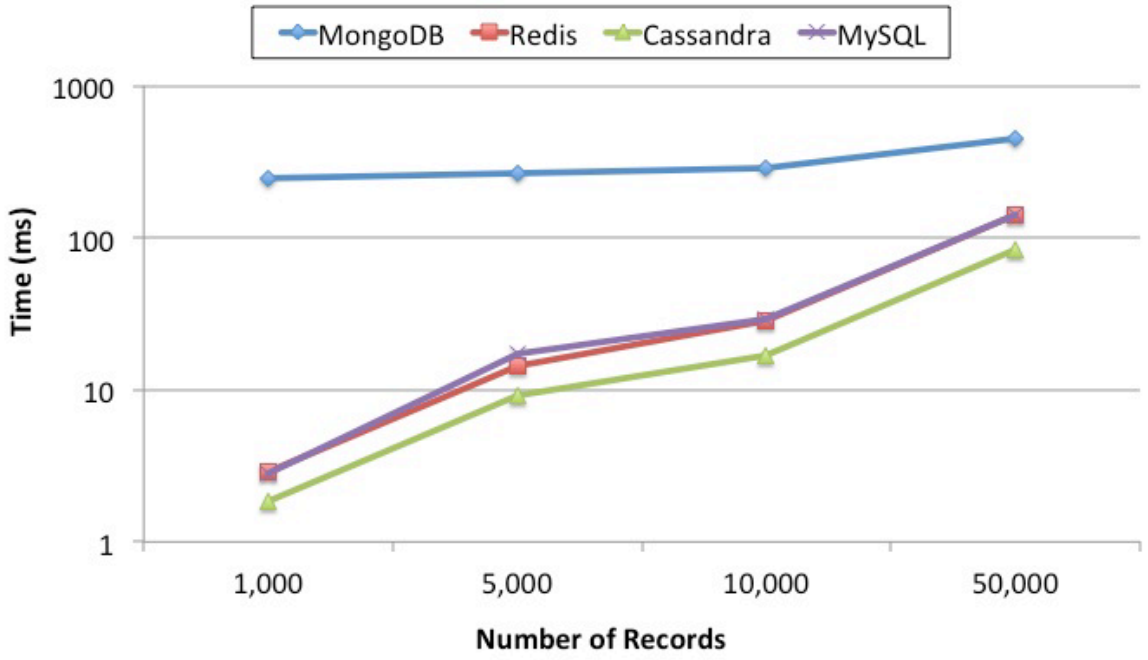

Figure 34. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 2: query 4.

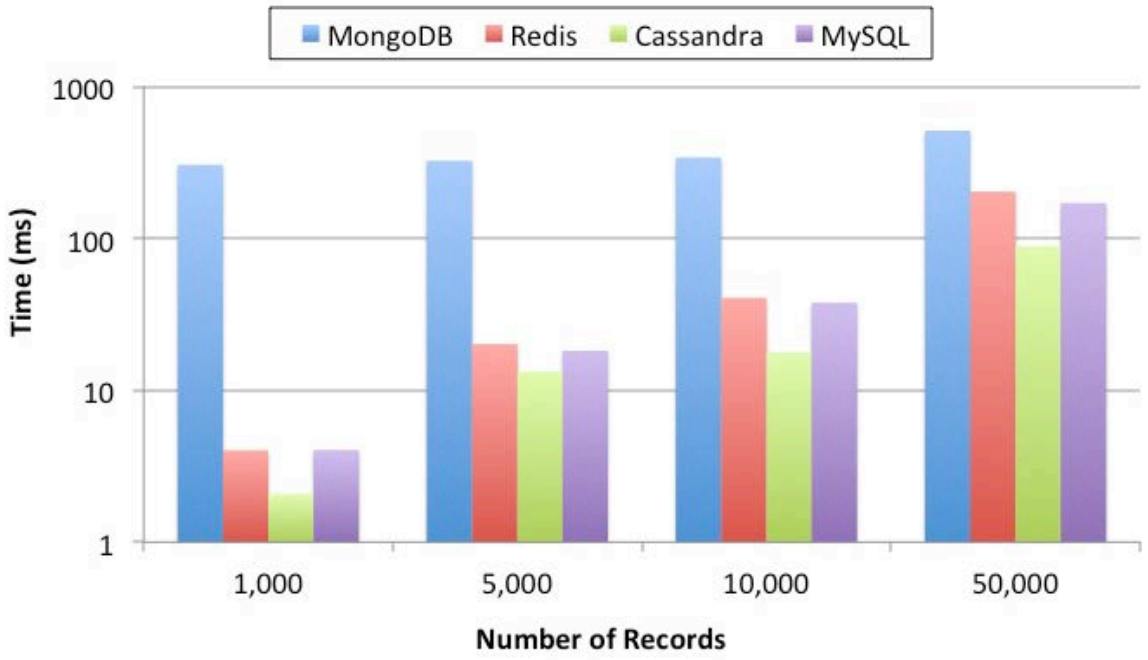

Figure 35. Variation in query time (base-10 log scale) of experiment 2: query 5.

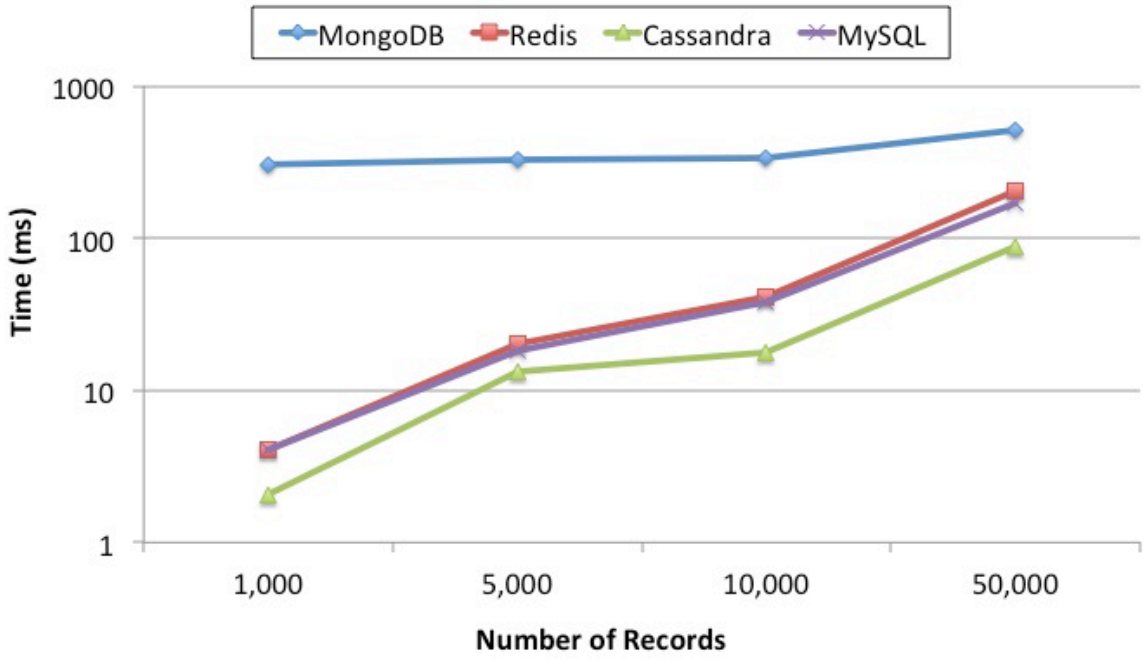

Figure 36. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 2: query 5.

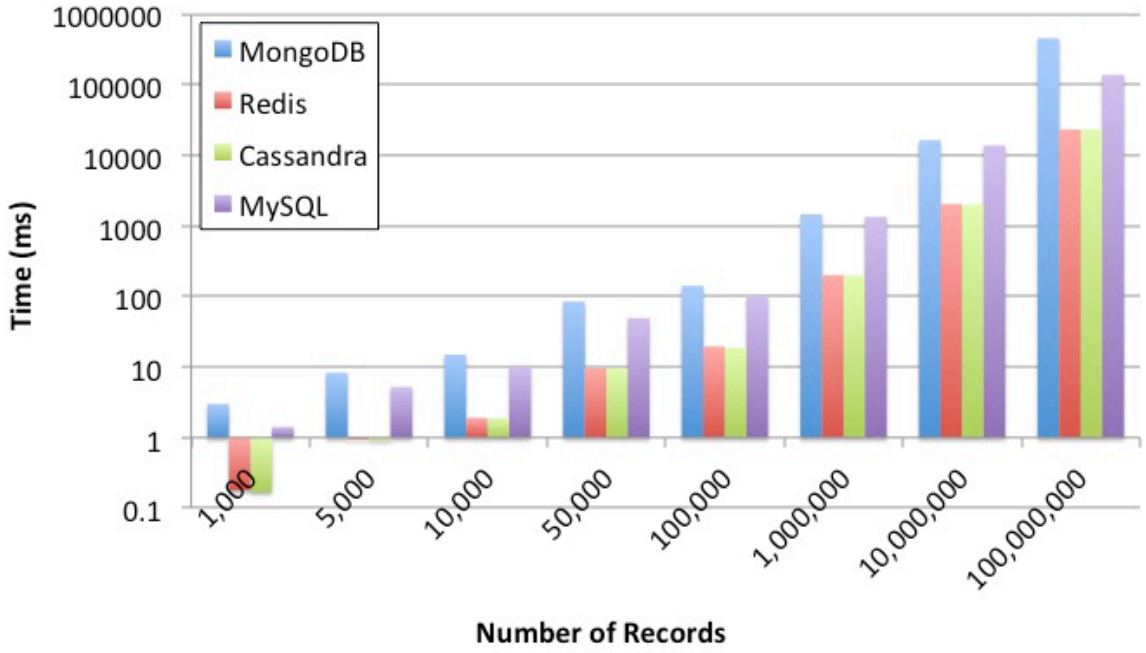

Figure 37. Variation in query time (base-10 log scale) of experiment 2 in larger databases: query 1.

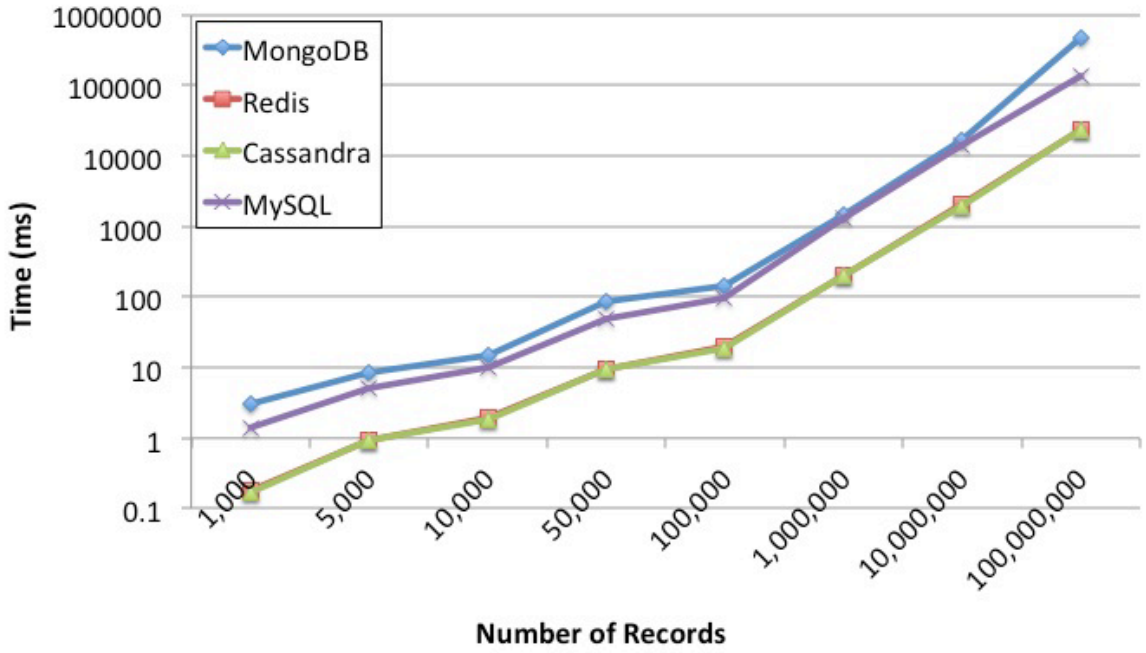

Figure 38. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 2 in larger databases: query 1.

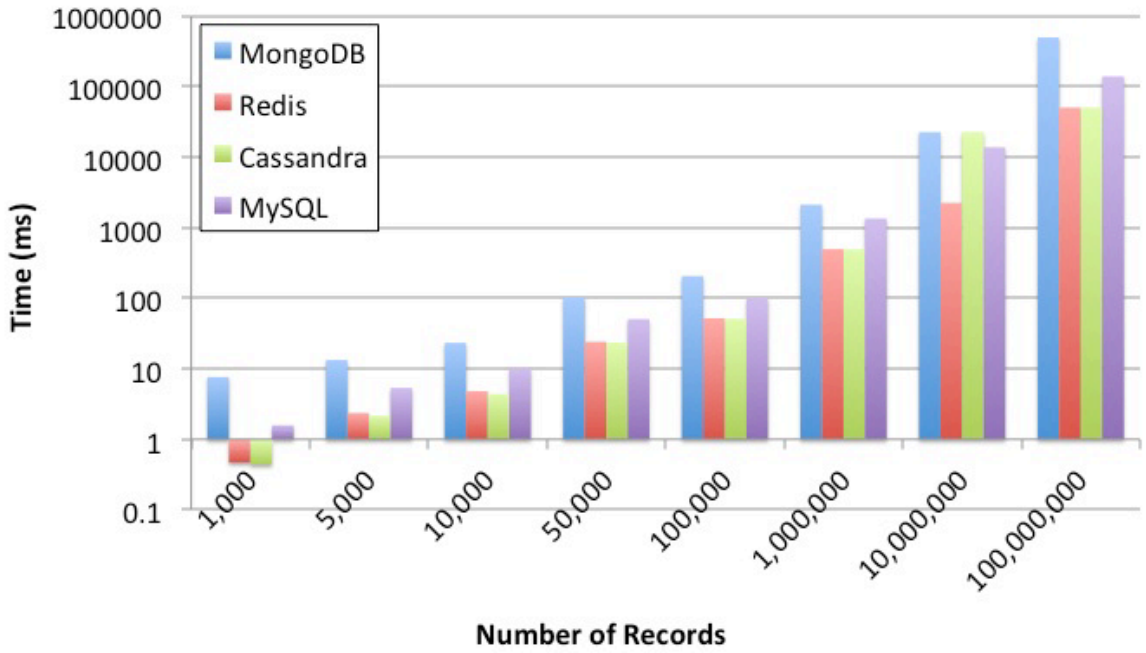

Figure 39. Variation in query time (base-10 log scale) of experiment 2 in larger databases: query 2.

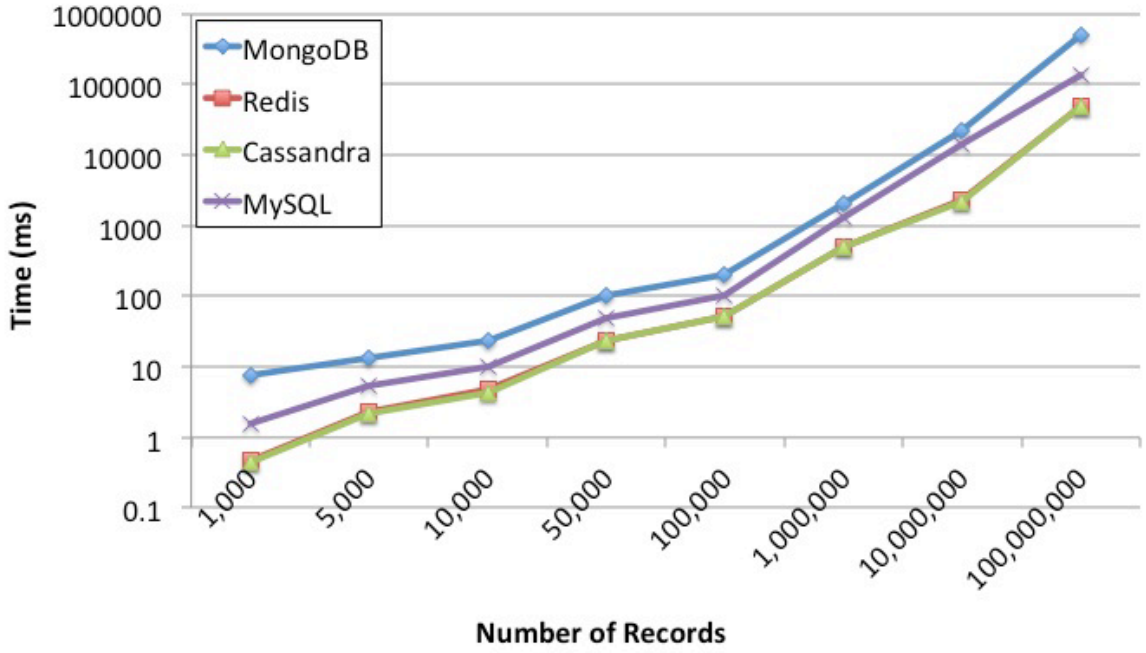

Figure 40. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 2 in larger databases: query 2.

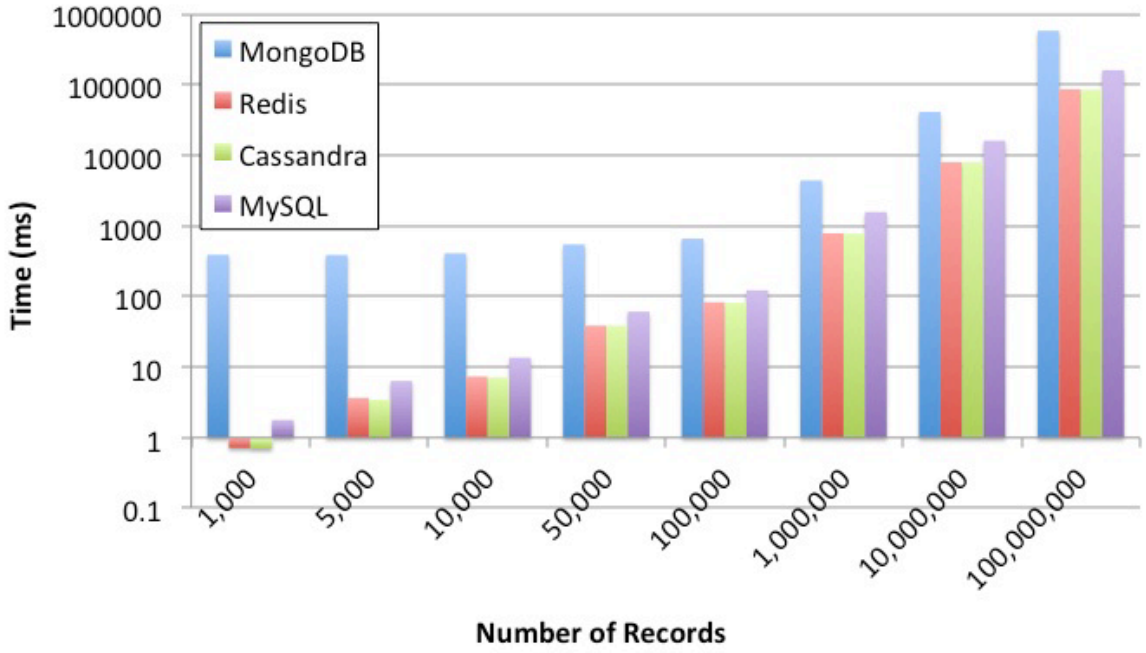

Figure 41. Variation in query time (base-10 log scale) of experiment 2 in larger databases: query 3.

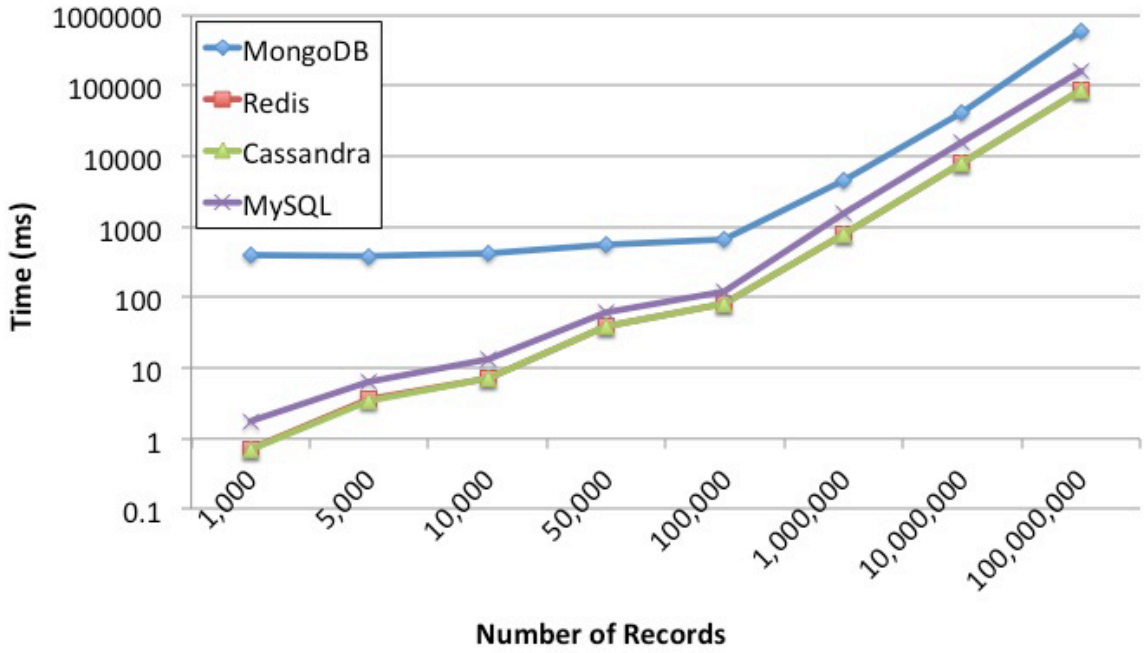

Figure 42. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 2 in larger databases: query 3.

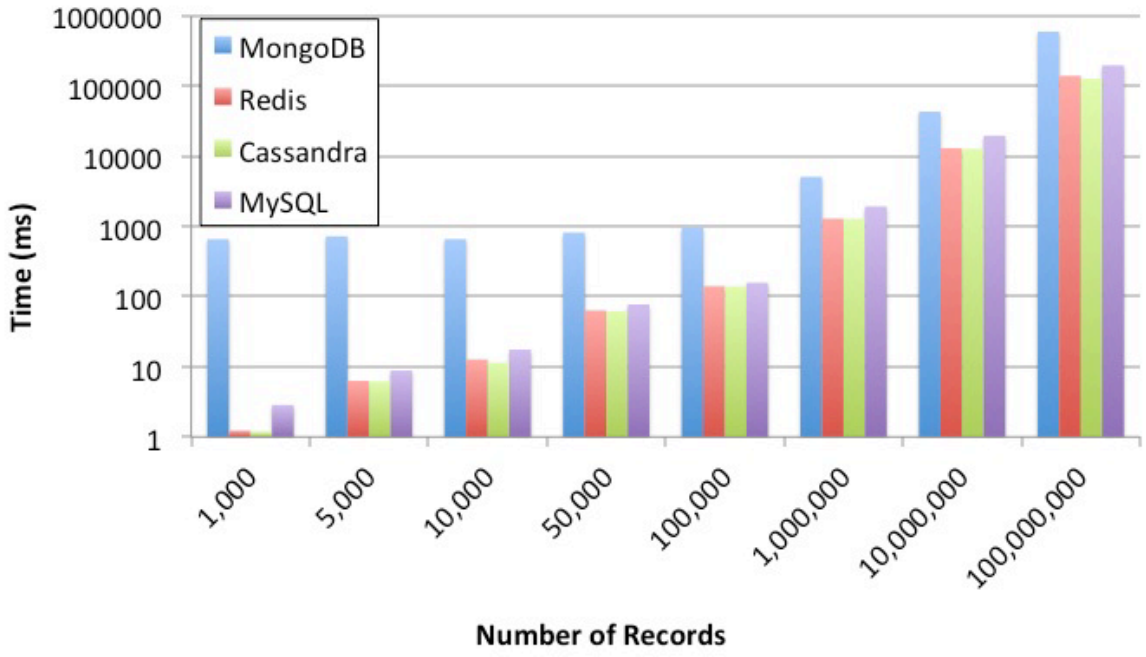

Figure 43. Variation in query time (base-10 log scale) of experiment 2 in larger databases: query 4.

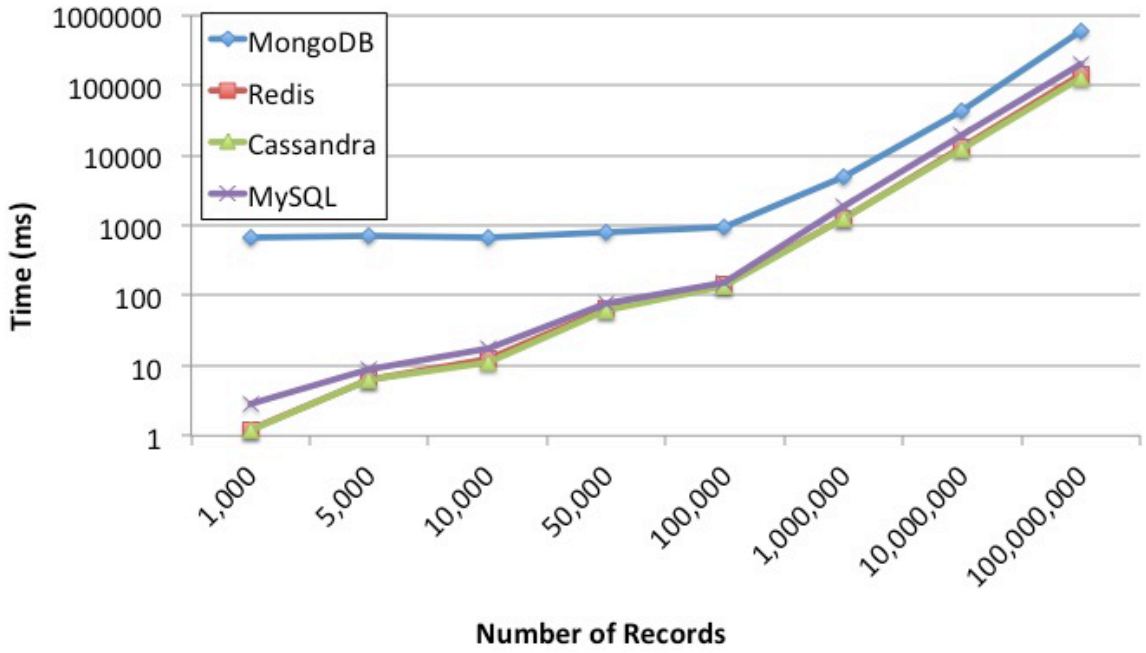

Figure 44. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 2 in larger databases: query 4.

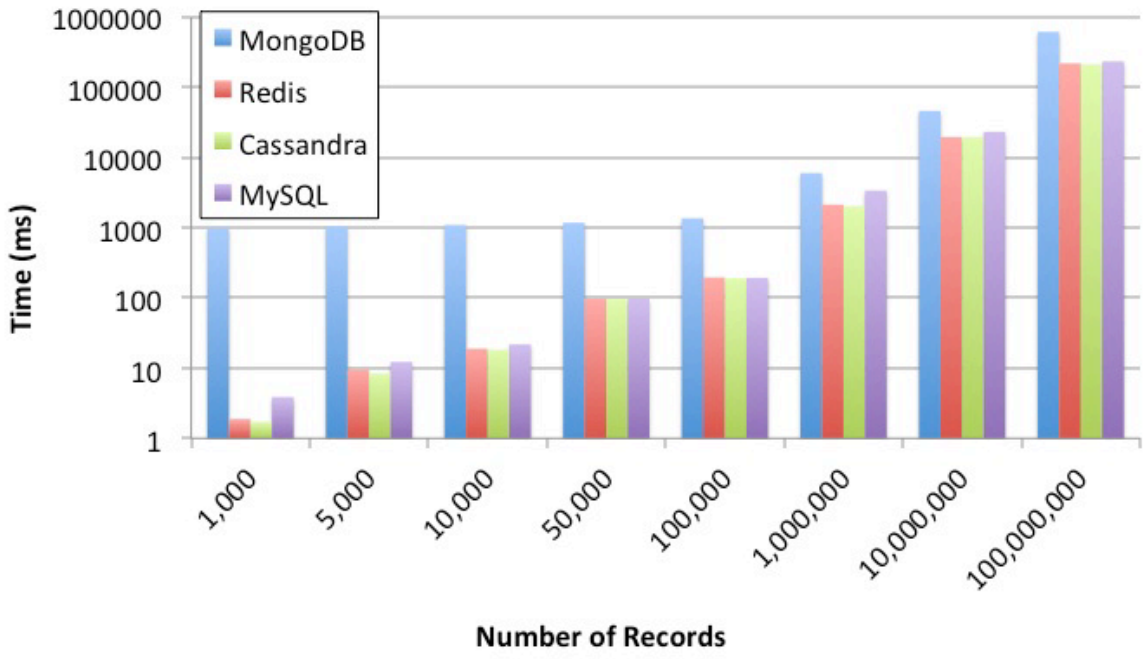

Figure 45. Variation in query time (base-10 log scale) of experiment 2 in larger databases: query 5.

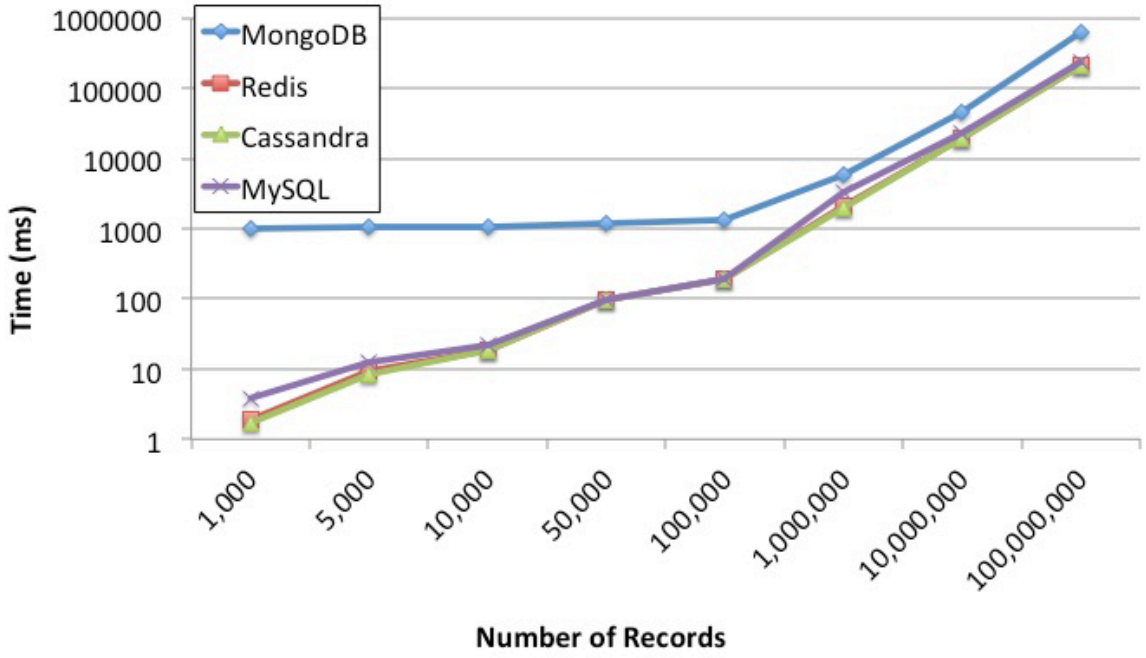

Figure 46. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 2 in larger databases: query 5.

