Rubrique

modeling and simulation

Freshness-Aware Metadata Management: Performance Evaluation with SWN models

Diallo Ousmane* — Mbaye Sene** — Idrissa Sarr***

* Département Mathématiques et d'informatique
Université Cheikh Anta Diop
Dakar
SENEGAL
ousmane81.diallo@ucad.edu.sn

** Département Mathématiques et d'informatique
Université Cheikh Anta Diop
Dakar
SENEGAL
mbaye.sene@ucad.sn

*** Département Mathématiques et d'informatique
Université Cheikh Anta Diop
Dakar
SENEGAL
idrissa.sarr@ucad.sn

ABSTRACT. Recent systems which are in general composed by several resources and distributed over a large-scale network, need high level models to be studied. In these complex systems such as peer-to-peer systems, efficient and fast queries routing is necessary for managing applications workload. Many works have been propose to deal with query routing, however none of these studies does not rely on Stochastic Well formed Petri Nets (SWN) for modelling proposed approach. We aim in this paper, to propose an algorithm for managing metadata needed to route queries. Moreover we use SWN models to evaluate and validate our approach. Our solution is freshness-aware, thus gain in the fact that stale data can be read under some limits. These limits are widley taken into account for managing metadata coherently.Our study takes into account the concurrency, the synchronization, the parallelism, the identity of the resources and their cooperation. We propose two SWN models for structuring metadata: one with strong consistency and another with weak consistency. Simulation are used to validate our approach and results obtainded demonstrate the feasability of our solution.

RÉSUMÉ. Les systèmes récents qui en général se composent de plusieurs ressources et sont repartis sur un réseau à large échelle ont besoin, pour être étudier, des modèles de haut niveau. Dans ces systèmes complexes, dans notre étude les systèmes P2P, le choix d'un routage rapide et efficace des requêtes utilisateurs est nécessaire pour une bonne performance d'exécution. Beaucoup de travaux ont été effectués sur le routage de requêtes, toutefois aucune de ces études n'emploie les réseaux de Pétri bien formés stochastiques (SWN) qui offrent une modélisation à haut niveau. Dans ce travail, nous proposons un algorithme de gestion des méta-données utilisées pour router les requêtes. De plus nous employons des modèles de SWN pour évaluer et valider notre approche. Notre solution exploite le relâchement des données, d'où un gain du fait qu'une donnée obsolète peut être lue sous une certaine limite. Ces limites sont prises en comptes pour la gestion de la cohérence des méta-données. Notre étude prend en considération la synchronisation, le parallélisme, l'identité des

ressources et leur coopération. Nous proposons deux modèles de SWN pour structurer les méta-données : un avec une cohérence forte et un autre avec cohérence relâchée. Des simulations sont employées pour valider notre approche et les résultats obtenus démontrent la faisabilité de notre solution.

KEYWORDS : peer-to-peer systems, transactions routing, metadata, performance evaluation, SWN, GreatSPN, WNSIM

MOTS-CLÉS : Systèmes pair- à-pair, routage de transaction, méta-données, évaluation de performances, SWN, GreatSPN, WNSIM

1. Introduction

Today, peer-to-peer infrastructures offer new opportunities for sharing data on a very large scale. Each node stores some data and can also accesses data store elsewhere. Furthermore each peer can handle requests sent by others nodes if it contains required data. The proposed architecture plans a mode of asynchronous replication [1] which tolerates a certain staleness between replica: one replica is slightly obsolete when it has not received all the transactions updates which are involved. It is thus necessary to route the requests transmitted by the applications towards the nodes of which the data are sufficiently fresh with regard to the needs of the requests. A fast and effective routing is possible thanks to the metadata which allow a good description of the address of the peers and their resources.

In this paper, we are interested in the adaptation of an algorithm of efficient queries routing in a peer-to-peer system with metadata. High level Petri Nets (SWN) are used to build the models. To reach our goal, our work is inspired by works made in [2]. These works present one solution for efficient query routing in a large-scale distributed database. They use *JuxMem* (a software for managing the shared memory) to manage metadata. *JuxMem* [3] uses two levels of locking to control the readings and writings on the catalog, one in shared mode for the readings and the other one in exclusive mode for writings. Locks mechanism leads to performance slowdown. In Web 2.0 applications context (eBay [4], Flickr [5], FaceBook [6]), the large scale and heterogenous environment make failed locking algorithm since failures can happen frequently. Applications can tolerate to read stale data under some limits. Such a model of replication is very used to improve read performance. For instance, in [10] the authors use tolerated staleness to route queries and reach load balancing. Our routing solution is freshness-based thus metadata can be managed without strong consistency.

