Efficient Transaction Routing in a Widely Replicated Database

DIENE Bassirou* — GUEYE Modou** — SARR Idrissa** — NDIAYE Samba**

* Dept Maths and Computer Science
Cheikh Anta Diop University
Dakar
Senegal
bassirou.diene@ucad.edu.sn

** Dept Maths and Computer Science
Cheikh Anta Diop University
Dakar
Senegal
gmodou@ucad.sn

** Dept Maths and Computer Science
Cheikh Anta Diop University
Dakar
Senegal
idrissa.sarr@lip6.fr

** Dept Maths and Computer Science
Cheikh Anta Diop University
Dakar
Senegal
samba.ndiaye@ucad.edu.sn

ABSTRACT. Large-scale environments such as P2P system provide tremendous of opportunities for storing shared data. They have very specific characteristics such as autonomy and heterogeneity of peers. However, having effective mechanisms for transactions routing, fast and consistent data access, is very challenging. To face this issue, we propose a transaction routing algorithm based on the availability and the process capacity of nodes. We have implemented our approach using FreePastry simulator. The results shows its feasibility and efficiency.

RÉSUMÉ. Les environnements à large échelle comme les systèmes P2P ont montré depuis des années toute leur importance dans la gestion et le stockage des données. Ils possèdent des caractéristiques très particulières telles que l'autonomie des pairs et leurs hétérogénéités. Cependant, avoir des mécanismes efficaces pour le routage des transactions, l'accès rapide aux données constitue un challenge. Pour faire face à ce défi, nous proposons un algorithme de routage de transactions qui s'appuie sur la disponibilité des nœuds du système et qui prend en compte leur puissance de traitement et leur capacité de stockage. Nous avons mis en œuvre notre approche à l'aide du simulateur FreePastry. Les résultats obtenus montrent son adaptabilité et son efficacité.

KEYWORDS : Database, P2P environment, transaction routing, availability

MOTS-CLÉS : Base de données, environnement P2P, traitement de transactions, disponibilité

1. Introduction

Today, P2P systems [4] and grid computing [5] are imposed themselves in computing and data storage area. P2P systems are generally used for data sharing. However their good qualities like well-scaling and fault-tolerance make them well-suited environments for applications which face tremendous workload.

Thus, nowadays, a large community of researchers turns towards P2P systems to build on them transactional distributed databases. Indeed, despite their complexity, these systems can be very appropriated when we have a high transactional load like in the reservation systems.

The availability of data is ensured by high replication through hundreds or even thousands of nodes and fault-tolerance algorithm [6]. This replication allows parallel treatments but introduces two major issues for researchers. First, replicated data must remain consistent. Secondly, to perform the quick as possible a transaction, we must find the fastest node to process it among all available nodes.

Solutions to theses issues have been proposed. DTR [1] Ganimed [7] are one of them. [1]improves the solution proposed by Leganet [2]. The algorithm used in DTR keeps data consistency but it is based on a strong assumption. It assumes a priori knowledge of transactions execution time. Thus, it could know to which node it should forward the request.

In this paper, we propose an efficient transaction routing in a widely replicated database. We focus particularly to second issue mentioned above. Our mechanism is built on DTR without its strong assumption mentioned above. It takes only into account the characteristics of the system like availability and process capacity of nodes.

The paper is organized as follows. Section 2 presents the architecture on which we rely to do the job, and some definitions of concepts that we have used. Section 3 describes our routing mechanism and its algorithm. Section 4 presents the results we have obtained through various experiments with FreePastry tool to demonstrate the validity and feasibility of our mechanism.

2. System Architecture and definitions of basic concepts

In this section, we first presente the description of our architecture based on the one presented in [1] and secondly define the replicated type and different concepts used in the transactions routing.

2.1. System Architecture

The global architecture of the system is shown in Figure 1. It comes with a distributed middleware that sits between the CNs (Client Nodes) and DNs (Data Nodes). The middleware consists essentially of TMs (Transaction Managers) and a Shared Directory.

– CNs: they represent end-users in our system. They send transactions to Transaction Manager (TM).

– DN: each Data Node uses a local Database Management System for storing data. It computes transactions that it receives from Transaction Managers.