6.0 SCALABILITY ON UPDATING WITH SCHEMA CHANGES

6.1 INTRODUCTION

In the Precision Medicine field, it is common find new discoveries because of technological advances. New discoveries challenge databases because they include novel information that is non-predefined for the database schema. At the same time, the availability of databases which contain novel data derived from recent discoveries is essential for researchers. In SQL systems it is problematic to include novel information in new table headers since the database schema needs to be redesigned entirely. NoSQL systems, on the other hand, facilitate the updating process, even if it requires including non-predefined information such as new headers or collections. NoSQL are schema-free.

Updating information is a challenging issue in big databases, whether SQL or NoSQL. The inclusion of new information increases the volume of data and forces users to modify queries to retrieve recently included information. These changes affect query times. Thus, for successful data management, database systems should adapt themselves in parallel with the new information, maintaining higher scalability by showing lower query times. For example if recent publications highlight the relevance of including a new variable such as “Population” in the database to detect breast cancer and this new information collected from patients needs to be included in a database, suitable DBMS should make this process easy.

In the following experiment, we studied the scalability of our previously selected database approaches to identify the one that can best adapt itself to this potential challenge in the management of clinical and genomic data.

In the following sub-sections we describe the details of our experiment in comparing performance and scalability using the previously described queries of varying complexity and an updating process that modified the database schema. In sub-section 6.2.1 we briefly present the performance and scalability results from the experiment. Also, we describe the results in a summary with our conclusions.

6.2 EXPERIMENT 3: COMPARING SCALABILITY ON UPDATING WITH SCHEMA CHANGES

To identify the database approach with highest scalability on updates with schema changes, we adapted the experiment used in Labrinidis et al. “Effect on queries: Schema changes.” We first updated the database with schema modifications by uploading non-predefined information (another column named “Population”) on the databases’ schema. In order to ask for the new information added, we made a new set of queries (Dynamic queries, explained in sub-section 3.2). These queries included a request for the new information that had been added to each patient’s record. We ran the dynamic queries and once the query times were obtained; we created tables to lay out the results, facilitating comparisons of query complexity and database size among the four database approaches. As a note, Table stores were completely redesigned and reloaded with a new schema since SQL technology has a rigid schema that does not easily accept changes. To identify the approach with the best performance in the updating process (schema

change) for each of the queries and database sizes, we selected the database approach that showed the lowest query time for each set, as described below in the experiment 3 workflow. All these calculations are described in section §6.2.1.1 Timing Performance. To identify the database approach with the best scalability in the updating process (schema change), we used the query times obtained above from measuring performance, but calculating for each database approach the difference in the query times from the 50,000-record DB minus the query times from the 1,000-record DB, and selecting the database approach with the lowest query time. All these calculations are described in section §6.2.1.1 Timing Scalability.

Experiment 3 was then repeated, but using supercomputing resources and even larger databases. These databases consisted of 1,000, 5,000, 10,000, 50,000, 100,000, 1 million, 10 millions and 100 millions records. To this purpose we used the same four database technologies: Document-MongoDB, Key-Value-Redis, Column-Cassandra and Table-MySQL. Performance and scalability results are annotated in Tables and Graphs, and described in section §6.2.1.1 Timing Performance and Timing Scalability, respectively.

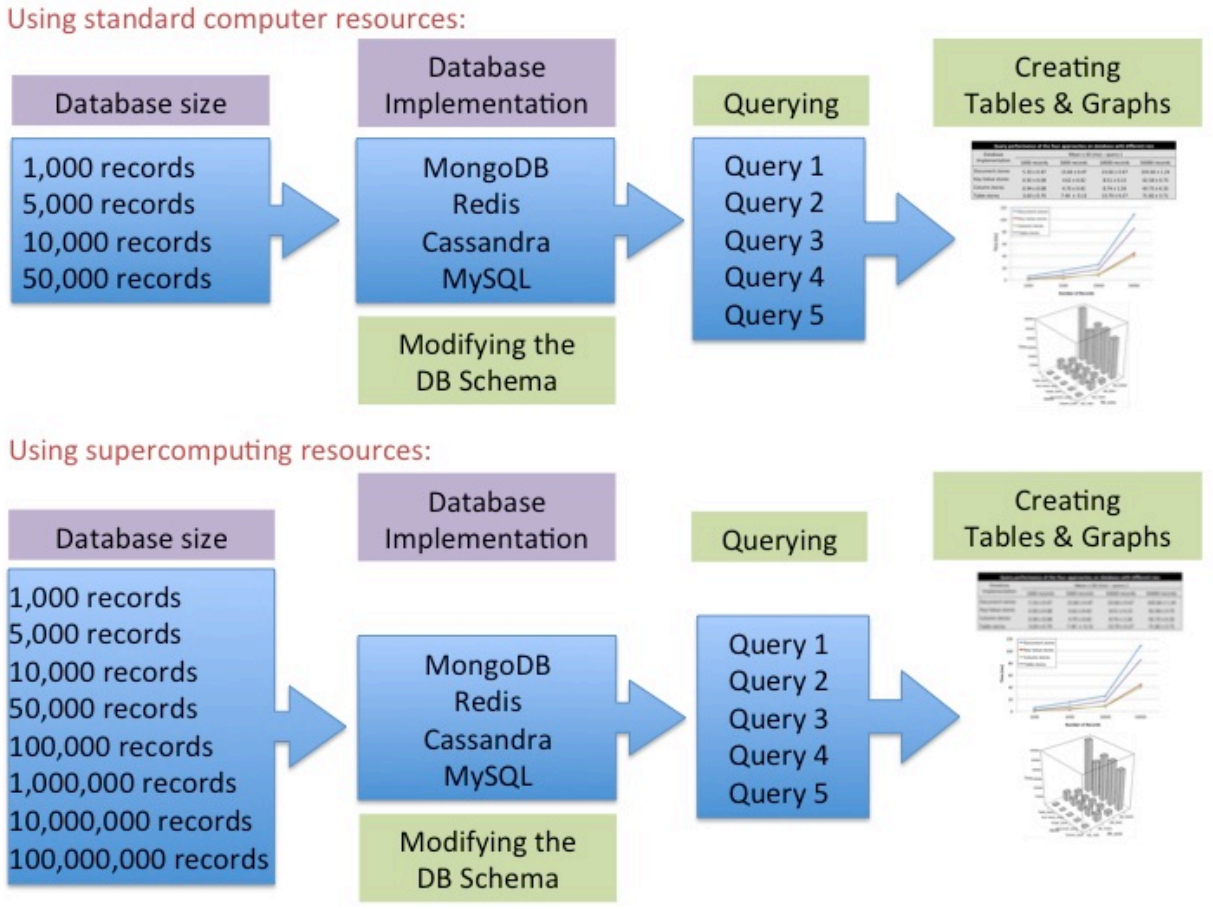

Figure 47. Workflow of Experiment 3.

6.2.1 Experimental Results

The query times of the four DBMS were evaluated by making five different queries with varying complexity and a instituting a database updating process (adding a new header: “Population”) that included schema change, as shown in Figure 47. The variation in query time in the database updating process was also studied. For each of the four database approaches, the time taken to make the queries with varying complexity, as described above, was measured with databases containing 1000, 5000, 10,000 and 50,000 patient records. In addition, using the same methodology but with supercomputer resources, the query time the four DBMS was measured

using the same number of records in databases and even larger databases -- with 100,000, 1,000,000, 10,000,000 and 100,000,000 patient records -- also shown in Figure 47. For each of the database sizes mentioned above, each query was made 3 times to calculate the average query time and the standard deviation (SD). The query times are given in Tables 23-32.

6.2.1.1 Query time results for queries of varying complexity and for different database sizes

Timing Performance

The timing performances for the simplest query are given in Table 23. For query 1 in the DB with 1,000 records, Cassandra resulted in the smallest query time. It was slightly faster than the MySQL and Redis, and more than 4 times faster than MongoDB. In the 5,000-record DB, Cassandra and MySQL were the fastest technologies. They were slightly faster than Redis and more than 2 times faster than MongoDB. In the 10,000-record DB, Cassandra had the shortest query time. It was slightly faster than MySQL and Redis and more than two times faster than MongoDB. Finally, in the 50,000-record DB, Cassandra was the fastest technology. It was slightly faster than the MySQL and Redis and almost two times faster than MongoDB.

The timing performances for the simplest query are given in Table 24. For query 2 in the DB with 1,000 records, MySQL and Cassandra had the smallest query times. They were slightly faster than Rdis and around 8 times faster than MongoDB. In the 5,000-record DB, Cassandra and MySQL were the fastest technologies. They were slightly faster than Redis and around 3 times faster than MongoDB. In the 10,000-record DB, Cassandra had the smallest query time. It was slightly faster than MySQL, almost two times faster than Redis and more than 3 times faster than MySQL. In the 50,000-record DB, Cassandra was the fastest technology. Its times were slightly faster than MySQL and Redis and more than 2 times faster than MongoDB.

The timing performances for the simplest query are given in Table 25. For query 3 in the DB with 1,000 records, Cassandra had the smallest query time. It was slightly faster than the MySQL and Redis and more than 104 times faster than MongoDB. In the 5,000-record DB, Cassandra was again the fastest technology. It was slightly faster than Redis and more than 23 times faster than MongoDB. In the 10,000-records DB, Cassandra again had the smallest query time. It was slightly faster than the MySQL and Redis stores and more than 13 times faster than MongoDB. In the 50,000-record DB, Cassandra was the fastest technology. It was slightly faster than MySQL and Redis and almost 5 times faster than MongoDB.

The timing performances for the simplest query are given in Table 26. For query 4 in the DB with 1,000 records, Cassandra had the smallest query time. It was around 2 times faster than the MySQL and Redis and almost 150 times faster than MongoDB. In the 5,000-record DB, Cassandra was the fastest technology. It was slightly faster than Redis and more than 31 times faster than MongoDB. In the 10,000-record DB, Cassandra had the smallest query time. It was around two times faster than the MySQL and Redis stores and more than 31 times faster than MongoDB. In the 50,000-record DB, Cassandra was again the fastest technology. Its results were around 2 times faster than the MySQL and Redis and more than 5 times faster than MongoDB.

The timing performances for the simplest query are given in Table 27. For query 5 in the DB with 1,000 records, Cassandra had the smallest query time. It was around 3 times faster than MySQL and Redis and more than 157 times faster than MongoDB. In the 5,000-record DB, Cassandra was the fastest technology as well. It was more than 2 times faster than MySQL and Redis and more than 33 times faster than MongoDB. In the 10,000-record DB, Cassandra had the smallest query times. It was around 2 times faster than the MySQL and Redis and 16 times faster

than MongoDB. In the 50,000-record DB, Cassandra was the fastest technology. It was around 2 times faster than the MySQL and Redis and almost 5 times faster than MongoDB.

The timing performances for the simplest query are given in Table 28. For query 1 in the database size of 1,000 records, Cassandra and Redis showed almost the same query times. In fact, Cassandra and Redis obtained results more than 7 and 20 times faster than MySQL and MongoDB, respectively. In the 5,000-record database, again Cassandra and Redis were the fastest technologies. In this database size, Cassandra and Redis were more than 5 and 8 times faster than MySQL and MongoDB, respectively. In the 10,000-record database, Cassandra and Redis again had almost the same query times, and both technologies were more than 5 and 8 times faster than MySQL and MongoDB, respectively. Finally, in the 50,000-record database, Cassandra and Redis had similar query times, respectively. In this database size, Cassandra and Redis were more than 4 and 7 times faster than MySQL and MongoDB, respectively. In the 100,000 records, Cassandra and Redis were the fastest technologies. In fact, they were more than 3 and 6 times faster than MySQL and MongoDB, respectively. In the 1-million records database, again Cassandra and Redis were the fastest technologies. For this database size, they were more than 5 and 6 times faster than MySQL and MongoDB. In the 10 million-records, Cassandra and Redis again were the fastest technologies. Both technologies were 6 and 7 times faster than MySQL and MongoDB, respectively. In 100 million records, Cassandra and Redis were the fastest technologies. In fact, they were more than 4 and 17 times faster than MySQL and MongoDB, respectively.

The timing performances for query 2 are given in Table 29. For this query in the database size of 1,000 records, Cassandra and Redis showed almost the same query times. In fact, Cassandra and Redis obtained results more than 2 and 15 times faster than MySQL and

MongoDB, respectively. In the 5,000-record database, again Cassandra and Redis were the fastest technologies. In this database size, Cassandra and Redis were slightly higher than MySQL and 4 times faster than MongoDB. In the 10,000-record database, Cassandra and Redis again had almost the same query times, and both technologies were slightly faster than MySQL and 4 times faster than MongoDB. Finally, in the 50,000-record database, Cassandra and Redis had similar query times, respectively. In this database size, Cassandra and Redis were slightly faster than MySQL and 3 times faster than MongoDB. In 100,000 records, Cassandra and Redis were the fastest technologies. In fact, they were slightly faster than MySQL and more than 3 times faster than MongoDB, respectively. In the 1-million records database, again Cassandra and Redis were the fastest technologies. For this database size, they were more than 2 and 3 times faster than MySQL and MongoDB. In 10 million-records, Cassandra and Redis again were the fastest technologies. Both technologies were again more than 2 and 3 times faster than MySQL and MongoDB. In 100 million records, Cassandra and Redis were the fastest technologies. In fact, they were more than 2 and 7 times faster than MySQL and MongoDB, respectively.

The timing performances for query 3 are given in Table 30. For this query in the database size of 1,000 records, Cassandra and Redis showed almost the same query times. In fact, Cassandra and Redis obtained results more than 2 and 492 times faster than MySQL and MongoDB, respectively. In the 5,000-record database, again Cassandra and Redis were the fastest technologies. In this database size, Cassandra and Redis were slightly faster than MySQL and more than 98 times faster than MongoDB, respectively. In the 10,000-record database, Cassandra and Redis again had almost the same query times, and both technologies were slightly faster than MySQL and more than 51 times faster than MongoDB, respectively. Finally, in the 50,000-record database, Cassandra and Redis had similar query times. In this database size,

Cassandra and Redis were slightly faster than MySQL and more than 13 times faster than MongoDB, respectively. In 100,000 records, Cassandra and Redis were the fastest technologies. In fact, they were slightly faster than MySQL and more than 7 times faster than MongoDB, respectively. In the 1-million records database, again Cassandra and Redis were the fastest technologies. For this database size, they were slightly faster than MySQL and 4 times faster than MongoDB. In 10 million-records, Cassandra and Redis again were the fastest technologies. Both technologies were again more slightly higher than MySQL and 4 times faster than MongoDB. Finally, in 100 million records, Cassandra and Redis were the fastest technologies. In fact, they were slightly faster than MySQL and 6 times faster than MongoDB.

The timing performances for query 4 are given in Table 31. For this query in the database size of 1,000 records, Cassandra and Redis showed almost the same query times. In fact, Cassandra and Redis obtained results 2 and 497 times faster than MySQL and MongoDB. In the 5,000-record database, again Cassandra and Redis were the fastest technologies. In this database size, Cassandra and Redis were slightly faster than MySQL and more than 130 times faster than MongoDB. In the 10,000-record database, Cassandra and Redis again had almost the same query times, and both technologies were slightly faster than MySQL and more than 53 times faster than MongoDB. In the 50,000-record database, Cassandra and Redis had similar query times. In this database size, Cassandra and Redis were slightly faster than MySQL and more than 12 times faster than MongoDB. In 100,000 records, Cassandra and Redis were the fastest technologies. In fact, they were slightly faster than MySQL and more than 5 times faster than MongoDB. In the 1-million records database, again Cassandra and Redis were the fastest technologies. For this database size, they were slightly faster than MySQL and 2 times faster than MongoDB. In 10 million-records, Cassandra and Redis again were the fastest technologies. Both technologies

were again slightly higher than MySQL and 2 times faster than MongoDB. Finally, in 100 million records, Cassandra and Redis were the fastest technologies. In fact, they were slightly faster than MySQL and more than 4 times faster than MongoDB.

The timing performances for query 5 are given in Table 32. For this query in the database size of 1,000 records, Cassandra and Redis showed almost the same query times. In fact, Cassandra and Redis obtained results 2 and 529 times faster than MySQL and MongoDB. In the 5,000-record database, again Cassandra and Redis were the fastest technologies. In this database size, Cassandra and Redis were slightly faster than MySQL and more than 115 times faster than MongoDB. In the 10,000-record database, Cassandra and Redis again had almost the same query times, and both technologies were slightly faster than MySQL and more than 52 times faster than MongoDB. Finally, in the 50,000-record database, Cassandra, Redis and MySQL had similar query times. In this database size, they were more than 12 times faster than MongoDB. In 100,000 records, MySQL, Cassandra and Redis were the fastest technologies. In fact, they were more than 7 times faster than MongoDB. In the 1-million records database, Cassandra and Redis were the fastest technologies. For this database size, they were slightly faster than MySQL and 2 times faster than MongoDB. In 10 million-records, Cassandra and Redis again were the fastest technologies. Both technologies were again slightly higher than MySQL and MongoDB. In 100 million records, Cassandra and Redis were the fastest technologies. In fact, they were slightly faster than MySQL and more than 2 times faster than MongoDB.

Table 23. Query performance of experiment 3: query 1.

Query performance of four DBMS using updated (schema chng) databases with different size				
Database Implementation	Mean ± SD (ms) – query 1			
	1000 records	5000 records	10000 records	50000 records
MongoDB	6.66 ± 0.47	17.66 ± 0.94	29.33 ± 3.29	115.66 ± 4.49
Redis	1.97 ± 0.16	9.86 ± 0.80	18.62 ± 0.08	93.1 ± 0.42
Cassandra	1.44 ± 0.05	7.22 ± 0.26	12.23 ± 0.62	61.18 ± 3.11
MySQL	1.57 ± 0.30	7.23 ± 0.03	14.01 ± 0.06	71.79 ± 2.40

Table 24. Query performance of experiment 3: query 2.

Query performance of four DBMS using updated (schema chng) databases with different size				
Database Implementation	Mean ± SD (ms) – query 2			
	1000 records	5000 records	10000 records	50000 records
MongoDB	12 ± 0	24.33 ± 0.47	40 ± 0.81	151 ± 2.16
Redis	2.32 ± 0.14	11.63 ± 0.70	22.25 ± 0.09	107.91 ± 5.01
Cassandra	1.53 ± 0.00	7.69 ± 0.04	13.63 ± 1.16	68.16 ± 5.81
MySQL	1.45 ± 0.19	8.12 ± 0.03	18.29 ± 1.67	80.52 ± 1.53

Table 25. Query performance of experiment 3: query 3.

Query performance of four DBMS using updated (schema chng) databases with different size				
Database Implementation	Mean ± SD (ms) – query 3			
	1000 records	5000 records	10000 records	50000 records
MongoDB	166.66 ± 0.47	185 ± 2.16	202.66 ± 1.69	372 ± 1.41
Redis	2.77 ± 0.12	13.85 ± 0.60	26.78 ± 0.19	133.93 ± 0.96
Cassandra	1.59 ± 0.02	7.99 ± 0.10	15.11 ± 0.12	75.58 ± 0.61
MySQL	2.08 ± 0.48	11.47 ± 1.27	21.76 ± 0.97	106.27 ± 1.75

Table 26. Query performance of experiment 3: query 4.

Query performance of four DBMS using updated (schema chng) databases with different size				
Database Implementation	Mean ± SD (ms) – query 4			
	1000 records	5000 records	10000 records	50000 records
MongoDB	248.66 ± 1.24	264.33 ± 0.47	287.33 ± 0.47	460.33 ± 1.24
Redis	3.96 ± 0.09	19.83 ± 0.455	38.99 ± 0.18	194.98 ± 0.93
Cassandra	1.66 ± 0.04	8.34 ± 0.23	16.73 ± 0.59	83.66 ± 2.99
MySQL	3.27 ± 0.79	16 ± 1.41	27.02 ± 0.02	136.55 ± 1.53

Table 27. Query performance of experiment 3: query 5.

Query performance of four DBMS using updated (schema chng) databases with different size				
Database Implementation	Mean ± SD (ms) – query 5			
	1000 records	5000 records	10000 records	50000 records
MongoDB	307.66 ± 0.94	328 ± 0.81	351 ± 4.32	522.66 ± 4.10
Redis	5.43 ± 0.39	27.18 ± 2	51.68 ± 1.33	258.43 ± 6.65
Cassandra	1.95 ± 0.13	9.79 ± 0.65	21.45 ± 1.45	107.25 ± 7.27
MySQL	5.20 ± 1.38	20.02 ± 0.66	36.65 ± 2.58	169.55 ± 5.17

Table 28. Query performance of experiment 3 in larger databases: query 1.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 1			
	1,000 records	5,000 records	1,0000 records	50,000 records
MongoDB	4 ± 0	9 ± 0	19 ± 0.81	91.33 ± 1.24
Redis	0.20 ± 0.003	1.02 ± 0.01	2.13 ± 0.02	11.88 ± 0.26
Cassandra	0.17 ± 0.01	0.98 ± 0.03	2.10 ± 0.01	11.36 ± 0.01
MySQL	1.55 ± 0.06	5.36 ± 0.01	11.07 ± 0.0004	50.38 ± 0.16
	100,000 records	1,000,000 records	10,000,000 records	100,000,000 records
MongoDB	164.3 ± 1.34	1506.1 ± 44.18	16765.3 ± 94.22	464490 ± 0.99
Redis	25.81 ± 0.39	222.60 ± 2.83	2249.63 ± 48.75	26826.26 ± 1567
Cassandra	23.36 ± 0.46	203.33 ± 3.15	2194.01 ± 22.59	25984.90 ± 83.04
MySQL	91.24 ± 0.19	1260.28 ± 1.33	13887.91 ± 356.4	131080 ± 0.11

Table 29. Query performance of experiment 3 in larger databases: query 2.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 2			
	1,000 records	5,000 records	1,0000 records	50,000 records
MongoDB	8.66 ± 0.47	14 ± 0	25 ± 0.81	110.66 ± 3.09
Redis	0.56 ± 0.02	2.82 ± 0.09	5.71 ± 0.30	29.35 ± 0.51
Cassandra	0.51 ± 0.03	2.59 ± 0.02	5.38 ± 0.05	28.49 ± 0.01
MySQL	1.66 ± 0.002	5.48 ± 0.01	11 ± 0.0008	51.68 ± 0.03
	100,000 records	1,000,000 records	10,000,000 records	100,000,000 records
MongoDB	236.1 ± 2.38	2078.5 ± 4.54	22688.5 ± 153.66	500450 ± 0.20
Redis	68.76 ± 0.56	591.65 ± 5.60	5965.57 ± 28.72	66055.33 ± 2575.2
Cassandra	66.26 ± 0.33	590.29 ± 1.58	5896.31 ± 35.43	65184.03 ± 41.7
MySQL	100.08 ± 0.52	1415.70 ± 40.66	13954.70 ± 30.04	139580 ± 0.31

Table 30. Query performance of experiment 3 in larger databases: query 3.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 3			
	1,000 records	5,000 records	1,0000 records	50,000 records
MongoDB	428.33 ± 7.40	430.33 ± 6.59	454.66 ± 7.58	583 ± 2.94
Redis	0.87 ± 0.01	4.39 ± 0.05	8.80 ± 0.07	44.09 ± 0.11
Cassandra	0.81 ± 0.01	4.25 ± 0.03	8.55 ± 0.05	40.13 ± 0.01
MySQL	1.83 ± 0.02	6.13 ± 0.006	14.66 ± 0.02	62.67 ± 0.06
	100,000 records	1,000,000 records	10,000,000 records	100,000,000 records
MongoDB	780.2 ± 27.82	3802.9 ± 49.74	36559.4 ± 215.05	589310 ± 0.38
Redis	100.83 ± 0.64	938.07 ± 2.35	9012.82 ± 37.53	98091.72 ± 1881.9
Cassandra	98.42 ± 0.29	921.31 ± 1.64	8792.17 ± 48.01	95359.64 ± 153.19
MySQL	123.23 ± 0.33	1581.32 ± 3.14	16227.49 ± 14.71	164910 ± 1.20

Table 31. Query performance of experiment 3 in larger databases: query 4.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 4			
	1,000 records	5,000 records	1,0000 records	50,000 records
MongoDB	672 ± 2.44	877.66 ± 5.31	719.33 ± 7.40	880.66 ± 17.96
Redis	1.35 ± 0.01	6.74 ± 0.08	13.57 ± 0.21	68.49 ± 0.72
Cassandra	1.29 ± 0.01	6.59 ± 0.01	13.25 ± 0.03	67.31 ± 0.04
MySQL	2.98 ± 0.01	9.21 ± 0.03	18.47 ± 0.02	78.16 ± 0.06
	100,000 records	1,000,000 records	10,000,000 records	100,000,000 records
MongoDB	780.2 ± 27.82	3802.9 ± 49.74	36559.4 ± 215.05	593870 ± 0.05
Redis	145.32 ± 1.36	1379 ± 7.98	13784.53 ± 166.95	147655.53 ± 1385
Cassandra	143.61 ± 0.45	1355.83 ± 4.69	13591.64 ± 19.05	137849.08 ± 293.1
MySQL	156.94 ± 0.37	1917.12 ± 5.39	19565.22 ± 21.99	198850 ± 0.14

Table 32. Query performance of experiment 3 in larger databases: query 5.

Query performance of four DBMS using databases with different size				
Database Implementation	Mean ± SD (ms) – query 5			
	1,000 records	5,000 records	1,0000 records	50,000 records
MongoDB	1022 ± 13.14	1123 ± 12.96	1136.33 ± 19.95	1269 ± 9.20
Redis	1.93 ± 0.01	9.75 ± 0.07	21.68 ± 0.04	100.78 ± 0.43
Cassandra	1.89 ± 0.03	9.50 ± 0.03	21.13 ± 0.01	98.96 ± 0.39
MySQL	3.95 ± 0.02	13.22 ± 0.05	22.71 ± 0.01	97.77 ± 0.03
	100,000 records	1,000,000 records	10,000,000 records	100,000,000 records
MongoDB	1626.7 ± 191.97	4590.8 ± 23.12	37480.8 ± 328.95	615870 ± 0.03
Redis	203.92 ± 2.97	2017.22 ± 49.87	22392.49 ± 354.90	206929.91 ± 3328
Cassandra	199.37 ± 3.51	2012.35 ± 17.80	22298.07 ± 19.61	205913.65 ± 224.6
MySQL	193.17 ± 0.21	2278.35 ± 22.28	23150.22 ± 47.35	235460 ± 0.06

Timing Scalability

The scalability of the database approaches, allowing querying of more patient records in a lesser amount of time, is described per query number in Figures 48-57.

As shown in Figures 48 and 49, in query 1, Cassandra had the lowest query time scaling from 1,000 to 50,000 records. In around 59.74ms it scaled to the largest database, 10.48ms less than the second fastest technology, MySQL. As shown in Figure 50 and 51, in query 2, Cassandra also had the lowest query time to scale to the largest database. It took 66.63ms to scale from 1,000 to 50,000 records, making it slightly faster than MySQL, which needed 79.07ms. As shown in Figure 52 and 53, in query 3, Cassandra was the fastest technology to scale to the largest number of records. It needed 73.99ms to scale to 50,000 records, making it again slightly faster than MySQL, which took 104.19ms. As shown in Figures 54 and 55, in query 4, Cassandra was the fastest technology to reach the larger database as well; it took 88ms, slightly faster than MySQL and Redis, which took 133.27ms and 191.02ms, respectively. Finally, as shown in Figures 56 and 57, in query 5, Cassandra was again the fastest technology to scale from 10,000 to 50,000 records. It took 105.3ms to scale to the largest database, slightly higher than Redis and MySQL, which took 253ms and 164.35ms, respectively.

As shown in Figures 58 and 59, in query 1, Cassandra and Redis took the least time to scale from 1,000 to 100,000,000 records. Specifically they took 26826.06 and 25984.73 ms, respectively, to scale to the largest database. As shown in Figures 60 and 61, in query 2, Cassandra and Redis took the least time to scale to the largest database. Specifically, they took 66054.77 and 65183.52 ms, respectively, to scale from 1,000 to 100,000,000 records. As shown in Figures 62 and 63, in query 3, Cassandra and Redis were the fastest technology to scale to the largest number of records. They needed 98090.85 and 95358.83 ms, respectively, to scale from

1,000 to 100,000,000 records. As shown in Figures 64 and 65, in query 4, Cassandra and Redis were the fastest technology to reach the largest database; they took 147654.18 and 137847.79 ms, respectively. Finally, as shown in Figures 66 and 67, in query 5, Cassandra and Redis were again the fastest technology to scale from 1,000 to 100,000,000 records. They took 206927.98 and 205911.76 ms, respectively, to scale to the largest database.

6.2.1.2 Summary

As in to the two previous experimental sections, MongoDB had larger query times, as expected for all queries and database sizes. As shown in Figures 48-57, similar to the last experiment, for the simple queries, the performances of Cassandra and Redis and MySQL stores were similar when the size of the database was smaller but diverged significantly as the number of records increased to 10,000 records and beyond, highlighting the query speed of Cassandra and Redis

In this experiment, the performance of Cassandra and Redis was significantly faster than MongoDB for a large number of records. Using standard computing resources, the timing was remarkable for more complex queries and large databases, with the query time of Cassandra store for the largest database and most complex query being at least 2 times faster than the other NoSQL approaches and almost 2 times faster than the SQL approach as well. By using supercomputing resources (Figures 58-67), the query time of Cassandra and Redis for the 100 million-record database and the most complex query was almost 3 times faster than the MongoDB (NoSQL approach) and slightly faster than MySQL (SQL approach), even though the Cassandra and Redis had significantly more keys to process the updated non-predefined information because of its default design. These results are probably because NoSQL approaches, such as those based on Cassandra and Redis, have been built to manipulate large amounts of new information that include new headers. This is in contrast to SQL database

technologies that, even if they have been fully optimized, hardly ever deal with schema changes and, consequently, with large new-header updated data.

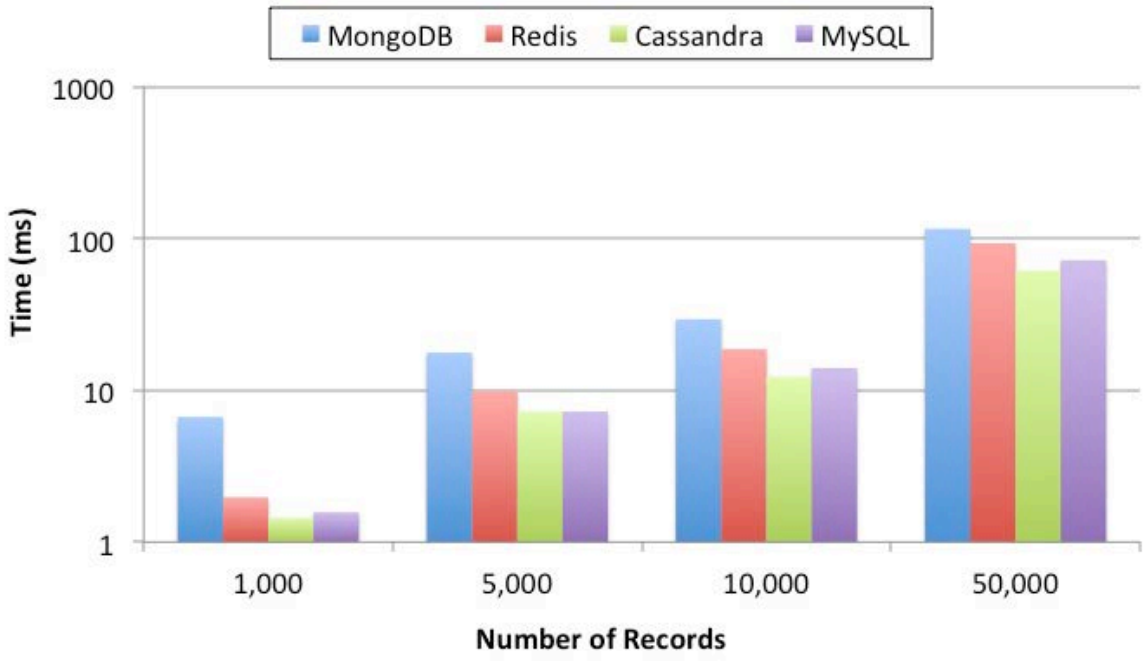

Figure 48. Variation in query time (base-10 log scale) of experiment 3: query 1.

