A NoSQL-Based Approach for Real-Time Managing of Embedded Data Bases

Afef gueidi
Faculty of Mathematical, Physical and Natural Sciences of Tunis (FST)
Tunis El Manar University, TUNISIA
Email: gueidi.afef@gmail.com

Hamza GHARSELLAOUI
National Institute of Applied Siences and Technology (INSAT)
Carthage University, TUNISIA
National Engineering School of Carthage (ENI-Carthage)
Carthage University, TUNISIA
Email: gharsellaoui.hamza@gmail.com

Samir BEN AHMED
Faculty of Mathematical, Physical and Natural Sciences of Tunis (FST)
Tunis El Manar University, TUNISIA
Email: samir.benahmed@fst.rnu.tn

***Abstract*—This paper deals with a multi-objective extracting, managing and interrogating problem for a reconfigurable real-time embedded databases. In nowaday industry, embedded databases which display a mixture of multimedia signals and big data with modal interfaces need to be highly reconfigurable to meet real-time constraints, data stores optimization problems and to solve requirements problem in order to achieve high scalability and availability. In this case, new methods are required and NoSQL database created for solving the mentioned sub-problems and to be able to store big data effectively, demand for high performance when extracting, managing and interrogating these embedded databases. We also discuss the advantages and disadvantages of a NoSQL approach.**

***Keywords*-component; formatting; style; styling;**

I. Introduction

Nowadays, due to the growing class of portable systems, such as personal computing and communication devices, embedded and real-time systems contain new complex software which are increasing by the time. This complexity is growing because many available software development models don't take into account the specific needs of embedded and systems development. The software engineering principles for embedded system should address specific constraints such as hard timing constraints, limited memory and power use, predefined hardware platform technology, and hardware costs. On the other hand, the new generations of embedded control systems are addressing new criteria such as flexibility and agility [Gharsellaoui, 2012]. For these reasons, there is a need to develop tools, methodologies in embedded software engineering and dynamic reconfigurable embedded control systems as an independent discipline. Each system is a subset of tasks. Each task is characterized by its worst case execution times (WCETs) C_i^{p,ψ_h}, an offset (release time) a_i^{p,ψ_h}, a period T_i^{p,ψ_h} and a deadline D_i^{p,ψ_h} for each reconfiguration scenario ψ_h, (h $\in$ 1..M, we assume that we have M reconfiguration scenarios) and on each processor p, (p $\in$ 1..K, we assume that we have K identical processors numbered from 1 to K), and n real-time tasks numbered from 1 to n that composed a feasible subset of tasks entitled ξ_{old} and need to be scheduled [Gharsellaoui, 2012]. The general goal is to be reassured that any reconfiguration scenario ψ_h changing the implementation of the embedded system does not violate real-time constraints: i.e. the system is feasible and meets real-time constraints even if we change its implementation after any reconfiguration scenario. In [Gharsellaoui, 2013] and in order to obtain this optimization, the authors propose an intelligent agent-based architecture in which a software agent is deployed to dynamically adapt the system to its environment by applying reconfiguration scenarios. A reconfiguration scenario ψ_h means the addition, removal or update of tasks in order to save the whole system on the occurrence of hardware/software faults, or also to improve its performance when random disturbances happen at run-time. In this paper, a software agent in the intelligent agent-based architecture interrogates NoSQL embedded database which ensures predictive and organized software reuse to achieve high scalability and availability and to optimize response time. This process for solving the mentioned sub-problems and to be able to store big data effectively, demand for high performance when extracting, managing and interrogating these embedded databases is proven. Indeed, NoSQL databases are available as commercial and open source database management software from various