To this end, we replicate our catalog in order to achieve parallelism when accessing metadata. Therefore we minimize the impact of exclusive locking used whenever metadata are updated. The staleness is controlled by applications which defines their constraints. The staleness is measured by the number of allowed modifications already processed on the master catalog and not yet applied on the slave catalog.

The main objective of this work is to minimize average response time and to improve throughput. To reach our goal, we propose two models for structuring metadata: one with strong consistency and another with weak consistency. These models are built with the formalism of Stochastic Well formed Petri Nets, GreatSPN tool and the symbolic simulator WNSIM are used to obtain numerical values of the performance indices.
The rest of the paper is organized as follows: we recall in section 2, the main related work. We describe our architecture for the query routing in Section 3. In the section 4, we briefly present the stochastic well-formed Petri Nets (SWN). We assume that the reader is familiar with basic properties of (Generalized) Stochastic Petri Net ((G) SPN), Colored Petri Nets and even of SWNs. In the section 5, we estimate the performances of both systems proposed thanks to the SWNs. Finally, we conclude in the section 6.

2. Related work

In the literature, many works were dedicated to effective solutions of transactions routing in distributed systems such as peer-to-peer [7, 8]. Enormous progress was made to

improve the performance of the distributed database (DDB) and to make more effective the algorithms of transactions routing [9]. However, in the all research which was carried out within the framework, we did not see any work using the SWN formalism to model the adaptation of these algorithms. Thus we aimed to complete previous studies by taking into account the synchronization, parallelism, cooperation and concurrency between encountered entities in complex systems such as a large-scale transactional database systems. In complex systems, particularly the widely distributed databases, the transactions routing consists to route efficently the transactions sent by applications towards the accuracy nodes or peers, i.e. which contain good data to be manipulated [8]. It is thus necessary to route the requests transmitted by the applications towards of which nodes the data are sufficiently fresh with regard to the needs of the requests. For a fast and effective routing you need a good management to the metadata which allow a good description of the coordinates of the peers and their resources.

3. System architecture

We assume a replicated distributed database with an asynchronous replication model [1]. Applications can tolerate to read stale data under some limits. Such a model of replication is very used to improve read performance. For instance, in [10] the authors use tolerated staleness to route queries and reach load balancing. We distinguish two kinds of queries, update queries which modify data and read-only queries which merely read data. To route queries, we use metadata to locate data accessed by queries. Metadata store both data location and transaction processed on a replica. Thus, update queries imply to modify metadata where read-only queries do not lead any writes. Finally, we suppose that update operations are lower compared to read-only operations.

We aim to process fast read queries blocked by locks used in *Juxmem* when a write operation is handled concurrently.

Our architecture (see figure 1) is inspired by works made in [2]. We leverage the system architecture presented in [2] by using a master-slave schema for storing metadata and a new component into the router structure to split the workload and route it to the appropriate metadata copy.

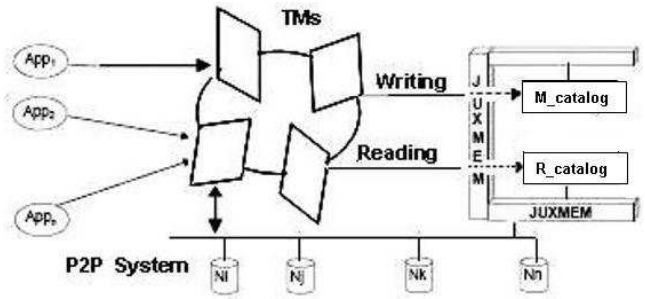

Figure 1. *Our system architecture*

– Description of the architecture:
Appk: is an application client which send request.

TM: (transaction manager) is the router, thus intercept transactions sent by client application and route them to database nodes. When it receives a transaction, it chooses a database node for its execution according to loads and freshnesses (and possibly other parameters).

Catalog: is the building blocks wich store the metadata. We distinguish *M_Catalog* from the *R_Catalog*. The first acts as a master where the former is a slave. In other words, metadata are replicated in such that writes operations are handled on the master copy where read operations are routed to the slave copy.

Ni: is a database node which stores a copy of the data. Data are managed by a local DBMS (Database Management System). Therefore transactions are controlled and validated by local DBMS.