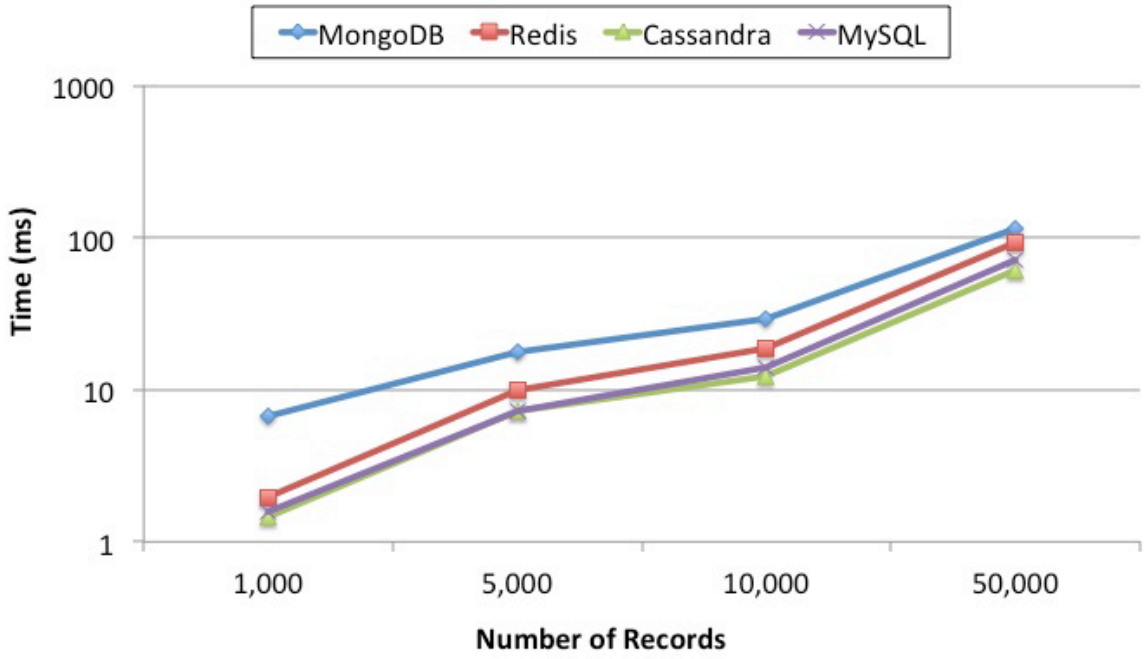

Figure 49. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 3: query 1.

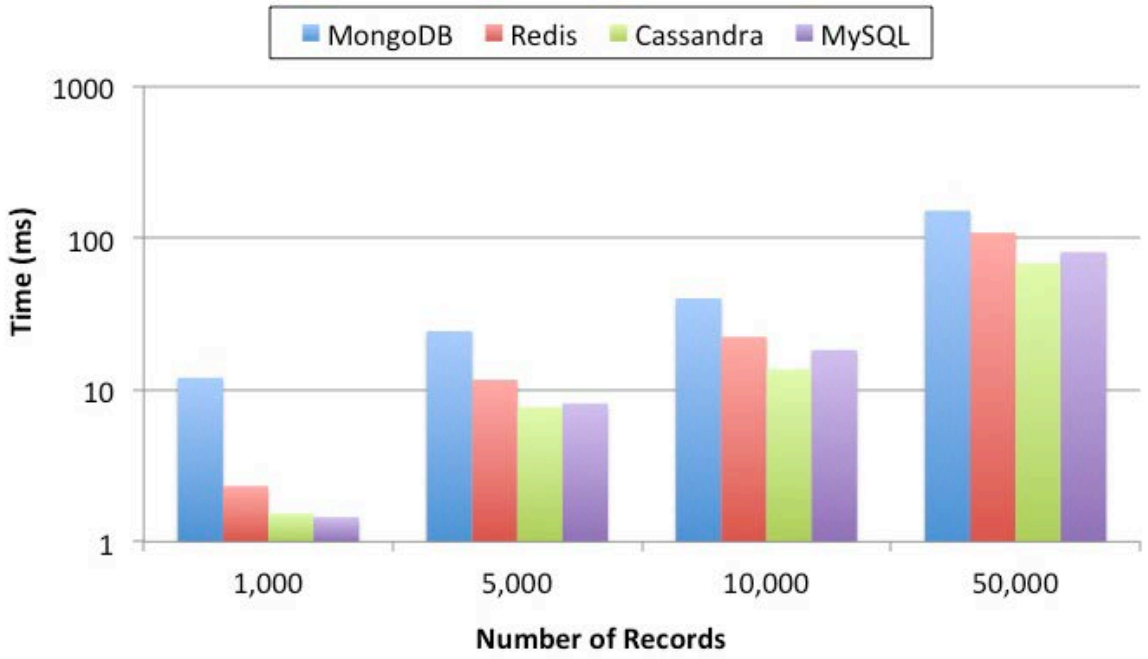

Figure 50. Variation in query time (base-10 log scale) of experiment 3: query 2.

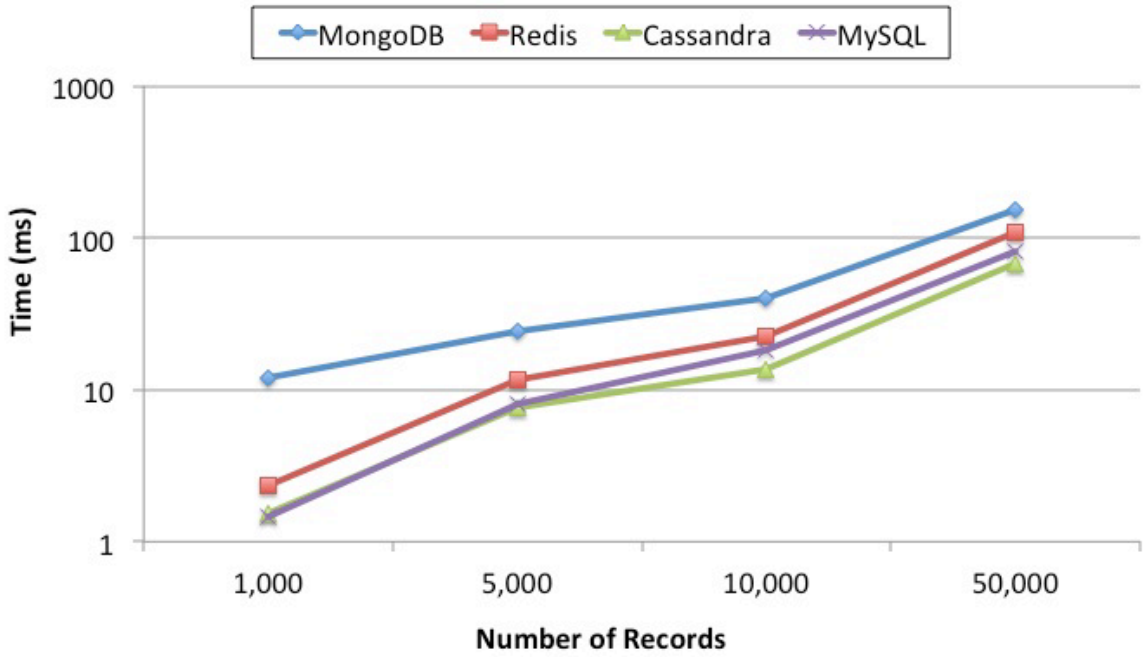

Figure 51. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 3: query 2.

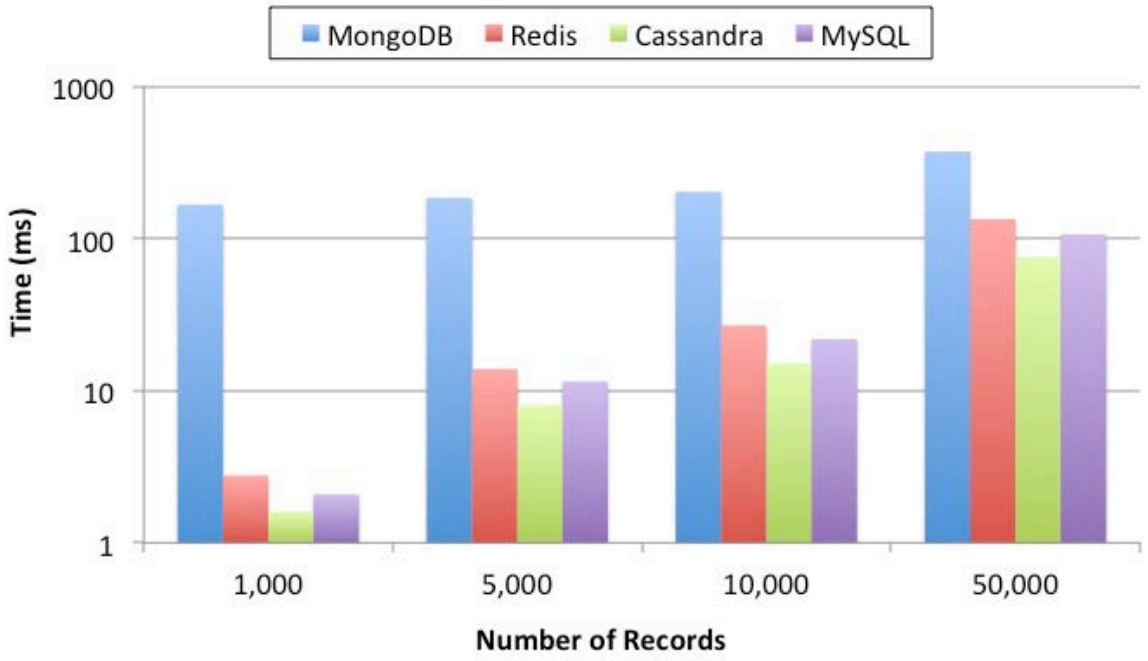

Figure 52. Variation in query time (base-10 log scale) of experiment 3: query 3.

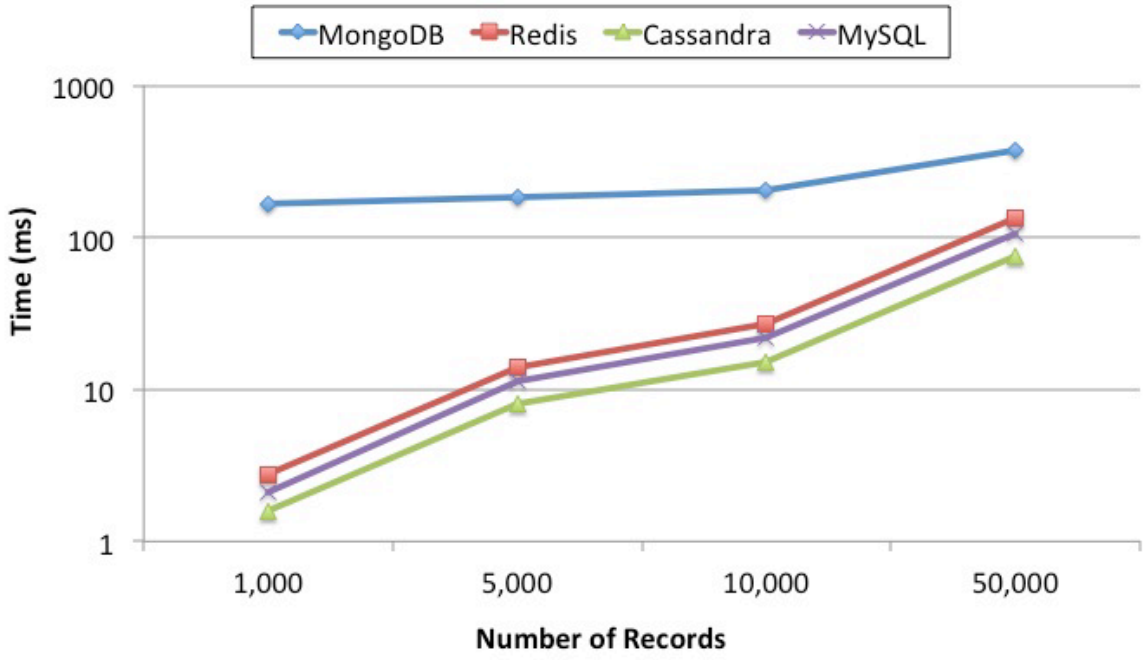

Figure 53. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 3: query 3.

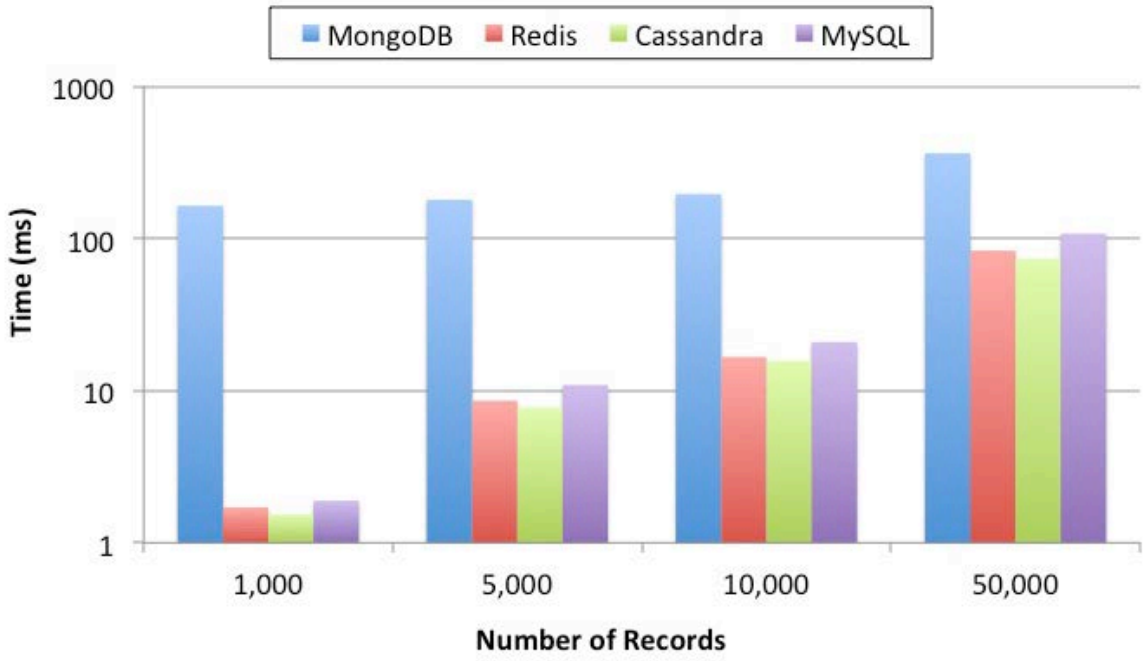

Figure 54. Variation in query time (base-10 log scale) of experiment 3: query 4.

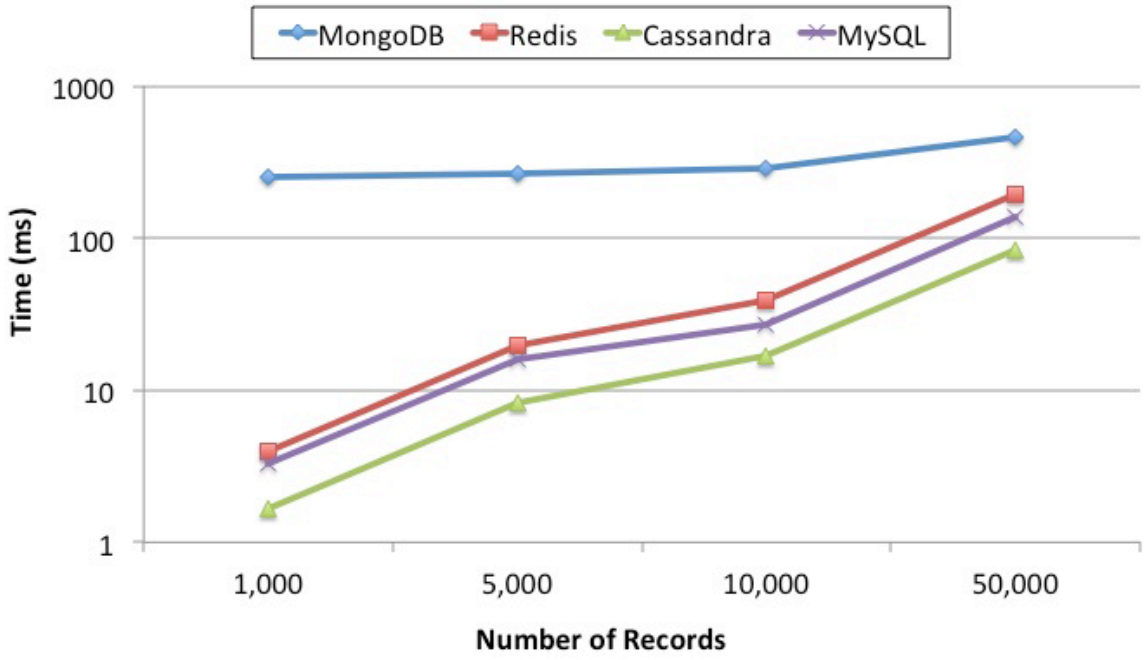

Figure 55. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 3: query 4.

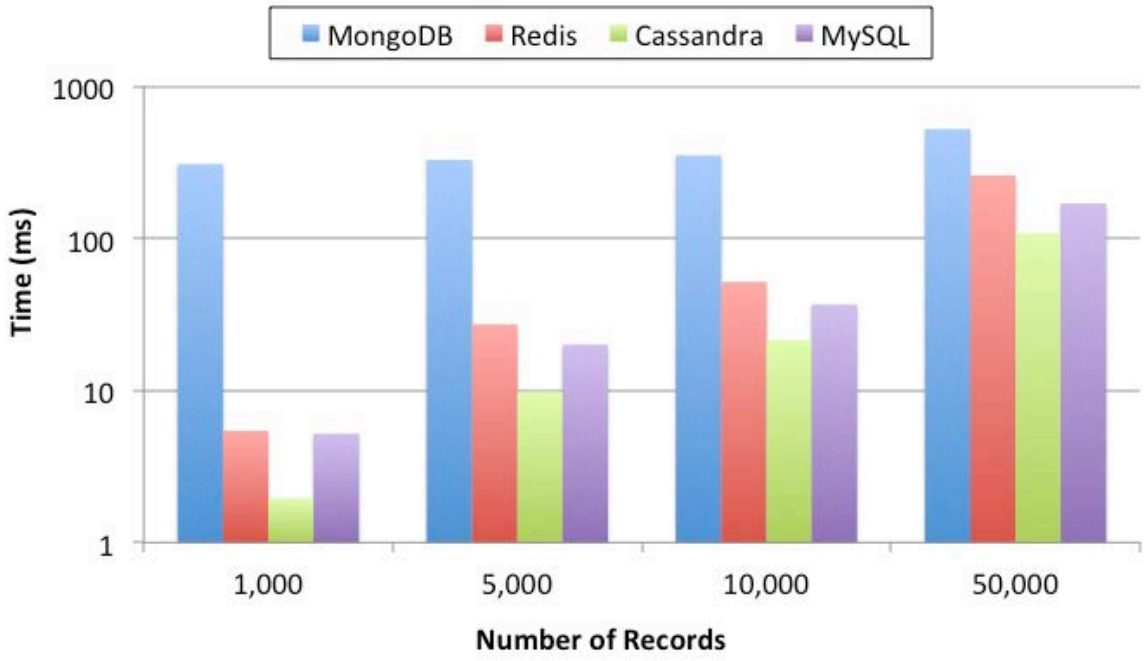

Figure 56. Variation in query time (base-10 log scale) of experiment 3: query 5.

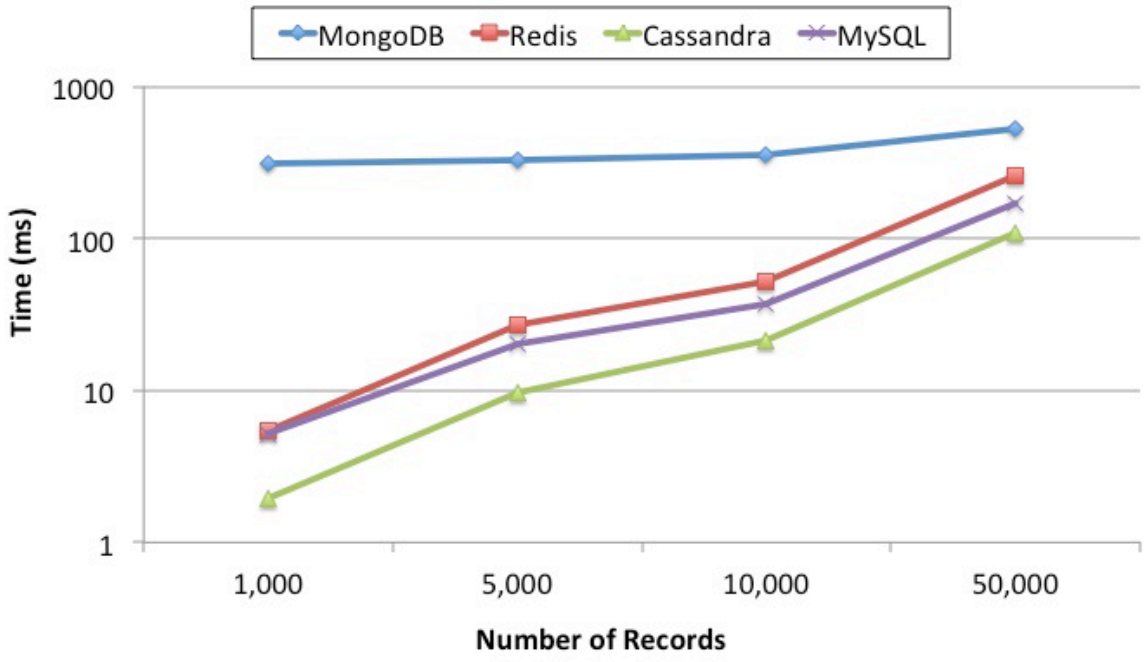

Figure 57. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 3: query 5.

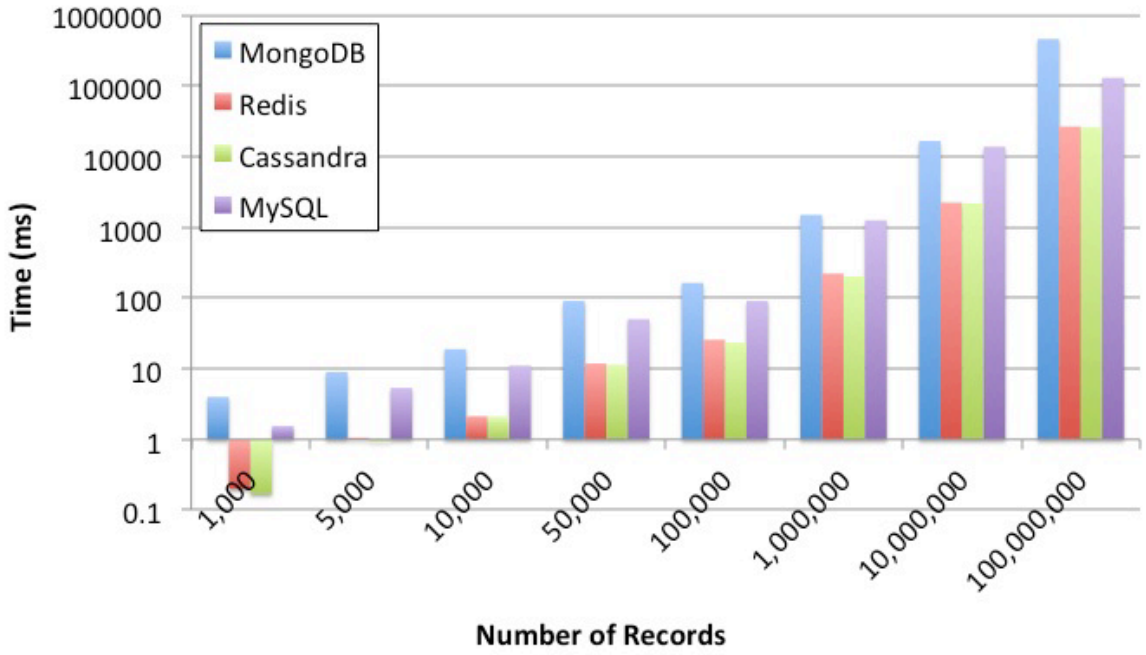

Figure 58. Variation in query time (base-10 log scale) of experiment 3 in larger databases: query 1.

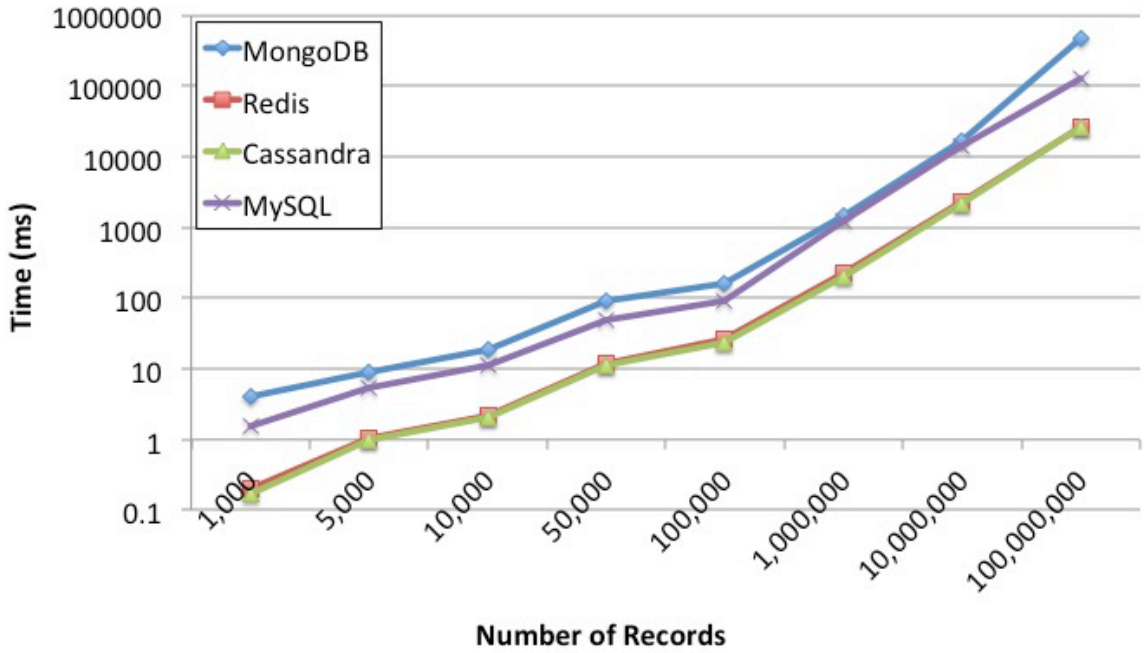

Figure 59. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 3 in larger databases: query 1.

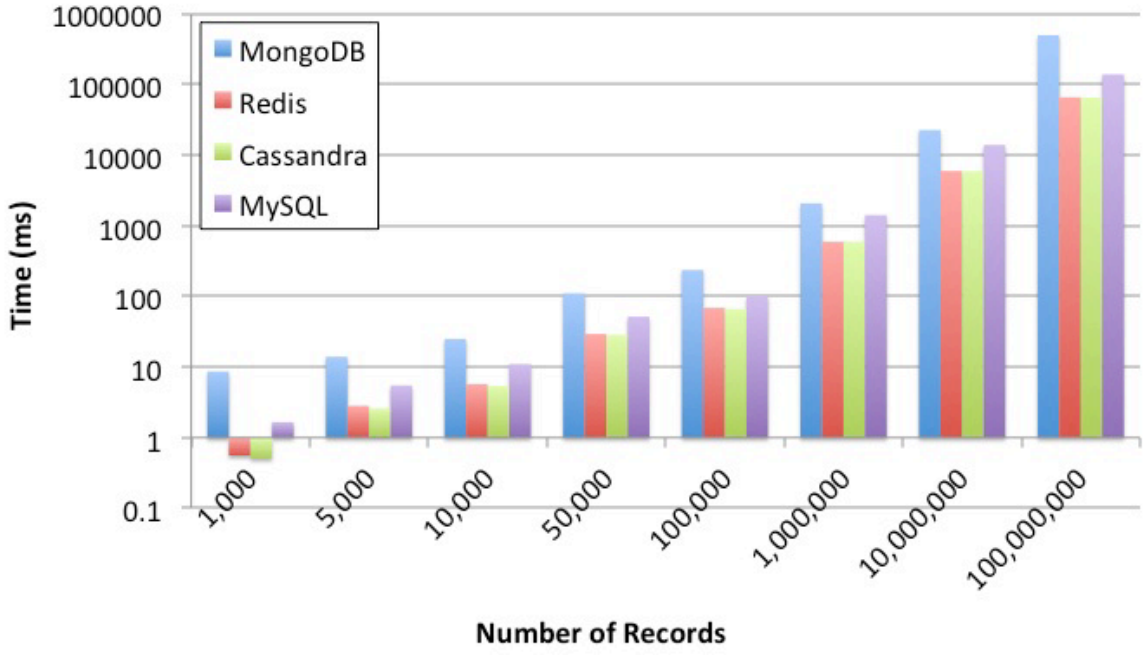

Figure 60. Variation in query time (base-10 log scale) of experiment 3 in larger databases: query 2.

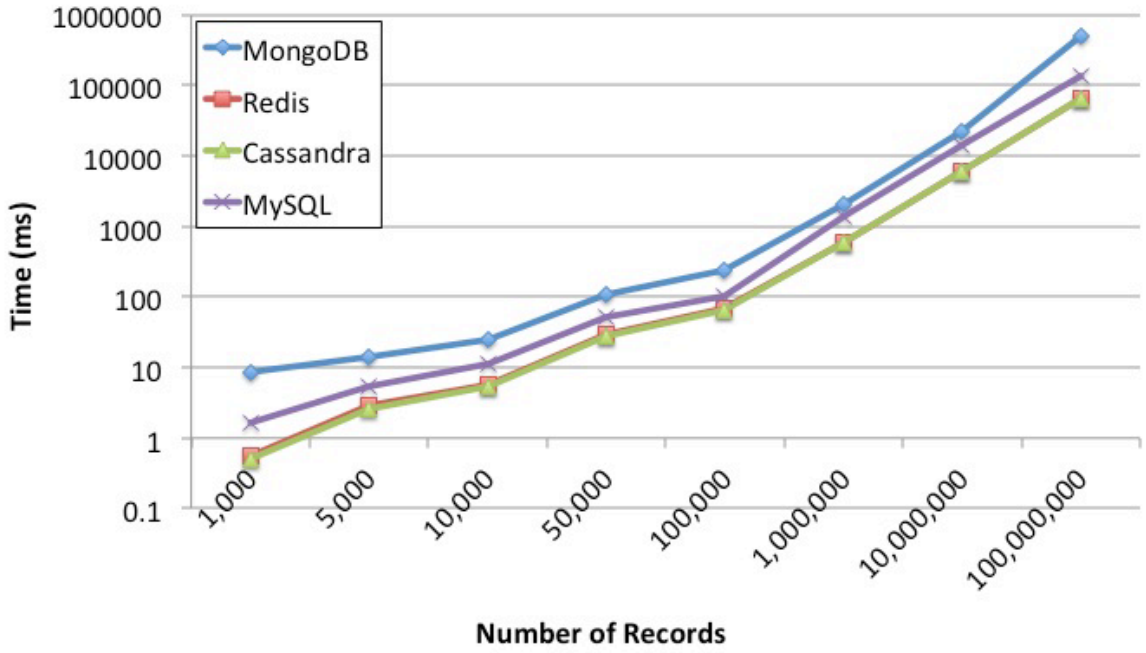

Figure 61. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 3 in larger databases: query 2.

