


DATABASE SYSTEM SUGGESTIONS FOR THE INTERNET OF THINGS (IOT) SYSTEMS

Mustafa Utku KALAY,

Computer Engineering Department, Yıldız Technical University, 34220, Istanbul, Turkey, utku@ce.yildiz.edu.tr

 <https://orcid.org/0000-0002-8002-0268>

Received: 19.01.2018, Accepted: 16.05.2018
*Corresponding author

Review Article
DOI:10.22531/muglajsci.418488

Abstract

Internet of Things (IoT) is an interconnection of different types of information assets in which data is continuously generated and transmitted over the Internet. Technologies of the sensor, RFID, GPS, mobile devices, and Internet-enabled actuators play a significant role in IoT systems. IoT brings out new challenges in terms of data and information management because it is not easy to collect and manage a large amount of heterogeneous data that is aggregated at very high velocity as well as to retrieve and manage the information that is hidden within this large volume of data.

In this paper, I discuss the main factors affecting the efficiency of data management in IoT systems, specifically query processing and transaction management. There are many lessons learned from traditional database systems, distributed systems, and sensor networks, however, traditional solutions are often inadequate to meet the needs of applications in such a complex ecosystem, namely IoT. In traditional database systems, for instance, query operations are usually local, and execution costs depend on the current processor power and other resource constraints (i.e. memory). On the other hand, transaction management mechanisms guarantee the ACID properties in order to provide overall data integrity. It is apparent that different types of IOT applications that operate on heterogeneous, streaming, real-time, and geographically distributed large data will significantly change the well-known aspects of querying and transaction management. Context-aware querying, distributed querying, MapReduce computing model and flexible transaction models such as web-based transaction handling are some of the current issues discussed in this paper. With the succinct yet comprehensive information presented in this work, I intend to provide a guide for researchers in the IoT systems, especially in the context of database systems.

Keywords: Internet of Things, Parallel DBMS, NoSQL, MapReduce, streaming data processing

IOT SİSTEMLERİ İÇİN VERİTABANI SİSTEM ÖNERİLERİ

Öz

Nesnelerin İnterneti (IoT), verilerin sürekli olarak üretilip İnternet üzerinden iletiildiği farklı tip bilgi kaynaklarından oluşan bir ağıdır. Sensörler, telsiz frekans tanıma (RFID) cihazları, küresel konumlandırma sistemleri (GPS), mobil cihazlar ve İnternet özellikli aktüatör teknolojileri IoT sistemlerinde önemli bir rol oynamaktadır. IoT, veri ve bilgi yönetimi açısından yeni zorluklar getiriyor, çünkü çok yüksek hızda üretilen büyük miktarda heterojen veriyi toplamak ve işlemenin zorluğu yanında, bu büyük veride gizlenen bilgileri almak ve yönetmek de kolay değildir.

Bu makalede, IoT sistemlerinde veri işleme verimliliğini etkileyen temel faktörleri, özellikle sorgulama ve hareket yönetimini ele alıyorum. Geleneksel veri tabanı sistemlerinden, dağıtık sistemlerden ve sensör ağlarından öğrenilen çok sayıda dersler vardır, ancak geleneksel çözümler, IoT gibi karmaşık bir ekosistemdeki uygulamaların ihtiyaçlarını karşılamada çoğunlukla yetersiz kalmaktadır. Geleneksel veri tabanı sistemlerinde, örneğin, sorgulama işlemleri, genellikle yereldir ve yürütme maliyetleri mevcut işlemci gücü ve bellek gibi kaynak kısıtlamalarına bağlıdır. Diğer taraftan geleneksel hareket yönetimi mekanizmaları, genel veri bütünlüğünü sağlamak için ACID özelliklerini garanti eder. Heterojen, sürekli, gerçek-zamanlı ve coğrafi olarak dağınık büyük veri üzerinde çalışan farklı tip IoT uygulamalarının, sorgulama işleminin ve hareket yönetiminin iyi bilinen yönlerini önemli ölçüde değiştireceği açıktır. İçeriğe duyarlı sorgulama, dağıtılmış sorgulama, MapReduce hesaplama modeli ve web tabanlı hareket yönetimi gibi esnek işlem modelleri bu makalede ele alınan güncel konulardan bazılarıdır. Bu çalışmadaki kısa fakat kapsamlı bilgilerle, IoT sistemlerinde, özellikle veri tabanı sistemleri üzerine, çalışan araştırmacılar için bir kılavuz sağlamayı amaçladım.

