Architectural Recovering Model for Distributed Databases: A Reliability, Availability and Serviceability Approach

Ramon Hugo de Souza, Paulo Arion Flores,
Mário Antônio Ribeiro Dantas and Frank Siqueira
Federal University of Santa Catarina (UFSC)
Distributed Computing Research Laboratory (LaPeSD)
Email: ramonh@inf.ufsc.br, pauloarion@gmail.com,
mario.dantas@ufsc.br and frank.siqueira@ufsc.br

***Abstract*—The early theoretical studies that presented conceptual definitions for the mapping of Quality of Experience (QoE) through Quality of Service (QoS), in an approach focused on Distributed Databases (DDBs), showed the directions to be threshed in a way to accomplish a hybrid DDBs' QoE-QoS set of evaluation criteria. That evaluation set being classified explicitly as partial, since QoE is a known multidimensional concept, hard to be explicitly well-defined. With these newfound equations being useful specifically to system-level purposes, without focusing on the user explicitly, as the usual QoE approach, these studies presented a new abstraction that allows web storage cloud services to adequately explore and assess requirements for contracted services. This study presents a novel architectural model to deal with reliability, availability and serviceability issues, in order to provide a solution giving QoS-like guarantees to DDB systems. With its origins in this set of evaluation criteria, this new kind of architectural solution aims to provide more stability by dealing with statistical guarantees. The proposed architecture focuses on these evaluations to choose recovery procedures, in order to avoid future unexpected behaviour, dealing with horizontal and vertical DDB's solutions to keep the services according to a Service Level Agreement (SLA).**

***Index Terms*—Availability, big data, cloud services, distributed database architectures, quality of experience, quality of service, reliability, serviceability.**

I. Introduction

This study aims to show the use of the evaluation equations of a theoretical mapping model of *Quality of Experience* (*QoE*) through *Quality of Service* (*QoS*) [1], focused on *Distributed Databases* (*DDBs*), within a replication architecture model that deals with horizontal and vertical distributed solutions focused on choosing the most fit recovery procedures to be followed.

As presented in the above mentioned work about the theoretical model, "the absence of *quality* definitions to *distributed databases*, allied with the *network-related concepts* and its holistic nature" [1] led to the formulation of this new evaluation model in order to fill the conceptual gap in the *DDB* literature, as presented in the bibliometric study of the paper.

This theoretical model was a first attempt in order to ground the *QoE* definitions in the *DDB* field with tangible metrics. It assumes that "*QoE* is a concept that it is not only limited to the use of a system or service" [2], but considers that "the evaluation can be partially defined as such to provide to a system, or service, the ability to auto-evaluate its completeness" [1], presenting the definition 1.

Definition 1: The multidimensional concept of *QoE* can be partially evaluated as *QoS completeness*, if the *QoS* is statistically well-defined, for system-level purposes.

And with base on these concepts and formulations, presented in section II, and further analysed in section III, this study follows the directions to achieve a solution by defining a new architectural model, focused on reliability and availability, with effects also over serviceability, using this first theoretical model, with the architecture as presented in section IV. A consideration for the *resources and quality* equations is also presented to consider the *formulas' framework* as a whole.

The main behavioural procedures of the proposed architecture are also presented in the algorithmic form in section IV, and detailed in section V, to illustrate the path yet to be threshed in the future implementations.

Also the behaviour of an architectural implementation for the model, using a multi-ring structure, with some specific considerations, is presented in section IV to illustrate the possibilities of dealing with a *two-level* service provider architecture.

II. Theoretical Map-Model of QoE through QoS

The work [1] presented a first theoretical model, based on the technical literature about distributed networks in clouds and management and evaluation of *QoS* in *DDBs* [3, 4, 5], that allows *QoS-based QoE* evaluations to be applied to *DDBs*. With grounds on the definition of the concepts of "*R&Q (resources and quality) capacity*: with execution, storage, networking and workloads constraints; and *RAS*[1] *(reliability/availability/serviceability) capacity*: with reliability, availability and serviceability or maintainability constraints -

[1]RAS: A term originally used by IBM to describe the robustness of their mainframe computers [6].