IT vendors for handling large amounts of unstructured, semi-structured and hybrid data at an amazing performance with reduced complexity and cost. NoSQL databases are specifically designed for low cost commodity hardware, mostly used for storage and access of data across multiple storage clusters. Features of NoSQL databases includes non-relational, distributed, open-source and horizontally scalable. Other characteristics includes schema-free, easy replication support, simple API, eventually consistent (BASE not ACID) and so on.
Our study concerns embedded databases which display a mixture of multimedia signals and big data with modal interfaces need to be highly reconfigurable to meet real-time constraints, data stores optimization problems and to solve requirements problem in order to achieve high scalability and availability. Indeed, currently, embedded databases are omnipresent electronic devices and embedded technologies presented in all industrial areas like telecommunication, avionic, automotive, medicine, etc,. On one hand, the volume of data is increasing at an enormous rate in these embedded technologies and on the other hand, the cost associated with scaling of the relational RDBMs is began also very expensive. Also, nowadays, industry and especially embedded systems have now to deal with faster and faster evolutions of their users requirements and must be able to adapt their behavior system and to meet real-time constraints. As a consequence, embedded databases have to evolve in order to become more reconfigurable which satisfy embedded systems needs and has very speed response time and low-power cost at the same time [Gajendran, 2012]. In contrast, NoSQL data stores are designed to scale well horizontally and run on commodity hardware. The term ”NoSQL” was first coined in 1998 by Carlo Strozzi to distinguish his solution from other RDMBS solutions which utilize SQL (Strozzis NoSQL still adheres to the relational model). He used the term NoSQL just for the reason that his database did not expose a SQL interface. Recently, the term NoSQL (meaning ”not only SQL”) has come to describe a large class of databases which do not have properties of traditional relational databases and which are generally not queried with SQL (structured query language) [Gajendran, 2012]. The term revived in the recent times with big companies like Google/Amazon using their own data stores to store and process huge amounts of data as they appear in their applications and inspiring other vendors as well on these terms [Gajendran, 2012].
The ability of optimizing the resource allocation in a distributed environment through the management of expansion and contraction of available tasks is an important feature in NoSQL DBMS. With the changes in available embedded systems, it is possible to automatically redistribute the data or to have different shards of data. This ability is important to the performance of the database because it influences the latency of the system. NoSQL database was designed to overcome limitations of relational database in supporting distributed processing of data. For this reason and in order to optimize the whole reconfigurable real-time embedded systems, we will adopt in our present and original work the NoSQL database for the optimization of multi-objective extracting, managing and interrogating of a reconfigurable real-time embedded databases to meet real-time constraints and to reduce storage memory and response time in some critical applications.
The paper is organized as follows. In the next section, we describe the NoSQL database statement and its four categories. In section 3, we present the literature review study and given the advantages and disadvantages of NoSQL database. We give in section 4, a our original proposed NoSQL-based approach for real-time managing of embedded databases to highlight our study. Finally, we present conclusions and perspectives to our work in section 5.

II. NoSQL Databases

Generally, NoSQL isn’t relational, and it is designed for distributed data stores for very large scale data needs (e.g. Facebook or Twitter accumulate Terabits of data every day for millions of its users), there is no fixed schema and no joins. Meanwhile, relational database management systems (RDBMS) ”scale up” by getting faster and faster hardware and adding memory. NoSQL, on the other hand, can take advantage of ”scaling out” - which means spreading the load over many commodity systems [Mikayel, 2011]. The acronym NoSQL was coined in 1998, and while many think NoSQL is a derogatory term created to poke fun at SQL, in reality it means ”Not Only SQL” rather than ”No SQL at all.” The idea is that both technologies (NoSQL and RDBMSs) can co-exist and each has its place. Companies like Facebook, Twitter, Digg, Amazon, LinkedIn and Google all use NoSQL in some way - so the term has been in the current news often over the past few years.

A. Services Model

The processing to be performed as part of activities management can be grouped into five broad categories: computing services, analysis, archive services, display and CRUD services. The CRUD services (Create, Read, Update and Delete) are low-level services that enable document management. Indeed, RDBMSs have their limitations like these three following problems:

1) RDBMSs use a table-based normalization approach to data, and that’s a limited model. Certain data structures cannot be represented without tampering with the data, programs, or both.
2) They allow versioning or activities like: Create, Read, Update and Delete. For databases, updates should never be allowed, because they destroy information. Rather, when data changes, the database should just

add another record and note duly the previous value for that record.
3) Performance falls off as RDBMSs normalize data. The reason: Normalization requires more tables, table joins, keys and indexes and thus more internal database operations for implement queries. Pretty soon, the database starts to grow into the terabytes, and that's when things slow down [Mikayel, 2011].

B. The Four Categories of NoSQL:

In this subsection, we will describe the four categories of NoSQL databases.

1) Key-values Stores: The main idea here is using a hash table where there is a unique key and a pointer to a particular item of data. The Key/value model is the simplest and easiest to implement. But it is inefficient when you are only interested in querying or updating part of a value, among other disadvantages [Mikayel, 2011].

2) Column Family Stores: These were created to store and process very large amounts of data distributed over many machines. There are still keys but they point to multiple columns. The columns are arranged by column family [Mikayel, 2011].