– TMs: they are responsible for routing transactions from a Client Node towards a Data Node. They are organized around a logical ring in order to facilitate the collaborative

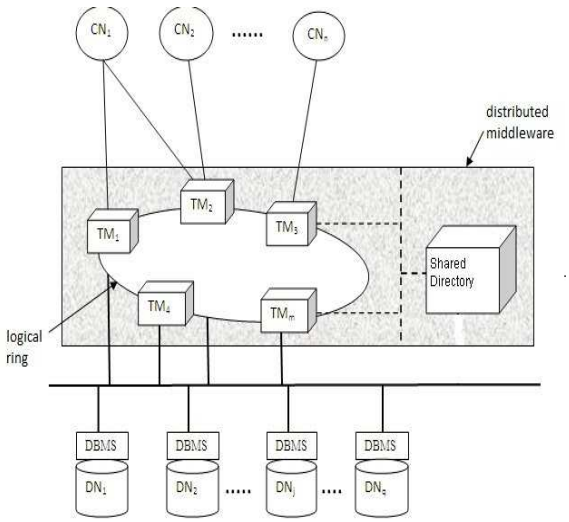

Figure 1. *Global System Architecture.*
detection of failures [1].

– Shared Directory: it contains details of transactions processed on the Data Nodes, i.e. for each relation and for each Data Node, the list of transactions executed and the running transactions. This information is used to calculate the freshness of a DN, but also to determine the synchronised transactions if the Data Node needs to be refreshed.

2.2. Replication and basic concepts

1) ***Replication:*** our system uses an asynchronous symmetric replication [3]. We distinguish three different categories of transactions:

- Update transactions are composed of one or several SQL statements which update the database.

- Queries are read-only transactions. In this case the Data Nodes (DN) did not need to be refreshed all the time.

- Refresh transactions are used to propagate update transactions to other replicas for refreshment. A refresh transaction can be made either by reviving the original transaction or propagating its effects to the database as a sequence of write operations.

2) ***freshness and tolerated staleness:*** the work maked in [2] allow us to define the terms of ***freshness of a replica (or Data Node)*** and ***tolerated staleness*** of a Transaction. These concepts are used in our routing mecanisms.

3) ***Availability:*** we assume that when a transaction executes, it uses almost all the resources of the Database Management System and a Data Node processes transactions one after another. In each DN is defined a queue or buffer with a given size that will contain all transactions sent to that node and which are not yet executing in other words all pending transactions on it. The availability of the DN will depend on a parameter called ***degree of availability*** corresponding to the number of transactions remaining for the queue of the Data Node is full. If this number is positive, then the DN is considered available.The DN sends a message containing its degree to a TM to declare his candidacy for the execution of a transaction. Thus, after this declaration the Transaction Manager

uses the algorithm defined in Section 3.2 to send it transactions.

3. Routing mechanism

In this section, we describe how transactions are routed. Initially, we present the mechanism used for the treatment and then terminated by the proposed algorithm.

3.1. Transactions routing mechanism

The transaction routing adopted is a pull-based[1] mechanism. In this approach, the Data Nodes must inform the TM about their availability to execute a transaction. Once a Data Node is available, the Transaction Manager should immediately send it transactions.

The execution of transactions is illustrated by the Figure 2.

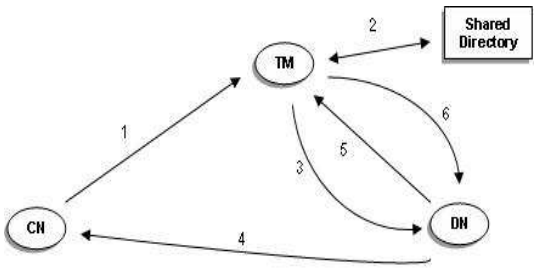

Figure 2. *Transactions routing mechanism.*

– The Client Nodes send either a read-only query or an update to a Transaction Manager (TM) with a tolerated staleness (step 1 on the figure 2). Based on [2], the tolerated staleness of an update is always equals to zero. A Client Node is connected to one or several Transaction Managers. A Transaction Manager is selected by using the Round Robin's method. During routing, a Client Node assigns for each transaction its identifier (i.e. its IP address and port). This in order to allow a Data Node to send its response directly to the CN which has submitted the transaction.