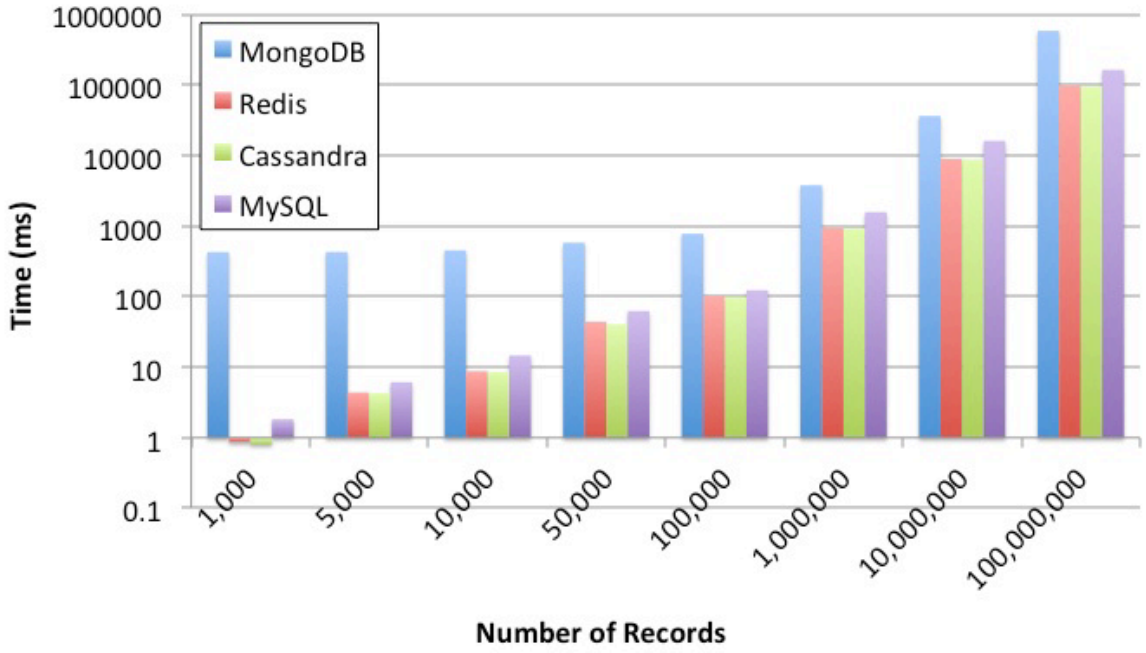

Figure 62. Variation in query time (base-10 log scale) of experiment 3 in larger databases: query 3.

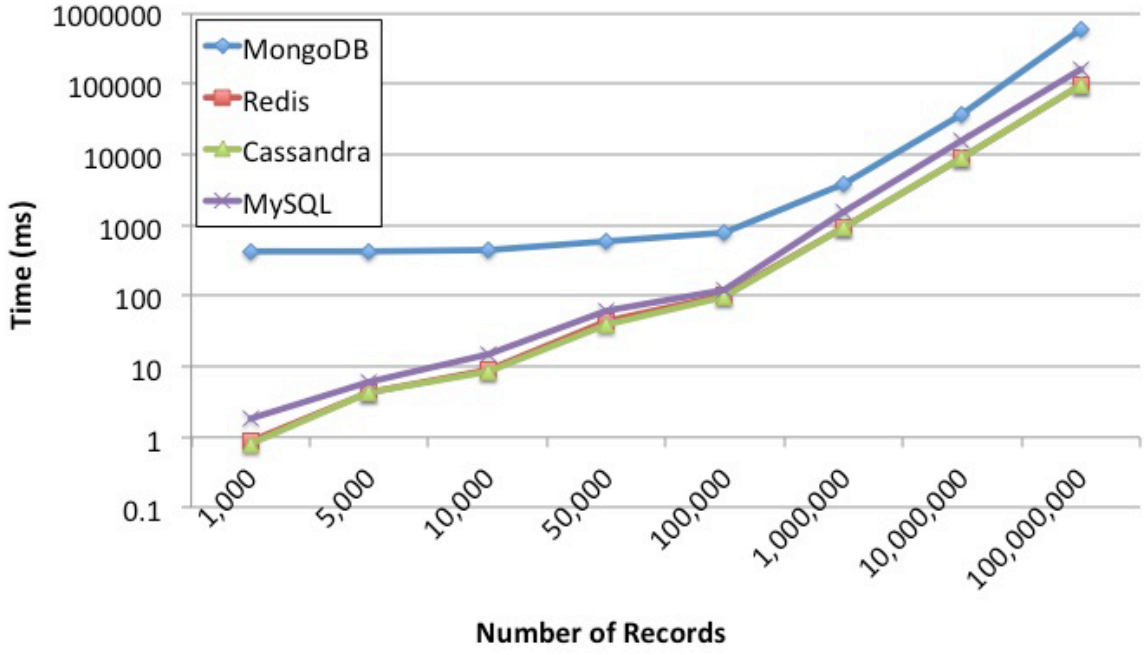

Figure 63. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 3 in larger databases: query 3.

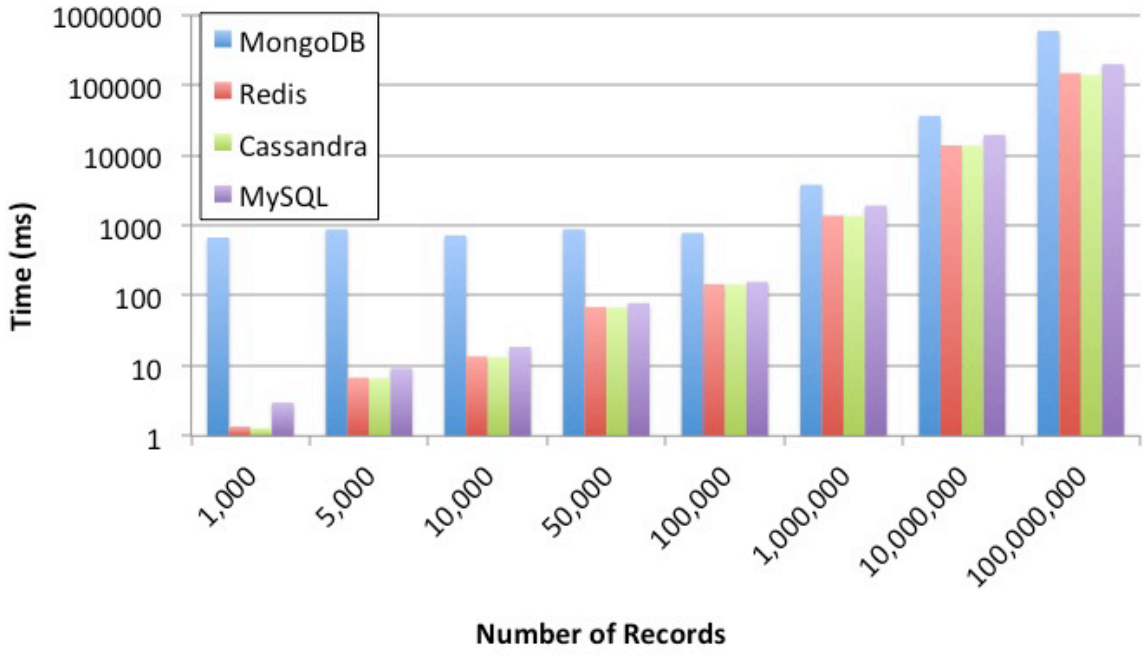

Figure 64. Variation in query time (base-10 log scale) of experiment 3 in larger databases: query 4.

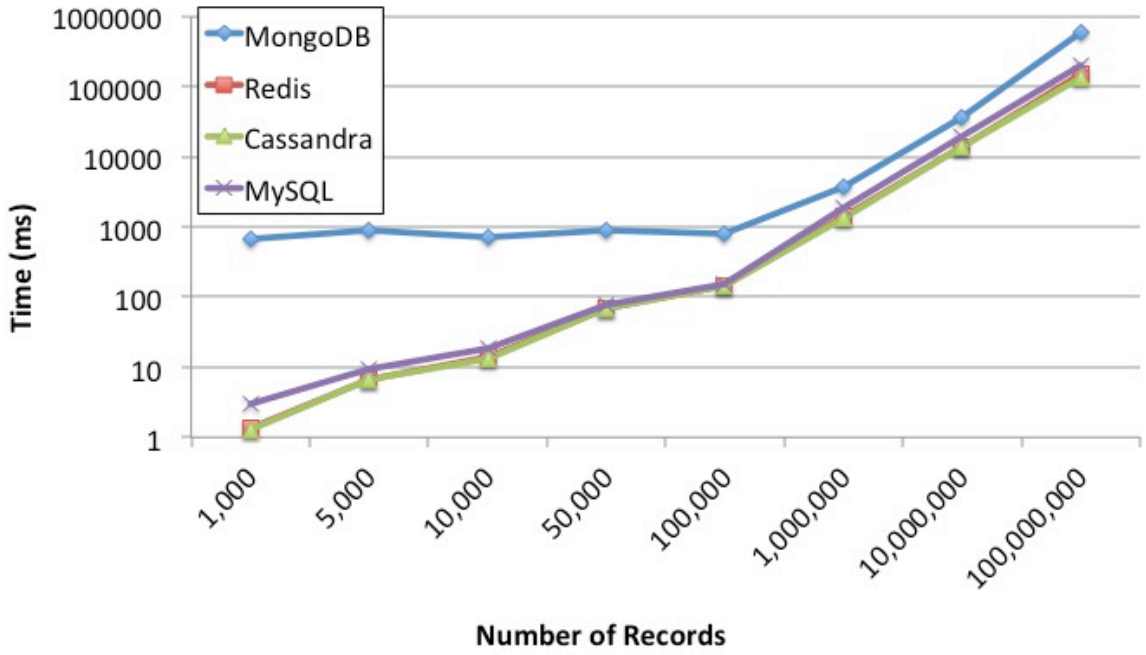

Figure 65. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 3 in larger databases: query 4.

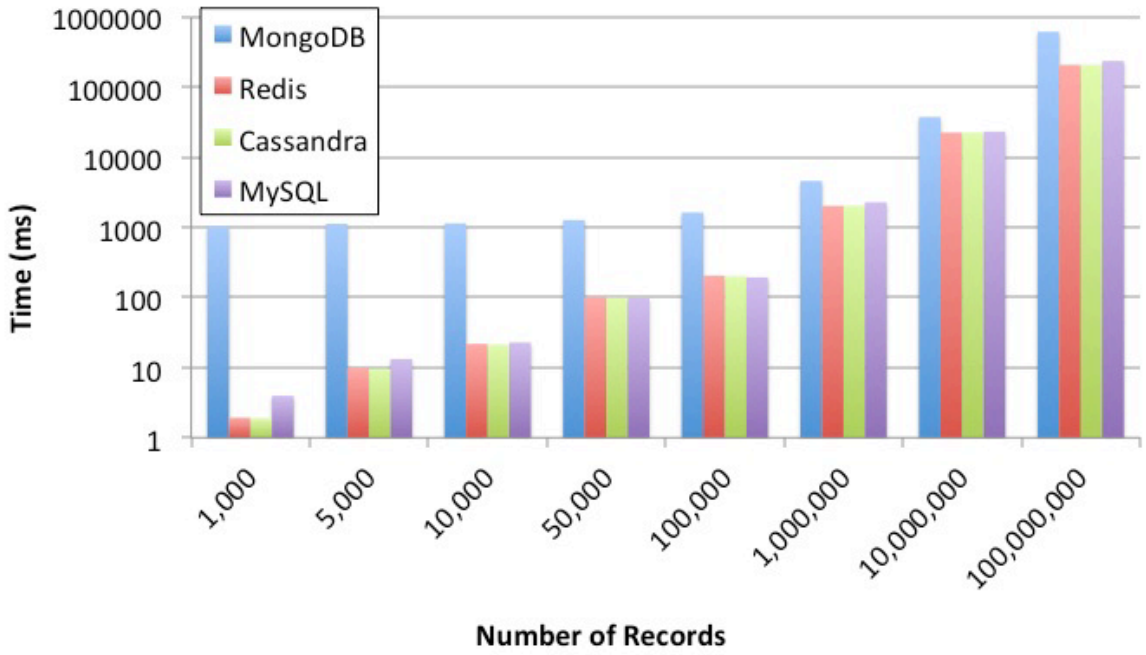

Figure 66. Variation in query time (base-10 log scale) of experiment 3 in larger databases: query 5.

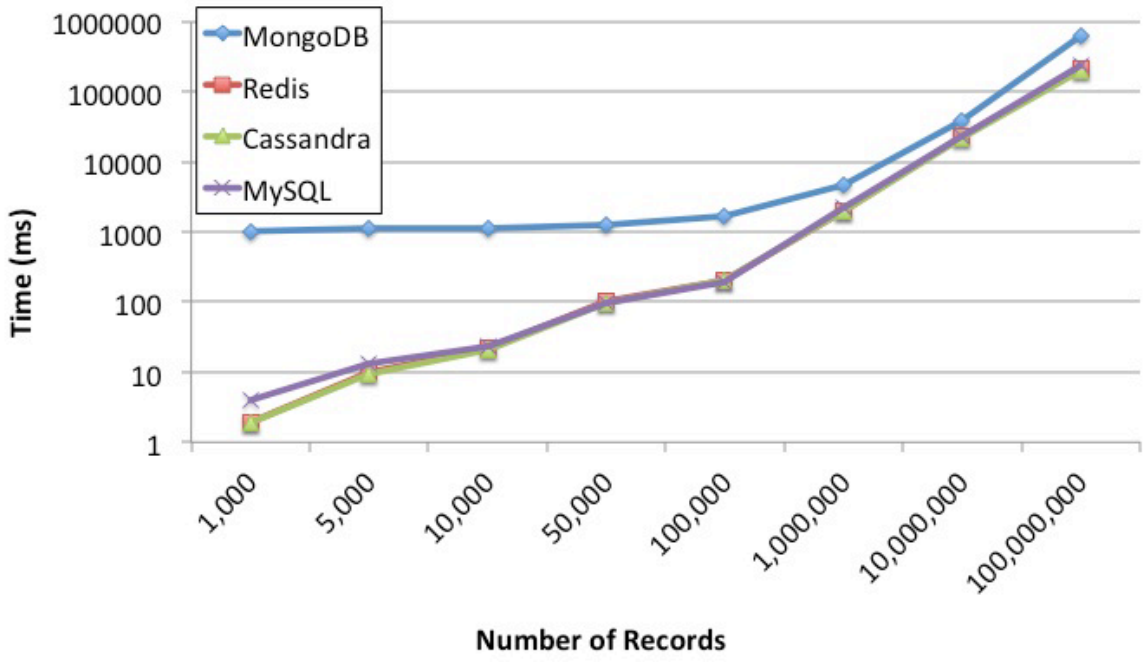

Figure 67. 2-D line graph showing the variation in query time (base-10 log scale) of experiment 3 in larger databases: query 5.

7.0 CONCLUSIONS AND FUTURE WORK

NoSQL database management systems appear to be are designed to handle the heterogeneous and dynamic data used in Precision Medicine. The goal of this work was to identify the most suitable database management system for the growing big data coming mostly from clinical and genomic information. In this process, we detailed needed measures that guided our decisions around the obstacles implied in recent technologies.

To understand performance and scalability even more clearly, experiments like those presented will continue to be necessary. We have presented three experiments for performance and scalability using database management systems that had not been applied in the way we applied them, to the Precision Medicine domain. Our experiments detail and evaluate database management system (NoSQL and SQL) characteristics to analyze their performance and scalability. Our experiments were able to measure performance with different updating processes, including database schema changes and non-schema changes, frequently occurring in Precision Medicine. Finally, we concluded which was the fastest DBMS and pointed out how its data orientation explains its advantage in handling big clinical and genomic data.

In this section, we present a brief overview of the key contributions, limitations and work presented in this dissertation. We then conclude with a summary of potential future research work that could stem from this project.

7.1 CONTRIBUTIONS

The main contributions of this dissertation are as follows:

Scalable measurements. We present novel experiments to measure different dimensions of performance and scalability (data size, system size). These methods draw on performance and scalability measurements of big databases, response times for querying different data that is growing in size. Results show that systems adapt invariantly toward upward database sizes.

Combined approaches. We combine novel performance and scalability methodologies with updating processes (with and without schema changes) to understand the adaptation of database systems to imminent computational procedures in biology, specifically in the clinical and genomic field. These methods take into account performance and scalability measurements of big databases and response times for querying different data with active updating processes. Query times from the used systems resulted invariant regarding growing database sizes.

Conglomerated definition. We present a combined concept of Precision Medicine based on available data. Databases included highly personal information from demographic, clinical and genomic data to indicate a set of suggested data in Precision Medicine to use in the future in Healthcare. This concept is based on a broad definition of the structure of human populations, medical information contained in the EHR, and genome sequencing.

Data structured method. We arranged the Precision Medicine data, specifically PGRR-TCGA information, into five different data models (document, key-value, column, graph and table store) through a series of text mapping and manipulating processes. These novel structures address the lack of semantic interoperability for EHR and genomic data, since in the U. S. it seems that there are no standards for semantic interoperability of health care data, nor for demographic, clinical or genomic data (our proposed Precision Medicine data concept); there are

only syntactic standards in clinical data (such as the HL7 v3 RIM). In other words, there is no standard format that has a definition for Precision Medicine data. Thus, our proposal addresses this fundamental issue in Precision Medicine.

Potential standardization. Results from these studies have the potential to identify the NoSQL approach most suited to the available Precision Medicine data (data size) and system resources (system size). This feature makes our experiments worth using to further evaluate the performance and scalability of any type of data.

Broad availability. We have integrated our novel designs and concepts to allow further comparisons of our approaches to other NoSQL technologies (i.e., Riak, CouchDB, among others) using our conglomerated definition of Precision Medicine. In other words, scientists will be able to use our methodology to evaluate nascent NoSQL systems and suggest new systems that can better adapt themselves for management of the growing big data in Precision Medicine.

Public Health Relevance. Our project has a wider public health impact since we are addressing a possible solution, using NoSQL technologies such as Column and Key-Value stores (Cassandra and Redis), to the effective management of Precision Medicine data. This solution is an attempt to decrease the high demands on the Healthcare system by bringing us closer to cures to diseases such as cancer. Our results promote the integration of data, including science, genomic, biotechnology, and medical records, which integration is the basic aim of the new Precision Medicine initiative attempting to revolutionize the US Healthcare system.

7.2 CONCLUSIONS

A review of this dissertation based on the evaluation of performance, scalability and updating processes in different data management systems indicates that NoSQL Cassandra and Redis are viable alternate to the relational database approach as they provide better query performance, scalability and flexibility in updating processes, qualities needed for the intense data filtration and schema modification required in the Precision Medicine field. Meanwhile, the use of MongoDB databases is a promising solution to handle big clinical and genomic data because of its portability that allow data sharing and accessing (human readable) easily; however, more work is needed to deal with performance issues so as to accelerate its functionality (query performance and scalability) in big data management.

The performance and scalability of NoSQL databases were here evaluated using real clinical and genomic data from real patients under the TCGA project. We conclude that the use of NoSQL technologies such as Cassandra and Redis is advantageous, given their lack of need for a schema and updating processes (without worrying about a table mapping framework). NoSQL approaches, especially Cassandra and Redis (Tables 33-34), provide better query performance, scalability and updating flexibility than other DBMS, enabling effective data management in Precision Medicine. In general, MySQL were found to be flexible for handling clinical and genomic data but fell short in terms of performance and scalability comparing to NoSQL based on Cassandra and Redis; they also experienced difficulties dealing with updating processes, which are common procedures in Precision Medicine.

Table 33. DBMSs with the lowest query times according to different database size and complex queries using standard computing resources.

Query No.	1,000 records	5, 000 records	10,000 records	50,000 records
Experiment 1				
1	Redis/Cassandra	Redis/Cassandra	Redis/Cassandra	Redis/Cassandra
2	Redis/Cassandra	Redis/Cassandra	Redis/Cassandra	Redis/Cassandra
3	Redis/Cassandra	Redis/Cassandra	Redis/Cassandra	Redis/Cassandra
4	Cassandra	Cassandra	Cassandra	Cassandra
5	Cassandra	Cassandra	Cassandra	Cassandra
Experiment 2				
1	Redis/Cassandra	Redis/Cassandra	Cassandra/Redis	Cassandra/Redis
2	Redis/Cassandra	Redis/Cassandra	Redis/Cassandra	Redis/Cassandra
3	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
4	Cassandra	Cassandra	Cassandra	Cassandra
5	Cassandra	Cassandra	Cassandra	Cassandra
Experiment 3				
1	Cassandra/MySQL	Cassandra/MySQL	Cassandra/MySQL	Cassandra
2	MySQL	Cassandra/MySQL	Cassandra/MySQL	Cassandra
3	Cassandra	Cassandra	Cassandra	Cassandra
4	Cassandra	Cassandra	Cassandra	Cassandra
5	Cassandra	Cassandra	Cassandra	Cassandra

Table 34. DBMSs with the lowest query times according to different database size and complex queries using supercomputing resources.

Query No.	1,000 records	5, 000 records	10,000 records	50,000 records
Experiment 1				
1	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
2	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
3	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
4	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
5	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
Experiment 2				
1	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
2	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
3	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
4	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
5	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
Experiment 3				
1	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
2	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
3	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
4	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
5	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis

Query No.	100,000 records	1 million records	10 million records	100 million records
Experiment 1				
1	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
2	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
3	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
4	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
5	MySQL	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
Experiment 2				
1	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
2	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
3	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
4	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
5	Cassandra/MySQL	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
Experiment 3				
1	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
2	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
3	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
4	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis
5	MySQL/Cassandra	Cassandra/Redis	Cassandra/Redis	Cassandra/Redis

We found that we could anticipate the most suitable DBMS in performance and scalability terms by knowing the filtration process for finding patients with a specific pattern. It seems that by knowing in advance the expected results from queries, we could select an appropriate DBMS for specific patients' matching process. For example, by using 1,000 to 50,000 records, in queries that match less than 10 data patterns (i.e., demographic and clinical characteristics) with a high number of matching records expected (i.e., 7 in every 25 records) (Tables 35-46), Redis seems to be the technology that could show the lowest query times by using standard computing resources and Cassandra the one by using supercomputing resources. However, in the same range of patients but with a more stringent filtration process (complex queries) with a lower number of matching records expected, Cassandra seems to be the best technology to deal with these higher filtration process using standard computing resources and Cassandra and Redis by using supercomputing resources. In summary, it appears that Redis can slightly handle a higher amount of record's matches than Cassandra using standard computing resources, highlighting it as the ideal choice for simple queries; however, both technologies are ideal to handle large quantity of matches using supercomputing resources. On the other hand, Cassandra seems to be ideal for high levels of filtration processes (complex queries) that are expected to return a fewer number of matches using both, standard computing and supercomputing resources. Overall, Cassandra seems to deal better with lower amounts of matched data than Redis.

In general, we noticed that Cassandra increases query performance by reducing the number of requests as a result of its data storage orientation. This column design appears to advantage in performance and scalability compared to other NoSQL approaches in managing clinical and genomic data because of its capacity to seek small data components placed in

columns as opposed to reading simply row by row like Redis. Thus, at some point clinical and genomic information can be placed in columns so that we can take advantage of using this column-oriented design. In addition, as this technology is used in data warehouses, it seems this technology is more likely to effectively handle big clinical and genomic data used in Precision Medicine, as shown in Figures 68-97.

We identify that MongoDB stores reduce the time to input clinical and genomic information from patients into to a database, especially when JSON files are structured accordingly. It shows to advantage when handling hierarchically complex information such as clinical and genomic data. MongoDB display higher portability since they are human readable. Thus, they are worth considering in the future, once their performance issues are resolved because its characteristics that allows effective storage and management of clinical and genomic data. In summary, though MongoDB shares characteristics similar to the other two NoSQL approaches, they fall short in terms of query performance and scalability, as shown in Figures 68-97.

We found that making queries using MongoDB is less effective compared to using the MySQL since the relational model used by big corporations is already integrated among systems and applications but undergone thorough optimization throughout the years. As MongoDB are recent technologies, more work is needed in the design of large databases. The future use of this specific NoSQL approach is hindered by the familiarity of current database managers with most popular SQL approaches.

This dissertation attempts to evaluate the currently available NoSQL approaches using most of the available data models. We focused on clinical and genomic information from patients with cancer, retrieving biological sense data relevant for the oncology field. The

feasibility of using NoSQL approaches in other medical fields, such as immunology, could be tested by looking at performance and scalability in retrieving different clinical and genomic information. Finally, the recent versions of NoSQL databases could also affect database performance since in the future performance needs may be detected and improved by particular NoSQL developers.

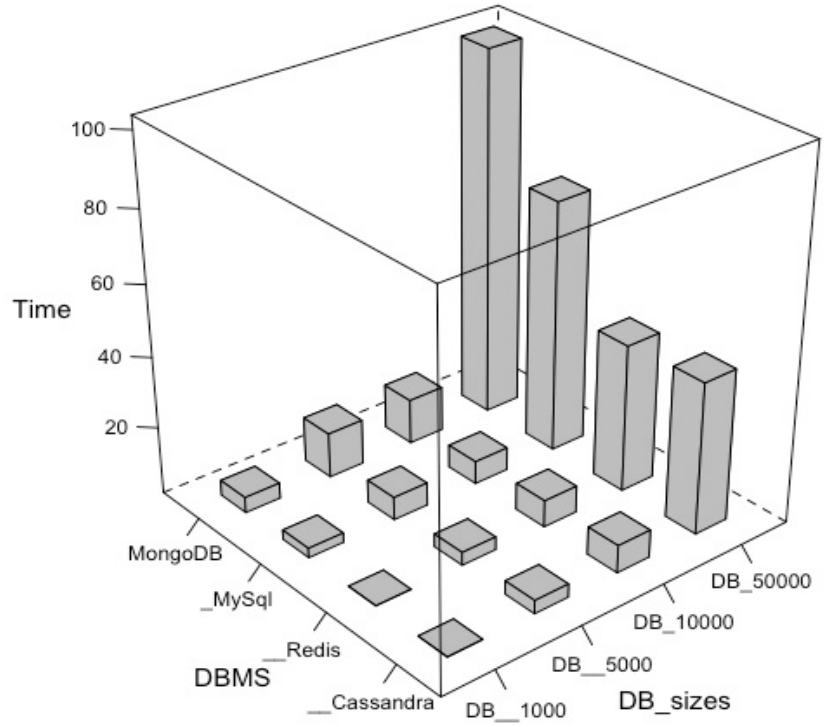

Figure 68. Variation in query time of experiment 1: query 1.

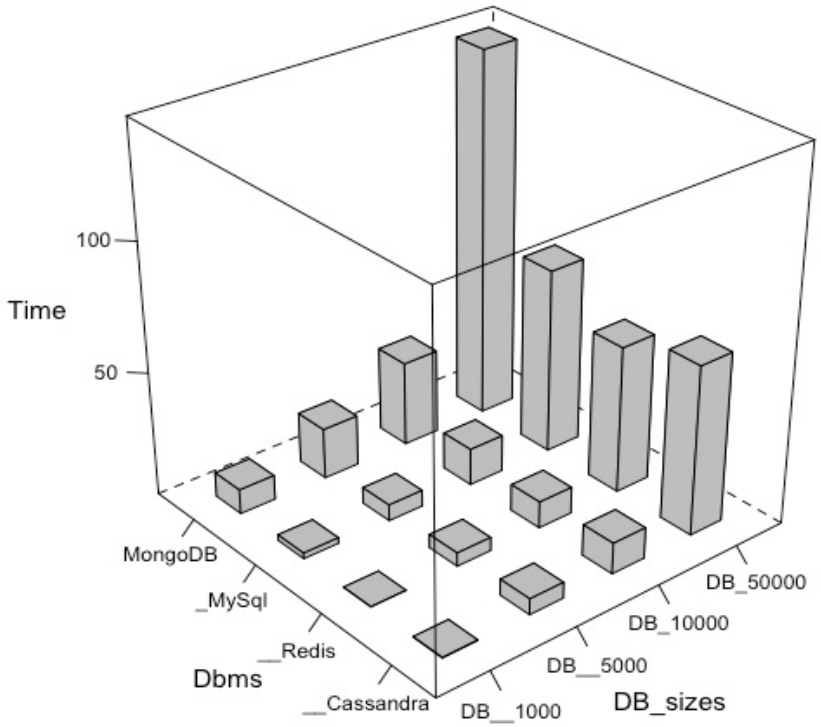

Figure 69. Variation in query time of experiment 1: query 2.

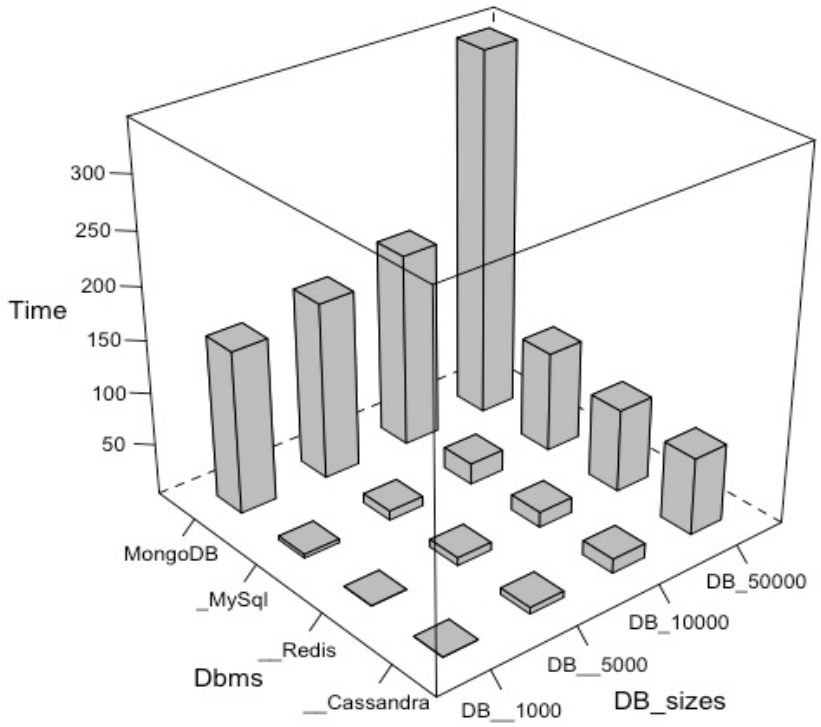

Figure 70. Variation in query time of experiment 1: query 3.

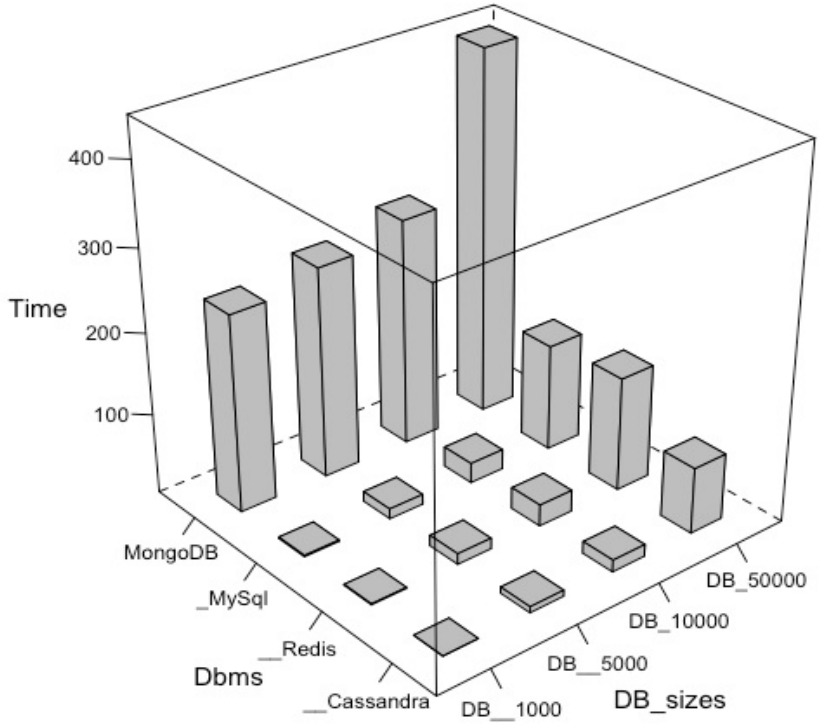

Figure 71. Variation in query time of experiment 1: query 4.