Anahtar Kelimeler: Nesnelerin İnterneti, Paralel VTYS, NoSQL, MapReduce, sürekli veri işleme

Cite

Kalay, M.U., (2018). "Database system suggestions for the internet of things (IOT) systems", *Mugla Journal of Science and Technology*, 4(1), 411-423.

1. Introduction

Dissemination of data and processing has been evolved for many years so that several variants of distributed systems with different names have emerged. Among them, the Internet is a global computer network which provides various information and communication facilities. The degree of heterogeneity, scalability, availability, and autonomy of individual sites are the main aspects of distributed processing systems. Since optimizing data processing performance is already crucial for centralized systems, it is considered to be a matter of paramount importance for large-scale data processing within distributed systems.

When we look at the evolution of the Internet, there are a number of salient milestones such as World Wide Web (WWW), Web2.0, social media and Internet of Things (IoT), each one has many different types of applications, standards, and protocols inside. While WWW is just the "internet of content", Web2.0 evolves to exchanging structured information via web services. By means of SOAP (Simple Object Access Protocol), a web-service messaging protocol, web-serviced "programs" that run on different operating systems interact with each other using HTTP and XML and/or JSON (JavaScript Object Notation). There is a more recent easy-to-use protocol as an alternative to SOAP nowadays, named as REST (Representational State Transfer), with a more flexible architecture.

With adding smartphones and their applications to "internet of services" with an affordable broadband access has exposed "internet of people", named as social media. The most distinguished feature of this current era is the user-generated content websites, like Facebook, Twitter, and LinkedIn etc. Most social media websites publish their APIs that enables websites (or applications) to customize the social data easily and to process with a better-integrated manner. Nowadays, as an example, there are many REST-compliant APIs embedded in various programming languages, and, special query languages over social data like Facebook Query Language. This has been a revolution in terms of data access and processing within web-based applications.

The last revolution in the way of using the Internet is named as the Internet of Things (IoT) that enables machines to communicate autonomously with other machines. "Machine" or "Thing" means here everyday objects such as a pen, refrigerator or even ourselves. Connecting all those "things" to Internet transparently is the main promise of IoT. When everyday objects become part of information systems and end-user applications, it is apparent that people's everyday life may change dramatically.

Managing this unprecedented quantity of digital data generated both from people's daily lives (social data) and from the "things" with a smooth integration bring about many new approaches, applications, protocols, algorithms under the name of data science. All those innovations range from marketing industry to even

database internals. Manufacturing, healthcare, and transportation are just some of the industries to profit from the IoT sector. By 2020, it is expected that the number of Internet-enabled devices used in manufacturing and daily life will be more than 50 billion [1].

For all stages of evolution of the Internet, from WWW to IoT, we can ask the question: "What is the motivation for all those rapid innovations?" My short answer is "to develop a more integrated environment in which machines and humans seamlessly interact with each other in a more context-aware manner". In such an extremely heterogeneous environment, every entity including persons, places, computers, software (all "things") should interact with each other so that the system is able to behave in a manner consistent with the users' work at the moment. In fact, this is the context-awareness and its indications can be seen at the beginning of WWW, and even at all types of computing systems in history. The context-aware computing systems (also known as Pervasive Computing or Ubiquitous Computing systems) customize its services based on the user's preferences. The context may be any type of information about entities depending on time, location and weather mostly.

As an example, think of a context-aware framework that predicts traffic status based on real-time data collected from sensors in the vehicles/roads (for vehicle or road current condition) and historical traffic data and the daily social activities in the city and weather. In such an environment, the drivers may be routed instantly under different levels of congestions to get a better traffic density distribution overall. Having smooth integration of more sensitive information collected from different types of sensors, the context-aware system will be able to tune the traffic distribution better.

There are many standardization problems and technical challenges before IoT effectively takes part in our daily life. Among these challenges, data processing, concurrency, heterogeneity, scalability, security, privacy, identification, and addressability are some issues of IoT. In this paper, I focus on database issues related to IoT systems. In next section, I start describing traditional DBMS technology. Inadequacies of the relational model for IoT systems, NoSQL databases, MapReduce framework, flexible transaction models and streaming data processing will be presented in section 3. I conclude the paper in Section 4.