978-1-5090-0679-3/16/$31.00 ©2016 IEEE

including the fault tolerance constraints" [1] as main families of *QoS guarantees* to deal with cloud services, and potentially with *DDBs*, the work presents family-based evaluations.

Tables I and II present a general description of the metrics of each family of the model presented in [1].

Execution	Server capacity metric → CPU: number → CPU Frequency: GHz → RAM size: GBs → Storage size: GBs → Replication: number of replicas Instance mean starting time metric → Time up - time requested: avg in minutes Instance standard deviation starting time metric → Standard deviation of (Time up - time requested) Response mean time metric (synchronous operation) → Mean time of response: avg in milliseconds Response standard deviation time metric (synchronous operation) → Standard deviation time of response: avg in milliseconds Completion mean time metric (asynchronous task) → Mean time of response: avg in seconds Completion standard deviation time metric (asynchronous task) → Standard deviation time of response: avg in milliseconds
Storage	Storage device capacity metric → Storage device capacity: GBs Storage horizontal scalability metric → Permissible storage changes to increased workloads: GBs Storage replication metric → Number of replicas: Minimum number guaranteed
Networking	Network capacity metric → Bandwidth, throughput: MBs/s → Delay, jitter: milliseconds Application capacity metric → Application capacity: requests/min
Workloads	Server horizontal scalability metric → Permissible changes to increased workloads: number of virtual servers in resource pool Server vertical scalability metric → Permissible capacity fluctuoations to workloads: fluctuation number of CPUs and RAM size in GBs

TABLE I: QoS guarantees of R&Q capacity.

In [1] is also presented the classification of the *DDB's* services to be evaluated as: insertion, update, deletion and selection. And with the QoS parameters well-defined, as presented in tables I and II, the defined services can be then evaluated.

A. QoS Guarantees and QoE Measurement Metrics

Table III lists the *RAS guarantees* used in the evaluation formulas for the insertion, update, deletion and selection services, as first presented in [1]. Note that since the *serviceability*

Reliability	Success rate metric → Successful service outcomes: number of successful responses in percentage Freshness rate metric → Access update ratio (AUR): $\frac{AccessFrequency}{UpdateFrequency}$
Availability	Uptime rate metric → Percentage of service uptime: $\frac{TotalUptime}{TotalTime}$
Serviceability	Outage duration metric → Duration of a single outage: outage end time - outage begin time Mean-time between failures metric → Time between consecutive service failures: $\frac{NormalOperationalPeriodDuration}{NumberOfFailures}$ Mean-time to switchover metric → Time to switchover from a failure: minutes Mean-time system recovery metric → Time to a complete recovery from a service failure: minutes

TABLE II: QoS guarantees of RAS capacity.

Reliability	Success rate metric → Successful service outcomes: number of successful responses in percentage Freshness rate metric → Access update ratio (AUR): $\frac{AccessFrequency}{UpdateFrequency}$
Availability	Uptime rate metric → Percentage of service uptime: $\frac{TotalUptime}{TotalTime}$

TABLE III: QoS RAS guarantees for the insertion, update, deletion and selection capacities evaluation formulas.

capacity deals with related measurable variables that backup the guarantees for the *reliability* and *availability capacities*, it does not need to be directly explicit in the evaluation formulas for the system's behavioural analysis.

The two *RAS* equations, namely *system* (1) and *user* (2) equations, deal with the *availability capacity* of *uptime rate* in the *system* case, evaluating the insertion, update, deletion and selection *availability* as a whole, and the *reliability capacity* of *success rate* in the *user* case, evaluating the *reliability* of services strictly from the user's point of view.

The selection equations also evaluates the *reliability capacity* of *freshness rate* together with the *uptime rate* and the *success rate* for the system and user cases, respectively, to consider the freshness of the selected data.

$$QoE_{RAS}^{system}(ins|upd|del) = \frac{UptimeRate}{SLA(Ins|Upd|Del.RASGuarantee)} \quad [0;1]$$
$$QoE_{RAS}^{system}(sel) = \frac{UptimeRate * FreshnessRate}{SLA(Sel.RASGuarantee)} \quad [0;1] \quad (1)$$

$$QoE_{RAS}^{user}(ins|upd|del) = \frac{SuccessRate}{SLA(Ins|Upd|Del.RASGuarantee)} \quad [0;1]$$
$$QoE_{RAS}^{user}(sel) = \frac{SuccessRate * FreshnessRate}{SLA(Sel.RASGuarantee)} \quad [0;1] \quad (2)$$

These *Service Level Agreement* (*SLA*) related variables, presented on equations 1 and 2, are directly correlated values, and the equations are the completeness of the expected *uptime* and *success rates*, respectively, with the *freshness rate* in the selection cases.

