Publication avant LAFMC

Titre	Database replication in large scale systems: optimizing the number of replicas
Auteurs	Modou Gueye, Idrissa Sarr, **Samba Ndiaye**
Référence	EDBT/ICDT '09 Proceedings of the 2009 EDBT/ICDT Workshops
Editeur	ACM
Pages	3-9
Année	2009
DOI	10.1145/1698790.1698794
URL	https://dl.acm.org/citation.cfm?id=1698794
Index	
ISBN	978-1-60558-650-2
Encadreur	Oui
Extrait d'une thèse	Non

Source details

ACM International Conference Proceeding Series

Scopus coverage years: 1999, from 2002 to 2016

Publisher: ACM

Subject area: Computer Science: Computer Vision and Pattern Recognition | Computer Science: Computer Networks and Communications | Computer Science: Software | View all

View all documents >

CiteScoreTracker 2018

Last updated on *11 February, 2019*
Updated monthly

0.70 = Citation Count 2018 / Documents 2015 - 2017 = 2 749 Citations to date > / 3 908 Documents to date >

ACM DIGITAL LIBRARY

SIGN IN SIGN UP

SEARCH

Check out a preview of the *next* ACM DL

Database replication in large scale systems: optimizing the number of replicas

Full Text: PDF Get this Article

Authors: Modou Gueye UCAD-FST, Dakar, Senegal
Idrissa Sarr UPMC Paris Universitas, LIP6 Lab, France
Samba Ndiaye UCAD-FST, Dakar, Senegal

2009 Article

Published in:
· Proceeding
EDBT/ICDT '09 Proceedings of the 2009 EDBT/ICDT Workshops
Pages 3-9

Saint-Petersburg, Russia — March 22 - 22, 2009
ACM New York, NY, USA ©2009
table of contents ISBN: 978-1-60558-650-2 doi>10.1145/1698790.1698794

In distributed systems, replication is used for ensuring availability and increasing performances. However, the heavy workload of distributed systems such as web2.0 applications or Global Distribution Systems, limits the benefit of replication if its degree (i.e., the number of replicas) is not controlled. Since every replica must perform all updates eventually, there is a point beyond which adding more replicas does not increase the throughput, because every replica is saturated by applying updates. Moreover, if the replication degree exceeds the optimal threshold, the useless replica would generate an overhead due to extra communication messages. In this paper, we propose a suitable replication management solution in order to reduce useless replicas. To this end, we define two mathematical models which approximate the appropriate number of replicas to achieve a given level of performance. Moreover, we demonstrate the feasibility of our replication management model through simulation. The results expose the effectiveness of our models and their accuracy.

Database Replication in Large Scale Systems: Optimizing the Number of Replicas

Modou Gueye
UCAD-FST
Dakar, SENEGAL
gmodou@ucad.sn

Idrissa Sarr
UPMC Paris Universitas
LIP6 Lab, FRANCE
idrissa.sarr@lip6.fr

Samba Ndiaye
UCAD-FST
Dakar, SENEGAL
ndiayesa@ucad.sn

ABSTRACT

In distributed systems, replication is used for ensuring availability and increasing performances. However, the heavy workload of distributed systems such as web2.0 applications or Global Distribution Systems, limits the benefit of replication if its degree (i.e., the number of replicas) is not controlled. Since every replica must perform all updates eventually, there is a point beyond which adding more replicas does not increase the throughput, because every replica is saturated by applying updates. Moreover, if the replication degree exceeds the optimal threshold, the useless replica would generate an overhead due to extra communication messages. In this paper, we propose a suitable replication management solution in order to reduce useless replicas. To this end, we define two mathematical models which approximate the appropriate number of replicas to achieve a given level of performance. Moreover, we demonstrate the feasibility of our replication management model through simulation. The results expose the effectiveness of our models and their accuracy.

1. INTRODUCTION

New applications such as Web2.0 applications and Global Distribution Systems manage huge amount of data and deal with heavy workloads. The challenge for these applications is to ensure data availability and consistency in order to deal with fast updates.

