International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 3 Issue: 3 1704 - 1708

Are NoSQL Data Stores Useful for Bioinformatics Researchers?

A comparative study of storing and querying strategies for proteomics mass-spectrometry data

Borong Shao, Tim OF Conrad
Freie Universität at Berlin,
Berlin, Germany
Research Campus MODAL, Zuse Institute Berlin,
Berlin, Germany
email: borong.shao@fu-berlin.de, conrad@mi.fu-berlin.de

***Abstract*—**The big data challenge in bioinformatics is approaching. Data storage and processing, instead of experimental technologies, are becoming the slower and more costly part of research. Biological data typically have large size and a variety of structures. The ability to efficiently store and retrieve the data is important in bioinformatics research. Traditionally, large datasets are either stored as disk-based flat-files or in relational databases. These systems become more complicated to plan, maintain and adjust to big data applications as they follow rigid table schema and often lack scalability, e.g. for data aggregation. Meanwhile, non-relational databases (NoSQL) emerge to provide alternative, flexible and more scalable data stores.

In this study, we aim to quantitatively compare the latencies of different data stores on storing and querying proteomics datasets. We show benchmarks for typical relational and non-relational systems for both, in-memory and disk-based configurations and compare them to a simple flat-file based approach. We will focus on the latencies of storing and querying proteomics mass spectrometry datasets and the actual space consumption inside the data stores. Experiments are carried out on a local desktop with medium-sized data, which is the typical experimental settings of individual bioinformatics researchers. Results show that there are significant latency differences among the considered data stores (up to 30 folds). In certain use cases, flat file system can achieve comparable performance with the data stores.

Keywords*-relational vs. non-relational databases, proteomics data, storing and querying latencies*

*****

I. INTRODUCTION

Nowadays, the advances in high-throughput technologies lead to the exponential growth of molecular biological data. Discovering useful information from these data is one of the main endeavors in bioinformatics. In order to achieve it, large amounts and varieties of biological data such as DNA, protein sequences, microarrays and proteomics data need to be stored, retrieved and analyzed. Although new algorithms and pipelines are developed constantly, the gap between the amount of data produced and the amount of data analyzed is still growing [1, 2]. Biological data is eligible for the name “big data" which is often characterized by three “V" properties: volume, velocity, and variety. An efficient data store is required to address the big data challenge in bioinformatics [3-5].

There are mainly two types of database systems: traditional relational databases and non-relational (NoSQL) databases. In relational databases, data are stored in a number of cross-referenced tables and queried through relational algebra operations. Relational databases provide ACID properties (atomicity, consistency, isolation, and durability), which guarantee reliable database transactions. At the same time, this limits the scalability of the databases [6, 7]. As stated in Eric Brewer's CAP theorem [8], a system can have only two properties out of these three properties: consistency, availability, and partition-tolerance. For systems that require ACID transactional properties, relational database is a good option. However, for systems that can relax ACID constraints but address availability and scalability, NoSQL databases may provide alternative options. NoSQL databases have several categories for different types of applications. There are key-value databases such as DynamoDB, column-oriented databases such as HBase and Cassandra, document-based databases such as CouchDB and MongoDB, and graph databases such as Allegro Graph and Neo4j [9]). They generally relax the ACID constraints and provide BASE properties (basically available, soft state, and eventual consistency) instead [10]. The lower level of ACID compliance is traded off for higher availability [6, 11], flexibility [12] and scalability.

Biological data are commonly stored as flat files or in relational databases [13]. Once the data are stored, most of the operations on the data are queries, which serve as the first step of data mining or knowledge discovery [14]. In bioinformatics data analysis, ACID compliance is usually not the critical issue but efficient data mining is [15]. For example, the mass spectrometry data of patients are generated and stored. These data, usually from Megabytes to Gigabytes, need to be queried over and over again to be used in computations such as biomarker identification and protein identification. Therefore, an ideal data store should have low latencies in storing and querying data, while maintaining the consistency. NoSQL data stores are useful to deal with the storage and processing of large volume of data when the structure of the data does not require a relational model [7]. Meanwhile, updates of the data are guaranteed to be propagated to all nodes eventually. It is therefore interesting to investigate whether NoSQL techniques can provide benefits in bioinformatics applications.