2. Traditional DBMS Technology: Relational (Parallel) DBMS

In this section, I briefly review of traditional DBMS technologies. I attempt to make some clarifications about the need for new approaches on traditional DBMS, which has already great wealth and value in IT industry. The most important actuator in this evolution is, of course, the evolution of the Internet as I discussed in the introduction.

In a relational database management system (RDBMS), data is stored in tables (relations), conforming to a predefined schema. Design process basically includes many phases like normalization, materialization to satisfy the proper constraints for a particular application. SQL, a declarative data manipulation language, dramatically increase application development and end-user productivity. Since DBMS is expected to serve multiple clients concurrently in a real-time manner, query processing and optimization need considerable attention in terms of performance. Having stored the critical (that is, statistical) information (with regular refreshments) needed for query evaluation in system catalog tables, almost the best execution plan is generated in today's centralized DBMSs. Query execution plan, represented as a query tree, includes many low-level physical operations like filtering, joining, sorting, merging, grouping, materialization, etc. Each one can be executed with different algorithms (i.e. merge-join, nested loop join, hash join etc.) based on different system conditions. It is not an easy task to execute many query trees together within a limited amount of resources without sacrificing the high system throughput.

Another aspect of DBMS technology is about the data integrity (or reliability). When some users update data while others concurrently read data, reliability and efficiency receive the second considerable attention. This brings out the notion of transactions which is the other essential part of DBMSs. This notion must satisfy some properties so-called ACID. Many transactions, called as indivisible programs access database concurrently with interleaving of read/write actions from different transactions. While the concurrency is indispensable for system utilization and hence performance, it is a major threat to data integrity (i.e. consistency and isolation). Concurrent operation of transactions is mostly accomplished with lock-based protocols in today's relational database systems. On the other hand, ensuring the atomicity and durability of database transactions is implemented with rollback function for an incomplete transaction and commit function for providing durability with some protocols such as undo-redo, undo-only and redo-only. There are many well-studied and efficient algorithms both in query execution and transaction processing that satisfy reliable data processing.

Now, I turn to very large databases, namely distributed or parallel DBMSs, that is, a database is distributed over many machines. When data storage and computation are distributed, many database system internals needs to be changed. While it is possible to take advantage of parallelism for some execution algorithms with minor modifications, joining tables on many servers is a difficult task for distributed relational databases [2]. Since relational model system solutions, in general, have the scalability problem, "one size fits all" would become more problematic for highly dynamic systems like IoT. On the other hand, transaction processing algorithms (related to lock management and commitment) is very complex in relational distributed databases. Assuring

atomicity of distributed transaction that has components at several sites and its execution in a serializable manner with other distributed transactions require likewise complex algorithms such as 2-phase commit, distributed recovery and distributed locking.

3. Motivations for New Generation Data Management for IoT Systems

NoSQL, the so-called "Not Only SQL" systems are very large database systems distributed over large-scale data storage. NoSQL databases stores and processes irregular or heterogeneous data in a massively parallel manner over a set of commodity computers [3]. Typically generated data in IoT systems are very irregular and heterogeneous and need to be handled with more flexible techniques than those in relational systems. Today, Google, Amazon, Facebook, Twitter etc., most of the largest web companies are using NoSQL based solutions. The reason for severe impact of this new generation databases over traditional relational systems (like parallel DBMS) is that all important design philosophies on system internals, ranging from data modeling/storing and processing and system integrity dramatically change based on some motivations. In the following subsections, I present 5 motivations for a novel distributed database system, especially with regard to characteristics of IoT systems. In fact, first four have already been motivations for NoSQL databases. In some motivations, I briefly introduce the related current technologies, such as MapReduce, BASE principles and stream processing. In the last subsection, I present the most accepted classification of NoSQL data models with a sample IoT application that fits each model.

3.1. Motivation 1: Data Modelling

First changes are in data storage models. After the 2000s, by means of ever-increasing advancement in web technologies, a significant part of database research has motivated to store and process semi-structured data. The idea here is that the real world can be represented better in semi-structured data since entities in the real world are not so regular (or structured) so that at any time additional attributes can be introduced while the formers may be absent in some of the newer data items. In very heterogeneous and dynamic systems like IoT, rapid inter-relations of things constantly change the database's physical and logical schema requirements. Thus, the schema-less approach would be the best-suited solution.