One solution to face this problem is to use replication. Although replication is used to ensure either read performance and write performance, improving both read and write performance simultaneously is a more challenging task [4]. To tackle only read performance, master-slave replication is widely used. With this approach, read-only queries are performed on the slave nodes and update queries are sent to the master node. Conversely, to face read and write performance, multi-master replication allows each replicas to store a full copy of the database, thus read or write operations can be handled anywhere. Furthermore, some synchronisation is needed to meet the mutual consistency requirement. To limit the synchronisation, which can lead to aborts and thus system scalability slowdown, some solution use lazy multi-master replication [16, 6] or delegate the consistency management to the middleware layer [13, 18, 4]. The heavy workload of Web2.0 applications or Global Distribution Systems, limits the benefit of replication if its degree (i.e., the number of replicas) is not controlled. Since every replica must perform all updates eventually, there is a point beyond which adding more replicas does not increase the throughput, because every replica is saturated by applying updates. Moreover, if the replication degree exceeds the optimal threshold, the useless replica would generate an overhead due to extra communication messages.

Many solutions have been proposed in the field of database replication, such as [13, 11, 12]. Some solutions include freshness control, for instance [16, 6, 9, 1, 15, 6, 5]. Some other, focus on data availability or fault-tolerant service, such as [2, 7, 8, 19]. We base our work on the DTR approach [18], since it offers update anywhere and freshness control features, and is designed for Global Distribution Systems.

DTR proposed a solution which controls the freshness of replicas in order to improve the performance of concurrent updates. Furthermore, DTR availability has been enhanced in [17] by using a middleware-based replication. However none of these previous works attempt to compute which replication threshold will reduce the overhead involved by the management of replicas. Indeed, the formal model described in [5] for controlling replication freshness, presents good performances in terms of response time and network traffic. Unfortunately, eventual replicas faults are not taken into account, and reducing communication messages can be improved by limiting the number of replicas. The goal of this article is to limit the overhead involved by managing useless replicas and to bring the following contributions:

• A replication management solution, based on the characteristics of the system. In order to ensure data availability, we propose a model that estimates the degree of replication with respect to the resources or the volatility of the system. We propose two ways to define the appropriate number of replicas: (i) one based on the required system availability and the frequency of nodes failures and (ii) another which takes into account the tolerated staleness of queries and node capabilities in terms of throughput.

DAMAP 2009, March 22, 2009, Saint Petersburg, Russia.

• An evaluation of our approach on a large scale simulator. It demonstrates the feasibility of our approach and measures the accuracy of our results.

The rest of this paper is organized as follows. We first present in Section 2 the system architecture together with the replication and freshness model. Section 3 describes our replication solution based on a probabilistic approach. Section 4 presents an analytical model to define the replication degree. Section 5 presents the performances evaluation of our replication approach through simulation. Section 6 concludes.

2. SYSTEM AND REPLICATION MODEL

In this section we introduce our system and the replication model upon which our approach relies.

2.1 Replication Model

We assume a single database with n relations $R^1, ..., R^n$ that is fully replicated at m nodes $N_1, ..., N_m$. We use a lazy multi-master (or update everywhere) replication scheme. Each node can execute any incoming transaction and is called the initial node of the transaction. Other nodes are later refreshed by propagating the transaction through refresh transactions. We distinguish between three kinds of transactions:

• An update transaction is a sequence of SQL statements : at least one of them updates the database.

• A refresh transaction is used to propagate update transactions to the other nodes for refreshment.

• A query is a read-only transaction. Thus, it does not need to be refreshed.

Let us note that, because we assume a fully replicated database (*i.e.* we do not consider partial replication), we do not need to deal with distributed transactions, *i.e.* each transaction can be entirely executed at a single node.

2.2 System Model and Definitions

We base our solution in DTR2 architecture system [17]. In DTR2, we distinguish three kinds of nodes: *Client Nodes* (*CN*), *Transaction Manager Nodes* (*TM*), *Data nodes* (*DN*) and *Shared Directory nodes* (*SDN*). CNs send transactions to TMs which route them for execution to DNs by using the SDNs which store the metadata. Queries may access stale data, provided it is controlled by applications. In other words, applications can associate a *tolerated staleness* with queries.