There have been a number of qualitative or conceptual studies comparing relational and non-relational databases [9, 10, 16]. They compare databases in terms of data models, query models, consistency models, scalability, maturity, etc. But in practice, it is helpful to have results from quantitative experiments to draw useful conclusions. There have also been a few quantitative studies to compare different databases in biological applications, where some of the data stores are employed in certain use cases. For example, experiments are performed on storing and querying clinical data with a XML-based data store [17]. They conclude that XML database can

store clinical data flexibly but it has higher query latencies than MS-SQL database. In [18], Neo4j and PostgreSQL are used to store and query the STRING human protein interactions network. The queries aim at solving graph processing problems in bioinformatics such as finding the best scoring path between two proteins. The results show that Neo4j can offer great speedups over relational databases. But depending on the types of queries, graph database may not be necessary for graph data.

In this study, we aim to compare different data stores for proteomics mass-spectrometry data. This type of data is important because it fosters a better understanding of diseases, biomarker identification and drug development [19, 20]. Since proteomics data do not have graph data structures, graph data stores are not included in the study. We compare the latencies of one relational database, three NoSQL data stores, and the flat file system on storing and querying mass spectrometry (MS) data, as well as their data sizes. Our choice of the data stores is based on their popularity, availability, and representativity. The four data stores are the representatives of four main database categories. They also cover both in memory and disk-based configurations, as listed below:

- Relational database (MySQL, standard disk-based and in memory configuration)
- Document-oriented database (MongoDB, disk-based)
- Column-oriented database (HBase, disk-based)
- Key-value database (Redis, in memory)

II. METHODS

We perform benchmark studies on storing and querying MS data using different data stores. This section introduces the employed data stores, the respective data models and the experimental settings.

A. Data Stores and Data Models

Each data store has alternative data models to store MS data. We decide the data model for each data store based on its distinguishing features. For example, MongoDB is document-oriented so we store each sample file in one document; HBase is column-oriented thus we store each sample file in one HBase table column. Below we introduce the individual databases and adopted data models.

1) MySQL: MySQL is the most widely used open-source RDBMS (relational database management system). To store MS samples, we create a table with three columns: sample number, m/z value and intensity value. An index is built on the sample number column to accelerate the search for multiple samples. The MySQL table structure is illustrated in Table I. We use both InnoDB engine and MEMORY engine for storing and querying data. Note that for inserting data, we use the bulk load operation to insert one sample file with one statement.

2) MongoDB: MongoDB is a document-oriented data store. It stores data in collections. A MongoDB collection contains documents. A document is composed of field-value pairs. MongoDB can store data flexibly in documents with embedded data models, instead of breaking it into relational table structures. It also supports aggregation operations for complex queries. We store the MS sample files in one collection with one document for one sample. The key of the document is the sample number. Within each document, the field-value pairs store the m/z value-intensity value pairs in the corresponding sample file. As MongoDB does not support float values as field names, we store the m/z value, e.g., 1000.02 as 1000_02 instead. The collection structure is illustrated in Table II.

3) HBase: HBase is a column-oriented data store. It stores data with HTables. A HTable has rows and column families. Data within a row are grouped by column families and data within a column family are identified by column qualifiers. A row key, a column family, a column qualifier and a version number (if present) can exactly specify a cell in a HTable. As suggested in the HBase manual, to achieve better performance, the number of column families should be kept low - usually not more than two or three. Thus we define the m/z values as the row keys and intensity values as the only column family, which has one column for each sample. The HBase table structure is illustrated in Table III.

4) Redis: Redis is an in-memory key-value data store. It is different from a traditional key-value data store in which string keys are associated with string values. In Redis, keys are binary safe so any binary sequence can be used as a key, from a string to an image file. The values can hold complex data structures such as list, set, sorted set, hash, etc. We use both Redis hash and string data models to store MS data and compare their performance. The hash uses field-value pairs to store the m/z value-intensity value pairs. The string simply stores all lines of the sample file as a string. The key of a hash or a string is the sample number. The Redis hash and string data models are illustrated in Table IV.