The *RAS* variables are about the services' *reliability*, *availability* and *serviceability* completeness, as the name of the equation's family suggests. Table IV lists the *R&Q guarantees* used in the evaluation formulas for the insertion, update, deletion and selection services, as first presented in [1]. Notice that these are time evaluations, being the *storage*, *networking* and *workload capacities*, capacities with measurable variables that give backup guarantees for the *execution capacity*, which do not need to be directly explicit in the evaluation formulas for the evaluation of the system's behaviour. Also the *completion metrics* are not used in the evaluations, since only synchronous operations are considered.

Execution	Instance mean starting time metric → Time up - time requested: avg in minutes Instance standard deviation starting time metric → Standard deviation of (Time up - time requested) Response mean time metric (synchronous operation) → Mean time of response: avg in milliseconds Response standard deviation time metric (synchronous operation) → Standard deviation time of response: avg in milliseconds

TABLE IV: QoS R&Q guarantees for the insertion, update, deletion and selection capacities evaluation formulas.

The two *R&Q* equations, namely *system* (3) and *user* (4) equations, deal with the *execution capacities* of *starting mean time* + n_1* *standard deviation* and *response mean time* + n_2* *standard deviation*, with n_1 and n_2 usually between 1 and 2, in the *system* case, evaluating the *execution* mean response time as a whole, and the *execution capacities* of *instance starting and reponse times* in the *user* case, evaluating the *response time* in the strict point of view of the user for every transaction individually.

$$QoE_{R\&Q}^{system}(ins|upd|del|sel) = \frac{SLA(Ins|Upd|Del|Sel.R\&QGuarantee)}{(InstanceMeanStartTime + n_1 * InstanceStdDevStartTime) + (ResponseMeanTime + n_2 * ResponseStdDevTime)} \quad [0;1] \quad (3)$$

$$QoE_{R\&Q}^{user}(ins|upd|del|sel) = \frac{SLA(Ins|Upd|Del|Sel.R\&QGuarantee)}{InstanceStartingTime + ResponseTime} \quad [0;1] \quad (4)$$

These *SLA* related variables, presented on equations 3 and 4, are directly correlated values and the equations are the completeness of the expected *system mean response time* and *user transaction response time*, respectively.

The *R&Q* variables are about the services' response times as agreed, dealing with the *resources available* and its *quality of response*, as the name of the family suggests.

As presented on [1], the system could also be evaluated as a whole combining the *RAS* and *R&Q* completeness together, as shown on equations 5 and 6.

$$QoE_{total}^{system}(ins|upd|del|sel) = QoE_{RAS}^{system}(ins|upd|del|sel) * QoE_{R\&Q}^{system}(ins|upd|del|sel) \quad [0;1] \quad (5)$$

$$QoE_{total}^{user}(ins|upd|del|sel) = QoE_{RAS}^{user}(ins|upd|del|sel) * QoE_{R\&Q}^{user}(ins|upd|del|sel) \quad [0;1] \quad (6)$$

III. Metrics Analysis

The focus of this work is into equations 1 and 2, and how to define an architectural model considering the *RAS* parameters, being also presented the analysis of equations 3 and 4, to show the impact of the whole model in the prediction of systems' behaviour, and how they could be used in order to avoid future unexpected behaviour with *on demand* data analysis.

For equation 1, $QoE_{RAS}^{system}(ins|upd|del|sel)$, let's consider $UptimeRate = rate$ for the insertion, update and deletion cases, and $UptimeRate$ * $FreshnessRate = rate$ for the selection case. This $rate$ should be an individual value for each operation type, being here considered as one individual value for the sake of simplicity.