***Definition** 1.* Staleness can be defined through various measures [9]. In this paper, we only consider one measure, defined as the number of missing updates. The average staleness tolerated by a query is denoted by $\bar{S}$.

Furthermore, DNs use a local DBMS to store data and execute the transactions received from the TMs. Thus, the number of transactions performed by a DN during a while, have a significant impact on the overall performances.

***Definition** 2.* The theoretical throughput of a DN, denoted χ_D, is the average number of transactions that a DN can process per unit time.

For instance, if χ_D is equal to 15, assuming that K units time lasted, the number of transactions processed is equal to $15 * K$.

3. PROBABILISTIC REPLICATION THRESHOLD

In this section, we first define the probability that a node failure occurs with respect to our architecture. Then, we define the threshold beyond which performances do not increase.

3.1 Model and Frequency of Node Failure

3.1.1 Failure Model

We consider only two kinds of components: the nodes that process the transactions, and the communication links between the nodes. At runtime, each of these components may fail, leading to a node or communication failure. In this section, we assume fail-stop failures: a node is either working correctly or not working at all (it is down). We also assume that communication failure may occur but is far less frequent than node failure.

3.1.2 Frequency of Node Failure

The frequency of failures in distributed systems has been widely studied. Based on the work of [10], we assume that the failures follow a Poisson distribution, we can derive the next formula.

$$P_k = \frac{(\lambda t)^k}{k!} e^{-\lambda t} \tag{1}$$

P_k represents the probability of k failures during the time interval t where λ is the failure rate. Effectively, it must exist better way than Poisson distribution to model failures. For example distributions such as Weibull used in [3] for predicting a system availability gives a good precision. Nevertheless our mean goal is not to define the probability of failures or to predict them, but to define the number of needed replicas using this probability.

3.2 Probabilistic Threshold

Replication can tolerate failures, because if a node fails, we will use another replica to replace it. However the degree of replication must be monitored and adjusted with respect to the frequency of failures for minimizing the cost of managing the mutual consistency. Then, given a frequency of node failures, our goal is to estimate how much replicas are needed to ensure system availability. We define system availability as follows:

***Definition** 3.* A system is available as long as there is at least one DN to perform the read or write operations. This DN can be stale, thus it needs to be refreshed by processing a set of refresh transactions before applying incoming operations.

To reach our goal, we use the previous formula which gives the probability that k failures occur during the time interval t. We also assume that the failed nodes do not recover during the time t. Then, we define the replication threshold, denoted k, that is needed to ensure the availability of the system:

$$k > \lceil \sqrt{\frac{1}{4} + \frac{2\lambda t}{P_{tol}}} - \frac{1}{2} \rceil \quad (2)$$

P_{tol} is the tolerated number of nodes that can fail without compromising the system. So, given the probability of failures that the application tolerates and the failure rate, we assess the degree of replication required to guaranty that our system will remain available during a time interval t. For more details, see *Appendix A*.

Example. Given a system with a failure rate equals to 0.1 ($\lambda = 1$) and a time interval t equal to 100s. We obtain the average number of failures during the time interval t with $\lambda * t = 10$. Then, by using Formula 2, we get $k > \lceil 12.1534295 \rceil$. Thus, with 13 replicas, the system remains available during t despite the occurrence of 10 node failures.

4. ANALYTIC REPLICATION THRESHOLD

We propose now an analytical method for bounding the replication degree. We distinguish update transactions from queries. First, we present an estimation of the number of queries processed during a given time interval. Then, we describe our analytical model to estimate the replication threshold.

4.1 Number of processed queries

We assume that $\bar{S}$, χ_D, and the incoming workload (number of incoming transactions and queries) are known. We also assume that the number of queries is very high compared to the number of updates, and the network latency is fixed.