3.2. Motivation 2: Data Processing

Another motivation is the need for fast processing of big data distributed over multiple heterogeneous nodes. Generally, although it is not impossible, it is not easy to express complex events in such environments with SQL. Parallel processing tasks may be expressed with database queries and if needed, with user-defined functions however large-scale and heterogeneous data manipulation is not efficient with SQL internal algorithms. Because, as pointed out earlier, executing

join algorithms within a distributed environment becomes complex as the number of joining tables increases [4]. Thus there are 2 important drawbacks of parallel RDBMS, first is the deficiency of SQL expressive power, the second is the complexity and parallel execution performance degradation.

For example, extracting the outgoing links from a set of HTML documents is difficult to express in SQL systems. Another example can be given from the famous Google Maps which provides navigation guide based on real-time traffic information. The applications should process all road segments within the requested area, then render and display with the most appropriate route. These applications and many others inherently consist of a variety of data types (i.e. audio, video, images, time series), and they cannot be easily processed with classical SQL join and/or aggregation algorithms. Instead, natural language processing, linear algebra, machine learning, text search/mining, and graph algorithms are some of the analytical techniques for processing the emerging data types [5].

Instead of traditional SQL execution algorithms, NoSQL databases usually use the Map-Reduce (MR) model [6] for processing the large amounts of data. There are many discussion papers [7-9] and research blogs [10] on these two technologies. Among them, I briefly present two of them comparing the two classes of systems in detail [7, 8]. First, parallel DBMS and MR both process data over "shared nothing" architectures. Secondly, relational DBMSs requires a pre-defined strict schema, whereas MR directly processes data in any arbitrary format. Many differences in indexing, compression optimizations, programming models, data distribution strategies, and query execution strategies even make these technologies incomparable in a fair manner.

According to its "simplicity" goal, MR framework consists of only 2 functions, Map and Reduce, in which key-value data pairs are processed. Once map function divides the computation work into smaller sub-problems based on the "keys" and then distribute them to available nodes, the aggregate function accumulates distributed individual results and then forms the end result. The number of maps and reduce instances and their locations and organizing the system-level tasks on each node are decided by global MR scheduler. Parallel relational database systems also apply this parallel execution framework, however, the main problem is the fact that the process to load data into relational tables conforming to a rigid predefined schema and parallel query optimization take longer than the MR system [8]. Conversely, MapReduce can process data without loaded into a database. Additionally, there are many valuable implementation tricks to diminish the execution costs in MR system internals [6].

In IoT systems, the operations are typically "append" nature that is heterogeneous data is progressively generated in sensors, as distinct from intensive "updates" in OLTP tables (a kind of relational database). If the sensor data is going to be analyzed especially only

once or twice, loading them into a relational database would be unacceptably slow. Another difficulty is the need for additional software to load heterogeneous data into the database. Even though all these difficulties are handled smoothly, rapid inter-relations of "things" constantly change the database's physical and logical schema requirements [7].

On the other hand, the salient advantage of the relational system is the efficient usage of built-in index structures (like B-tree) even if SQL programmer does not need to be aware of [8]. This is also possible in MR systems but MR programmers must "implement" indexes to accelerate the accesses to the data required for the application. I strongly recommend reading those comparison papers for interested readers [7-10].

Now, the question is: "Where, in IoT systems do we prefer parallel DBMS over MapReduce or vice versa?" As pointed in [11], once data in IoT framework is generated by the end-point "things", they are, if not all of them, reported to aggregation points and then periodically pushed up into the network (typically, cloud) and subsequently to a parallel DBMS. This enables a globalized view of data and understanding of critical long-term trends of applications which is important for data mining. On the other hand, real-time and localized services usually do not propagate data further up to parallel DBMS that is far in the framework, instead, autonomous processing units nearby the sensor nodes may be used. (Nowadays, this type of processing is named as edge or fog computing). I think that MapReduce style is best suited for edge processing so that real-time data is processed in quasi-real-time without loading into DBMS. Although the sensor nodes do not have powerful characteristics (like CPU, battery, memory, bandwidth) and Map Reduce is, on the contrary, originally designed for running on a cluster of commodity servers, specialized Map-reduce frameworks may be best suited for IoT data processing. The works in [12, 13] strengthen this idea.

MapReduce framework seems to be the most appropriate for IOT systems since sensor data is repetitive in nature and as exemplified at the beginning of this section, typical calculations usually require linear analytical algorithms that can be easily transformed to map and reduce functions. On the other hand, latency-sensitive applications such as identifying potential fraud, authentication, and recommending personalized content need special processing techniques different from Map Reduce. Because Map Reduce provides a fast batch processing of big data and does not concern velocity challenges. Traditional techniques may be better for such applications.