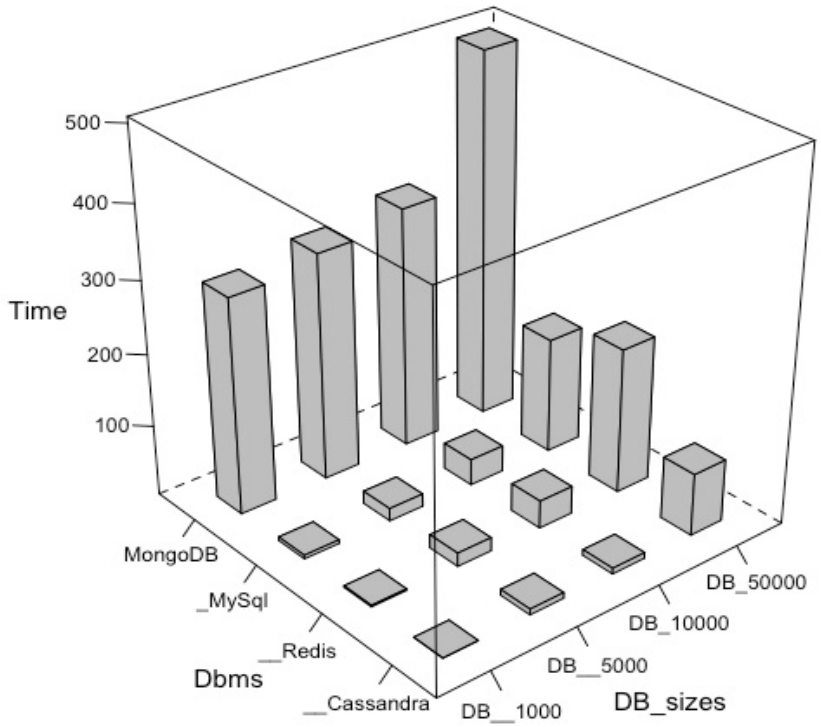

Figure 72. Variation in query time of experiment 1: query 5.

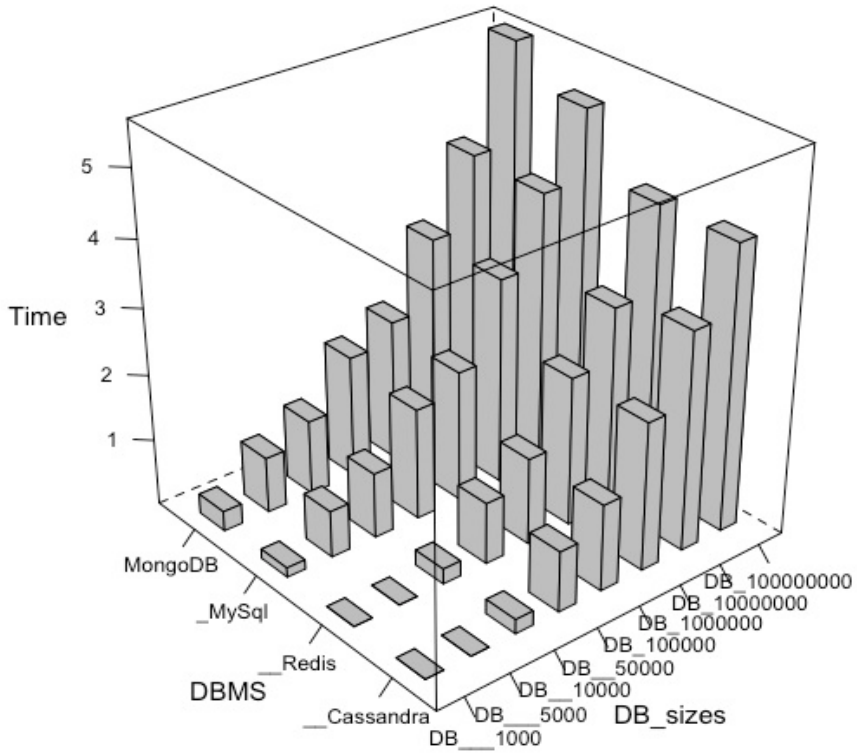

Figure 73. Variation in query time (base-10 log scale) of experiment 1 on larger databases: query 1.

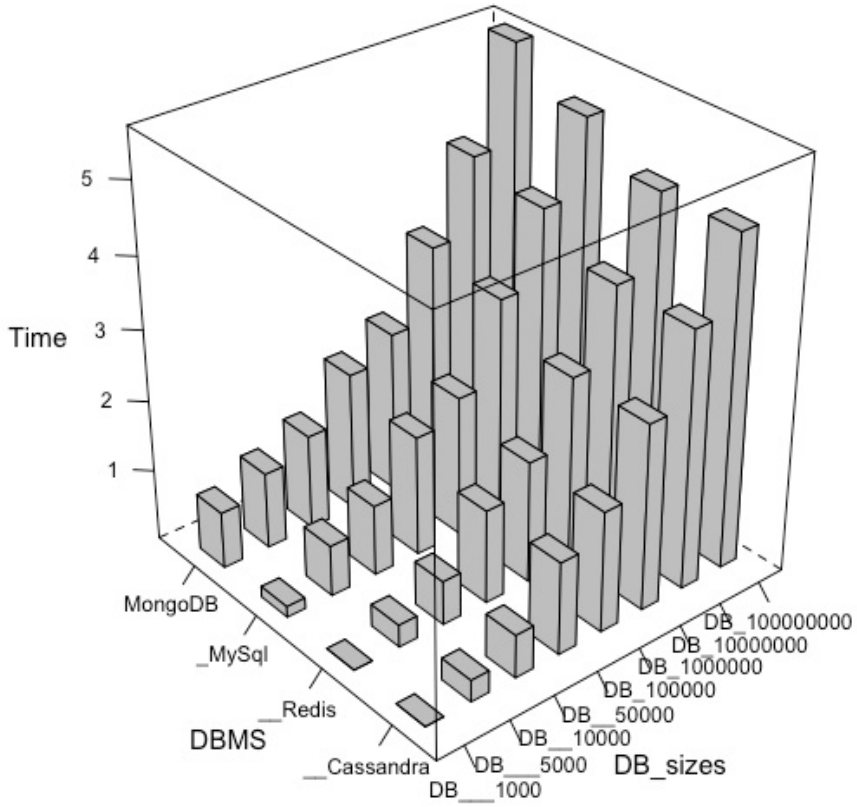

Figure 74. Variation in query time (base-10 log scale) of experiment 1 on larger databases: query 2.

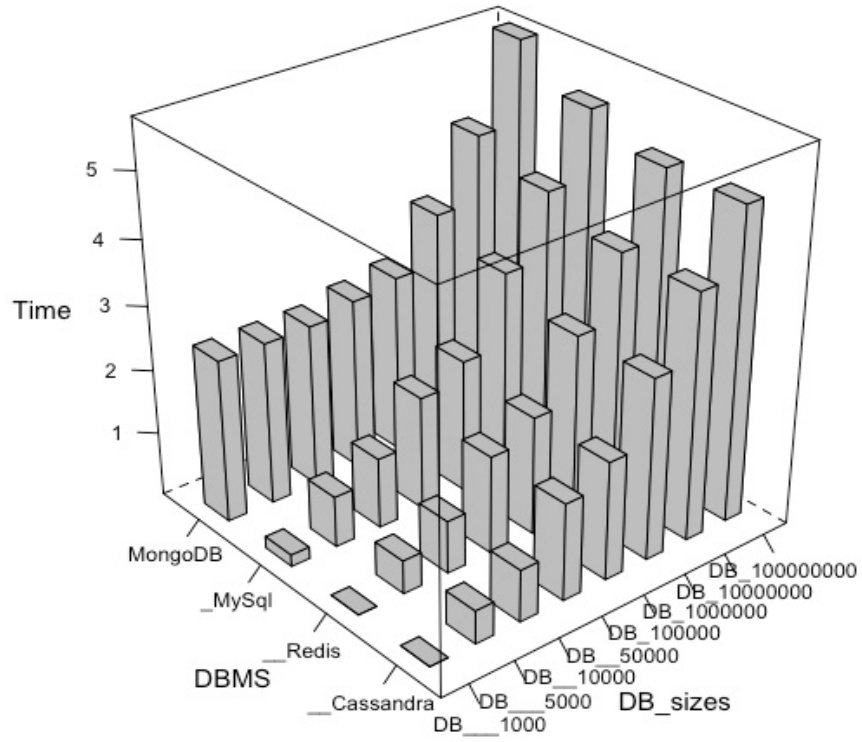

Figure 75. Variation in query time (base-10 log scale) of experiment 1 on larger databases: query 3.

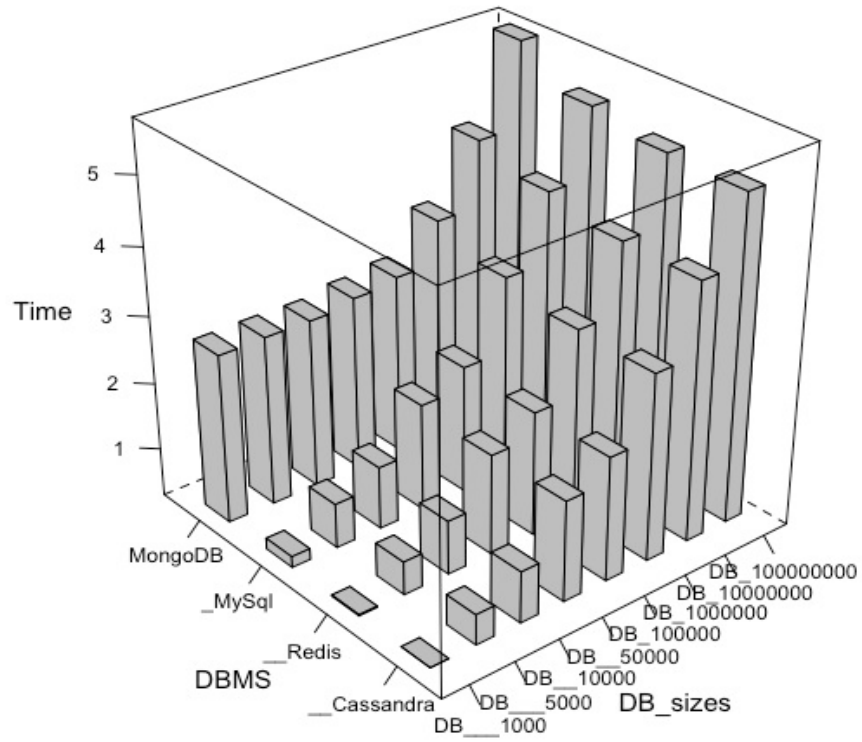

Figure 76. Variation in query time (base-10 log scale) of experiment 1 on larger databases: query 4.

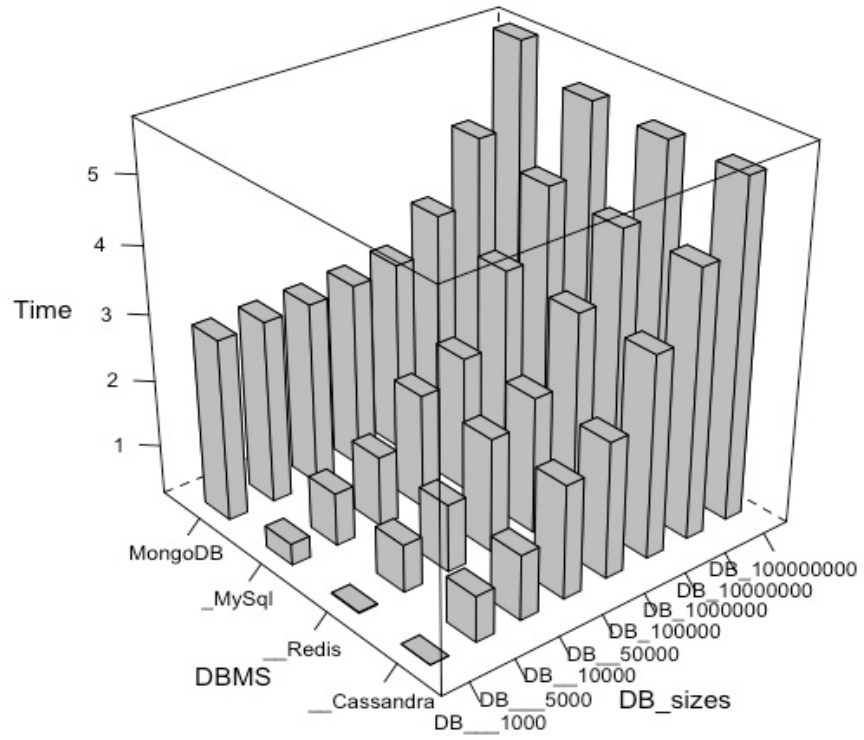

Figure 77. Variation in query time (base-10 log scale) of experiment 1 on larger databases: query 5.

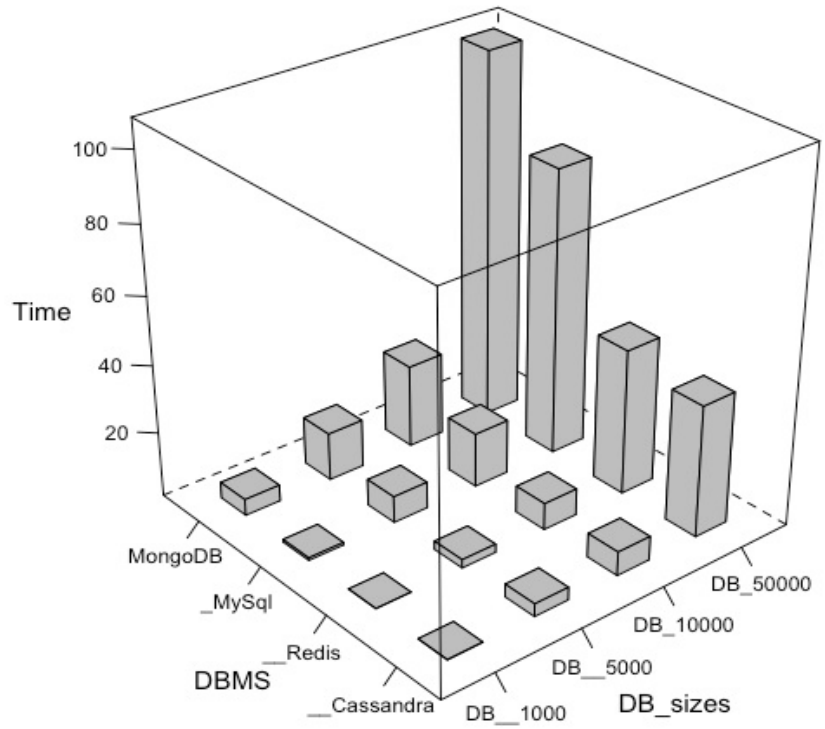

Figure 78. Variation in query time of experiment 2: query 1.

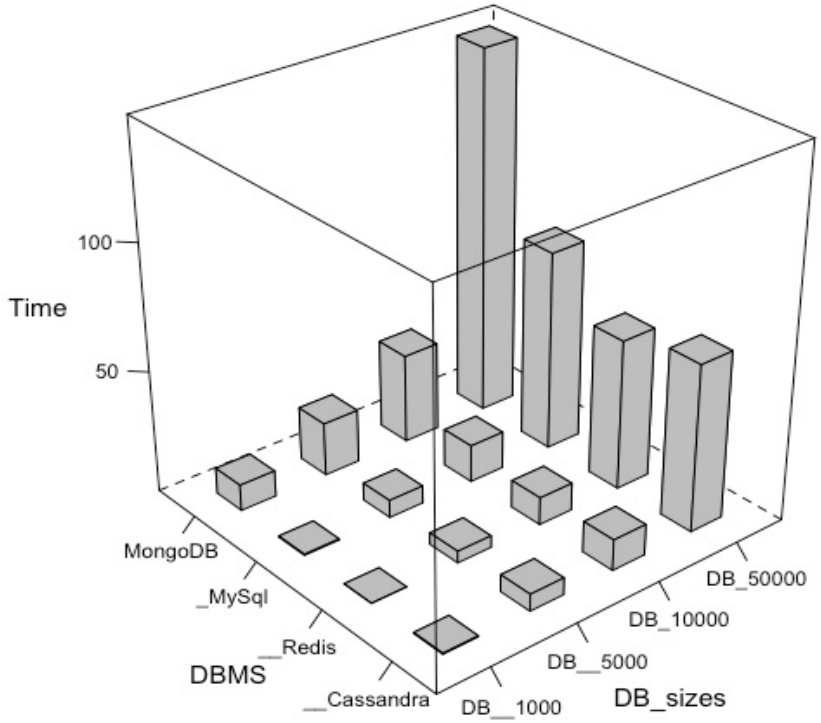

Figure 79. Variation in query time of experiment 2: query 2.

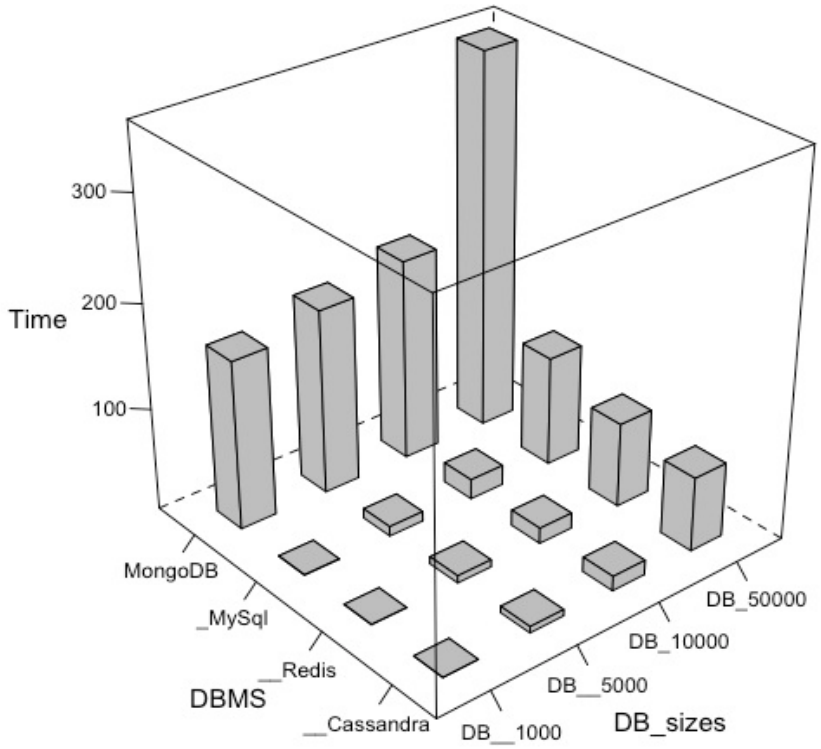

Figure 80. Variation in query time of experiment 2: query 3.

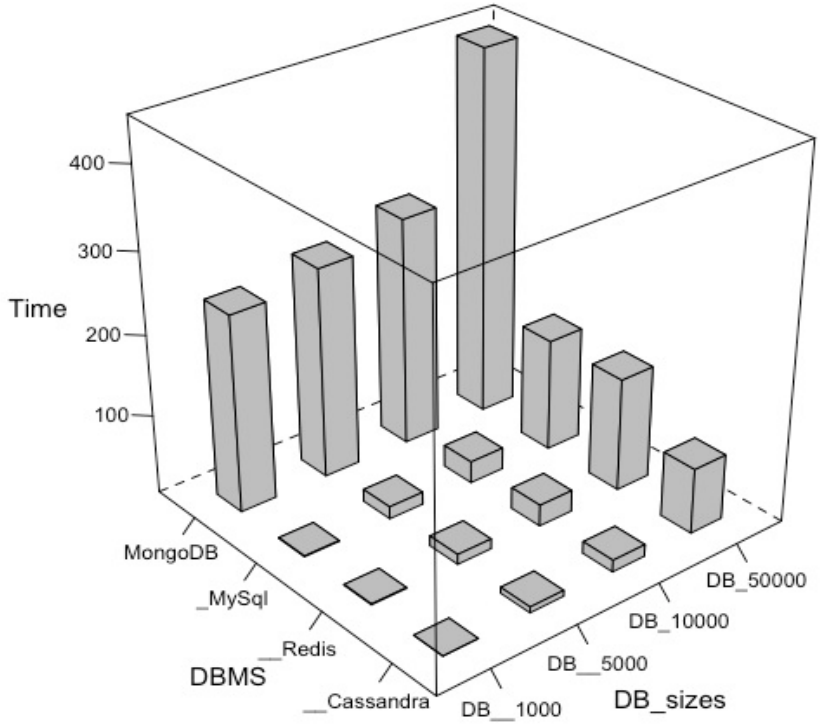

Figure 81. Variation in query time of experiment 2: query 4.

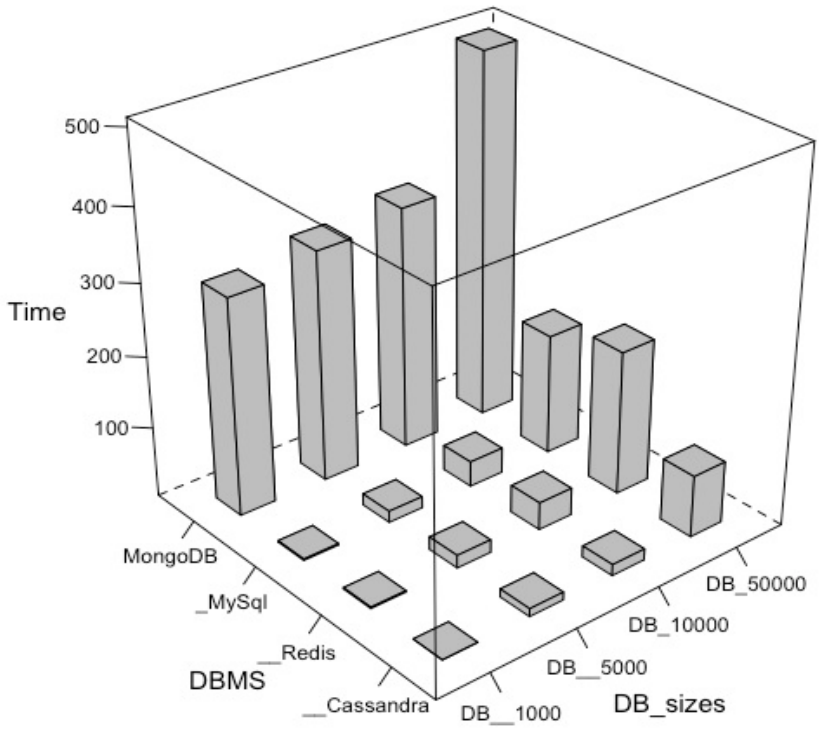

Figure 82. Variation in query time of experiment 2: query 5.

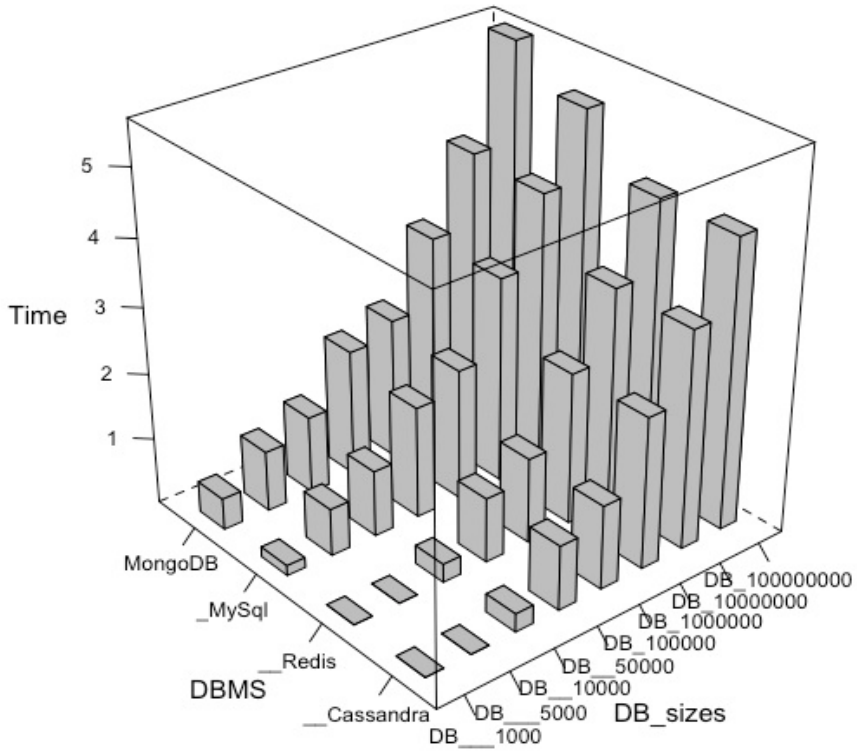

Figure 83. Variation in query time (base-10 log scale) of experiment 2 on larger databases: query 1.

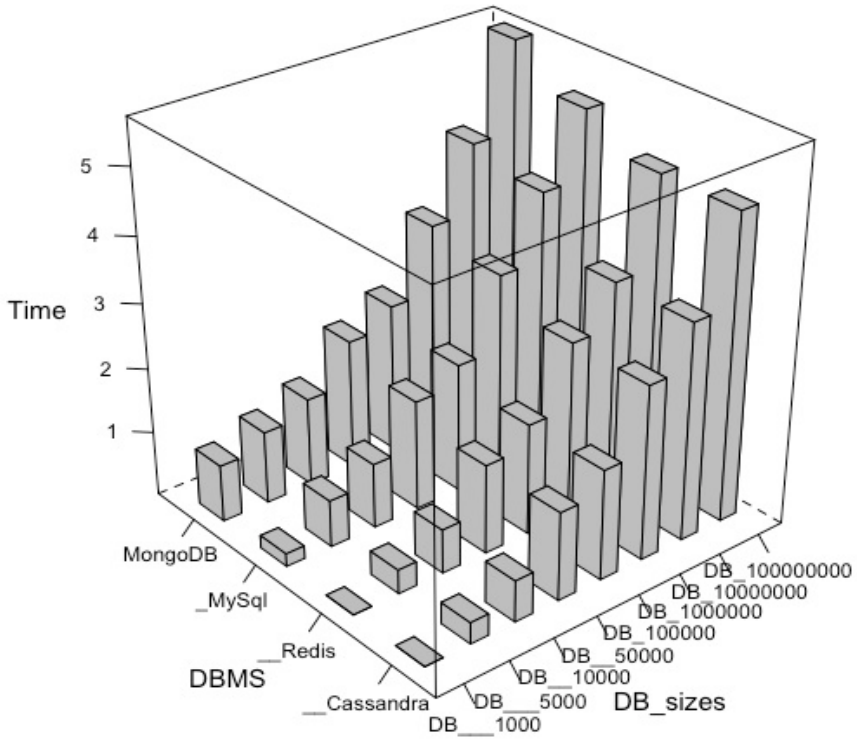

Figure 84. Variation in query time (base-10 log scale) of experiment 2 on larger databases: query 2.

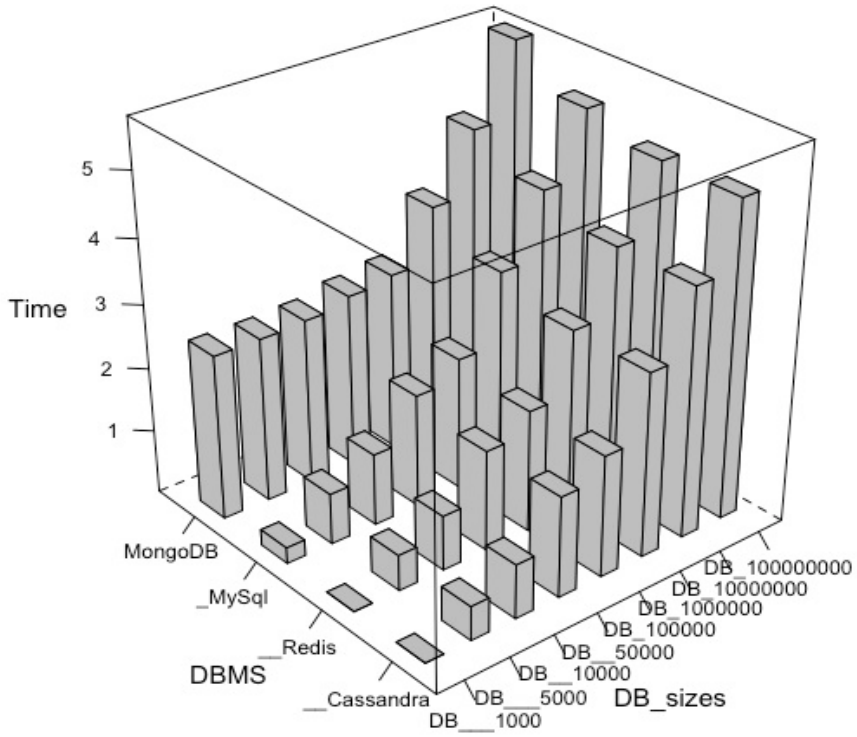

Figure 85. Variation in query time (base-10 log scale) of experiment 2 on larger databases: query 3.

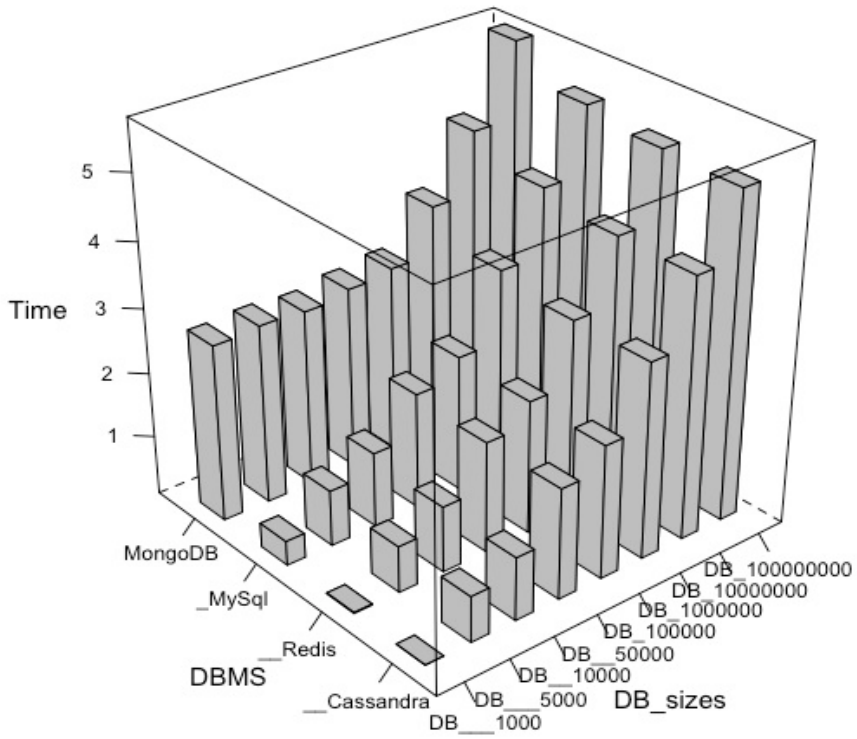

Figure 86. Variation in query time (base-10 log scale) of experiment 2 on larger databases: query 4.