Let be n_1, the average number of incoming updates per time unit, c_1, their average execution cost, n_2, the average number of incoming queries per time unit, and c_2, their average execution cost. The following formula gives the number of queries ($Q_p(k)$) processed by m DNs during k time units.

$$Q_p(k) = \frac{k\chi_D + \bar{S} * c_1 + (\frac{1-m}{m}) * c_2}{\frac{c_1}{r} + \frac{c_2}{m}} \quad (3)$$

r, is equal to $\frac{n_2}{n_1}$ and the estimated number of queries computed $Q_p(k)$ is a function of m. For more details, see *Appendix B*. Formula 3 is very convenient since it allows us to compute the theoretical number of queries performed during a given time interval based on the transactions and system characteristics. The Section 5 presents measures which confirm this formula.

4.2 Analytical Replication Threshold

The knowledge of this threshold is very significant since it avoids useless replicas. We distinguish two ways to compute this threshold, depending on the workload size, namely either *RT4HW* for high workload or *RT4LW* for low workload To this end, we use the equation 3 with some transformations and mathematical concepts.

RT4HW. To find the minimal number of replicas required to face high workload, we derive the equation 3 and use the increasing flow of processed queries (α) when m becomes high. Then, we obtain the following formula to determine the *RT4HW*:

$$m \geq \sqrt{\frac{c_2 r^2 (k\chi_D + \bar{S}c_1 - c_2 - \frac{c_1}{c_2})}{\alpha c_1^2}} - \frac{c_2 r}{c_1} \quad (4)$$

With $\alpha > 0$. The smaller is α, the better will be the threshold. We notice that our formula works well if the workload is so high that the system cannot perform all incoming queries. Intuitively, any added replica will increase the number of processed queries, then α is always greater than zero.

RT4LW. We study the case where the workload is low, or the DN throughput is so high that the system can execute all queries during the k time units. Thus, adding new replicas will not increase the number of processed queries (*i.e.* $\alpha \approx 0$). We conclude easily that our previous formula fails since $m \to \infty$ when $\alpha \to 0$. To face this case, we define the following formula:

$$m \geq \frac{c_2(kn_2 - 1)}{k(\chi_D - c_1 n_1) + \bar{S}c_1 - c_2} \quad (5)$$

For low workload, this formula provides a better threshold as it is demonstrated by experimentation (see Figure 2). For intermediate workload, using either Formula 4 or 5 depends on p. If $p \geq \frac{c_2 - \bar{S}c_1}{k} + c_1 n_1$ then use Formula 5, else use Formula 4. See *Appendix C* for details about these two formulas.

Example. Let us consider the following values for the parameters: $c_1 = 2, c_2 = 5, n_1 = 18, k = 59$ and $\bar{S} = 100$. The value of the sum $\frac{c_2 - \bar{S}c_1}{k} + c_1 n_1$ is then 32,694. For a throughput χ_D lower than this value, the Formula 4 must be used. This is the case shown in the Figure 1. If we change the values of c_1 and c_2 respectively with 1 and 3, the value of the previous sum becomes equal to 16,355 and then, for a throughput equal to 30, the Formula 5 must be taken as we can see it with the Figure 2.

5. EXPERIMENTAL EVALUATION

In this, section we validate our approach through simulation by using Peersim [14]. Peersim is a P2P system simulator developed with Java. We have extended Peersim classes in order to implement our experiments. Our first goal is to check the accuracy of our formulas which estimate the replication threshold. To this end, we compare the analytical throughput (computed with the Formula 3) with the experimental throughput (measured after running the simulation). We ran a set of experiments, varying the number of replicas from 2 to 100.

To set up our simulation experiments, we choose $k = 60$, the number of updates $n_1 =$ 1000, the number of queries $n_1 =$2000, and the average of tolerated staleness equals 100. The cost of queries c_2 is always greater than c_1 and during all the experimentation we take $c_2 = 2.5 * c_1$. This specific setup was chosen, because we deal with Web2.0 applications which received a number of read operations more than write operations, and cost of processing updates is smaller than cost of processing queries.