TABLE I. DATA SCHEMA IN MYSQL TABLE FOR STORING MS DATA

Sample number (smallint(6))	***M/z value (float)***	***Intensity value (smallint(6))***
1	1000.02	29
1	1000.12	21
...	...	...
N	9999.68	5

TABLE II. DATA SCHEMA IN MONGODB COLLECTION FOR STORING MS DATA

Sample:1	1000_02:29	1000_12:21	...	9999_68:1
Sample:2	1000_02:96	1000_12:91	...	9999_68:10
...	...	...	...	...
Sample:N	1000_02:35	1000_12:34	...	9999_68:1

TABLE III. DATA SCHEMA IN HBASE TABLE FOR STORING MS DATA

Row key	***ColumnFamily***
1000.02	sample:1 29 sample:2 96 … sample:N 35
1000.12	sample:1 21 ... sample:N 11
...	...
9999.68	... sample:N 1

TABLE IV. HASH OR STRING DATA MODEL IN REDIS FOR STORING MS DATA

Sample:1	1000.02 29 1000.12 21 … 9999.68 1
Sample:2	1000.02 96 1000.12 91 … 9999.68 10
...	...
Sample:N	1000.02 35 1000.12 34 … 9999.68 1

B. *Experimental Settings*

We use experimental approach to compare the latencies of MySQL, MongoDB, HBase, Redis, and flat file system on storing and querying MS data, as well as the data sizes in them. Java programs and JDBC (Java database connectivity technology) are used to access the databases and flat file system. Below we introduce the data, three use cases and the measurement.

1) Data: We use raw 1D mass spectrometry data in the form of m/z (mass-to-charge ratio) and intensity value pairs, stored in the widely used mzML format[1]. Among other meta information, each MS data sample consists of 42,381 value pairs (about 440KB) in the m/z range from 1000 to 10000 Da, which are encoded as binary strings. To measure the influence of parsing the mzML-XML structure and converting the binary encoding, we also perform experiments just using the m/z value and intensity value pairs, stored as numbers in text files. We will refer to this format as .dta format.

2) Use Cases: The goal of this study is to evaluate the suitabilities of the data stores for bioinformatics researchers. We therefore choose a few common use cases which occur during every day routine when working with MS data. We decide to use the following three examples to serve as proxy applications:

a) Storing new data: insert n (n = 50, 250, 500, 1000, 2000, . . . , 7000) number of MS data samples to each data store.

b) Range query (query1): select all m/z value-intensity value pairs from the available datasets where m/z values are between 1000.02 and 1500.02 Da. We will refer to this query type as query1 in the following sections.

c) Retrieve entire samples (query2): retrieve entire data samples for 10% of all available samples. We will call this query as query2 in the rest of this paper.

3) Measurement: All data stores are configured in standalone mode on a Debian Linux[2] desktop PC equipped with a 4-core Intel Xeon (R) CPU running at 3.3GHz, 7.78GB RAM and a 232.9GB SATA hard-disk drive. For each experiment, the computer is rebooted and only one database system is running. Latencies are measured within the respective experiment implementations. Space consumption of the data inside the database-systems is measured by querying the database management system directly[3]. The MS data files are available on the local hard-disk. All results are averaged over multiple runs.

III. RESULTS AND DISCUSSIONS

We measure the latencies of the use cases and the data size for each data store and plot them against the number of sample files (as shown from Fig. 1 to Fig. 4).

The results show significant performance differences among the data stores with respect to the latencies and data sizes, as far as our experiments with real proteomics (MS) data are concerned. In-memory data stores (Redis, MySQL with MEMORY engine) are generally faster than disk-based data stores (MongoDB, HBase, and MySQL with InnoDB engine). All storage systems show a linear dependency of latencies and data size with respect to the number of files. Thus for each data store, we calculate the average latency of storing or querying one MS sample, and the average size of one MS sample (as shown in Table V). Each table entry therefore does not reflect the average measurement per MS sample given only one observation, e.g., n=7000 samples, but the average measurement across all experiments (n = 50, 250, 500, 1000, 2000... 7000). The experimental results provide useful information and also raise new questions, which require a deeper understanding of the technical details of individual data stores. Below we discuss a few basic observations across all data stores.