The $SLA(Ins|Upd|Del|Sel.\ RASGuarantee)$ is the completion guarantee over the defined $rate$ value, but since it is a guarantee that considers an expected mean with a standard deviation, from this point on we will consider this kind of equation range guarantees splitted into equations like: $SLA_{RASGuarantee}^{mean}$[2], $SLA_{RASGuarantee}^{min}$ and $SLA_{RASGuarantee}^{max}$.

As an example let's consider a generic guarantee of:

$$\frac{rate}{SLA_{RASGuarantee}^{min}} \geq 1,$$

with the $SLA_{RASGuarantee}^{min}$ being the expected mean minus one times the standard deviation. And considering that example of *SLA agreement* with the guarantee of 68.2% of the values being into the range of the mean $\pm$ standard deviation in a normal distribution, we have 84.13% of the values above this considered minimum. Note that in this generic exemplification, with all the operations together, the effect over the selection guarantee is stronger in the $UptimeRate$, since it is multiplied by the $FreshnessRate$, then needing an upper $UptimeRate$ value when compared to the equation applicable to insertion, update and deletion.

And then we have that simply:

$$rate \geq SLA_{RASGuarantee}^{min}.$$

Being the measures of QoE_{RAS}^{system}[3] between 1 and 0.8413 acceptable for the one standard deviation consideration, but

[2]When not expliciting operations into an equation we, from this point on, will consider them dealing with the four operations.

[3]This is a complex kind of guarantee, not being a static value, but a mean and standard deviation combination guarantee.

also measures that should trigger at least soft recovery procedures to avoid future unexpected behaviour. While the *rate equation* results in a value above the given $SLA^{min}_{RASGuarantee}$ the guarantees are being kept.

Considering the $UptimeRate$ parameter, a first solution to avoid the value of the *rate equation* getting close to the given $SLA^{min}_{RASGuarantee}$, or less, is to deal with horizontal scalability, and then to start new database replicas. This decision is not straightforward, because the availability metric is not able to identify if a service is on, but only if the service can be reached. It is also a complex evaluation point, since the service could deal with a limited requisitions queue. A better solution could be to deal with vertical scalability to increase the queue size or to increase the processing capacity and accelerate the dequeuing process, since these solutions will have less impact on the $FreshnessRate$ parameter, that affects the selection QoE^{system}_{RAS} evaluation. But the $FreshnessRate$ parameter aside, the horizontal scalability is the natural choice to solve this kind of problem in a long term.

Another issue that must be taken into account, when choosing between horizontal and vertical strategies, is the time to replicate a database with the horizontal strategy, or the availability of resources when considering the vertical strategy. And with these issues taken into account, a solution that appears to fit is the possibility of dealing with a limited number of non-prioritary replicas in a special pool of dormant replicas, that can be *awakened* to a prioritary level, migrating to the pool of serving replicas.

The considered trigger to start a soft maintenance process, either horizontal or vertical, could be the $rate$ to reach the decreasing value of $SLA^{mean}_{RASGuarantee}$, since the system is expected to work with that mean value. A prioritary maintenance procedure is needed when the system achieves $SLA^{min}_{RASGuarantee}$.

In the same way, *energy saving procedures* could be triggered when the system achieves $SLA^{max}_{RASGuarantee}$, sending replicas to a non-prioritary pool, or decreasing the resources vertically.

Since horizontal replication is a more long term solution, it should be prioritarily used when it is verified that a system is not well designed. The vertical solution could be used for a faster response, and to solve punctual problems as verified by QoE^{user}_{RAS}, that deals with the $SuccessRate$. Then the vertical solution could be used in a way to verify problems of *uptime* related to *success* first, as a momentary fluctuation, and if the fluctuation persists the system should then trigger horizontal replication procedures.

For example, let's assume a *DDB* with s servers, each one with an abstract capacity denoted by c. In a moment the QoE^{system}_{RAS} reaches the value $SLA^{min}_{RASGuarantee}$ the s servers should be vertically scaled to a capacity $c = c + n$.

Existing an expected mean value for the capacity, the capacity level could be analysed from time to time to decide if the number of servers should grow. If so, the value of the capacity in the active servers should return to the expected c value when a new server replica is awakened from a pool of dormant replicas.