The Figure 1 shows the threshold beyond which the number of processed queries does not increase any more. Moreover, it highlights that our analytical results provide a useful approximation. Indeed, the threshold obtained either analytically or by simulation is almost the same.

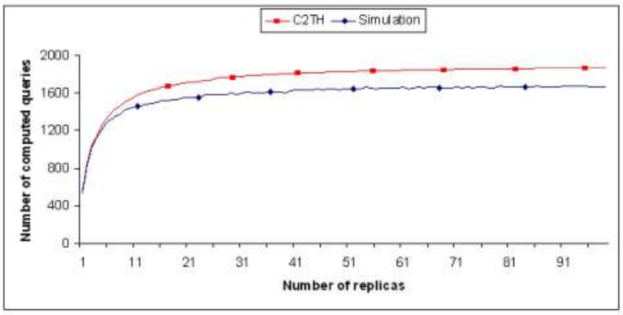

Figure 1: Analytical vs. Real throughput

Furthermore, we increase the DN throughput by a factor of 2 in order to measure the lack of precision by using Formula 4 when workload decreases. The results in Figure 2 show the large gap between the analytical and the measured threshold (more than 1500 queries) by using Formula 4, where Formula 5 produce a negligible gap. Ongoing works will try to validate our model with a real system in order to prove that it works well.

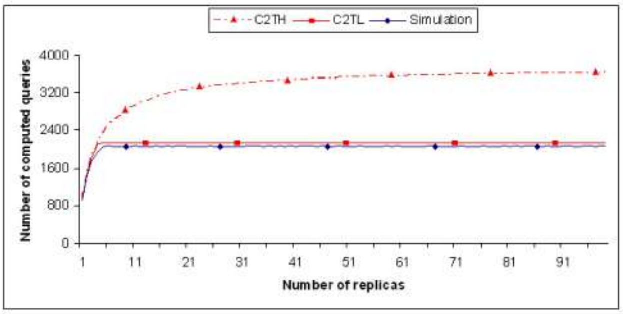

Figure 2: Analytical vs. Real throughput

6. CONCLUSION

This article deals with replication threshold in the context of distributed databases. Beyond this threshold, no performance benefit would be noticed. We propose a model to estimate this threshold by two theoretical methods, one is probabilistic and the other one is analytical. Our probabilistic method, based on the Poisson distribution, computes the replication threshold k with respect to the needed availability. This replication threshold depends on the arrival rate of replicas failures and the probability of system failure. The analytic method allows us to assess the replication threshold by taking into account the mains factors affecting transactions processing. Our experimental simulation demonstrates the feasibility of our approach and measures the precision of our estimates.

Appendix A. Let us be S a system, λ the average of the arrival rate of failures, P_{tol}, the node probability tolerated during t. In order to ensure data availability durind t, we need to find k, the number of replicas such that $P_k > P_{tol}$. Assume m be the number of replicas and :

$$\forall m < k, \Rightarrow P_k < P_{tol} \leq P_m$$

Setting m = k - 1 leads to:

$$P_1 + 2P_2 + ... + mP_m \geq P_{tol} + 2P_{tol} + ... + mP_{tol} \quad (6)$$

Then we can deduce:

$$\sum_{n=1}^{m} nP_n \geq \frac{m(m+1)}{2} P_{tol} \quad (7)$$

If we substitue P_n for its value in 7, we get:

$$e^{-\lambda t} \sum_{n=1}^{m} \frac{(\lambda t)^n}{(n-1)!} \geq \frac{m(m+1)}{2} P_{tol} \quad (8)$$

When we add positives terms to the right of the inequality 8, we have:

$$\lambda t.e^{-\lambda t} [\sum_{n=0}^{m-1} \frac{(\lambda t)^n}{(n)!} + \sum_{n=m}^{\infty} \frac{(\lambda t)^n}{(n)!}] \geq \frac{m(m+1)}{2} P_{tol} \quad (9)$$

Then we can get:

$$\lambda t.e^{-\lambda t} \sum_{n=0}^{\infty} \frac{(\lambda t)^n}{(n)!} \geq \frac{m(m+1)}{2} P_{tol} \quad (10)$$

By taking the limited development near zero, we get

$$\sum_{n=0}^{\infty} \frac{(\lambda t)^n}{(n)!} = e^{\lambda t}$$

Therefore the inequality 10 becomes:

$$\lambda t \geq \frac{m(m+1)}{2} P_{tol} \quad (11)$$

Hence, we solve the inequality $m^2 + m - \frac{2\lambda t}{P_{tol}} \leq 0$ which give us the Formula 2 estimating the number of needed replicas to keep the data available.