3.3. Motivation 3: System Recovery

This motivation is about fault-tolerance. As data sets are disseminated, fault-tolerant computation becomes more crucial. Horizontal data scaling that is, partitioning data across multiple machines is good in terms of scalability whereas it brings out new problems during execution

[14]. Apart from hardware/software problems, the systems may abort some low-priority tasks in some processing units for higher-priority tasks. In such cases, it would be inefficient to re-execution of all other tasks in other processing units, especially for long-running queries. Likely, parallel RDBMS, unfortunately, handles faults during execution with such a coarse-grained approach. This is actually due to the fact that 2-phase commit protocol is robust but very strict (Classical database textbooks like [4] are good sources for interested readers). Therefore, flexible, and global fault tolerance techniques at the system level and/or application level are needed. In fact, horizontal partitioning with a high level of fault-tolerance has been the basic motivation of NoSQL systems. With successful horizontal scaling in NoSQL databases, system resources (cluster nodes at the high granularity) are used most efficiently, providing that a high availability without sacrificing parallel processing power. This is good for IoT systems, because, they are run typically over wireless sensor networks which have usually low-band and non-reliable connections. In contrast, to accommodate growth, relational database management systems vertically scale up by increasing the capacity of system hardware.

Lastly, MapReduce-based systems are well-known for their simple yet efficient mechanisms to handle different kinds of failures [15].

3.4. Motivation 4: System Integrity

Distributed databases that are processing web data with high performance and availability are not expected to adhere to ACID properties due to the fact that while ACID guarantees system integrity at the high level, it causes execution performance degradation. However, complete integrity is not a concern for all applications. For example, while bank transactions require a high level of consistency, in social networking applications, it is more important to serve the millions of simultaneous users in the fastest way. It is clear that in such applications, availability is more important. As a trade-off, however, consistency can be delayed a bit. Basically, in No SQL databases, Consistency and Isolation properties can be relaxed to gain availability by modifying the behavior of long duration locks. Because they are known as the main bottleneck for system global availability. Additionally, reducing the read latency are usually achieved by replicating data over multiple nodes [9].

Resulting new principles are named as the BASE (Basically Available, Soft State, Eventual consistency). The BASE is already considered as a better model for web applications. Basically, BASE systems allow queries to read dirty data. In such systems, clients may experience some inconsistencies as updates are in progress, however, the system guarantees that the data will eventually reach the expected consistent state [9].

To sum up, the main idea is to provide the best combination of consistency and availability for each

application. This flexible approach is best suited for heterogeneous application environments like IoT.

3.5. Motivation 5: Streaming Processing

Instead of store and pull model of traditional database systems, integrated data stream processing is typically based on the data, that is, the system intelligently calculates new results as data arrive. Therefore, in such systems, continuous querying is required. Continuous querying aims continuously evaluating streaming data by using incremental algorithms without optimizing queries from scratch as much as possible [14]. IoT systems typically generate append-only streams and continuous query processors are needed over this activity data. Moreover, due to the enormous and unlimited nature of the data flow, all history cannot be stored for future use. Thus, intelligent sampling and filtering algorithms are needed for a better archiving.

The characteristics mentioned above changes the data processing style within database systems. Additionally, significant limitations in expressing streaming data with SQL emerges many new research topics under the name of DSMS(data stream management systems). A fundamental concept in data stream systems is the sliding “window”, defined as a basic processing unit. [16].

A key research issue for DSMSs is deciding on the best data model and query language. In database literature, there have been many proposals to model the behavior of streaming data, having different query languages associated with them [17-19]. Additionally, NoSQL databases relax many of the traditional constraints associated with streaming data. For example, Apache Kafka [20], is a distributed streaming platform that stores and processes stream of records in a fault-tolerant durable way.

3.6. NoSQL Databases

By its flexible nature, different data models and processing frameworks (like MapReduce) have emerged under the name of NoSQL. I now introduce the most accepted classification of NoSQL data models: key/value, document, wide-column, and graph data models [16]. I will not describe each model in detail here, however, I point out their some distinguishing characteristics that may be important for IoT systems. Each model is characterized to get better performance for different applications. For each model, I give some sample IoT application that fits this model.