When the cause of the unexpected measure is the $Freshness\ Rate$, the *access update ratio* can also be increased to correct the expected $QoE^{system}_{RAS}(sel)$, as an alternative solution.

For equation 2, the QoE^{user}_{RAS}, we have an evaluation that relates to solutions of vertical approach, as mentioned before, focusing on solving problems related to *success rate* for momentary problems.

Considering this kind of approach, if the resources stabilize in an unexpected value, let's say above the expected capacity level, the system should consider to trigger a horizontal scalability procedure and wake up a replica from a dormant pool. If the opposite occurs, and the resources stabilize below the expected capacity level, the system may consider to trigger vertical scalability's energy saving procedures, such as reducing the processing power of the active nodes. Or, in a capacity based overview, to reduce the capacity.

These punctual evaluations may also indicate other availability problems, that may only be solved by starting instantly the horizontal procedures, adding new service providers to the pool of services.

It is interesting to notice that both the horizontal and the vertical strategies could be started to solve problems with one specific service, but the scalability procedures will usually affect all the services provided by a *DDB* system.

For equations 3 and 4, namely $QoE^{system}_{R\&Q}$ and $QoE^{user}_{R\&Q}$, we have *time evaluations*, as mentioned before. And for *time evaluations* the natural solution seems to relate with the vertical procedures focused on the processing capacity.

The *R&Q* system evaluation deals with the values considering the standard deviations, as agreed, and the *R&Q* user evaluation deals with the momentary measures, transaction by transaction. The first one is focused on detecting behavioural problems that could lead the system to future unexpected states, while the second is focused on detecting punctual problems, *on demand*, that need instant solutions.

When considering the $QoE^{system}_{R\&Q}$ we have the *SLA* guarantee time against the time to start an instance plus the agreed standard deviation for such, that is mostly the time to reach a server, which includes the time spent in a queue, with the response time plus the agreed standard deviation for it, that is the time of processing the request. And with that given evaluation it is possible to infer for the need of (1) more processing power, or (2) more service providers in the pool, when the starting time is not responding as specified, (3) more processing power when the response time is not according to the expected mean, but stable, or (4) more service providers in the pool to solve a high standard deviation state.

The $QoE^{user}_{R\&Q}$ evaluates the momentary expected time against the instance starting time plus the response time for a transaction specifically. This is the total amount of time to receive a response to a given request.

With the $QoE^{user}_{R\&Q}$ evaluation, punctual momentary response time problems can be identified. For example, problems

with a specific server could be identified, and the problematic service provider could be restarted or replaced by one of the replicas provided by the pool of dormant replicas.

If the waiting time reaches a limit, the client can signalize a possible problem with the responding service provider. If the signalling persists, the system, that could be specified as a Database Management Systems (DBMS), should inquire the provider to check for problems.

IV. RAS Architectural Model

The presented model considers two categories of pool with service providers: The *service provider's pool*, with the services available and ready to be used, being provided by the main servers; and the *backup pool*, a pool with dormant replicas, that are awakened from time to time, when the replicas in the *service provider's pool* achieve *checkpoints*, with messages from the *borderline nodes*, that are service providers responsible for sending the *checkpoint* data to the servers in the *backup pool*.

This is a solution to avoid the time-consuming operations to create new database replicas from the start when dealing with horizontal scalability procedures. With the replicas of the *backup pool* already in a state close to the state of the online replicas, it is quicker to have a new working replica in the same state as the functional replicas on the *service provider's pool* when needed.

Figure 1 illustrates a simple example with a multi-ring message exchanging implementation, inspired in the example described in [7], dealing with two full rings in the *service provider's pool*. The rings have two common nodes for a simple fault tolerance, in a way that if node 2 goes down, edges *a* and *b* become active, and if node 3 goes down, edges *c* and *d* become active.