Appendix B. Assume that n_1 is the average number of incoming updates per unit time and c_1, their average execution cost. We let n_2 be the average number of incoming queries per unit time and c_2 their average execution cost. Likewise $\bar{S}$ is the average staleness and χ_D the average throughput of replicas. The system receives both updates and queries. Let be r the ratio of n_2 to n_1. $r = \frac{n_2}{n_1}$

Let us number the queries in their arrivals order, so Q_i is the i^{th} query. Likewise U_j is the j^{th} incoming update. Let us assume that the number of updates and queries arrivals per time unit is enough to consider them regulars. Because of this, the arrival of Q_i supposes the one of U_j with: $j = \frac{i}{r}$

Let us assume that at least a query is computed, and Q_i the last query handled by the system during a while (a number of units time), and m the number of system replicas. By taking into account the previous updates, the computation of Q_i on a replica has a global cost equal to:

$$CQ_i = \left(\frac{i}{r} - \bar{S}\right) c_1 + \left(\frac{i-1}{m}\right) c_2 + c_2 \tag{12}$$

The term $\left(\frac{i}{r} - \bar{S}\right) c_1$ represents the number of updates that must be computed by the replica for reaching the staleness required by the query. Then, it cannot be negative. The second term of the Formula ($\left(\frac{i-1}{m}\right) c_2$) means that $i-1$ queries are yet arrived before Q_i and are sent uniformly on the m replicas. The last term is the average execution cost of Q_i. Let us k be the execution duration in number of time units. Thus, we can set:

$$CQ_i \leq k\chi_D \tag{13}$$

In other words:

$$\left(\frac{i}{r} - \bar{S}\right) c_1 + \left(\frac{i+m-1}{m}\right) c_2 \leq k\chi \tag{14}$$

The large inequality represents the amount of works needed to compute Q_i on the replica. With this last formula, we can distinguish two cases:(i) the replica is fully used during all the execution, then we have the equality case, (ii) the replica is partially used, so the strict inequality is considered . Now, we consider that the replica is fully occupied and the case in which the replica is partially occupied will be described later. Then let set:

$$\left(\frac{i}{r} - \bar{S}\right) c_1 + \left(\frac{i+m-1}{m}\right) c_2 = k\chi_D \tag{15}$$

That involves:

$$\frac{i}{r}c_1 + \frac{i}{m}c_2 - \bar{S}c_1 + \left(\frac{m-1}{m}\right) c_2 = k\chi \tag{16}$$

Therefore:

$$i\left(\frac{c_1}{r} + \frac{c_2}{m}\right) = k\chi_D + \bar{S}c_1 + \left(\frac{1-m}{m}\right) c_2 \tag{17}$$

Hence, we can deduce the formula:

$$i = \frac{k\chi_D + \bar{S}c_1 + \left(\frac{1-m}{m}\right) c_2}{\frac{c_1}{r} + \frac{c_2}{m}} \tag{18}$$

This theoretical result is the number of queries computed by the system during k time units. It assumes a good transactional load balance and takes its full importance since it allows us to find a replication threshold. The next appendix talks about this point.

Appendix C. As shown in section 4, we can follow the system evolution with our analytical method. Then, it follows that we can find a sufficient number of replicas which must be a good replication threshold. Let us remind that the knowledge of this value is very important since it would allow to avoid the addition of unless replicas.