First, key/value model simply store key-value pairs in distributed hash tables. Since lookup for a key item is extremely fast and the scalability is the best among others it may be used in applications like managing user profiles/sessions. Dynamo, Amazon's Highly Available Key-value Store is a major contributor to this model.

Second, document model stores each record within a standard document format. Various query and analytics tools can query both semi-structured data elements within document objects and the structure of the document itself. Document-oriented databases are

well suited for different types of applications that require management of different documents (namely content or blog management) like text, email, multimedia, XML and user-generated tweets/comments.

Third, wide-column databases store data in columns rather than rows by means of key/value pairs. This type of storage is good for data compression and aggregate queries. The column-store technology is particularly suitable to process big data in MapReduce framework. In the IoT traffic analytics example that I gave in the introduction, statistical machine learning algorithms on historical data and sensitive real-time information collected from sensors can be processed in MapReduce framework in a quasi-real-time manner. BigTable started in 2004 by Google is a major contributor in this model.

Last, Graph databases stores data within a set of nodes for objects, edges for objects' relationships, and properties for object and relationship attributes expressed as key/value pairs. RDF stores, a semantic web database is a good example that can be stored and processed in graph databases [21].

Figure 1 below shows the NoSQL family in terms of data complexity versus scalability. For example, social network and semantic data are considered as complex data and best managed with graph data model, while very large-scale of simple key/value collections are considered as simple data and best managed with key/value or wide-column data model.

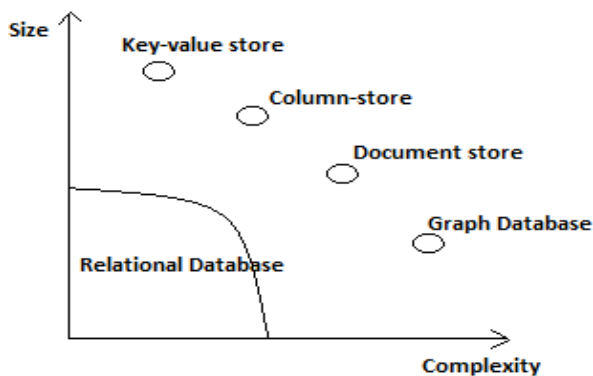


Figure 1. Comparison of NoSQL family with relational model in terms of complexity and scalability [16].

In the last decade, there have been many evaluations and comparisons over different types of databases including relational and NoSQL family in order to reveal their differences in performance, usage, and complexity [9], [22, 23]. Among them, extensive test results for four popular databases (MySQL, MongoDB, CouchDB, and Redis) located in the cloud are presented in a thesis [9]. The tests including basic read/write operations with different workloads are run to measure the average latency for each database system. The workloads include two typical IoT data types: *sensor scalar data* and *multimedia data*. The experiments are conducted with many other setup parameters such as the number of concurrent clients, query types and database configurations. The test results report the latency of bulk insert, read/write latency and index support in the query

performance. This thesis shows the popularity of NoSQL databases against conventional relational database systems, especially for IoT systems. I believe that this valuable thesis can be a good starting point for those who are curious the pros and cons of current database systems.

4. Conclusion

Having a better understanding of how collective data is used with the evolution of the Internet, intelligent algorithms, and tools to analyze this information may add greater efficiency to our lives with making our society safer and healthier. IoT's main promise is to integrate the physical objects seamlessly with the Internet, specifically with the web services. Eventually, "things" can become active participants in our daily life and business processes. Services may interact with these "smart objects" by querying their state and any related information. Since IoT systems generate a large amount of data at very high velocity, appropriate data management is maybe the most critical part of this ecosystem. In this paper, I listed 5 motivations to better understand the need for new approaches over traditional DBMS design principals. In the first 4 motivations, I introduced schema-less approach, MapReduce data processing framework with horizontal scalability and high fault-tolerance, then discussed the relaxing some of the ACID properties which are considered as the distinguished issues of emerging database technologies. At the last motivation, I introduce streaming data processing and its importance for IoT systems.

I conclude that while NoSQL databases open new perspectives providing improved availability and flexibility that is a need in IoT systems, it is apparent that there is still much more room for future research for data management of IoT systems.

5. Acknowledgment

I thank Veli Hakkoymaz for his valuable feedback on this paper.