– When a TM receives a transaction from a CN, it puts this newcomer at the end of a queue dedicated to store them. One specialized thread ensures that. Another thread is responsible for leading transactions towards DNs in computing routing algorithm described below (step 3). This thread relies on the shared directory to retrieve or store metadata about transactions or DNs (step 2). In our system, the Data Nodes must notify TM of their availability for the execution of a transaction when they are available. This allows the TM to send them a set of transactions corresponding to their degree of availability.

– A DN after to have executed a transaction sends the response directly to the Client Node which had initiated the demand (step 4). It sends also an acknowledgement of execution to the TM (step 5). Therefore, the latter does the necessary in the metadata in order to maintain the consistency of the database, and then sends in its turn an acknowledgement of receipt to the DN (step 6). This processing and all its subjacent mechanisms described in DTR allow us to keep data consistency.

1. There is also the approach PUSH which it's the Transaction Manager who determine the Data Node who will provide an optimal time processing to send him transactions

To send transactions, the TM doesn't consider only the freshness of Data Nodes but also the tolerated staleness of transactions.

3.2. Routing algorithm

As we have already said, we have defined on the Transaction Manager a queue where transactions from the Client Nodes are ordered before being routed. The discipline used to send transactions is the FIFO (First In First Out).

The Data Nodes send messages to the Transaction Manager so to declare their availability. These Messages don't contain only their degree of availability but also the last transaction that they have received.

When a DN is available, the Transaction Manager uses the algorithm presented bellow to select transactions to send it. The number of transactions depends mainly on its degree of availability but also on the state of coherence of its database.

Algorithm 1 Routing Algorithm.

```
Require: m (corresponding number of Data Node available)
 1: for each DN_i do
 2:    F(Ni)=detFresh();
 3: end for
 4: while m>0 do
 5:    F(N)=detFreshestDn()
 6:    F(Tk)=detStalenessTolered()
 7:    if F(N)>F(Tk) then
 8:       send(Tk)
 9:       removed(Tk)
10:       decrease(m)
11:    else
12:       Tsync=determineTsync()
13:       nbrTrans=numberTrans(Tsync)
14:       if nbrTrans<n-1 then
15:          send(Tsync+Tk)
16:          removed(Tk)
17:          decrease(m)
18:       else
19:          send(Traf)
20:          decrease(m)
21:       end if
22:    end if
23: end while
```

Let's assume that ***m*** ($m>1$) Data Nodes declare to Transaction Manager that they are available to receive transactions. The Transaction Manager receiving these messages determines the freshness of each Data Nodes.

After determining their freshness, the TM selects the freshest DN and compares its freshness with the tolerated staleness of the transaction being at the top of its queue. If the Data Node is enough fresh to receive the transaction then this latter is sent to it. As soon as a transaction is sent to a DN, it is removed from the queue. So that, in the next loop, the transaction which followed the removed one is at the top of the queue. This DN is quoted

unavailable for next treatments. If the DN is not enough fresh to receive the transaction, the TM determines all transactions to propagate to DN in order to enhance its freshness and to do it to reach the tolerated staleness of the transaction.

The TM compares the number of transactions in this set of transactions to propagate with the degree of availability of DN. If this number is lower than its degree of availability, all transactions are sent with the transaction at the top of the queue itself. Otherwise, the TM selects, in this set, a number of transactions equal to the degree of availability of DN and sends it them.

4. Experimental Validation

In this section we demonstrate the feasibility and efficiency of our solution through simulation. To this end, we used FreePastry [8] a simulator of P2P systems that allows us to simulate the performance of our system. In these simulations we have as mains objectives to verify the fastness and the scalability of our algorithm: firstly, we show the impact of the number of Data Node on the average execution time of transactions; secondly, we show that our algorithm ensure load balancing; and finally we conclude by a comparison of our solution with the Round Robin approach.

4.1. Impact of number of nodes on the average time of execution

The first experiment aims to determine the average time of transaction execution. We watch the evolution of the average time of execution by increasing the number of Data Nodes. We set to 10 the number of Client Nodes and each of them sends 100 transactions (i.e updates or read-only queries). We do to vary the number of Data Nodes between 50 and 750 and several times we evaluate the global average execution time of transactions.