3) Document Databases: These were inspired by Lotus Notes and are similar to key-value stores. The model is basically versioned documents that are collections of other key-value collections. The semi-structured documents are stored in formats like JSON. Document databases are essentially the next level of Key/value, allowing nested values associated with each key. Document databases support querying more efficiently [Mikayel, 2011].

4) Graph Databases: Instead of tables of rows and columns and the rigid structure of SQL, a flexible graph model is used which, again, can scale across multiple machines. NoSQL databases do not provide a high-level declarative query language like SQL to avoid overtime in processing. Rather, querying these databases is data-model specific. Many of the NoSQL platforms allow for RESTful interfaces to the data, while other offer query APIs. Generally, the best places to use NoSQL technology is where the data model is simple; where flexibility is more important than strict control over defined data structures; where high performance is a must; strict data consistency is not required; and where it is easy to map complex values to known keys [Mikayel, 2011].

C. Some Examples of When to Use NoSQL

The following are some examples when to use NoSQL database:

Logging/Archiving.
Log-mining tools are handy because they can access logs across servers, relate them and analyze them.

Social Computing Insight.
Many enterprises today have provided their users with the ability to do social computing through message forums, blogs etc.

External Data Feed Integration.
Many companies need to integrate data coming from business partners. Even if the two parties conduct numerous discussions and negotiations, enterprises have little control over the format of the data coming to them. Also, there are many situations where those formats change very frequently - based on the changes in the business needs of partners.

Front-end order processing systems.
Today, the volume of orders, applications and service requests flowing through different channels to retailers, bankers and Insurance providers, entertainment service providers, logistic providers, etc. is enormous. These requests need to be captured without any interruption whenever an end user makes a transaction from anywhere in the world. After, a reconciliation system typically updates them to back-end systems as well as updates the end user on his/her order status.

Real-time stats/analytics.
Sometimes, it is necessary to use the database as a way to track real-time performance metrics for websites (page views, unique visits, etc.). Tools like Google Analytics are great but not real-time - sometimes it is useful to build a secondary system that provides basic real-time stats. Other alternatives, such as 24/7 monitoring of web traffic, are a good way to go, too.

III. LITERATURE REVIEW

In the last days, there are so many NoSQL systems that it's hard to get a quick overview of the major trade-offs involved when evaluating relational and non-relational systems in non-single-server environments. We will give in this section some preliminaries about NoSQL database and its characteristics.

A. Background

NoSQL databases are finding significance in bigdata analytics, real time applications and Online Transaction Processing (OLTP). Also, most RDBMSs guarantee so called ACID transactions. ACID is an acronym of four properties. These are, in order [Ullman, 2008]:

- **Atomicity:** Transactions are atomic. That is, a transaction is executed either in its entirety or not at all.
- **Consistency:** Every transaction takes the database from one valid state to another.
- **Isolation:** A running transaction will not interfere with another transaction.

- **Durability:** The effect of a transaction must persist and thus never be lost.

Most NoSQL databases share a common set of characteristics [Strauch, 2012]. Naturally, since NoSQL is a broad concept, more or less all of these characteristics have exceptions. Still, they can serve to give a general idea about what NoSQL databases are:

- **Distributed:** NoSQL databases are often distributed systems where several machines cooperate in clusters to provide clients with data. Each piece of data is commonly replicated over several machines for redundancy and high availability.
- **Horizontal scalability:** Nodes can often be dynamically added to (or removed from) a cluster without any downtime, giving linear effects on storage and overall processing capacities. Usually, there is no (realistic) upper bound on the number of machines that can be added.
- **Built for large volumes:** Many NoSQL systems were built to be able to store and process enormous amounts of data quickly.
- **BASE instead of ACID:** Brewer's CAP theorem [Brewer., 2012] states that a distributed system can have at most two of the three properties Consistency, Availability and Partition tolerance. For a growing number of applications, having the last two are most important. Building a database with these while providing ACID properties is difficult, which is why Consistency and Isolation often are forfeited, resulting in the so called BASE approach [Brewer., 2012].

- SQL is unsupported: While most RDBMSs support some dialect of SQL, NoSQL variants generally do not. Instead, each individual system has its own query interface. Recently, the Unstructured Data Query Language 6 was proposed as an attempt to unify the query interfaces of NoSQL databases [Jackson., 2012].