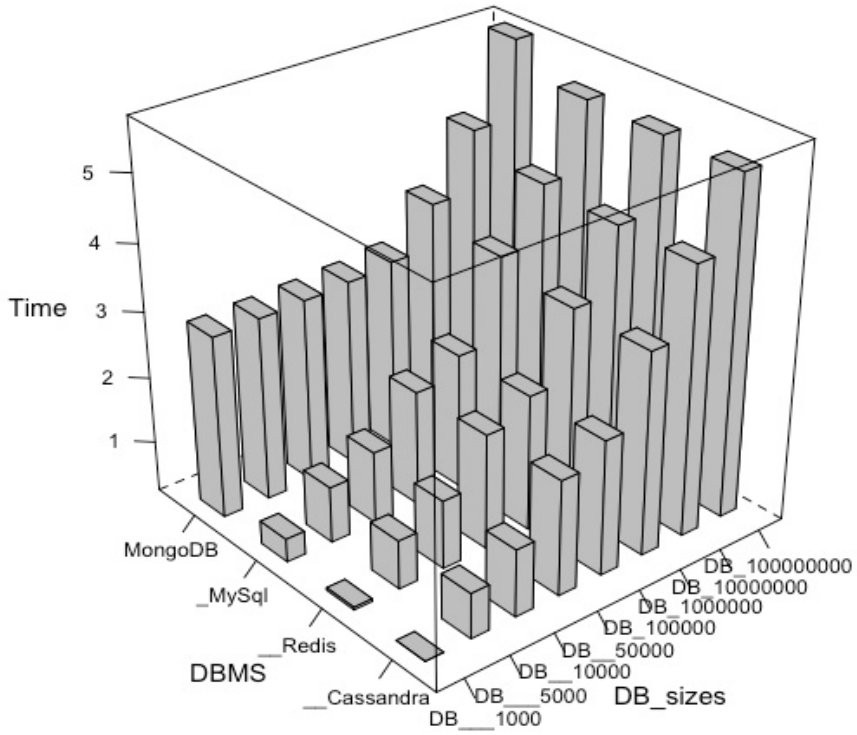

Figure 87. Variation in query time (base-10 log scale) of experiment 2 on larger databases: query 5.

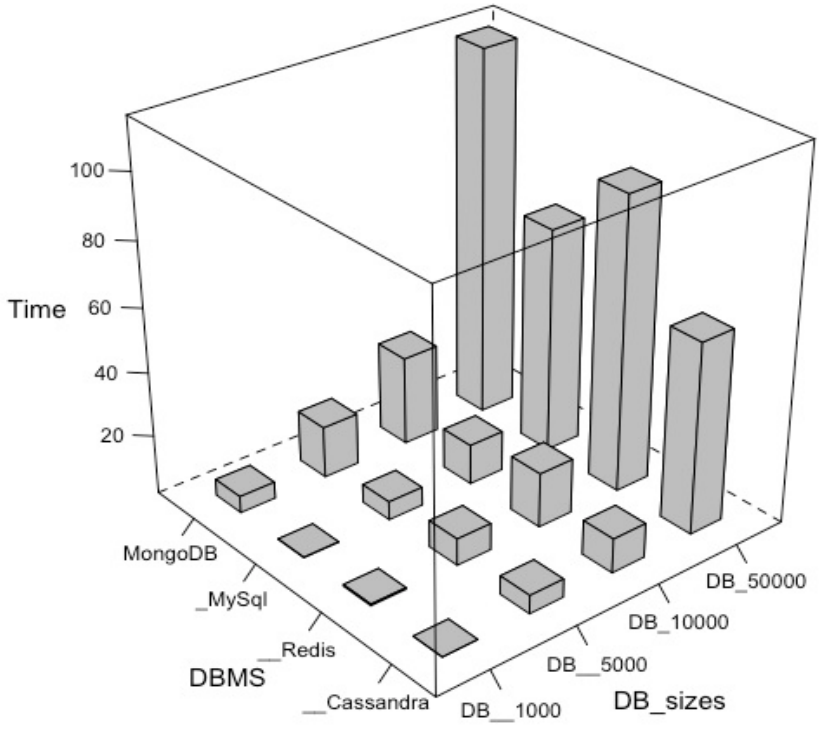

Figure 88. Variation in query time of experiment 3: query 1.

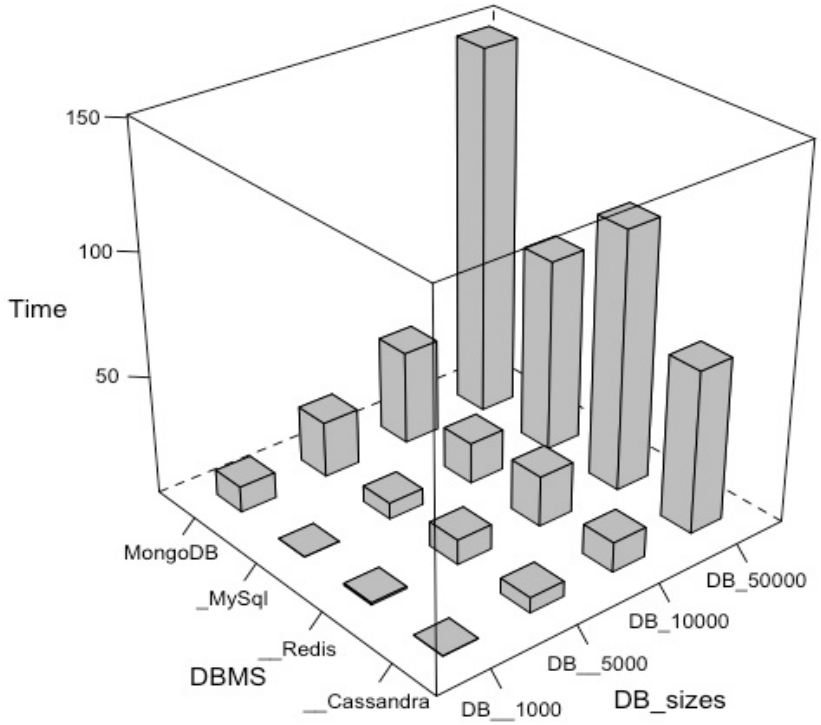

Figure 89. Variation in query time of experiment 3: query 2.

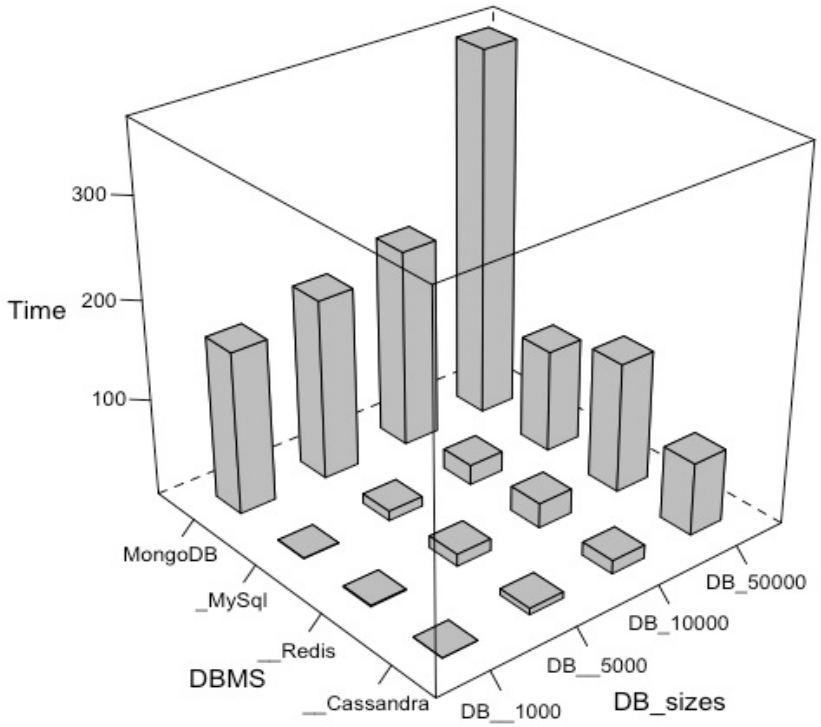

Figure 90. Variation in query time of experiment 3: query 3.

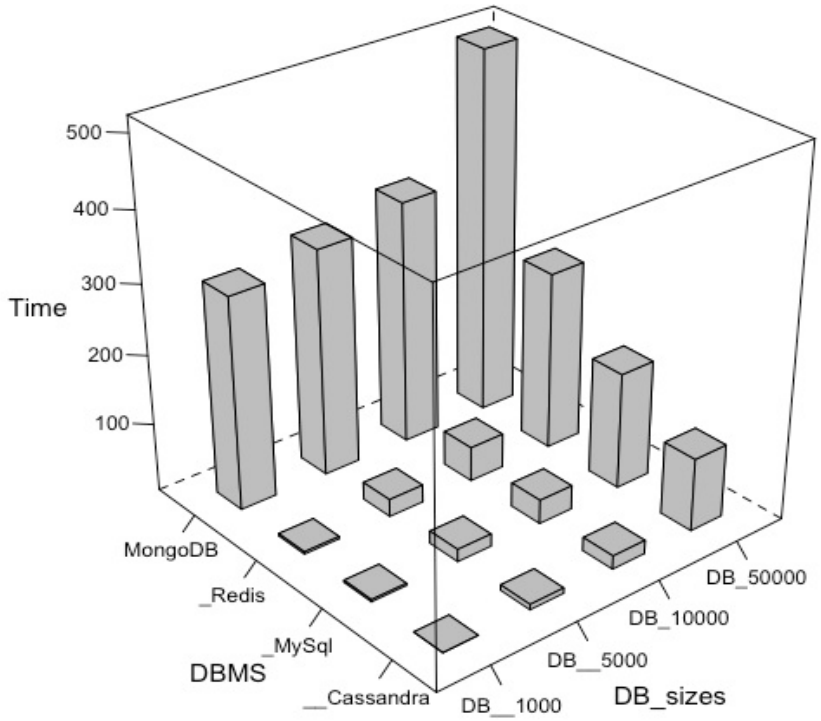

Figure 91. Variation in query time of experiment 3: query 4.

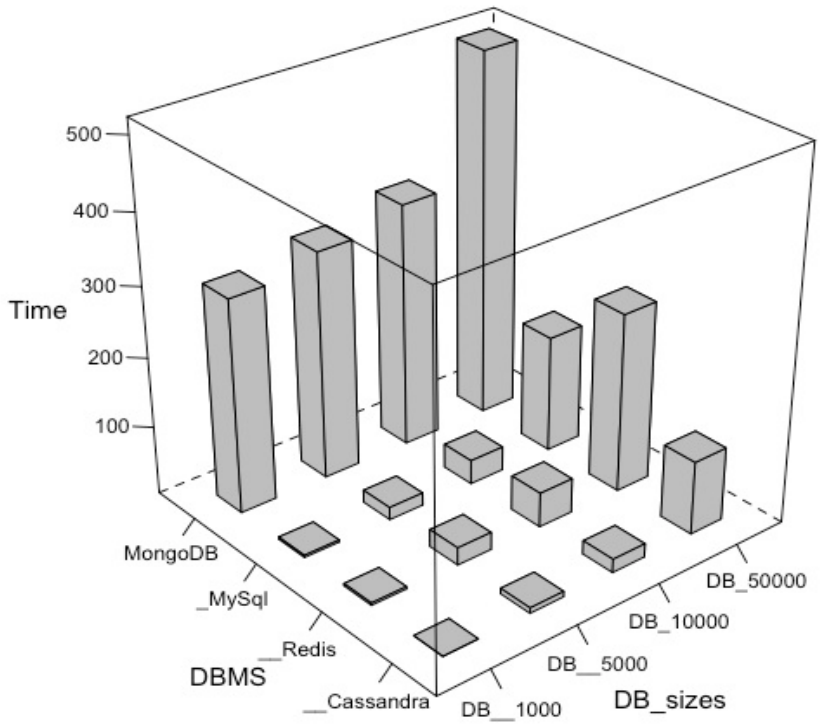

Figure 92. Variation in query time of experiment 3: query 5.

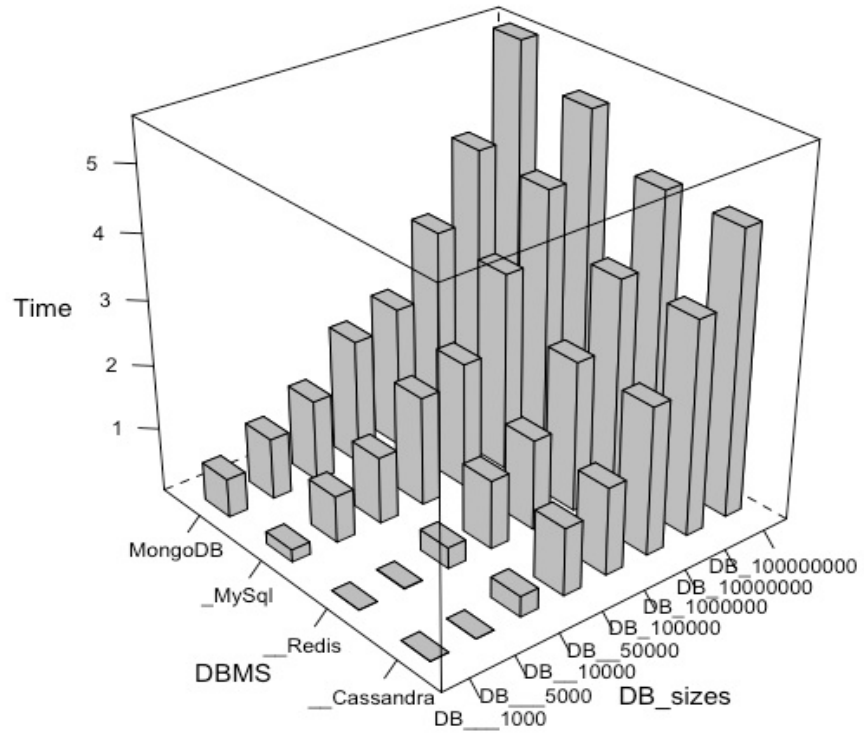

Figure 93. Variation in query time (base-10 log scale) of experiment 3 on larger databases: query 1.

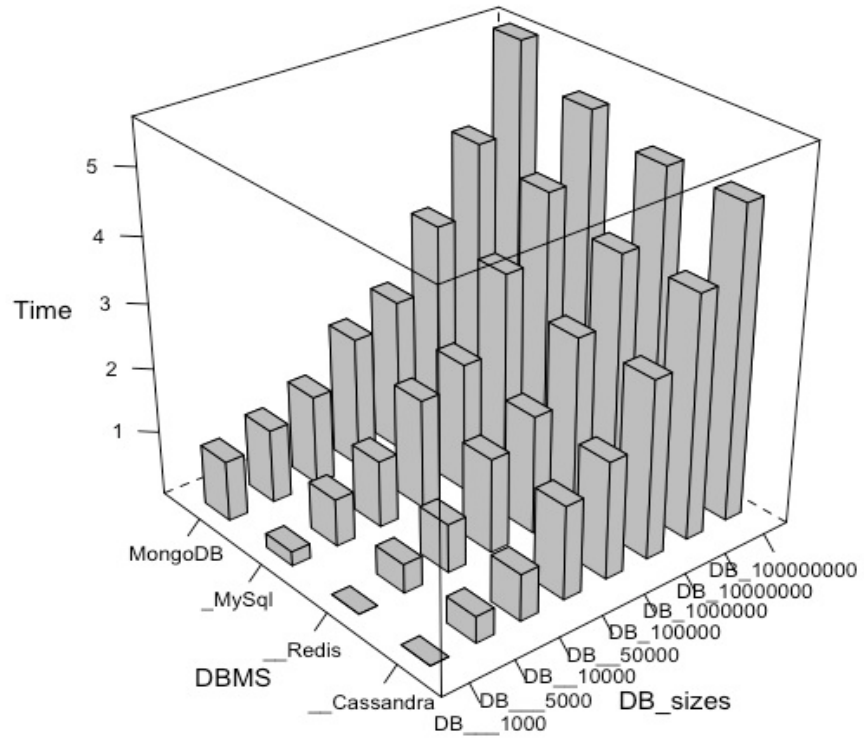

Figure 94. Variation in query time (base-10 log scale) of experiment 3 on larger databases: query 2.

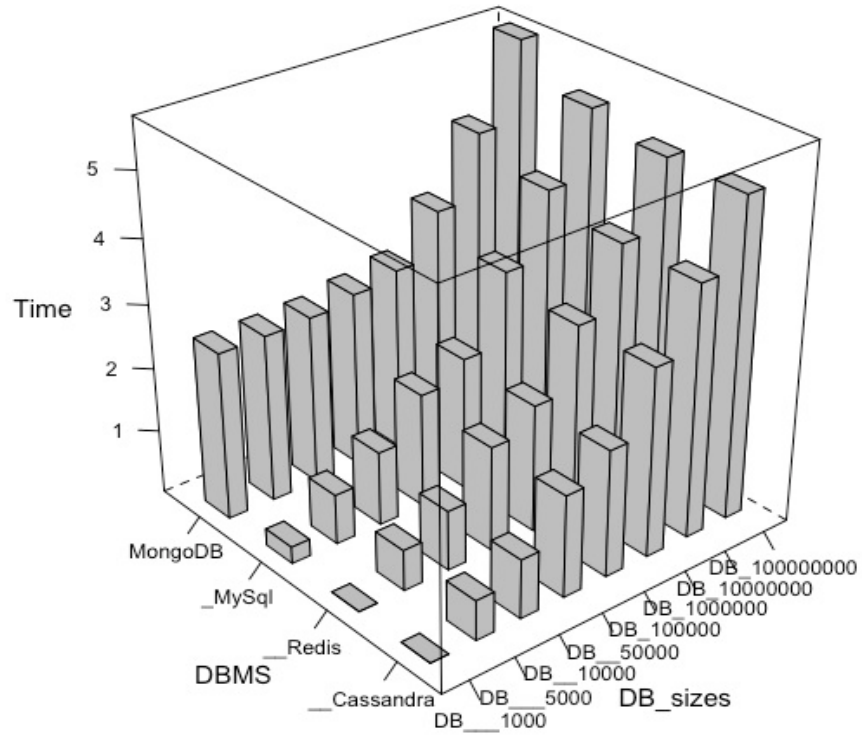

Figure 95. Variation in query time (base-10 log scale) of experiment 3 on larger databases: query 3.

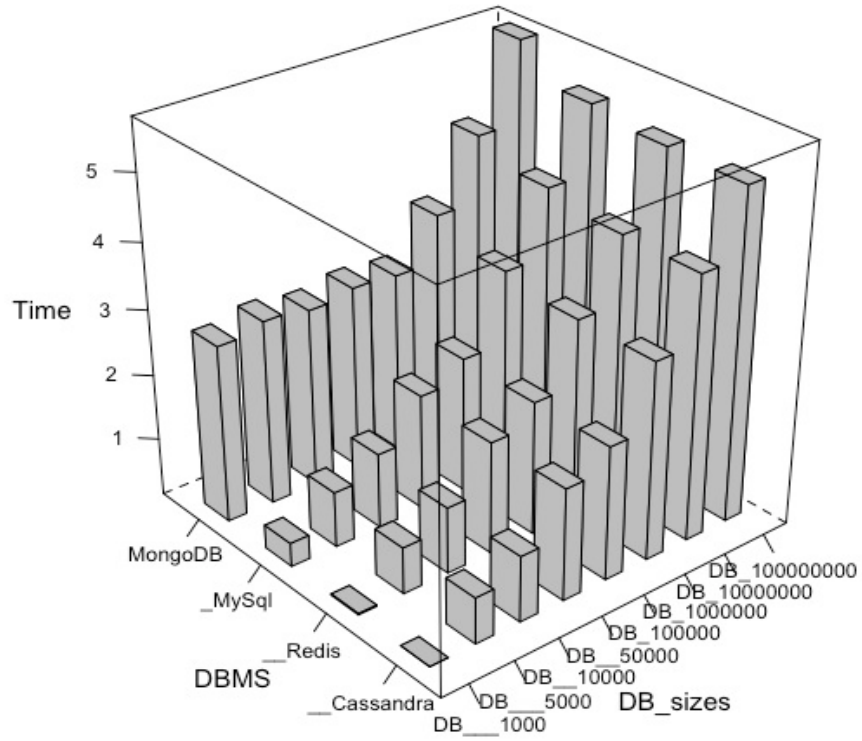

Figure 96. Variation in query time (base-10 log scale) of experiment 3 on larger databases: query 4.

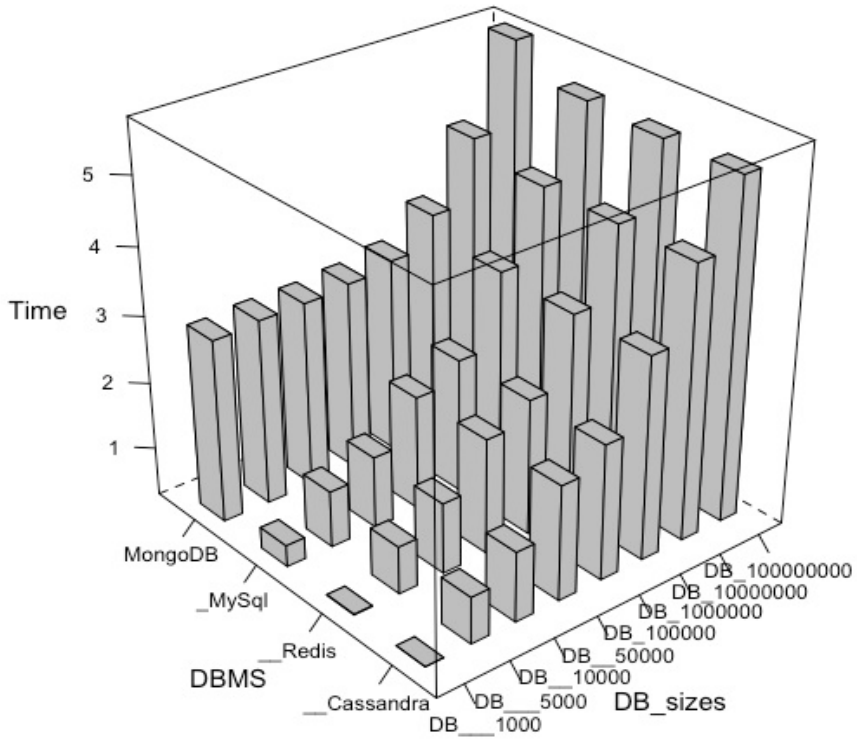

Figure 97. Variation in query time (base-10 log scale) of experiment 3 on larger databases: query 5.

Table 35. Descriptions of database files, Query outputs and setup database effort using MongoDB in Experiment 1.

Database size	File size	Q1 output: No. of Records	Q2 output: No. of Records	Q3 output: No. of Records	Q4 output: No. of Records	Q5 output: No. of Records	No. of programming lines	Approx. No. of days to DB setup
Using standard computing resources								
MongoDB								
1,000	1.6 MB	280	280	280	20	20	162	10
5,000	8 MB	1,400	1,400	1,400	100	100	162	10
10,000	16 MB	2,800	2,800	2,800	200	200	162	10
50,000	80 MB	140,000	140,000	140,000	1000	1000	166	10
Using supercomputing resources								
MongoDB								
1,000	1.6 MB	280	280	280	20	20	162	10
5,000	8 MB	1,400	1,400	1,400	100	100	162	10
10,000	16 MB	2,800	2,800	2,800	200	200	162	10
50,000	80 MB	140,000	140,000	140,000	1,000	1,000	166	10
100,000	0.16 GB	280,000	280,000	280,000	2,000	2,000	172	10
1,000,000	1.6 GB	2,800,000	2,800,000	2,800,000	20,000	20,000	182	10
10,000,000	16 GB	28,000,000	28,000,000	28,000,000	200,000	200,000	192	10
100,000,000	0.16 TB	280,000,000	280,000,000	280,000,000	2,000,000	2,000,000	202	10

Table 36. Descriptions of database files, Query outputs and setup database effort using Redis in Experiment 1.

Database size	File size	Q1 output: No. of Records	Q2 output: No. of Records	Q3 output: No. of Records	Q4 output: No. of Records	Q5 output: No. of Records	No. of programming lines	Approx. No. of days to DB setup
Using standard computing resources								
Redis								
1,000	1.9 MB	280	280	280	20	20	6441	90
5,000	9.5 MB	1,400	1,400	1,400	100	100	6441	90
10,000	19 MB	2,800	2,800	2,800	200	200	6441	90
50,000	95 MB	140,000	140,000	140,000	1000	1000	6441	90
Using supercomputing resources								
Redis								
1,000	1.9 MB	280	280	280	20	20	6441	90
5,000	9.5 MB	1,400	1,400	1,400	100	100	6441	90
10,000	19 MB	2,800	2,800	2,800	200	200	6441	90
50,000	95 MB	140,000	140,000	140,000	1,000	1,000	6441	90
100,000	190 MB	280,000	280,000	280,000	2,000	2,000	6441	90
1,000,000	1.9 GB	2,800,000	2,800,000	2,800,000	20,000	20,000	6441	90
10,000,000	19 GB	28,000,000	28,000,000	28,000,000	200,000	200,000	6441	90
100,000,000	0.19 TB	280,000,000	280,000,000	280,000,000	2,000,000	2,000,000	6441	90

Table 37. Descriptions of database files, Query outputs and setup database effort using Cassandra in Experiment 1.

Database size	File size	Q1 output: No. of Records	Q2 output: No. of Records	Q3 output: No. of Records	Q4 output: No. of Records	Q5 output: No. of Records	No. of programming lines	Approx. No. of days to DB setup
Using standard computing resources								
Cassandra								
1,000	1.66 MB	280	280	280	20	20	187	60
5,000	8.3 MB	1,400	1,400	1,400	100	100	187	60
10,000	16.6 MB	2,800	2,800	2,800	200	200	187	60
50,000	83 MB	140,000	140,000	140,000	1000	1000	187	60
Using supercomputing resources								
Cassandra								
1,000	1.66 MB	280	280	280	20	20	187	60
5,000	8.3 MB	1,400	1,400	1,400	100	100	187	60
10,000	16.6 MB	2,800	2,800	2,800	200	200	187	60
50,000	83 MB	140,000	140,000	140,000	1,000	1,000	187	60
100,000	166 MB	280,000	280,000	280,000	2,000	2,000	187	60
1,000,000	1.66 GB	2,800,000	2,800,000	2,800,000	20,000	20,000	187	60
10,000,000	16.6 GB	28,000,000	28,000,000	28,000,000	200,000	200,000	187	60
100,000,000	0. 166 TB	280,000,000	280,000,000	280,000,000	2,000,000	2,000,000	187	60

Table 38. Descriptions of database files, Query outputs and setup database effort using MySQL in Experiment 1.

Database size	File size	Q1 output: No. of Records	Q2 output: No. of Records	Q3 output: No. of Records	Q4 output: No. of Records	Q5 output: No. of Records	No. of programming lines	Approx. No. of days to DB setup
Using standard computing resources								
MySQL								
1,000	0.44 MB	280	280	280	20	20	225	10
5,000	2.2 MB	1,400	1,400	1,400	100	100	225	10
10,000	4.4 MB	2,800	2,800	2,800	200	200	225	10
50,000	22 MB	140,000	140,000	140,000	1000	1000	225	10
Using supercomputing resources								
MySQL								
1,000	0.44 MB	280	280	280	20	20	225	10
5,000	2.2 MB	1,400	1,400	1,400	100	100	225	10
10,000	4.4 MB	2,800	2,800	2,800	200	200	225	10
50,000	22 MB	140,000	140,000	140,000	1,000	1,000	225	10
100,000	44 MB	280,000	280,000	280,000	2,000	2,000	225	10
1,000,000	0.44 GB	2,800,000	2,800,000	2,800,000	20,000	20,000	225	10
10,000,000	4.4 GB	28,000,000	28,000,000	28,000,000	200,000	200,000	225	10
100,000,000	0.044 TB	280,000,000	280,000,000	280,000,000	2,000,000	2,000,000	225	10

Table 39. Descriptions of database files, Query outputs and setup database effort using MongoDB in Experiment 2.

Database size	File size	Q1 output: No. of Records	Q2 output: No. of Records	Q3 output: No. of Records	Q4 output: No. of Records	Q5 output: No. of Records	No. of programming lines	Approx. No. of days to DB setup
Using standard computing resources								
MongoDB								
1,000	1.61MB	280	280	280	20	20	162	10
5,000	8.05 MB	1,400	1,400	1,400	100	100	162	10
10,000	16.1 MB	2,800	2,800	2,800	200	200	162	10
50,000	80.5 MB	140,000	140,000	140,000	1000	1000	166	10
Using supercomputing resources								
MongoDB								
1,000	1.61MB	280	280	280	20	20	162	10
5,000	8.05 MB	1,400	1,400	1,400	100	100	162	10
10,000	16.1 MB	2,800	2,800	2,800	200	200	162	10
50,000	80.5 MB	140,000	140,000	140,000	1,000	1,000	166	10
100,000	0.161 GB	280,000	280,000	280,000	2,000	2,000	172	10
1,000,000	1.61 GB	2,800,000	2,800,000	2,800,000	20,000	20,000	182	10
10,000,000	161 GB	28,000,000	28,000,000	28,000,000	200,000	200,000	192	10
100,000,000	0.161 TB	280,000,000	280,000,000	280,000,000	2,000,000	2,000,000	202	10

Table 40. Descriptions of database files, Query outputs and setup database effort using Redis in Experiment 2.

Database size	File size	Q1 output: No. of Records	Q2 output: No. of Records	Q3 output: No. of Records	Q4 output: No. of Records	Q5 output: No. of Records	No. of programming lines	Approx. No. of days to DB setup
Using standard computing resources								
Redis								
1,000	2 MB	280	280	280	20	20	6441	90
5,000	10 MB	1,400	1,400	1,400	100	100	6441	90
10,000	20 MB	2,800	2,800	2,800	200	200	6441	90
50,000	100 MB	140,000	140,000	140,000	1000	1000	6441	90
Using supercomputing resources								
Redis								
1,000	2 MB	280	280	280	20	20	6441	90
5,000	10 MB	1,400	1,400	1,400	100	100	6441	90
10,000	20 MB	2,800	2,800	2,800	200	200	6441	90
50,000	100 MB	140,000	140,000	140,000	1,000	1,000	6441	90
100,000	200 MB	280,000	280,000	280,000	2,000	2,000	6441	90
1,000,000	2 GB	2,800,000	2,800,000	2,800,000	20,000	20,000	6441	90
10,000,000	20 GB	28,000,000	28,000,000	28,000,000	200,000	200,000	6441	90
100,000,000	0.2 TB	280,000,000	280,000,000	280,000,000	2,000,000	2,000,000	6441	90

Table 41. Descriptions of database files, Query outputs and setup database effort using Cassandra in Experiment 2.

Database size	File size	Q1 output: No. of Records	Q2 output: No. of Records	Q3 output: No. of Records	Q4 output: No. of Records	Q5 output: No. of Records	No. of programming lines	Approx. No. of days to DB setup
Using standard computing resources								
Cassandra								
1,000	1.68 MB	280	280	280	20	20	187	60
5,000	8.4 MB	1,400	1,400	1,400	100	100	187	60
10,000	16.8 MB	2,800	2,800	2,800	200	200	187	60
50,000	84 MB	140,000	140,000	140,000	1000	1000	187	60
Using supercomputing resources								
Cassandra								
1,000	1.68 MB	280	280	280	20	20	187	60
5,000	8.4 MB	1,400	1,400	1,400	100	100	187	60
10,000	16.8 MB	2,800	2,800	2,800	200	200	187	60
50,000	84 MB	140,000	140,000	140,000	1,000	1,000	187	60
100,000	168 MB	280,000	280,000	280,000	2,000	2,000	187	60
1,000,000	1.68 GB	2,800,000	2,800,000	2,800,000	20,000	20,000	187	60
10,000,000	16.8 GB	28,000,000	28,000,000	28,000,000	200,000	200,000	187	60
100,000,000	0. 168 TB	280,000,000	280,000,000	280,000,000	2,000,000	2,000,000	187	60

Table 42. Descriptions of database files, Query outputs and setup database effort using MySQL in Experiment 2.