As shown in Figure 3, the increase in the number of Data Node has a considerable influence on the average time execution. We observe a decrease in global execution time. This is explained by the fact that the treatment is shared between the Data Nodes. At 750 DNs, we note a decrease of 67% of global execution time.

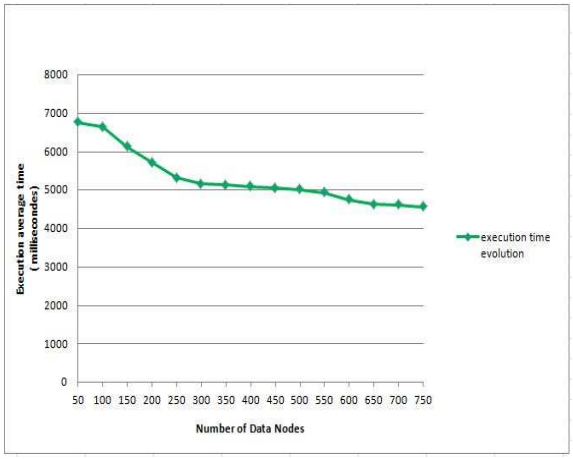

Figure 3. *Impact of number of nodes on the average time of execution, number of CNs fixed.*

In Figure 4, we vary the number of transactions between 100 and 1000. We have led three experiments where the number of DNs is fixed respectively to 250, 500 and 750 DNs. Then we evaluate the global average time for each of these experiments. We observe in the Figure 4 that the execution time increases significantly when we increase the number of transactions sent by the Client Nodes. We also note that the impact of the number of DNs on the average execution time of transactions has the same trend that the one in Figure 3. Indeed, for the same number of transactions, more the number of DNs is important, smaller the execution time is.

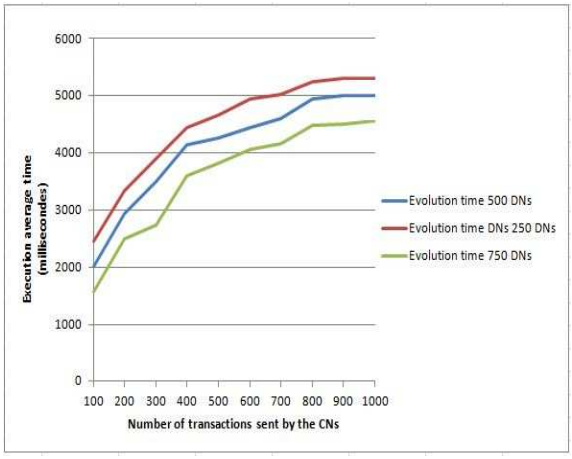

Figure 4. *Impact of number of nodes on the average time of execution, number of CN varied.*

With these two experiences we can see the positive impact that the increasing of the number Data Nodes lead to transactions processing.

4.2. Load Balancing

To watch our system load balancing, we use the mathematical variable ***coefficient of variation*** which represents the ratio of the standard deviation to the mean value of a distribution. To do this, for each experiment we determine the number of transactions executed by each Data Node. Typically, a small coefficient of variation denotes a low standard deviation, this means that the different values of a distribution are clustered around the average one. In other words, smaller is the coefficient of variation, more uniform is the distribution.

During this experiment, the number of Data Nodes varies between 50 and 750 and the number of Client Nodes is fixed. As shown in Figure 5, we notice a constant coefficient of variation with lower values since they are between 0.36 and 0.47. With these values, we can say that the transactions are almost uniformly distributed over the Data Nodes. These results show that the difference in the number of transactions performed by each DN is very high. This allows us to say that load balancing is provided by our solution.

4.3. Comparing our approach with Round Robin

To demonstrate the performance of our approach, we compare its global execution time of transactions to the global execution time given by a Round Robin approach in

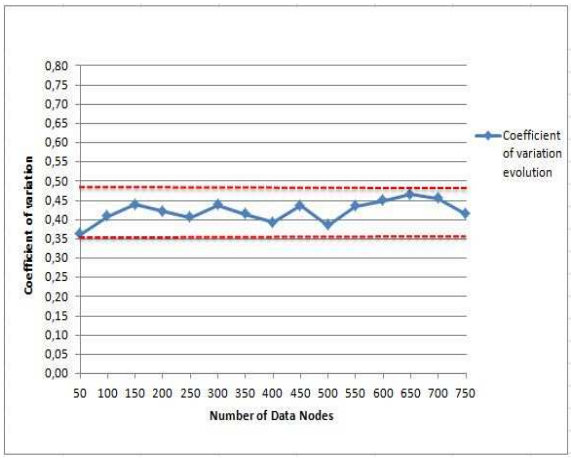

Figure 5. *Variation des coefficients de variation en fonction des Data Nodes.*

the same context. In Round Robin approaches, the Data Nodes are selected in turn for transactions routing.