6. References

- [1] Welbourne E., Battle L., Cole G., Gould K., Rector K., Raymer S., Balazinska M., and Borriello G., "Building the internet of things using rfid: The rfid ecosystem experience," *Internet Computing, IEEE*, vol. 13, no. 3, pp. 48-55, 2009.
- [2] Leavitt N., "Will nosql databases live up to their promise?" *Journal Computer Vol. 43, Issue 2* (2010), 12-14.
- [3] Moniruzzaman A.B.M., Hossain S., (2013). "NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison." *Int J Database Theory Appl.* 6.
- [4] Ramakrishnan R., and Gehrke J., *Database Management Systems (Third Edition)*. McGraw-Hill, Boston, 2003.
- [5] Laney D., "3-D Data Management: Controlling Data Volume, Velocity and Variety" *Research Note, META Group*, February 2001.

- [6] Dean J. and Ghemawat S., "Mapreduce: simplified data processing on large clusters." *Communications of the ACM* 51, 1 (2008), 107-113.
- [7] Dean J., Ghemawat S., "MapReduce: A Flexible Data Processing Tool." *Commun. ACM.* 53. 72-77. (2010)
- [8] Pavlo A., Paulson E., Rasin A., Abadi D.J., DeWitt D.J., Madden S., and Stonebraker M., "A comparison of approaches to large-scale data analysis" In *Proceedings of the 2009 ACM SIGMOD International Conference ACM Press, New York, 2009*
- [9] Phan T. A. M., Nurminen J. K. and Francesco M. Di, "Cloud Databases for Internet-of-Things Data," 2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing(CPSCoM),Taipei,117-124.
- [10] Dewitt D., and Stonebraker M., "MapReduce: A Major Step Backwards" Available: <http://databasecolumn.vertica.com/database-innovation/mapreduce-a-majorstep-backwards/>
- [11] Abu-Elkheir M., Hayajneh M., Ali NA.. "Data Management for the Internet of Things: Design Primitives and Solution". *Sensors* (Basel, Switzerland). 2013;13(11):15582-15612. doi:10.3390/s131115582.
- [12] Jonathan A., Ryden M., Oh K., Chandra A. and Weissman J., "Nebula: Distributed Edge Cloud for Data Intensive Computing," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 11, pp. 3229-3242, Nov. 1 2017.
- [13] Satoh I., "MapReduce-Based Data Processing on IoT," *International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCoM), Taipei, 2014*, pp. 161-168.
- [14] Babu S. and Herodotou H.. "Massively Parallel Databases and MapReduce Systems" *Foundations and Trends in Databases* Vol. 5, No. 1 (2012) 1–104c 2013 DOI: 10.1561/1900000036
- [15] Memishi B., Ibrahim S., Pérez M.S., Antoniu G. "Fault Tolerance in MapReduce: A Survey." *Resource Management for Big Data Platforms. Computer Communications and Networks.* Springer, Cham(2016)
- [16] Francesco C., Massimo D. S., Vincenzo M., Picariello A., Schreiber F. A., Tanca L. *Data Management in Pervasive Systems, Data-Centric Systems and Applications book series (DCSA), (2015) DOI: <https://doi.org/10.1007/978-3-319-20062-0>*
- [17] Arasu A., Babu S., Widom J., "The cql continuous query language: semantic foundations and query execution." *J. Int. J. Very Large Data Bases* 15(2), 121–142 (2006)
- [18] Chandrasekaran S., Cooper O., Deshpande A., Franklin M.J., Hellerstein J.M., Hong W., Krishnamurthy S., Madden S.R., Reiss F., Shah M.A.; "Telegraphcq: continuous dataflow processing." In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pp. 668–668. ACM, New York (2003)
- [19] Chen J., DeWitt D.J., Tian F., Wang Y., "NiagaraCQ: a scalable continuous query system for internet databases." In: *ACM SIGMOD Record*, vol. 29, pp. 379–390. ACM, New York (2000)
- [20] Apache Kafka. Available: <https://kafka.apache.org/intro>
- [21] Liebig T., Vialard V., Opitz M., and Metzl S., "GraphScale: Adding Expressive Reasoning to Semantic Data Stores." *Demo Proceedings of the 14th International Semantic Web Conference (ISWC 2015)*
- [22] Bartholomew D., "Sql vs. nosql". *Linux Journal* 2010, 195 (2010), 4.
- [23] Hecht R. and Jablonski S., "Nosql evaluation: A use case oriented survey" In *Cloud and Service Computing (CSC), 2011 International Conference*, IEEE, pp. 336-341.