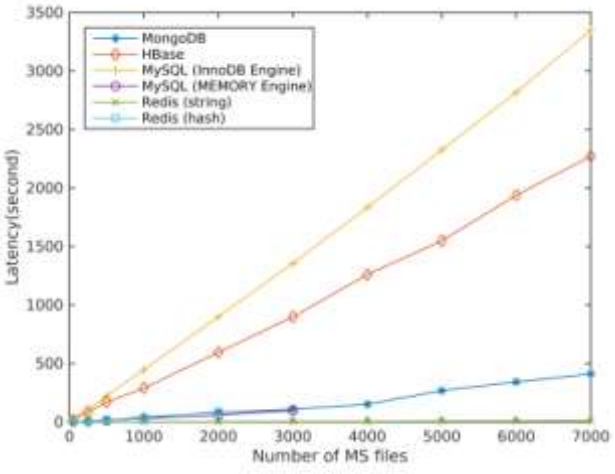

Figure 1 Latencies of storing data to the data stores

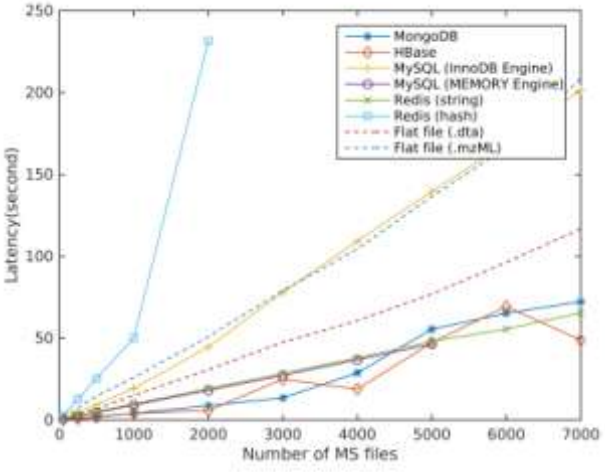

Figure 2 Latencies of querying data by m/z range (query1)

1 The mzML format was introduced by the HUPO-Proteomics Standards Initiative, see [6] for more details.
2 Linux kernel version: SMP Linux 3.2.0-4-amd64
3 MySQL: size of the database, MongoDB: size of the collection, HBase: size of the HTable, Redis: full memory footprint.

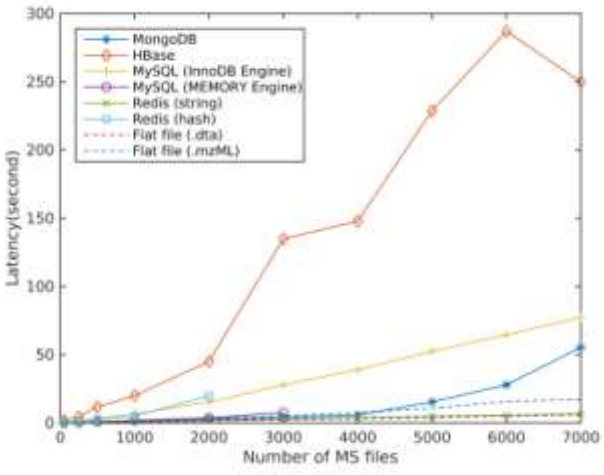

Figure 3 Latencies of querying entire MS samples (query2)

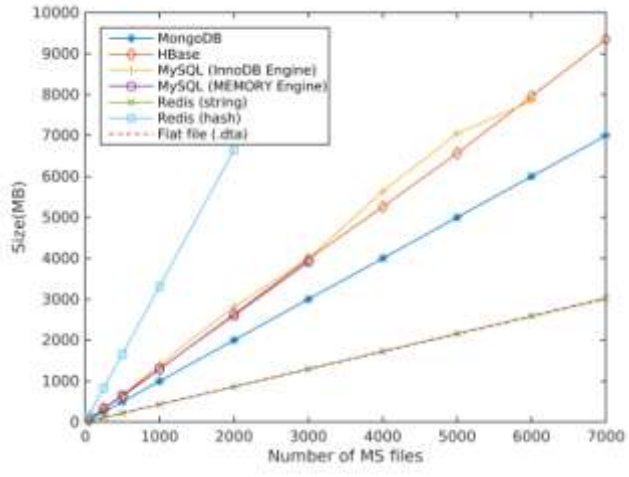

Figure 4 Space of the data used inside the respective database systems

A. *Better memory utilization results in lower latencies*

As expected, the two in memory data stores (Redis and MySQL with MEMORY engine) have lower write and query latencies compared with disk-based data stores (MongoDB, HBase, and MySQL with InnoDB engine). It is interesting to observe that the actually disk-based MongoDB also has very low latencies. This is because MongoDB uses memory-mapped files ("RAM disk") which first utilizes (all) available memory before using the hard-disk. If the available memory is exceeded the average latencies increase from 6.79ms to 10.9ms per MS sample for query1 and from 2.71ms to 8.2ms per MS sample for query2. In summary, if the data are available in memory (instead of disk), the access latencies are much lower.