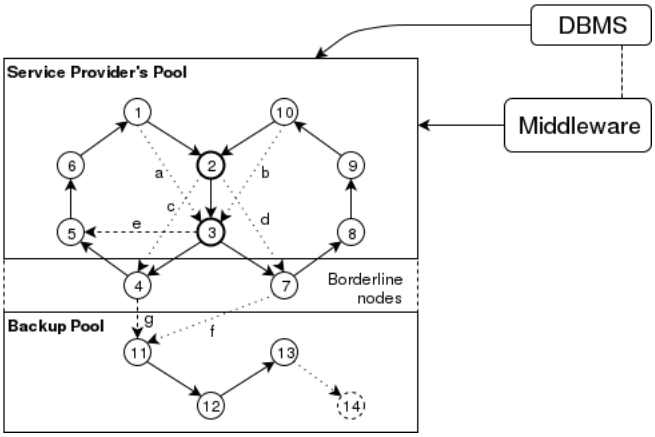

Fig. 1: RAS architecture model with multi-ring message exchanging implementation and simple fault tolerance example.

Nodes 4 and 7 are called *borderline nodes*, because they are in the *service provider's pool* and partially in the *backup pool*. The "partially" is because the edge *g*, or the edge *f* if node 4 goes down, wake up the nodes in the *backup pool* every time the system reaches a *checkpoint* and refreshes the data, using the partially connected ring in the *backup pool*, by calling a *checkpoint* procedure. Being the *backup pool* an approach to, at a certain degree, deal with the consistency, availability and partition tolerance (CAP) theorem [8] with no impact over the *service provider's pool*.

When a new node is required in the *service provider's pool*, let's say node 11, a new node replica should be instantiated in the *backup pool*, as node 14, and added to the ring of backup replicas. This new node should be replicated based on the nodes in the *backup pool*, to avoid impact over the system in the *service provider pool's* side.

Taking into account that the whole architecture is hosted in a cloud-based environment, the structure can be vertically scaled very easily due to the elastic nature of the host platform.

The multi-ring message exchange is an interesting approach to avoid excess of communication, but it is shown here just as a possible implementation for the architecture as defined.

In this model, as presented in figure 1, a middleware attached to the DBMS analyses the behaviour of the system as a whole, and controls the decisions for performing the horizontal and vertical maintenance procedures.

Notice that to scale horizontally will result in complexity costs in order to maintain data consistency, due to the fact that latency increases when the horizontal scalability procedures tries to keep the consistency [9]. To avoid these latency costs, the creation of new replicas is done with base only on the replicas on the *backup pool* side.

Without considering the fault tolerance illustration and the exemplification by the multi-ring architecture, that are not the focus of this work, based on the analysis made on section III and on the architecture presented in this section, the main procedures of the expected behaviour in the developed model are presented in algorithm 1.

The procedures needed to build this model consist of a *RAS* minimum and maximum vertical and horizontal guarantee verification procedures and also the checkpoint signaling procedure to the *borderline nodes* to deal with the dormant replicas in the *backup pool*.

V. Algorithm procedures

It is possible to observe that in algorithm 1 are provided procedures in accordance with the given *QoE-QoS* equations in order to build a solution managing the *database architecture system's* resources according to the *rate* variation.

Three procedures are presented in algorithm 1:

- The Main procedure, that constantly checks the *rate*, calling the $RAS_{min}(time)$ procedure when the $SLA^{min}_{RASGuarantee}$ is achieved, being the $RAS_{min}(time)$ a procedure that start mechanisms to provide more resources, or calling the $RAS_{max}(time)$ procedure when the $SLA^{max}_{RASGuarantee}$ is achieved, in order to release unnecessary resources;
- the $RAS_{min}(time)$ procedure, that iterates while the value of the QoE^{system}_{RAS} is kept under the expected value of $SLA^{mean}_{RASGuarantee}$, growing the value of the capacity c by a defined value of n, respecting the