B. CAP theorem and NoSQL database classification

In 2000, Professor Eric Brewer put forward the famous CAP theorem. That is, Consistency, Availability, tolerance of network Partition. CAP theorem's core idea is a distributed system cannot meet the three district needs simultaneously, but can only meet two. According to CAP theorem and different concerns of NoSQL database, a preliminary classification of NOSQL databases is as follows [Nathan, 2010]:

- **Concerned about consistency and availability (CA):** Part of the database is not concerned about the partition tolerance, and mainly use of Replication approach to ensure data consistency and availability. Systems concern the CA are: the traditional relational database, Vertica (Column-oriented), Aster Data (Relational), Greenplum (Relational) and so on.
- **Concerned about consistency and partition tolerance (CP):** Such a database system stores data in the distributed nodes, but also ensure the consistency of these data, but support not good enough for the availability. The main CP system: BigTable (Column-oriented), Hypertable (Column-oriented), HBase (Column-oriented), MongoDB (Document), Terrastore (Document), Redis (Key-value), Scalaris (Key-value), MemcacheDB (Key-value), Berkeley DB (Key-value).
- **Concerned about availability and partition tolerance (AP):** Such systems ensure availability and partition tolerance primarily by achieving consistency, AP's system: Voldemort (Key-value), Tokyo Cabinet (Keyvalue), KAI (Key-value), CouchDB (Documentoriented), SimpleDB (Document-oriented), Riak (Document-oriented).

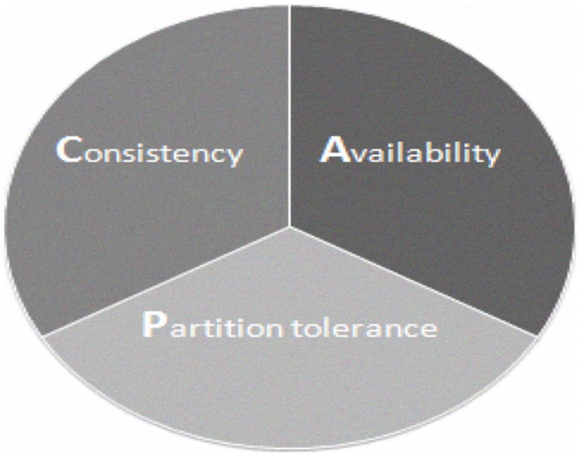

Figure 2: CAP Theorem

In our work, we will adopt the last one (the availability and partition tolerance (AP)) in order to solve a multi-objective extracting, managing and interrogating problem for a reconfigurable real-time embedded databases and to achieve high scalability and availability. In this case, NoSQL database created for solving the mentioned problem and to be able to store big data effectively, demand for high performance when extracting, managing and interrogating these embedded databases.

IV. PROPOSED APPROACH

In this section and in order to solve the described problem and to achieve the objective, in the following we will provide an optimization algorithm that takes smaller interrogation response time and big data store in embedded data bases using NoSQL. NoSQL database was designed to handle large volumes of data processing by removing some supports that existed in RDBMS. One of these supports is ad-hoc query. Our proposed algorithm is described like the following:

A. Proposed Algorithm

The dynamic based NoSQL database guarantee test in terms of residual time, which is a convenient parameter to deal with both normal and overload conditions in embedded databases is presented here.

```
Algorithm GUARANTEE($\xi$; $\sigma_a$)
begin t = get current time();
$R_0$ = 0;
$d_0$ = t;
Insert $\sigma_a$ in the ordered task linked list;
$\xi$` = $\xi \bigcup \sigma_a$;
k = position of $\sigma_a$ in the task set $\xi$`;
for each task $\sigma_i$` such that i $\geq$ k do {
$R_i$ = $R_{i-1}$ + ($d_i$ - $d_{i-1}$) - $c_i$;
if ($R_i$ < 0) then return ("Not Guaranteed");
}
return ("Guaranteed");
end
```

B. Design elements extraction and SPL design extraction

In order to position our proposed approach in a unified framework that fulfills the real-time constraints, we propose some characteristics to be respected.
The organization and structure of the unified framework that fulfills the real-time constraints is retrieved based on the following construct rules:

- **R1:** Each real-time embedded system is geo-localized, either to an industry or to mobile devices. thus, it is necessary to physically move resources around to perform the real-time embedded systems.
- **R2:** A real-time embedded system is composed of tasks. A task is characterized by its worst case execution times (WCETs) C_i^{p,ψ_h}, an offset (release time) a_i^{p,ψ_h}, a period T_i^{p,ψ_h} and a deadline D_i^{p,ψ_h}. The task must respect these temporal characteristics with more or less regular repetitions.
- **R3:** The tasks to be scheduled are predefined and listed in a NoSQL database ordered by such criteria. For each task, we must define the types of parameters. This NoSQL database must be available and regularly maintained and updated in order to be ready for any embedded system reconfiguration, interrogation, tasks modifications or data extraction.
- **R4:** Continually assess tasks and allow for modifications concerning the evolution of the real-time system at run-time providing intervention and reconfiguration in order to save the whole system on the occurrence of hardware/software faults, or also to improve its performance when random disturbances happen at run-time.
- **R5:** A random disturbance is defined as any external event like adding, removal of tasks or just modification of task parameters in order to optimize the response time and the storage capacity of the embedded system database.

C. Discussion of NoSQL Proposed Approach

This proposed process permits us, first, to derive design enriched with optional and mandatory real-time constraints to be highly reconfigurable to meet real-time constraints, data stores optimization problems and to solve requirements problem in order to achieve high scalability and availability. To the authors knowledge, no unified approach exists to handle similar situations for different real-time embedded systems. This paper work allows service providers to manage geo-localized (distributed) tasks based on the NoSQL database regardless of the nature of their services. Finally, we propose to implement the design of our proposed NoSQL database using MongoDB and UML Marte profile for the conception model enriched with the information and real-time constraints in order to re-obtain the feasibility of the embedded systems under study. This NoSQL approach presents several advantages over other technologies in the sense that is easier to structure the data to specific real-time needs; the design is closer to the implementation and development is very simple and it is easier to modify processing and evolve data structures. Nonetheless, our proposed approach suffers from some drawbacks, including difficulty expressing integrity real-time constraints in embedded databases, installation difficulty and lack of the real-time control of the transactions. We will work hardly in the next issues in order to handle these difficulties.

V. CONCLUSION

This paper describes the background of NoSQL, after understand the four categories of NoSQL, then the advantages and disadvantages of NoSQL database were analyzed. After a brief introduction of the CAP theorem, classification of NoSQL database was proposed. Finally, our proposed approach based on the NoSQL approach, was presented in such away that will help user to choose NoSQL database in practice. However, NoSQL database still have various limitations, like using NoSQL in cloud computing and internet of things which is not clear enough and we will be to strengthen our future research work in these two areas.

REFERENCES

[Brewer., 2012] Brewer., E. (2012). Towards robust distributed systems. *Accessed January 25, 2012. July 2000. url: http://www.cs.berkeley.edu/ brewer/cs262b-2004/PODC-keynote.pdf.*

[Gajendran, 2012] Gajendran, S. K. (2012). A survey on nosql databases. *masters.dgtu.donetsk.ua.*

[Jackson., 2012] Jackson., J. (2012). Couchbase, sqlite launch unified nosql query language. *Accessed January 25, 2012. July 2011. url: http://www.arnnet.com.au/ article/395469/couchbase sqlite launch unified nosql query language.*, pages 395–469.

[Strauch, 2012] Strauch, C. (2012). *NoSQL Databases*. Accessed January 25, 2012. Feb. 2011. url: http://www.christof-strauch.de/nosqldbs.pdf.

[Nathan, 2010] Nathan H. (2010). *Visual Guide to NoSQL Systems*. http://blog.nahurst.comlvisual-guide-to-NoSQL-systems.

[Ullman, 2008] Ullman J.D., Garcia-Molina H., and Widom J.,. (2008). *Database Systems: The Complete Book*. Prentice Hall.

[Mikayel, 2011] Mikayel V.,. (2011). *Picking the Right NoSQL Database Tool*. http://www.monitis.com/blog/2011/05/22/picking-the-right-nosql-database-tool/.

[Gharsellaoui, 2012] Gharsellaoui H., Khalgui M., and Ben Ahmed S.,. (2012). *New Optimal Solutions for Real-Time Reconfigurable Periodic Asynchronous Operating System Tasks with Minimizations of Response Time*. IJSDA 1.4 (2012): 88-131. Web. 2 Feb. 2016. doi:10.4018/ijsda.2012100105.

[Gharsellaoui, 2013] Gharsellaoui H., Khalgui M., and Ben Ahmed S.,. (2013). *An EDF-based Scheduling Algorithm for Real-time Reconfigurable Sporadic Tasks*. Proceedings of the 8th International Joint Conference on Software Technologies, ”ICSOFT” 2013: 377-388, Reykjavik, Iceland.