Database size	File size	Q1 output: No. of Records	Q2 output: No. of Records	Q3 output: No. of Records	Q4 output: No. of Records	Q5 output: No. of Records	No. of programming lines	Approx. No. of days to DB setup
Using standard computing resources								
MySQL								
1,000	0.44 MB	280	280	280	20	20	225	10
5,000	2.2 MB	1,400	1,400	1,400	100	100	225	10
10,000	4.4 MB	2,800	2,800	2,800	200	200	225	10
50,000	22 MB	140,000	140,000	140,000	1000	1000	225	10
Using supercomputing resources								
MySQL								
1,000	0.44 MB	280	280	280	20	20	225	10
5,000	2.2 MB	1,400	1,400	1,400	100	100	225	10
10,000	4.4 MB	2,800	2,800	2,800	200	200	225	10
50,000	22 MB	140,000	140,000	140,000	1,000	1,000	225	10
100,000	44 MB	280,000	280,000	280,000	2,000	2,000	225	10
1,000,000	0.44 GB	2,800,000	2,800,000	2,800,000	20,000	20,000	225	10
10,000,000	4.4 GB	28,000,000	28,000,000	28,000,000	200,000	200,000	225	10
100,000,000	0.044 TB	280,000,000	280,000,000	280,000,000	2,000,000	2,000,000	225	10

Table 43. Descriptions of database files, Query outputs and setup database effort using MongoDB in Experiment 3.

Database size	File size	Q1 output: No. of Records	Q2 output: No. of Records	Q3 output: No. of Records	Q4 output: No. of Records	Q5 output: No. of Records	No. of programming lines	Approx. No. of days to DB setup
Using standard computing resources								
MongoDB								
1,000	1.7 MB	280	280	280	20	20	167	10
5,000	8.5 MB	1,400	1,400	1,400	100	100	167	10
10,000	17 MB	2,800	2,800	2,800	200	200	167	10
50,000	85 MB	140,000	140,000	140,000	1000	1000	171	10
Using supercomputing resources								
MongoDB								
1,000	1.7 MB	280	280	280	20	20	167	10
5,000	8.5 MB	1,400	1,400	1,400	100	100	167	10
10,000	17 MB	2,800	2,800	2,800	200	200	167	10
50,000	85 MB	140,000	140,000	140,000	1,000	1,000	171	10
100,000	170 MB	280,000	280,000	280,000	2,000	2,000	177	10
1,000,000	1.7 GB	2,800,000	2,800,000	2,800,000	20,000	20,000	187	10
10,000,000	17 GB	28,000,000	28,000,000	28,000,000	200,000	200,000	197	10
100,000,000	0.17 TB	280,000,000	280,000,000	280,000,000	2,000,000	2,000,000	207	10

Table 44. Descriptions of database files, Query outputs and setup database effort using Redis in Experiment 3.

Database size	File size	Q1 output: No. of Records	Q2 output: No. of Records	Q3 output: No. of Records	Q4 output: No. of Records	Q5 output: No. of Records	No. of programming lines	Approx. No. of days to DB setup
Using standard computing resources								
Redis								
1,000	2.1 MB	280	280	280	20	20	193	90
5,000	10.5 MB	1,400	1,400	1,400	100	100	193	90
10,000	21 MB	2,800	2,800	2,800	200	200	193	90
50,000	105 MB	140,000	140,000	140,000	1000	1000	193	90
Using supercomputing resources								
Redis								
1,000	2.1 MB	280	280	280	20	20	193	90
5,000	10.5 MB	1,400	1,400	1,400	100	100	193	90
10,000	21 MB	2,800	2,800	2,800	200	200	193	90
50,000	105 MB	140,000	140,000	140,000	1,000	1,000	193	90
100,000	210 MB	280,000	280,000	280,000	2,000	2,000	193	90
1,000,000	2.1 GB	2,800,000	2,800,000	2,800,000	20,000	20,000	193	90
10,000,000	21 GB	28,000,000	28,000,000	28,000,000	200,000	200,000	193	90
100,000,000	0.21 TB	280,000,000	280,000,000	280,000,000	2,000,000	2,000,000	193	90

Table 45. Descriptions of database files, Query outputs and setup database effort using Cassandra in Experiment 3.

Database size	File size	Q1 output: No. of Records	Q2 output: No. of Records	Q3 output: No. of Records	Q4 output: No. of Records	Q5 output: No. of Records	No. of programming lines	Approx. No. of days to DB setup
Using standard computing resources								
Cassandra								
1,000	1.7 MB	280	280	280	20	20	6691	60
5,000	8.5 MB	1,400	1,400	1,400	100	100	6691	60
10,000	17 MB	2,800	2,800	2,800	200	200	6691	60
50,000	85 MB	140,000	140,000	140,000	1000	1000	6691	60
Using supercomputing resources								
Cassandra								
1,000	1.7 MB	280	280	280	20	20	6691	60
5,000	8.5 MB	1,400	1,400	1,400	100	100	6691	60
10,000	17 MB	2,800	2,800	2,800	200	200	6691	60
50,000	85 MB	140,000	140,000	140,000	1,000	1,000	6691	60
100,000	170 MB	280,000	280,000	280,000	2,000	2,000	6691	60
1,000,000	1.7 GB	2,800,000	2,800,000	2,800,000	20,000	20,000	6691	60
10,000,000	17 GB	28,000,000	28,000,000	28,000,000	200,000	200,000	6691	60
100,000,000	0.17 TB	280,000,000	280,000,000	280,000,000	2,000,000	2,000,000	6691	60

Table 46. Descriptions of database files, Query outputs and setup database effort using MySQL in Experiment 3.

Database size	File size	Q1 output: No. of Records	Q2 output: No. of Records	Q3 output: No. of Records	Q4 output: No. of Records	Q5 output: No. of Records	No. of programming lines	Approx. No. of days to DB setup
Using standard computing resources								
MySQL								
1,000	0.442 MB	280	280	280	20	20	241	10
5,000	2.21 MB	1,400	1,400	1,400	100	100	241	10
10,000	4.42 MB	2,800	2,800	2,800	200	200	241	10
50,000	22.1 MB	140,000	140,000	140,000	1000	1000	241	10
Using supercomputing resources								
MySQL								
1,000	0.442 MB	280	280	280	20	20	241	10
5,000	2.21 MB	1,400	1,400	1,400	100	100	241	10
10,000	4.42 MB	2,800	2,800	2,800	200	200	241	10
50,000	22.1 MB	140,000	140,000	140,000	1,000	1,000	241	10
100,000	44.2 MB	280,000	280,000	280,000	2,000	2,000	241	10
1,000,000	0.442 GB	2,800,000	2,800,000	2,800,000	20,000	20,000	241	10
10,000,000	4.42 GB	28,000,000	28,000,000	28,000,000	200,000	200,000	241	10
100,000,000	0.0442 TB	280,000,000	280,000,000	280,000,000	2,000,000	2,000,000	241	10

7.3 LIMITATIONS

System storage is always an issue when we deal with big data. Storage limitations forced us to run one database approach at the time. Thus, the importing process had to be repeated each time we shifted from one database approach to another, making our work more time consuming and elaborate than was expected. Further projects could focus on comparing fewer technologies (i.e. two technologies) with larger databases than handling more technologies in order to avoid long hours spent importing data.

We found that each NoSQL database approach showed different results. However, by looking in the literature, it appears that within each data model (i.e., Document store) different

technologies can show different results. For example MongoDB uses Document stores but could have different performance and scalability than CouchDB, which uses the same data model. Thus, our results are likely not reflective of the database model itself as a solution to managing the Precision Medicine data, but of the database technology used (in this case MongoDB). However, overall there are few studies that have examined this issue, and our methods will be available to be used in the future for measuring other database technologies that use the same data model. As a note, we selected our database approaches according to the most popular NoSQL approaches registered on a LinkedIn skill index (as of September 2014), having tracked mentions of NoSQL databases in its member profiles. Future studies can use another resource to select the most popular NoSQL databases in order to study their performance and scalability using clinical and genomic data.

The issue about aware of the frequent updating processes from PGRR to TCGA was solved by saving the portion of data of interest at a particular point in time and running the queries consequently. This solution saved us time in actively having to import data from PGRR every time we needed to shift to a different database approach and allowed us to avoid the possibility of bias in our comparison results since we would be using different patients' information. In addition we also avoided having to annotate the updates realized by the PGRR and having to report the database size each time we make the importing process.

A serious limitation of this work was the use of a non-curated source of data that prevented us from building algorithms to accurately extract data to posteriorly use in the different database technologies. The TCGA clinical information offered data that was poor in quality in terms of accommodation and syntax. By accessing the data through the command line, the TCGA information appeared incomplete: many blank spaces, words with incorrect spelling,

extra spaces, conglomerated adjacent information, and even misplaced data appeared with different headers. This poor data quality forced us to manually curate the information, which also consumed more time than expected. This limitation forced us to include less information and to model the rest of information needed. Future projects should include publicly available data but of acceptable clinical data quality. As the shared data culture grow and the field of Precision Medicine develops, it is more likely that upcoming publicly available information that is highly sensitive but anonymous will be more easily accessible to improve the design of this type of database project in data source terms.

7.4 FUTURE RESEARCH DIRECTIONS

The work in this dissertation involved a wider evaluation of NoSQL database management systems, and we hope that it will eventually encourage the wider use of NoSQL systems in the integration of clinical and genomic data. We believe there is a substantial body of future research in the area of NoSQL databases regarding performance and scalability since the data is growing, new NoSQL database updates are forthcoming and new NoSQL approaches are being developed.

Future work includes the use of NoSQL databases that work with the same data model, such as MongoDB and CouchDB, to develop NoSQL-Document-model databases with clinical and genomic data and evaluate performance and scalability. This work will give more comprehensive results on the performance and scalability of NoSQL approaches.

Prospect work also includes the mixture of different models in a database since probably specific clinical or genomic information are better stored and managed in a particular data model. To this purpose, more emphasis should be placed in the characteristics of the data and the

database functionality. Thus, works focused in the study of all models but in each model requested a specific either clinical or genomic information could provide novel information about which data model is more suitable to a specific source of data. In other words, new studies should elaborate queries just focus in either clinical or genomic information to bring new information that provide confidence for the use of different data models in a database, contributing in the finding of suitable databases in Precision Medicine.

Since data can be managed in either ways online or offline forms, we focused on online databases but upcoming studies should probably be more detailed in this form in order to point more accurately the most suitable data model in manipulating data in real time context. Online form is important in Precision Medicine since hospital systems would need real time analytics to integrate continuously generated clinical and genomic information from patients that probably need to be compared to other patients with similar or different treatments. Technically, new studies should report the lowest systems query latency that better support user's requests. Also in online way, future studies should focus on multiple users and data availability using different data models. This information would provide evidence to select suitable databases for modern applications in Personalized Medicine.

Since data production is growing exponentially, our project could be continued by running it in larger databases, multiple databases or big hospital databases. This means using complete sets of clinical and genomic information for more than 100 million patients to study the performance and scalability of NoSQL approaches.

This project can be extended by including and requesting another type of genomic data representative of Precision Medicine, for example, including and requesting data from epigenetic analysis using DNA methylation (i.e., Methyl-Seq) and Histone modifications (i.e., ChIP-Seq).

The information retrieved would provide relevant information on the effective data management of multiple genomic information using NoSQL approaches. Our results indicate that NoSQL approaches potentially enable the integration of a variety of genomic data, inspiring this line of research. In functional genetics our studied NoSQL approaches could be used to manage data, such as proteomic, transcriptomic, and metabolomics, providing a better understanding of successful data integration and, consequently, the genetic basis of diseases.

In the area of clinical data, this project could be extended to include a variety of clinical data terms. Beyond using written clinical data, imaging data could also be stored using NoSQL technologies, as it is already used in the movie industry. Thus, the inclusion of clinical images, such as MRI, fMRI, TC, and Echography among others, would provide better integration of clinical and genomic data.

The successful integration of clinical and genomic data shown in our project, offering the capability of accessing and analyzing heterogeneous clinical and genomic data, could inspire the development of cognitive programming in Precision Medicine.

7.5 DISCUSSION

Our results show that NoSQL approaches, especially Cassandra and Redis, outperform MySQL (SQL) approach in terms of performance and scalability, and have the advantage in updating processes, as we originally hypothesized. These results were based on performances of queries that requested specific clinical and genomic information.

In terms of performance and scalability, we used very specific queries that had the function of filtering relevant information for Precision Medicine. Some technologies had faster

query times than others for specific queries and database sizes. We believe that we could predict the most suitable technology based on the query complexity (increased filtering process) and database size. For example, in our simplest query 1, using standard computing resources, Cassandra and Redis were the technologies with the lowest query times; the function of this query was to filter simple demographic characteristics of the patients' records. In our query 2, also using databases with 1000 to 50,000 records, Redis store showed the lowest query time; query 2 requires a more complex filtration process than query 1: it filtered patients with similar demographic and clinical characteristics. Finally, in the more complex queries 3, 4 and 5, using databases with 1000 to 50,000 records, Cassandra was the fastest technology; these last queries filter demographic, clinical, treatment, structural and functional genomic characteristics. Thus, using the mentioned database sizes, we can predict that for more complex queries (higher levels of filtration), the most suitable technology would be Cassandra and for more simple queries (lower levels of filtration) the most suitable technology would be Redis store. In larger database sizes, 100,000 to 10 million, using supercomputing resources, for either simple or complex queries, Cassandra and Redis seems to be the most suitable technology in terms of performance. Thus, it seems that large computational resources increase the suitability of these two technologies, based in two different data models, to successfully manipulate Precision Medicine data.

We implemented MySQL as one big table in order to compete in performance and scalability issues with NoSQL technologies. We also could have chosen to split information into different tables, which would have significantly reduced its performance in contrast to NoSQL technologies. Thus, we demonstrate that by fitting all information in one table, this SQL approach could be compared at some point to NoSQL approaches in terms of performance and

scalability. In general, using standard computing resources, MySQL showed similar lower query times than Cassandra in simpler queries during experiment 3. However, using supercomputing resources, MySQL showed slightly lower query times when using 100,000 records and the complex query than the other NoSQL approaches, with the exception of experiment 2 where Cassandra was the fastest technology. Overall, it seems that the MySQL does not reach the level of performance and scalability than NoSQL technologies such as Cassandra, when using large computational resources, as required in Precision Medicine. Moreover, using Tables could not be practical given ever increasing supercomputer resources. In reality, Table store would operate using multiple tables with joins, which would decrease the performance and scalability of this technology, supporting the idea of using of NoSQL approaches to manage big data bases using large computational resources.

We found different functionality in DBMS to measure query performance. SQL queries have a very well integrated function to measure query times, as well as to detect slow queries, for each query we made. In contrast, with NoSQL technologies, we had to invoke functions from each technology in order to measure the query time. In MongoDB we programmed the difference of two created variables to obtain the query time in milliseconds –execution_mills = after - before–. One of these variables was created before making each query –var before = new Date()–, and the second was created after making the query –var after = new Date()–. In Redis, we used a more direct method to calculate the query times; here, we customized the configuration of slow log to lower than 5 microseconds –config set slowlog-log-slower-than 5–. In Cassandra, we also use a direct method to calculate query times; this technology has a tracing function that can be turned on and off before and after each query is made to obtain query times in microseconds. The way we calculated the query times in milliseconds on MongoDB could have influenced the

results for this specific technology because it is not a direct way to calculate query times in contrast to MySQL and even to Redis and Cassandra, which, though they lack an automated way to measure query times, have functions that allow us to measure query times without the need to make extra programming lines. Overall, the lack of a robust time-measuring system in NoSQL approaches, in contrast to SQL approaches, could have made these technologies consume extra time and computational resources to assess query performance and scalability. Thus, future developmental work is necessary in NoSQL approaches regarding this time-measuring issue.

We also observed that the natural indexing found in the Cassandra model offers an advantage in performance and scalability. In fact, to make a query with a filtration function, the cqlsh requires that all requested headers be the primary key and that the function named “allow filtering” be invoked together with the query. Thus, it seems that the design of this technology help to run queries with filtering functions. This design could be a reason for the resulting fastest query times from this technology. In summary, our conclusions comprise a complex filtering process used in the Precision Medicine field using a technology where its data model design includes an integrated filtering process based on natural indexing. This natural indexing property provides a natural structure to order the data, resulting in interesting performance and scalability scores, which could allow it to successfully filter clinical and genomic patients’ data.

We further observe that the Redis model is merely a database that requests values from their keys. In fact, it seems this technology do not allow the use of regular expressions at all. Our queries were created in a way that we can use this technology to query our results. However it was challenging to request information for ranges of values since in the query you must include all possible results you could possible obtain. For example, in order to find people in a range of age, we had to include all the possible years included in the range we were looking for. Thus, for

queries where regular expressions are essential to retrieve specific information, this technology would require the support of object-oriented programs. However, using intermediate programs could reduce the performance of this technology since the amount of information resulting from millions of Keys and Values can easily saturate the memory of such intermediate programs used from the import to process the information. Thus, this important limitation, querying just keys and values, could make this technology difficult to use for requesting more complex data (filtering processes), such as requesting specific structural and functional genomic information with extremely large ranges of values. Since Precision Medicine requires the study of patients with similar specific clinical and genomic characteristics, future developments in the querying process are necessary to make this technology a stronger candidate for manipulating this type of data.

With respect to MongoDB, we observe that it provides advantages in database implementation and portability. Using MongoDB reduced the time needed to create and populate a database in comparison to the other NoSQL approaches. Minimal time was required to start the data importing process. This technology is well designed to easily implement a database since the documents are hierarchically ordered based on JSON files. Moreover, as already mentioned in the literature, this data format is human readable. Thus, information extracted from this data can be read without the use of intermediate software, adding a portability data feature to this technology. Once performance and scalability issues are addressed by developers, the attractive characteristics of this technology could make it a potential candidate for future management of clinical and genomic information.

About our experience in DBMS implementation, although we did not measure the time of each specific process during the database implementation such as the load time, update time, and

programming time, we subjectively measure the overall time taken to database implementation and the approximate number of lines written to set up each database. We spent contrasting time and wrote dissimilar number of programming lines to set up each database technology as shown on Tables 33-46. MongoDB and MySQL were technologies that required less time to be implemented, contrasting to Cassandra and Redis that were the technologies that took us longer to implement. In fact, MongoDB and MySQL were 3 and 9 times faster than Cassandra and Redis in their implementation process, subsequently. MongoDB required less programming lines than any other NoSQL technology, following by Cassandra and MySQL. Redis needed 40 times more programming lines than MongoDB to be implemented. Overall, we required less effort to implement MongoDB and MySQL than Cassandra and Redis.

We devised a series of clinical-genomic filtering queries based on relevant biological questions to identify patients that share similar information and grouping them. Results from these queries could be used by 1) physicians to aware about the prognosis of a typical treatment in a specific group of patients, and 2) researchers to identify structural and functional genomic information such as gene expression patterns that affect the efficacy among typical cancer therapies. In addition, these queries could be useful in public health research, especially in randomized clinical trials to group people with specific clinical and molecular characteristics to test new cancer treatments. In overall, we believe these type of queries, beyond answer scientific questions, could be a basic filtering process for Precision Medicine since they have the potential to cluster patients in order to find better ways to prevent, screen for, diagnose, or treat cancer.

We use pre-computed files since this is the trend of different companies in the genomic field to accelerate the computational processes. The creation of clinical and genomic pre-computed files helped us to optimize the storage process. For example, instead of storage a

whole variant calling file from each patient, we were able to identify and merge specific chromosomal information together with its genomic location that were relevant for the queries we would use in posterior steps of the project. We believe that this trend in creating multiple pre-computed files prior to populating a database is an effective way to save computational resources and manipulate big Precision Medicine data. A limitation of the pre-computed files is that they require the proper clinical and genomic knowledge from the database manager administrator. Clearly, it is important that people understand this very sensitive information prior to its being pre-processed and implemented in a database. The use of pre-computed files provided greater confidentiality for the information stored in our databases. We believed that these files have the potential to promote the use of clinical-genomic cognitive programming because of the insight of the data they provide to their developers.

The Precision Medicine field requires the use of automated algorithms that run on a robust database with higher performance and scalability. These database characteristics are important since algorithms can include trillions of queries. Thus, if one DB system is slower than another, even in one query by seconds or milliseconds, the proportion of delays in time is exponential when we talk about quantities in the range of trillions. Overall, the most suitable database system for Precision Medicine will be as fast as possible for the most frequent queries used in this field. In this dissertation, we studied and identified suitable technologies to manipulate big Precision Medicine data by developing five different queries with different complexities based on the definition of Precision Medicine, clustering patients with similar demographic, clinical and genomic information.

APPENDIX: DATA FEATURES

A. ETHICAL AND SAFETY CONSIDERATIONS

The ethical and safety considerations were already addressed by PGRR. Thus, it is expected that this managed environment instance already met the information security and regulatory requirements for using TCGA data (d.1). In addition, it is expected that PGRR already dealt with regulatory acts such as The Institutional Review Board (IRB), The Health Insurance Portability and Accountability Act (HIPPA) –privacy rule's right of access and health information technology, and The Genetic Information Nondiscrimination Act (GINA).

B. CHARACTERISTICS OF NOSQL DATABASES

Table 47. Attributes of data models and NoSQL technologies used in this project.

Features	Document stores/ MongoDB	Key-Value stores/ Redis	Column stores/ Cassandra
Integrity Model	BASE	BASE	BASE
Atomicity	Yes	Yes	Yes
Consistency	Yes	Yes	Yes
Isolation	No	Yes	No
Durability	Yes	Yes	Yes
MapReduce	Yes	Yes	Yes

Table 48. Benefits and limitations using NoSQL Databases.

NoSQL databases	
Benefits	Limitations
BASE transactions	ACID transactions (Except Graph stores)
Schema-less	Nascent technologies
Elastic scaling	Evolving in scalability terms
Lesser server cost	Few facilities for non-standard queries
Integrated caching	Not easier to code in NoSQL as in SQL
Larger data handling capability	Supported by relatively small companies
Cheaper server maintenance	Limited connectivity to different applications

C. FIGURES OF DATA STRUCTURE USING DIFFERENT DATA MODELS

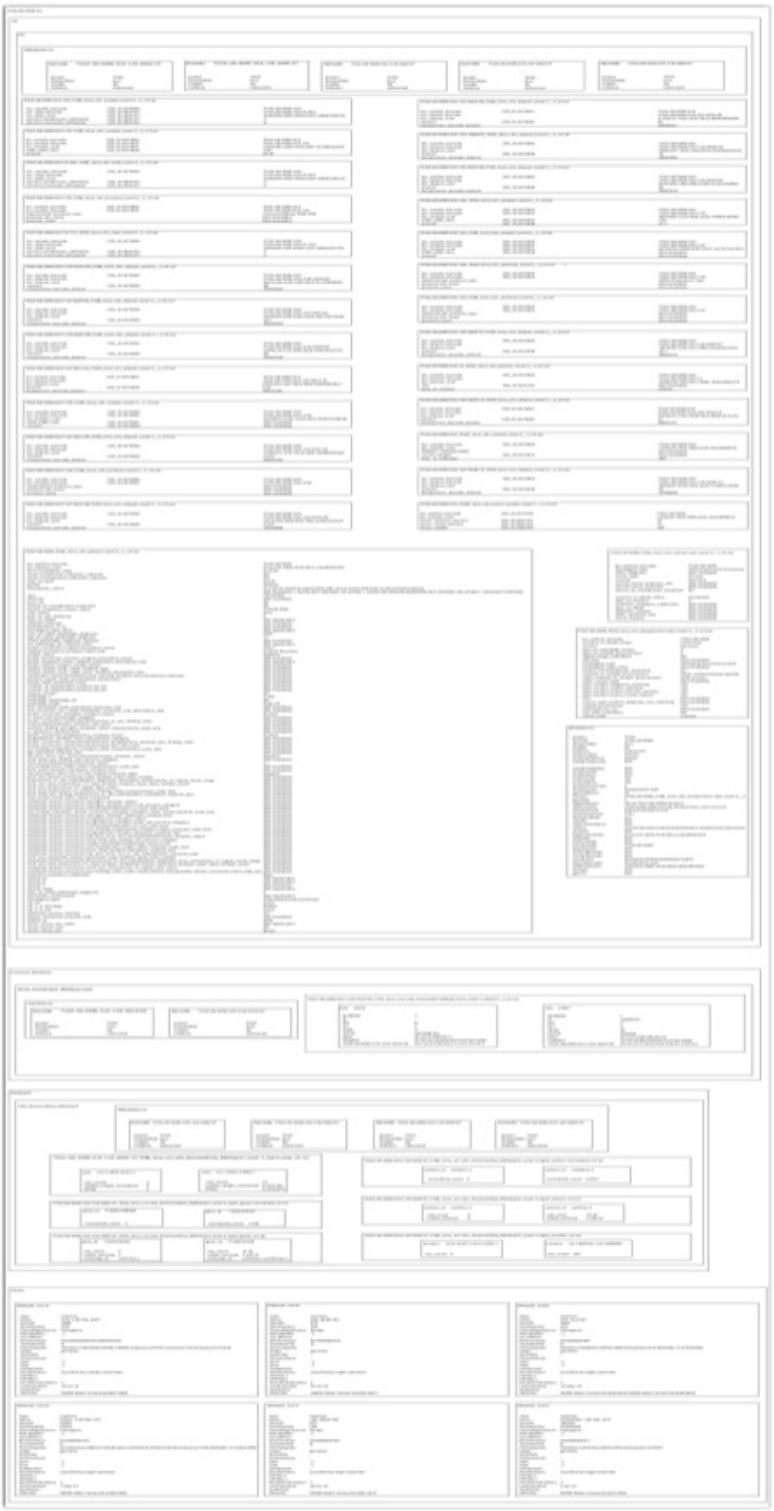

Figure 98. Data structure of Document store for MongoDB.

Figure 99. Data structure of Key-Value store for Redis.

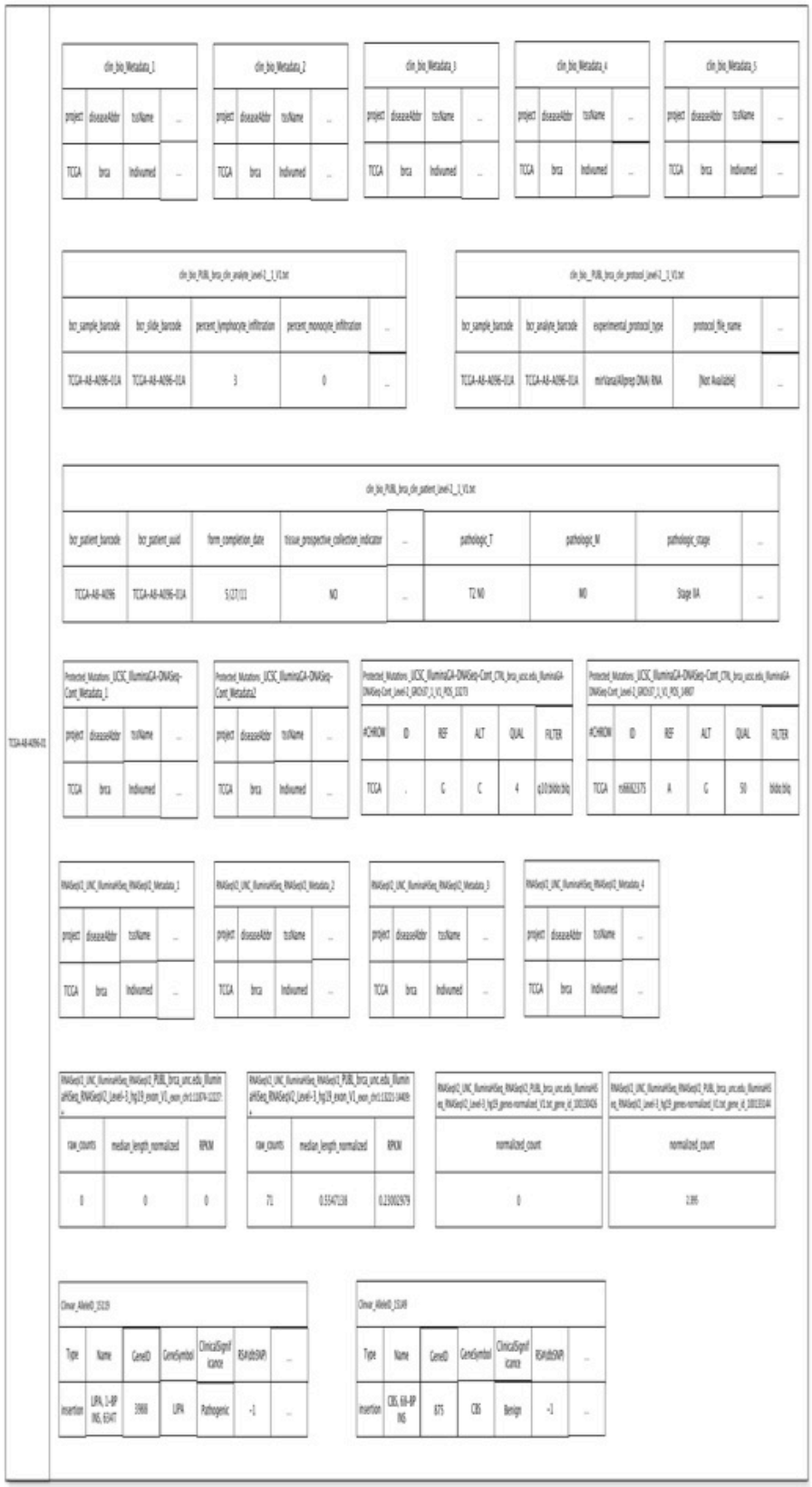

Figure 100. Data structure of Column store for Cassandra.

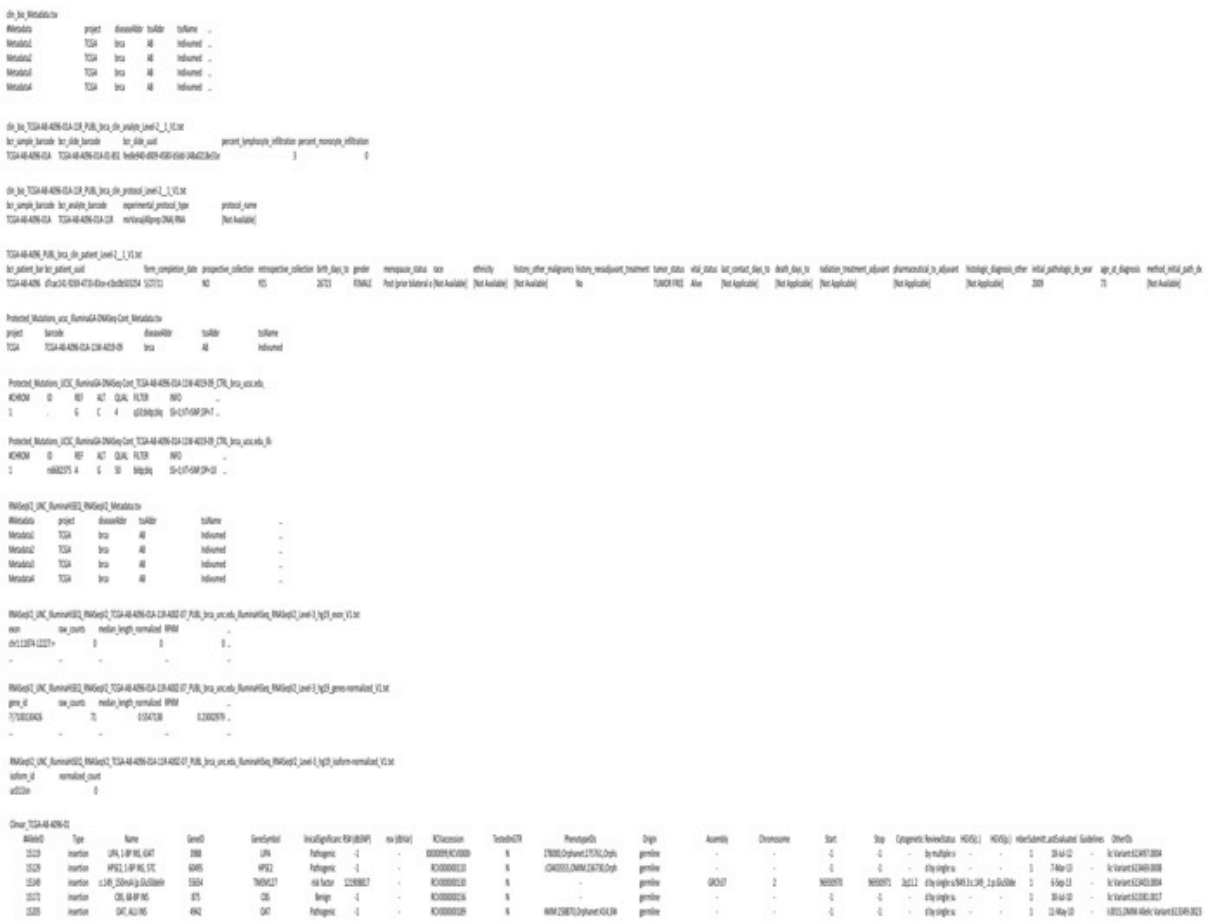

Figure 101. Data structure of Table store for MySQL.