Accessing data from disk and from memory are intrinsically different. Accessing data from a hard-disk is typically done through the SATA (serial ATA) interface. This has a theoretical maximum bandwidth of 750 MB per second. This is about 20 times slower than accessing the main memory (at a maximal bandwidth of about 14.9GB per second). Additional to this, seeking to the correct position of a file on a hard-disk takes about four milliseconds (using a standard 7200 RPM disk). Taken together, accessing data from the memory is on average 40.000 times faster than accessing data from the disk. Meanwhile, effects like operating system dependent page caching and hardware-based caching mechanisms for disk-reads can reduce the latency of disk read dramatically. The actual effect of the combination of different strategies can hardly be predicted these days due to the complexity of the used components.

B. *Flat file storage can achieve comparable query latencies*

Although querying from flat files involves disk reading, flat file storage achieves comparable query latencies compared to the databases. Flat file storage also has the lowest latency on querying MS data by samples (query2). As mentioned above, techniques such as prefetching and hardware-based caching accelerate the reading from disk, if the reads are sequential. Besides, performing queries in each single sample file avoids the overhead of loading large volume of data to the memory, which can cause page faults and disk swaps if the data does not fit in the memory. Recall that we use two MS file formats: .mzML and .dta in the experiments. The results show that querying .dta files has about half the latencies as querying .mzml files.

C. *Range queries are more expensive*

Our experiments show that querying data ranges is much more expensive than reading entire samples. This behavior is well known and occurs because (1) sequential access (reading a full dataset at once) is faster than random access ("reading a bit and then seeking to the entry point") and (2) databases often implement range queries as first returning all data fulfilling the lower bound and then filtering on the upper bound[4]. This can seriously lengthen the overall query times, and seems to be the case in all tested database systems.

D. *The trade-off between ACID compliance and other properties*

MySQL and HBase generally have higher write and query latencies than other data stores. At the same time, they guarantee higher level of data consistency, which inevitably requires more disk writing. MySQL provides ACID properties. HBase can provide ACID properties within the same row. In comparison, MongoDB does not guarantee ACID properties. It trades off ACID compliance for higher availability which contributes to better speed. As stated in the literatures [1, 10] and confirmed by our experiments, NoSQL data stores relax the ACID compliance for other properties, such as availability and horizontal scalability. Based on the experimental results, we conclude that this trade-off has a good potential for bioinformatics applications.

E. *The suitability of a data store depends on the use case*

In our experiments, the combinations of data stores and data models show different performance for every use case. We summarize our experience in Table VI.

TABLE VI. USE CASES IN WHICH THE DATA STORES MAY BE CONSIDERED

If you have …	*Recommended data store*
Unstructured or flexible data that require complex queries	MongoDB
Large data volume applications	HBase
Data that require a relational model and ACID transactional properties	MySQL
Data that do not require complex queries and can fit in the memory	Redis
Data that only require limited operations	Flat file

[4] There are numerous database-systems dependent approaches to optimize this behavior, which we did not consider in this work.

TABLE V. THE AVERAGE STORING AND QUERYING LATENCIES OF THE DATA STORES

Measurement per MS sample	Data Stores							
	MongoDB	*HBase*	*MySQL (InnoDB)*	*MySQL (MEMORY)*	*Redis (hash)*	*Redis (string)*	*Flat file (dta)*	*Flat file (mzM,L)*
Write latencies (millisecond)	44.45	322.71	490.55	30.98	38.71	1.69	-	-
Range query latencies (millisecond)	6.79 (10.9)[a]	5.60	23.97	10.09	64.80	10.20	15.47	28.38
Sample query latencies (millisecond)	2.71 (8.2)[a]	33.43	9.59	1.93	6.19	1.01	0.89	2.03
Data sizes (KB)	1024.0	1285.8	1851.0	1304.0	3325.7	434.66	440.32	669.1

a. The query latencies when the data size exceeds the available memory

IV. CONCLUSION

In bioinformatics, system-level investigations of cellular and molecular interactions involve large amounts of data. An efficient data store is necessary in this process. We use an experimental approach to compare the performance of a relational database (MySQL) and three non-relational data stores (MongoDB, HBase, Redis) on their latencies of storing and querying mass spectrometry data and the data sizes. We also perform the same queries on a flat file system (both dta and mzML formats) for comparison. To the best of our knowledge, this study is the first quantitative comparison among a relational database, NoSQL data stores, and flat file system in the context of bioinformatics applications, which can provide practical guide to bioinformatics researchers.