Let us consider the function $i = f(m)$ found in 18:

$$i = f(m) = \frac{k\chi_D + \bar{S}c_1 + \left(\frac{1-m}{m}\right) c_2}{\frac{c_1}{r} + \frac{c_2}{m}} \tag{19}$$

The figures presented above show the existence of the replication threshold. Beyond this number of replicas, the system performances do not increase. The variations of the curve of evolution are almost nil (*i.e.* the rate of change $\tau_{m,f}$ of the function f is very low).

Let us note that this derivative cannot be negative. Indeed, the number of processed queries increases with the number of replicas. However, this increase became much small, almost nil, when we exceed the replication threshold.

We can easily compute

$$\tau_{m,f} = \frac{f(m+h) - f(m)}{h} = \frac{c_2 r^2 \left(k\chi_D + \bar{S}c_1 - c_2 - \frac{c_1}{c_2}\right)}{(c_1 m + c_2 r)(c_1 m + c_2 r + c_1 h)} \tag{20}$$

Therefore

$$\lim_{h\to 0} \tau_{m,f} = \lim_{h\to 0} \frac{f(m+h) - f(m)}{h} = \frac{c_2 r^2 \left(k\chi_D + \bar{S}c_1 - c_2 - \frac{c_1}{c_2}\right)}{(c_1 m + c_2 r)^2} \tag{21}$$

Let us choose an enough small rate of change α but higher than zero and postulate:

$$\lim_{h\to 0} \tau_{m,f} \leq \alpha \tag{22}$$

This let us say that:

$$\frac{c_2 r^2 \left(k\chi_D + \bar{S}c_1 - c_2 - \frac{c_1}{c_2}\right)}{(c_1 m + c_2 r)^2} \leq \alpha \tag{23}$$

and then

$$m \geq \sqrt{\frac{c_2 r^2 \left(k\chi_D + \bar{S}c_1 - c_2 - \frac{c_1}{c_2}\right)}{\alpha c_1^2}} - \frac{c_2 r}{c_1} \tag{24}$$

As we are already said, we assume that at least a query is computed and then we suppose that the execution time k is enough higher to have $k\chi_D \geq c_2$. Otherwise, it would implicate that any query is not computed. The replication threshold given by the Formula 24 seems be better as the approximation value α is smaller. However, this approximation value must be taken with care. For instance, if we take 0.5 as approximation value, we have 81 replicas as replication threshold value. For an approximation value equals to 0.4, we have 91 replicas. The difference of approxmation values given equals 0.1 but 10 replicas must be added. The number of computed queries by 91 replicas is not larger that 0.004 percent than the one of 81 replicas while the number of added replicas is 0,123%.This shows the importance to choose a good approximation value for getting best replication threshold. The Formula 24 is usable if the system load in terms of numbers and costs of incoming queries and updates is heavy (see Section 5). If the system workload is light, the above formula may give a bad threshold. In other words, it exists a number of replicas smaller than the given threshold and sufficient for computing the set of incoming queries. Then, we define another formula for such cases. In these cases, all queries are computed. Then let assume that

$$i = f(m) = \frac{k\chi + \bar{S}c_1 + \left(\frac{1-m}{m}\right)c_2}{\frac{c_1}{r} + \frac{c_2}{m}} \geq kn_2 \tag{25}$$

kn_2 is the full number of incoming queries. By transposition, we get:

$$k\chi + \bar{S}c_1 + \frac{c_2}{m} - c_2 \geq kn_2 \left(\frac{c_1}{r} + \frac{c_2}{m}\right) \tag{26}$$

which implicates:

$$\frac{k(\chi - c_1 n_1) + \bar{S}c_1 - c_2}{c_2(kn_2 - 1)} \geq \frac{1}{m} \tag{27}$$

The term $k(\chi - c_1 n_1) + \bar{S}c_1 - c_2$ must be positive. If not, we obtain the contradiction $\frac{1}{m} \leq 0$ even though m is a positive number. This term is negative in the case where the throughput system is not sufficient for treating all queries. Thus we have:

$$m \geq \frac{c_2(kn_2 - 1)}{k(\chi - c_1 n_1) + \bar{S}c_1 - c_2} \tag{28}$$

This last condition is enough for the cases where the transactional system load is small comparing with its throughput. The replication threshold is there the smallest number of replicas which satisfies formula 25.