Algorithm 1 Reliability-Availability-Serviceability Procedures

```
1: procedure Main(time)
2:     while true do
3:         if rate ≤ SLA^min_RASGuarantee then
4:             call RAS_min(time)
5:         else if rate ≥ SLA^max_RASGuarantee then
6:             call RAS_max(time)
7:         else if rate = SLA^mean_RASGuarantee then
8:             c ← c_original
9:         end if
10:        wait(time)
11:    end while
12: end procedure
13: procedure RAS_min(time)
14:    while rate < SLA^mean_RASGuarantee do
15:        if c = c_max then
16:            if #replicas = #replicas_max then
17:                return
18:            end if
19:            Wake up a replica from the backup pool
20:            Add the replica to the service provider's pool
21:            c ← c_original
22:            Start the backup pool new replica procedure
23:        else
24:            c ← c + n
25:        end if
26:        wait(time)
27:    end while
28: end procedure
29: procedure RAS_max(time)
30:    while rate > SLA^mean_RASGuarantee do
31:        if c = c_original then
32:            if #replicas = #replicas_min then
33:                return
34:            else
35:                Remove one replica from the service provider's
   pool
36:            end if
37:        else
38:            c ← c − n
39:        end if
40:        wait(time)
41:    end while
42: end procedure
```

measuring interval of $time$, keeping the value of the *extended capacity* until the mean achieves the expected $SLA^{mean}_{RASGuarantee}$ again, or, if the capacity level reaches its maximum, awakening a new replica from the *dormant pool* and reseting c. Note that this procedure solution is a conservative solution, growing the capacity c until the $SLA^{mean}_{RASGuarantee}$, not $SLA^{min}_{RASGuarantee}$, is reached;

- and the $RAS_{max}(time)$ procedure, that runs in the opposite way of the $RAS_{min}(time)$ procedure, dealing with a loop that first checks if the mimimum accepted value for the resources capacity has been reached, and if not it decreases it by the defined value of n, or if so it removes a replica from the service provider's pool as an energy saving procedure.

VI. Conclusions and future work

This work indicates that is possible to create complex formulations, in order to deal with the evaluations as presented in [1], and by that being able to predict, and to avoid, system's misbehaviour before reaching unexpected states with a reliability, availability, and serviceability architectural model.

The proposed solutions presented are based on horizontal and vertical scalabilities applied to *DDB systems*, and this could be implemented directly into a *DBMS* to avoid these unexpected states. Being these concepts not new *per se*, but suffice to fill this lack of architecture conceptualization over the database area.

A simple algorithm, with some procedures considering the generic expected behaviour, was presented to illustrate how to further develop this study into a new implementation.

The next expected effort is to build an implementation of these concepts into a *DBMS*, or to have a module that allows a *DBMS* to deal with decisions based on this kind of evaluation.

References

[1] R. H. de Souza and M. A. R. Dantas, "Mapping QoE through QoS in an approach to DDB architectures: Research analysis and conceptualization," *ACM Computing Surveys (CSUR)*, vol. 48, November 2015.

[2] P. L. Callet, S. Möller, and A. Perkis, "Qualinet white paper on definitions of quality of experience," in *European Network on Quality of Experience in Multimedia Systems and Services (COST Action IC 1003)* (Lausanne, ed.), (Switzerland), March 2013. Version 1.2.

[3] T. Erl, R. Puttini, and Z. Mahmood, *Cloud Computing: Concepts, Technology & Architecture*. The Prentice Hall Service Technology Series from Thomas Erl, Prentice Hall, 1 ed., 2013.

[4] R. L. de Carvalho Costa and P. Furtado, "Quality of experience in distributed databases," in *Distributed and Parallel Databases* (S. Netherlands, ed.), vol. 29, pp. 361–396, 5-6 ed., October 2011.

[5] R. L. de Carvalho Costa and P. Furtado, "Providing quality of experience for users: The next dbms challenge," in *Computer* (I. C. Soc, ed.), vol. 46, pp. 86–93, 9 ed., September 2013.

[6] D. P. Siewiorek and R. S. Swarz, *Reliable computer systems: design and evaluation*. Natick, Mass. : A K Peters, 3th ed., 1998. p. 508.

[7] P. J. Marandi, M. Primi, and F. Pedone, "Multi-ring paxos," in *Conference on Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International*, (Boston, MA), IEEE, June 2012.

[8] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," in *ACM SIGACT*, vol. 33, pp. 51–59, ACM New York, NY, USA, June 2002.

[9] D. J. Abadi, "Consistency tradeoffs in modern distributed database system design," in *Computer*, vol. 45, pp. 37–42, IEEE, February 2012.