The results show that NoSQL data stores with proper data models can achieve lower write and query latencies and smaller database size than relational databases. Depending on the use case, flat file system can achieve comparable query performance as the databases. Above all, our study suggests that the suitabilities of databases need to be considered based on the application context. In the future, we will extend our study by comparing the performance of the databases in distributed mode, for example: HBase with Hadoop and HDFS, MongoDB with sharding technique, and MySQL cluster, which are applicable to a medium-sized bioinformatics data center.

ACKNOWLEDGMENT

This work was funded by the German Ministry of Research and Education (BMBF) project Grant 3FO18501 (Forschungscampus MODAL).

REFERENCES

[1] Schatz Michael C, Langmead Ben, and Salzberg Steven L. Cloud computing and the DNA data race. Nat Biotech, 28(7):691–693, jul 2010.

[2] Lin Dai, Xin Gao, Yan Guo, Jingfa Xiao, and Zhang Zhang. Bioinformatics clouds for big data manipulation. Biology Direct, 7(1):43, 2012.

[3] Marx Vivien. Biology: The big challenges of big data. Nature, 498(7453):255–260, jun 2013.

[4] Casey S. Greene, Jie Tan, Matthew Ung, Jason H. Moore, and Chao Cheng. Big data bioinformatics. Journal of Cellular Physiology, 229(12):1896–1900, 2014.

[5] Savage Neil. Bioinformatics: Big data versus the big C. Nature, 509(7502):S66–S67, may 2014.

[6] B.G. Tudorica and C. Bucur. A comparison between several nosql databases with comments and notes. In Roedunet International Conference (RoEduNet), 2011 10th, pages 1–5, June 2011.

[7] A. B. M. Moniruzzaman and Syed Akhter Hossain. Nosql database: New era of databases for big data analytics - classification, characteristics and comparison. CoRR, abs/1307.0191, 2013.

[8] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News, 33(2):51–59, June 2002.

[9] Clarence JM Tauro, Baswanth Rao Patil, and KR Prashanth. A comparative analysis of different NoSQL databases on data model, query model and replication model. In Proceedings of International Conference on "Emerging Research in Computing, Information, Communication and Applications" ERCICA. Elsevier, 2013.

[10] Ameya Nayak, Anil Poriya, and Dikshay Poojary. Type of NoSQL databases and its comparison with relational databases. International Journal of Applied Information Systems, 5(4):16–19, March 2013.

[11] Rick Cattell. Scalable SQL and NoSQL data stores. SIGMOD Rec., 39(4):12–27, May 2011.

[12] Paolo Atzeni, Francesca Bugiotti, and Luca Rossi. Uniform access to NoSQL systems. Information Systems, 43(0):117 – 133, 2014.

[13] Jeremy O. Baum Marketa J. Zvelebil. Understanding bioinformatics. Garland Science, 2008.

[14] François Bry and Peer Kröger. A computational biology database digest: Data, data analysis, and data management. Distrib. Parallel Databases, 13(1):7–42, January 2003.

[15] Yixue Li, Luonan Chen, Big Biological Data: Challenges and Opportunities, Genomics, Proteomics & Bioinformatics, Volume 12, Issue 5, October 2014.

[16] Clarence J M Tauro, Aravindh S, and Shreeharsha A.b. Comparative study of the new generation, agile, scalable, high performance NoSQL databases. International Journal of Computer Applications, 48(20):1–4, June 2012.

[17] Ken Ka-Yin Lee, Wai-Choi Tang, and Kup-Sze Choi. Alternatives to relational database: comparison of NoSQL and XML approaches for clinical data storage. Computer Methods and Programs in Biomedicine, 110(1):99–109, April 2013.

[18] Christian Theil Have and Lars Juhl Jensen. Are graph databases ready for bioinformatics? Bioinformatics, 2013.

[19] Sam Hanash. Disease proteomics. Nature 422: 226–232, 2003.

[20] Zhen Xiao, DaRue Prieto, Thomas P. Conrads, Timothy D. Veenstra, and Haleem J.Issaq. Proteomic patterns: their potential for disease diagnosis. Molecular and Cellular Endocrinology, 230(12):95 – 106, 2005.

[21] Eric Deutsch. mzml: A single, unifying data format for mass spectrometer output. PROTEOMICS, 8(14):2776–2777, 2008.

[22] VijaySrinivas Agneeswaran. Big-data theoretical, engineering and analytics perspective. In Big Data Analytics, volume 7678 of Lecture Notes in Computer Science, pages 8–15. Springer Berlin Heidelberg, 2012.

[23] N. Leavitt. Will nosql databases live up to their promise? Computer, 43(2):12–14, Feb 2010.