We use Formula 25 when the term $k(\chi - c_1 n_1) + \bar{S}c_1 - c_2$ is positive otherwise Formula 24 is required with an enough small rate of change α for having a good replication threshold.

7. REFERENCES

[1] F. Akal, C. Türker, H. Schek, Y. Breitbart, T. Grabs, and L. Veen. Fine-Grained Replication and Scheduling with Freshness and Correctness Guarantees. In *Int. Conf. on Very Large DataBase (VLDB)*, 2005.

[2] G. Antoniu, J. Deverge, and S. Monnet. How to Bring Together Fault Tolerance and Data Consistency to Enable Grid Data Sharing. *Concurrency and Computation: Practice and Experience*, 18(13), 2006.

[3] J. Brevik, D. Nurmi, and R. Wolski. Automatic methods for predicting machine availability in desktop grid and peer-to-peer systems. In *CCGRID '04: Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, pages 190–199, Washington, DC, USA, 2004. IEEE Computer Society.

[4] E. Cecchet, G. Candea, and A. Ailamaki. Middleware-based Database Replication: The Gaps Between Theory and Practice. *SIGMOD*, 2008.

[5] R. Gallersdörfer and M. Nicola. Improving Performance in Replicated Databases through Relaxed Coherency. In *VLDB 1995, Proceedings of 26th International Conference on Very Large Data Bases, Zurich, Switzerland*, 1995.

[6] S. Gançarski, H. Naacke, E. Pacitti, and P. Valduriez. The Leganet System: Freshness-aware Transaction Routing in a Database Cluster. *Journal of Information Systems*, 32(2), 2006.

[7] R. Guerraoui and A. Schiper. Software-Based Replication for Fault Tolerance. *IEEE Computer*, 30(40), 1997.

[8] R. Koo and S. Toueg. Checkpointing and Rollback-Recovery for Distributed Systems. *IEEE Transactions on Software Engineering*, 13(1), 1987.

[9] C. Le Pape, S. Gançarski, and P. Valduriez. Refresco: Improving Query Performance Through Freshness Control in a Database Cluster. In *Int. Conf. On Cooperative Information Systems (CoopIS)*, 2004.

[10] M. T. Özsu and Patrick Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, 1999.

[11] E. Pacitti, C. Coulon, Patrick Valduriez, and T. Ozsu. Preventive Replication in a Database Cluster. *Distributed and Parallel Databases*, 18(3), 2005.

[12] E. Pacitti, P. Minet, and E. Simon. Fast Algorithms for Maintaining Replica Consistency in Lazy Master Replicated Databases. *Int. Conf. on Very Large DataBase (VLDB)*, 1999.

[13] M. Patino-Martinez, R. Jimenez-Peres, B. Kemme, and G. Alonso. MIDDLE-R, Consistent Database Replication at the Middleware Level. *ACM Transactions on Computer Systems*, 28(4), 2005.

[14] PeerSim. http://peersim.sourceforge.net/.

[15] K. Ramamritham and C. Pu. A Formal Characterization of Epsilon Serializability. *IEEE Transactions on Knowledge and Data Engineering*, 07(6), 1995.

[16] U. Rohm, K. Bohm, H. Sheck, and H. Schuldt. FAS - a Freshness-Sensitive Coordination Middleware for OLAP Components. *Int. Conf. on Very Large DataBase (VLDB)*, 2002.

[17] I. SARR, H. Naacke, and S. Gançarski. Distributed Transaction Routing with Failure Management in a Large Scale Network. In *24 Journées de Bases de Données Avancées BDA*, 2008.

[18] I. SARR, H. Naacke, and S. Gançarski. DTR: Distributed Transaction Routing in a Large Scale Network. In *VECPAR'08 Workshop on High-Performance Data Management in Grid Environments (selected papers)*, 2008.

[19] F. B. Schneider. Implementing Fault-tolerant Services using the State Machine Approach: A tutorial. Technical report, ACM Computing surveys, 1990.