BIBLIOGRAPHY

1. PCAST. (2010). Executive Office of the President President's Council of Advisors on Science and Technology. Report to the President Realizing the Full Potential of Health Information Technology to Improve Healthcare for Americans: The Path Forward. [Accessed on August 20, 2014. Available at: http://www.whitehouse.gov/sites/default/files/microsites/ostp/pcast-health-it-report.pdf].

2. Benson, D.A., Karsch-Mizrachi, I., Lipman, D.J., Ostell, J., and Sayers, E.W. (2009). GenBank. Nucleic acids research 37, D26-31.

3. Benson, D.A., Karsch-Mizrachi, I., Lipman, D.J., Ostell, J., and Wheeler, D.L. (2008). GenBank. Nucleic acids research 36, D25-30.

4. Stein, L. (2013). Creating databases for biological information: an introduction. Curr Protoc Bioinformatics Chapter 9, Unit9.1.

5. Bellazzi, R. (2014). Big data and biomedical informatics: a challenging opportunity. Yearbook of medical informatics 9, 8-13.

6. Overby, C.L., and Tarczy-Hornoch, P. (2013). Personalized medicine: challenges and opportunities for translational bioinformatics. Personalized medicine 10, 453-462.

7. Canuel, V., Rance, B., Avillach, P., Degoulet, P., and Burgun, A. (2015). Translational research platforms integrating clinical and omics data: a review of publicly available solutions. Briefings in bioinformatics 16, 280-290.

8. Shoenbill, K., Fost, N., Tachinardi, U., and Mendonca, E.A. (2014). Genetic data and electronic health records: a discussion of ethical, logistical and technological considerations. Journal of the American Medical Informatics Association : JAMIA 21, 171-180.

9. Alexandros Labrinidis, N.R. A Performance Evaluation of Online Warehouse Update Algorithms. Technical Research report in in the CSHCN series for The Center for Satellite and Hybrid Communication Networks, NASA-sponsored Commercial Space Center, Department of Defense (DOD), University of Maryland and the Institute for Systems Research. [Accessed August 20, 2014. Available at http://drum.lib.umd.edu/bitstream/1903/5988/1/TR_98-63.pdf].

10. Mohammed, E.A., Far, B.H., and Naugler, C. (2014). Applications of the MapReduce programming framework to clinical big data analysis: current landscape and future trends. BioData mining 7, 22.

11. Lee, K.K., Tang, W.C., and Choi, K.S. (2013). Alternatives to relational database: comparison of NoSQL and XML approaches for clinical data storage. Computer methods and programs in biomedicine 110, 99-109.

12. Hilbert, M., and Lopez, P. (2011). The world's technological capacity to store, communicate, and compute information. Science (New York, NY) 332, 60-65.

13. Company, M. (2011). Big data: The next frontier for innovation, competition, and productivity. [Accessed on August 20, 2014. Available at: http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation].

14. Kavoussi, S.C., Huang, J.J., Tsai, J.C., and Kempton, J.E. (2014). HIPAA for physicians in the information age. Connecticut medicine 78, 425-427.

15. Wang, W., and Krishnan, E. (2014). Big data and clinicians: a review on the state of the science. JMIR medical informatics 2, e1.

16. SAS. (2011). How big is 'big data' in healthcare? [Accessed on April 04, 2015. Available at: http://blogs.sas.com/content/hls/2011/10/21/how-big-is-big-data-in-healthcare/].

17. Pasche, B., and Absher, D. (2011). Whole-genome sequencing: a step closer to personalized medicine. JAMA 305, 1596-1597.

18. Watson, J.D., and Crick, F.H. (1953). Molecular structure of nucleic acids; a structure for deoxyribose nucleic acid. Nature 171, 737-738.

19. (2004). Finishing the euchromatic sequence of the human genome. Nature 431, 931-945.

20. Goodman, D.M., Lynm, C., and Livingston, E.H. (2013). JAMA patient page. Genomic medicine. JAMA 309, 1544.

21. Gibson, W.M. (1971). Can personalized medicine survive? Can Fam Physician 17, 29-88.

22. Arnold, R.M., and Forrow, L. (1990). Rewarding medicine: good doctors and good behavior. Ann Intern Med 113, 794-798.

23. Langreth, R., and Waldholz, M. (1999). New era of personalized medicine: targeting drugs for each unique genetic profile. Oncologist 4, 426-427.

24. Maskos, U., and Southern, E.M. (1992). Oligonucleotide hybridizations on glass supports: a novel linker for oligonucleotide synthesis and hybridization properties of oligonucleotides synthesised in situ. Nucleic Acids Res 20, 1679-1684.

25. Augenlicht, L.H., Wahrman, M.Z., Halsey, H., Anderson, L., Taylor, J., and Lipkin, M. (1987). Expression of cloned sequences in biopsies of human colonic tissue and in colonic carcinoma cells induced to differentiate in vitro. Cancer Res 47, 6017-6021.

26. Augenlicht, L.H., Taylor, J., Anderson, L., and Lipkin, M. (1991). Patterns of gene expression that characterize the colonic mucosa in patients at genetic risk for colonic cancer. Proc Natl Acad Sci U S A 88, 3286-3289.

27. Kulesh, D.A., Clive, D.R., Zarlenga, D.S., and Greene, J.J. (1987). Identification of interferon-modulated proliferation-related cDNA sequences. Proc Natl Acad Sci U S A 84, 8453-8457.

28. Schena, M., Shalon, D., Davis, R.W., and Brown, P.O. (1995). Quantitative monitoring of gene expression patterns with a complementary DNA microarray. Science 270, 467-470.

29. Lashkari, D.A., DeRisi, J.L., McCusker, J.H., Namath, A.F., Gentile, C., Hwang, S.Y., Brown, P.O., and Davis, R.W. (1997). Yeast microarrays for genome wide parallel genetic and gene expression analysis. Proc Natl Acad Sci U S A 94, 13057-13062.

30. Zaidi, S.K., Young, D.W., Choi, J.Y., Pratap, J., Javed, A., Montecino, M., Stein, J.L., Lian, J.B., van Wijnen, A.J., and Stein, G.S. (2004). Intranuclear trafficking: organization and assembly of regulatory machinery for combinatorial biological control. J Biol Chem 279, 43363-43366.

31. Mattick, J.S., Amaral, P.P., Dinger, M.E., Mercer, T.R., and Mehler, M.F. (2009). RNA regulation of epigenetic processes. Bioessays 31, 51-59.

32. Martinez, N.J., and Walhout, A.J. (2009). The interplay between transcription factors and microRNAs in genome-scale regulatory networks. Bioessays 31, 435-445.

33. Tomilin, N.V. (2008). Regulation of mammalian gene expression by retroelements and non-coding tandem repeats. Bioessays 30, 338-348.

34. Online Mendelian Inheritance in Man. [Accessed on March 15. Available at: http://www.omim.org/].

35. Gusella, J.F., Wexler, N.S., Conneally, P.M., Naylor, S.L., Anderson, M.A., Tanzi, R.E., Watkins, P.C., Ottina, K., Wallace, M.R., Sakaguchi, A.Y., et al. (1983). A polymorphic DNA marker genetically linked to Huntington's disease. Nature 306, 234-238.

36. Riordan, J.R., Rommens, J.M., Kerem, B., Alon, N., Rozmahel, R., Grzelczak, Z., Zielenski, J., Lok, S., Plavsic, N., Chou, J.L., et al. (1989). Identification of the cystic fibrosis gene: cloning and characterization of complementary DNA. Science 245, 1066-1073.

37. Strachan T, R.A. (2010). Human Molecular Genetics.

38. Altmuller, J., Palmer, L.J., Fischer, G., Scherb, H., and Wjst, M. (2001). Genomewide scans of complex human diseases: true linkage is hard to find. Am J Hum Genet 69, 936-950.

39. Klein, R.J., Zeiss, C., Chew, E.Y., Tsai, J.Y., Sackler, R.S., Haynes, C., Henning, A.K., SanGiovanni, J.P., Mane, S.M., Mayne, S.T., et al. (2005). Complement factor H polymorphism in age-related macular degeneration. Science 308, 385-389.

40. Yamazaki, K., McGovern, D., Ragoussis, J., Paolucci, M., Butler, H., Jewell, D., Cardon, L., Takazoe, M., Tanaka, T., Ichimori, T., et al. (2005). Single nucleotide polymorphisms in TNFSF15 confer susceptibility to Crohn's disease. Hum Mol Genet 14, 3499-3506.

41. Ozaki, K., and Tanaka, T. (2005). Genome-wide association study to identify SNPs conferring risk of myocardial infarction and their functional analyses. Cell Mol Life Sci 62, 1804-1813.

42. Duerr, R.H., Taylor, K.D., Brant, S.R., Rioux, J.D., Silverberg, M.S., Daly, M.J., Steinhart, A.H., Abraham, C., Regueiro, M., Griffiths, A., et al. (2006). A genome-wide association study identifies IL23R as an inflammatory bowel disease gene. Science 314, 1461-1463.

43. Sladek, R., Rocheleau, G., Rung, J., Dina, C., Shen, L., Serre, D., Boutin, P., Vincent, D., Belisle, A., Hadjadj, S., et al. (2007). A genome-wide association study identifies novel risk loci for type 2 diabetes. Nature 445, 881-885.

44. Barrett, J.C., Hansoul, S., Nicolae, D.L., Cho, J.H., Duerr, R.H., Rioux, J.D., Brant, S.R., Silverberg, M.S., Taylor, K.D., Barmada, M.M., et al. (2008). Genome-wide association defines more than 30 distinct susceptibility loci for Crohn's disease. Nat Genet 40, 955-962.

45. Stahl, E.A., Raychaudhuri, S., Remmers, E.F., Xie, G., Eyre, S., Thomson, B.P., Li, Y., Kurreeman, F.A., Zhernakova, A., Hinks, A., et al. (2010). Genome-wide association study meta-analysis identifies seven new rheumatoid arthritis risk loci. Nat Genet 42, 508-514.

46. Rioux, J.D., Xavier, R.J., Taylor, K.D., Silverberg, M.S., Goyette, P., Huett, A., Green, T., Kuballa, P., Barmada, M.M., Datta, L.W., et al. (2007). Genome-wide association study identifies new susceptibility loci for Crohn disease and implicates autophagy in disease pathogenesis. Nat Genet 39, 596-604.

47. Cho, H.S., Byun, T.J., Ahn, S.B., Kim, T.Y., Eun, C.S., Jeon, Y.C., Kim, Y.S., and Han, D.S. (2008). [A case of familial Crohn's disease observed in a parent and his offspring]. Korean J Gastroenterol 52, 247-250.

48. Mathew, C.G. (2008). New links to the pathogenesis of Crohn disease provided by genome-wide association scans. Nat Rev Genet 9, 9-14.

49. Ghoussaini, M., Song, H., Koessler, T., Al Olama, A.A., Kote-Jarai, Z., Driver, K.E., Pooley, K.A., Ramus, S.J., Kjaer, S.K., Hogdall, E., et al. (2008). Multiple loci with different cancer specificities within the 8q24 gene desert. J Natl Cancer Inst 100, 962-966.

50. Lango Allen, H., Estrada, K., Lettre, G., Berndt, S.I., Weedon, M.N., Rivadeneira, F., Willer, C.J., Jackson, A.U., Vedantam, S., Raychaudhuri, S., et al. (2010). Hundreds of variants clustered in genomic loci and biological pathways affect human height. Nature 467, 832-838.

51. Gudmundsson, J., Sulem, P., Steinthorsdottir, V., Bergthorsson, J.T., Thorleifsson, G., Manolescu, A., Rafnar, T., Gudbjartsson, D., Agnarsson, B.A., Baker, A., et al. (2007). Two variants on chromosome 17 confer prostate cancer risk, and the one in TCF2 protects against type 2 diabetes. Nat Genet 39, 977-983.

52. Illumina, next generation sequencing. [Accessed on March 15. Available at: http://www.illumina.com/Documents/products/Illumina_Sequencing_Introduction.pdf].

53. SMI, next-generation sequencing. [Accessed on March 15. Available at: http://www.smi-online.co.uk/pharmaceuticals/uk/next-generation-sequencing?utm_source=P-075&utm_medium=Bentham Science&utm_campaign=WebBanner].

54. Choi, M., Scholl, U.I., Ji, W., Liu, T., Tikhonova, I.R., Zumbo, P., Nayir, A., Bakkaloglu, A., Ozen, S., Sanjad, S., et al. (2009). Genetic diagnosis by whole exome capture and massively parallel DNA sequencing. Proc Natl Acad Sci U S A 106, 19096-19101.

55. Thorisson, G.A., Muilu, J., and Brookes, A.J. (2009). Genotype-phenotype databases: challenges and solutions for the post-genomic era. Nature reviews Genetics 10, 9-18.

56. NIH. (2015). The Human Genome Project. [Accessed on April 28, 2015. Available at: http://ghr.nlm.nih.gov/handbook/hgp?show=all].

57. FORBES. (2012). How Cloud and Big Data are Impacting the Human Genome - Touching 7 Billion Lives. [Accessed on August 20, 2014. Available at: http://www.forbes.com/sites/sap/2012/04/16/how-cloud-and-big-data-are-impacting-the-human-genome-touching-7-billion-lives/].

58. Thorisson, G.A., Smith, A.V., Krishnan, L., and Stein, L.D. (2005). The International HapMap Project Web site. Genome research 15, 1592-1593.

59. Consortium, T.I.H. (2013). International HapMap Project. [Accessed on August 20, 2014. Available at: http://hapmap.ncbi.nlm.nih.gov].

60. Abecasis, G.R., Auton, A., Brooks, L.D., DePristo, M.A., Durbin, R.M., Handsaker, R.E., Kang, H.M., Marth, G.T., and McVean, G.A. (2012). An integrated map of genetic variation from 1,092 human genomes. Nature 491, 56-65.

61. consortium, G. (2015). 1000 Genomes A Deep Catalog of Human Genetic Variation. [Accessed on April 04, 2015. Available at: http://www.1000genomes.org].

62. Genomes. (2015). How much disk space is used by the 1000 genomes project? [Accessed on April 04, 2015. Available at: http://www.1000genomes.org/faq/how-much-disk-space-used-1000-genomes-project].

63. consortium, G.p. (2015). Statistics - 1000 Genomes A Deep Catalog of Human Genetic Variation. [Accessed on April 04, 2015. Available at: http://www.1000genomes.org/category/statistics].

64. Arstechnica. (2015). Most of what you read was wrong: how press releases rewrote scientific history. [Accessed on April 04, 2015. Available at: http://arstechnica.com/staff/2012/09/10/most-of-what-you-read-was-wrong-how-press-releases-rewrote-scientific-history/].

65. (2012). An integrated encyclopedia of DNA elements in the human genome. Nature 489, 57-74.

66. (2011). A user's guide to the encyclopedia of DNA elements (ENCODE). PLoS biology 9, e1001046.

67. Raney, B.J., Cline, M.S., Rosenbloom, K.R., Dreszer, T.R., Learned, K., Barber, G.P., Meyer, L.R., Sloan, C.A., Malladi, V.S., Roskin, K.M., et al. (2011). ENCODE whole-genome data in the UCSC genome browser (2011 update). Nucleic acids research 39, D871-875.

68. Space, I. (2013). ENCODE: Big Data, the Human Genome, and Non-Profit Global Enterprise. [Accessed on April 04, 2015. Available at: http://infospace.ischool.syr.edu/2013/01/24/encode-big-data-the-human-genome-and-non-profit-global-enterprise/].

69. NCBI. (2015). GenBank Overview. [Accessed on April 04, 2015. Available at: http://www.ncbi.nlm.nih.gov/genbank/].

70. Nakamura, Y., Cochrane, G., and Karsch-Mizrachi, I. (2013). The International Nucleotide Sequence Database Collaboration. Nucleic acids research 41, D21-24.

71. Cochrane, G., Karsch-Mizrachi, I., and Nakamura, Y. (2011). The International Nucleotide Sequence Database Collaboration. Nucleic acids research 39, D15-18.

72. National Cancer Institute. National Human Genome Research Institute. The Cancer Genome Atlas, Data Portal. [Accessed on August 20, 2014. Available at: https://tcga-data.nci.nih.gov/datareports/codeTablesReport.htm].

73. Program, N.C.I. (2014). Cancer Genomics Cloud Pilots Concept Briefing to the NCAB/BSA. [Accessed on April 04, 2015. Available at: http://deainfo.nci.nih.gov/advisory/ncab/165_0613/Komatsoulis.pdf].

74. Levy, S., Sutton, G., Ng, P.C., Feuk, L., Halpern, A.L., Walenz, B.P., Axelrod, N., Huang, J., Kirkness, E.F., Denisov, G., et al. (2007). The diploid genome sequence of an individual human. PLoS biology 5, e254.

75. Venter, J.C., Adams, M.D., Myers, E.W., Li, P.W., Mural, R.J., Sutton, G.G., Smith, H.O., Yandell, M., Evans, C.A., Holt, R.A., et al. (2001). The sequence of the human genome. Science (New York, NY) 291, 1304-1351.

76. Times, T.N.Y. (2007). Genome of DNA Discoverer Is Deciphered. [Accessed on April 04, 2015. Available at: http://www.nytimes.com/2007/06/01/science/01gene.html?_r=2&oref=slogin&].

77. YanHuang, the first asian diploid genome. [accessed on March 15. Available at: http://yh.genomics.org.cn/].

78. The Korean Genome Project. [accessed on March 15. Available at: http://koreangenome.org/index.php/Main_Page].

79. BGI. (2015). YanHuang - The First Asian Diploid Genome. [Accessed on April 04, 2015. Available at: http://yh.genomics.org.cn].

80. Ahn, S.M., Kim, T.H., Lee, S., Kim, D., Ghang, H., Kim, D.S., Kim, B.C., Kim, S.Y., Kim, W.Y., Kim, C., et al. (2009). The first Korean genome sequence and analysis: full genome sequencing for a socio-ethnic group. Genome research 19, 1622-1629.

81. Feero, W.G. (2013). Genomics in medicine: maturation, but not maturity. JAMA 309, 1522-1524.

82. Lupski, J.R., Reid, J.G., Gonzaga-Jauregui, C., Rio Deiros, D., Chen, D.C., Nazareth, L., Bainbridge, M., Dinh, H., Jing, C., Wheeler, D.A., et al. (2010). Whole-genome sequencing in a patient with Charcot-Marie-Tooth neuropathy. The New England journal of medicine 362, 1181-1191.

83. Worthey, E.A., Mayer, A.N., Syverson, G.D., Helbling, D., Bonacci, B.B., Decker, B., Serpe, J.M., Dasu, T., Tschannen, M.R., Veith, R.L., et al. (2011). Making a definitive

diagnosis: successful clinical application of whole exome sequencing in a child with intractable inflammatory bowel disease. Genetics in medicine : official journal of the American College of Medical Genetics 13, 255-262.

84. Mayer, A.N., Dimmock, D.P., Arca, M.J., Bick, D.P., Verbsky, J.W., Worthey, E.A., Jacob, H.J., and Margolis, D.A. (2011). A timely arrival for genomic medicine. Genet Med 13, 195-196.

85. Rios, J., Stein, E., Shendure, J., Hobbs, H.H., and Cohen, J.C. (2010). Identification by whole-genome resequencing of gene defect responsible for severe hypercholesterolemia. Human molecular genetics 19, 4313-4318.

86. Bainbridge, M.N., Wiszniewski, W., Murdock, D.R., Friedman, J., Gonzaga-Jauregui, C., Newsham, I., Reid, J.G., Fink, J.K., Morgan, M.B., Gingras, M.C., et al. (2011). Whole-genome sequencing for optimized patient management. Science translational medicine 3, 87re83.

87. Fay Chang, J.D., Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber. (2006). Bigtable: A Distributed Storage System for Structured Data. [Accessed on April 04, 2015. Available at: http://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf].

88. Giuseppe DeCandia, D.H., Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels. (2007). Dynamo: Amazon's Highly Available Key-value Store. [Accessed on April 04, 2015. Available at: http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf].

89. Cameron, D. (2014). Transforming "Big Data" into Knowledge. [Accessed on April 04,2015. Availabe at: http://hms.harvard.edu/news/transforming-big-data-knowledge%5D.

90. Ian Robinson, J.W.E.E. (2013). Graph Databases.(O'REILLY).

91. Haerder T, e.a. (1983). Principles of transaction-oriented database recovery. ACM Computing Surveys (CSUR) 15, 287-317

92. W., V. (2009). Eventually consistent. Communications of the ACM - Rural engineering development 52, 40-44.

93. Cios, K.J., and Moore, G.W. (2002). Uniqueness of medical data mining. Artificial intelligence in medicine 26, 1-24.

94. al., B.S.e. (1999). Semantic integration of semistructured and structured data sources. ACM Sigmod Record 28, 54-59.

95. Jain, M. (2012). Next-generation sequencing technologies for gene expression profiling in plants. Briefings in functional genomics 11, 63-70.

96. Ozsolak, F., and Milos, P.M. (2011). RNA sequencing: advances, challenges and opportunities. Nature reviews Genetics 12, 87-98.

97. Levenson, V.V., and Melnikov, A.A. (2012). DNA methylation as clinically useful biomarkers-light at the end of the tunnel. Pharmaceuticals (Basel, Switzerland) 5, 94-113.

98. Moya, A., Huisman, L., Ball, E.E., Hayward, D.C., Grasso, L.C., Chua, C.M., Woo, H.N., Gattuso, J.P., Foret, S., and Miller, D.J. (2012). Whole transcriptome analysis of the coral Acropora millepora reveals complex responses to CO(2)-driven acidification during the initiation of calcification. Molecular ecology 21, 2440-2454.

99. Nagalakshmi, U., Wang, Z., Waern, K., Shou, C., Raha, D., Gerstein, M., and Snyder, M. (2008). The transcriptional landscape of the yeast genome defined by RNA sequencing. Science (New York, NY) 320, 1344-1349.

100. D, M. (2011). Metagenomics: Current Innovations and Future Trends.

101. Hall, N. (2007). Advanced sequencing technologies and their wider impact in microbiology. The Journal of experimental biology 210, 1518-1525.

102. Church, G.M. (2006). Genomes for all. Scientific American 294, 46-54.

103. Illumina. (2015). Illumina. [Accessed on April 04, 2015. Available at: http://www.illumina.com/].

104. Corporation, R.D. (2015). 454 sequencing. [Accessed on April 04, 2015. Available at: http://454.com].

105. Technologies, L. (2015). Applied Biosystems. [Accessed on April 04, 2015. Available at: http://www.appliedbiosystems.com].

106. Los, R.K., van Ginneken, A.M., de Wilde, M., and van der Lei, J. (2004). OpenSDE: Row modeling applied to generic structured data entry. Journal of the American Medical Informatics Association : JAMIA 11, 162-165.

107. Prather, J.C., Lobach, D.F., Goodwin, L.K., Hales, J.W., Hage, M.L., and Hammond, W.E. (1997). Medical data mining: knowledge discovery in a clinical data warehouse. Proceedings : a conference of the American Medical Informatics Association / AMIA Annual Fall Symposium AMIA Fall Symposium, 101-105.

108. Rector AL, e.a. (1991). Foundations for an Electronic Medical Record. Published in Methods of Information in Medicine 30, 179-186.

109. B.M.e. (2014). Clinical Database: RDBMS v/s newer technologies (NoSQL and XML database); why look beyond RDBMS and Consider the newer. International Journal of Computer Engineering and Technology (IJCET) 5, 73-83.

110. Williams, M.S. (2014). Genomic medicine implementation: learning by example. American journal of medical genetics Part C, Seminars in medical genetics 166c, 8-14.

111. Hasso Plattner, M.-P.S. (2014). High-Performance In-Memory Genome Data Analysis, How In-Memory Database Technology Accelerates Personalized Medicine.(Springer).

112. Kasabov, N. (2014). Springer Handbook of Bio-/Neuro-Informatics.(Springer).

113. R., J. (1991). The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. .(WILEY).

114. Wescott, B. (2013). The Every Computer Performance Book, Chapter 3: Useful laws. CreateSpace.

115. J., C. (2013). Spanner: Google's Globally Distributed Database. ACM Transactions on Computer Systems (TOCS) 31.

116. Hesham El-Rewini, M.A.-E.-B. (2005). Advanced Computer Architecture and Parallel Processing.

117. (2014). The National Law Review (The Analysis Group, Inc.). Big Data in Health Care. [Accessed on October 20, 2014. Available at: http://www.natlawreview.com/article/big-data-health-care].

118. Centers for Medicare & Medicaid Services. EHR Incentive Programs. [Accessed on August 20, 2014. Available at: https://http://www.cms.gov/ehrincentiveprograms].

119. Office of the National Coordinator for Health Information Technology. Achieve Meaningful Use. [Accessed on August 20, 2014. Available at: https://http://www.healthit.gov/providers-professionals].

120. National Human Genome Research Institute. DNA Sequencing Costs. Data from the NHGRI Genome Sequencing Program (GSP). [Accessed on August 20, 2014. Available at: http://www.genome.gov/sequencingcosts/%5D.

121. PCAST. (2014). Executive Office of the President President's Council of Advisors on Science and Technology. Report to the President: Big Data and privacy: A technological Perspective. [Accessed on August 20, 2014. Available at: http://www.whitehouse.gov/sites/default/files/microsites/ostp/PCAST/pcast_big_data_and_privacy_-_may_2014.pdf.].

122. PGRR. Pittsburgh Genome Resource Repository. Institute for Personalized Medicine. University of Pittsburgh. [Accessed on August 20, 2014. Available at: http://www.pgrr.pitt.edu].

123. PGRR. (2014). TCGA. Pittsburgh Genome Resource Repository. Institute for Personalized Medicine. University of Pittsburgh. [Accessed on August 20, 2014. Available at: http://www.pgrr.pitt.edu/tcga].

124. Labrinidis A., D.A. (2011). Challenges and Opportunities with Big Data. In Purdue e-pubs. [Accessed on August 20, 2014. Available at: http://docs.lib.purdue.edu/cctech/1/].

125. Horwitz, R.I., Cullen, M.R., Abell, J., and Christian, J.B. (2013). Medicine. (De)personalized medicine. Science (New York, NY) 339, 1155-1156.

126. Sharan, R. (2014). Research in Computational Molecular Biology, 18th Annual International Conference, RECOMB 2014, Pittsburgh, PA, USA, Proceedings.(Springer).

127. Christopher J.O. Baker, G.B., Igor Jurisica. (2013). Data Integration in the Life Sciences, 9th International Conference, DILS 2013, Montreal, QC, Canada, Proceedings.(Springer).

128. Sunderraman, R. (2008). Oracle 10g Programming.(PEARSON, Addison Wesley).

129. Navathe, E. (2011). Fundamentals of Database Systems.(PEARSON, Addison Wesley).

130. Chodorow, K. (2013). MongoDB, The Definitive Guide, Powerful and Scalable Data Storage.(O'REILLY).

131. Tiago Macedo, F.O. (2011). Redis Cookbook, Practical techniques for Fast Data Manipulation.(O'REILLY).

132. Hewitt, E. (2011). Cassandra, The Definitive Guide, Distributed Data at Web Scale.(O'REILLY).

133. Mendes, C.L.a.R., D. A. . (1998). Integrated compilation and scalability analysis for parallel systems. In In International Conference on Parallel Architectures and Compilation Techniques. (Paris, France), pp pages 385–392.

134. Mendes, C.L.a.R., D. A. . (2004). Monitoring large systems via statistical sampling. . International Journal of High Performance Computing Applications 18(2):267–277.

135. PCAST. (2008). President's Council of Advisors on Science and Technology. Priorities for Personalized Medicine. [Accessed on August 20, 2014. Available at: http://www.whitehouse.gov/files/documents/ostp/PCAST/pcast_report_v2.pdf].

136. Wade, J.E., Ledbetter, D.H., and Williams, M.S. (2014). Implementation of genomic medicine in a health care delivery system: a value proposition? American journal of medical genetics Part C, Seminars in medical genetics 166c, 112-116.

137. Whitcomb, D.C. (2012). What is personalized medicine and what should it replace? Nature reviews Gastroenterology & hepatology 9, 418-424.

138. (2014). Standards for clinical use of genetic variants. Nature genetics 46, 93.

139. Biesecker, L.G., and Green, R.C. (2014). Diagnostic clinical genome and exome sequencing. The New England journal of medicine 370, 2418-2425.

140. (PSC), P.S.C. (2015). PSC's Computational Resources. [Accessed on April 04, 2015. Available at: http://www.psc.edu/index.php/computing-resources/].

141. NCBI. (2015). ClinVar. [Accessed on April 04, 2015. Available at: http://www.ncbi.nlm.nih.gov/clinvar/intro/].

142. Landrum, M.J., Lee, J.M., Riley, G.R., Jang, W., Rubinstein, W.S., Church, D.M., and Maglott, D.R. (2014). ClinVar: public archive of relationships among sequence variation and human phenotype. Nucleic acids research 42, D980-985.

143. Society, A.C. (2015). What are the risk factors for breast cancer? [Accessed on April 04, 2015. Available at: http://www.cancer.org/cancer/breastcancer/detailedguide/breast-cancer-risk-factors].

144. Institute, N.N.C. (2015). Breast Cancer Risk in American Women. [Accessed on August 20, 2014. Available at: http://www.cancer.gov/types/breast/risk-fact-sheet].

145. Janavičius, R. (2010). Founder BRCA1/2 mutations in the Europe: implications for hereditary breast-ovarian cancer prevention and control. The EPMA Journal 1, 397-412.

146. Popovici, V., Chen, W., Gallas, B.G., Hatzis, C., Shi, W., Samuelson, F.W., Nikolsky, Y., Tsyganova, M., Ishkin, A., Nikolskaya, T., et al. (2010). Effect of training-sample size and classification difficulty on the accuracy of genomic predictors. Breast cancer research : BCR 12, R5.

147. Wang, S., Pandis, I., Wu, C., He, S., Johnson, D., Emam, I., Guitton, F., and Guo, Y. (2014). High dimensional biological data retrieval optimization with NoSQL technology. BMC genomics 15 Suppl 8, S3.

148. Kent, W.J. (1982). IBM Technical Report: A Simple Guide to Five Normal Forms in Relational Database Theory. [Accessed on April 04, 2015. Available at: http://www.bkent.net/Doc/simple5.htm].