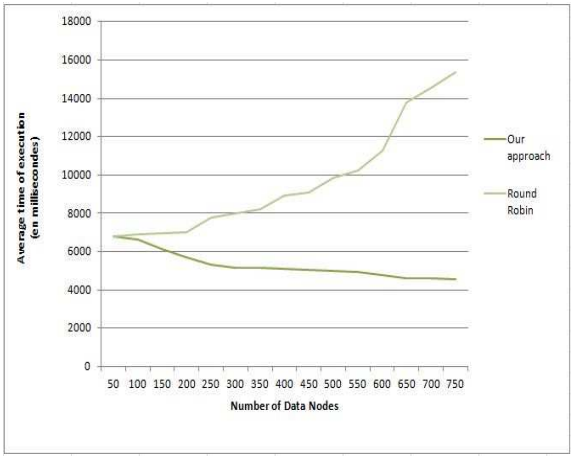

Figure 6. *Comparing our approach with Round Robin.*

In the figure 6, we see that the execution time for the Round Robin approach is higher than the one of our solution if the number of DNs exceeds 150. We observe in the case of Round Robin the execution time grows dramatically when the number of DNs increases. This is explained by the fact that an elected Data Node for the execution of a transaction may have transactions preceding it. So these latters must be sent first. This could increase the global execution time. Note that in the Round Robin strategy, the Data Nodes are refreshed during execution of transactions.

5. Conclusion

In this paper, we have presented a new approach to process transactions in widely replicated databases. Contrary to certain proposed solutions, our solution isn't based on strong assumptions like a priori knowledge of transaction execution time or components homogeneity even if we have large scale systems. Our transaction routing algorithm takes the difference of power processing of DNs by requiring that the DNs declare their availabilities. Because of this, more power a DN is, more available it is. Our algorithm performs the routing without knowing a priori the execution time of a transaction. The decision to execute transactions comes from Data Nodes which declare their availabilities through a parameter called degree of availability. The algorithm also relies on some concepts of [2] such as the tolerated staleness of transactions and the freshness of Data Nodes in order to improve load balancing. Using the network simulator FreePastry, we have validated our approach by conducting several experiments. The results show that our approach is well-suited for large scale systems.

6. References

[1] IDRISSA SARR, HUBERT NAACKE, STÉPHANE GANÇARSKI, "Routage Décentralisé de Transactions avec Gestion des Pannes dans un Réseau à Large Echelle", *Journées de Bases de Données Avancées (BDA)*, 2008.

[2] STÉPHANE GANÇARSKI, HUBERT NAACKE, ESTHER PACITTI, PATRICK VALDUREZ., "The Leganet System: Freshness-Aware Transaction Routing in a Database Cluster", *Journal 32(2), pp. 320-343, 2006*

[3] GARDARIN G., GARDARIN O., "Le Client Serveur", *Editions Eyrolles, Paris, 1997*

[4] ENG KEONG LUA, JON CROWCROFT, MARCELO PIAS, RAVI SHARMA AND STEVEN LIM, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes", *The Journal IEEE Communications Surveys and Tutorials, volume 7, 72-93, 2005*

[5] IAN FOSTER, C. KESSELMAN, EDITORS, "The Grid: Blueprint for a New Computing Infrastructure", *Morgan-Kaufmann, 1999*

[6] IDRISSA SARR, HUBERT NAACKE, STÉPHANE GANÇARSKI, "TransPeer: adaptive distributed transaction monitoring for Web2.0 applications", *SAC '10: Proceedings of the 2010 ACM Symposium on Applied Computing, March 2010*,

[7] C. PLATTNER, G. ALONSO, "Ganymed: scalable replication for transactional web applications", *In Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware, 155-174, 2004*,

[8] "http://www.freepastry.org/FreePastry/",