– Routing Protocol:

Our routing strategies is quite similar with those described in [2]. The main difference is the distinction made in order to access catalog either for write operation or read operation. Since several TMs can access simultaneously the same portion of the metadata, therefore metadata inconsistency can be expected. To ensure consistency despite simultaneous access, many solutions rely on lock mechanisms. For example, *Juxmem* uses locks to handle concurrent access. The major drawback of using locks is the blocked situation created by confliting operations (read-only and writes operations). To avoid that read operations are blocked by write operations, we use a master-slave mechanism. Precisely, the master copy of the catalog is used whenever a transaction (update) must be routed and slave copy remain avalaible for other operations (read-only queries). Thus, performances of read operations are improved. Moreover, some read-only queries can tolerate some staleness under some limits , thus, a stale copy of data (missing last updates) can be used. For example retrieving the number of seats of a flight does not necessary requires the lasts reservations.

1) In the case of a read-only, the TM accesses to $R\_catalog$ with the tolerated staleness. If the tolerated staleness is verified, the TM read directly on the $R\_catalog$. After retrieving all required informations, the TM chooses the optimal node (i.e. the node wich minimize the cost of execution). Finally, it routes the transaction to the chosen database node, which can carry out the execution. After validation, the database node sends directly the results to the client which initiate the transaction and notifies the TM the successful end of the execution. If tolerated stalness is not achieved, the we refresh the $R\_catalog$ before retrieving metadata. When the $R\_catalog$ is refreshed, orther TMs are notified in such that they postpone their access for a while.

2) If the TM receives a transaction, it contacts directly the master catalog, which always contains coherent metadata.

4. Stochastic Well formed Petri Nets

The need to have a high-level class allowing a direct analysis without passing in the unfolding gave birth to Well Formed Petri Nets (WN) [11]. In these networks, the colour classes and domains are structured well as well as the colour functions to allow the creation of algorithms making possible their study without passing by an unfolding. However, these networks did not take into account the stochastic character of certain networks, hence the extension of this class to the Stochastic Petri Nets which ends in the Stochastic Well-formed Petri Nets (SWN). These classes of high-level networks are suited well to

study very complex systems with concurrency, parallelism, synchronization and cooperation. In the SWN [11, 12], colour domains for every place and transition are cartesian product of basic colour classes usually modelling the various fields of the entities of the system, or the entities themselves for transition colour domains. A basic class is a set of entities which have the same behaviour or a comparable nature (class of servers, class of resources). The basic color class in our model is: $R = R1\_ \biguplus R2\_$ (requests of reading and requests of writing).

4.1. Simulation versus exact calculus in SWN

When the size of the system is not too large, we can use the PERFSWN tool [13] to compute exact performance indices. However, when the size and the complexity grow, the analytical resolution is out of reach of current tools because of the size of this generated chain. It is a general phenomenon and in most cases, simulation technics are used instead. The main result in this paper was thus obtained thanks to the new simulator WNSIM [14, 15]. WNSIM is an event driven stochastic Discrete Event Simulator (DES) whose models are generalized Stochastic Petri Nets (GSPN) or SWN. It allows the user to specify all parameters of the simulation run: size of the batch, length of the initialization phase, confidence level (5%) and confidence interval (0.95) i.e given a simulation result r_s of a performance measure r, the exact value r_e of r satisfies $Prob(r_e \in [0.95r_s, 1.05r_s]) \geq (1 - 0.05)$.

5. Performance evaluation with SWN models

Our goal is to compare performance indices of the mechanism of management of metadata consistency with *JuxMem* [3] for queries routing made on the works [2] and our new mechanism with weak consistency of metadata for read-only queries. To reach our goal, we propose two models; these models have the same elementary class R. All the places and transitions have the same colour domain (R). All the queries sent by peers(reading-only and writing) of the network is modelled by the marking of the place *pairs*. The marking of places $W\_waiting$ and $R\_waiting$ stand respectively for the transactions in wait of the writing processor and the reading queries in wait of that of reading. The current reading queries in processing are modelled by the marking of the place $reading$ and the current transactions in processing by the marking of the place $writing$. The marking of the place Log models the log of the transactions processed on the master catalog and not reproduced on its replica. The locking mechanism is represented by the marking of the place $Synch$. The most important transitions of these two models are:

– Transitions **begin_reading** and **end_reading** (subclass of reading requests: R1_): these transitions model the beginning/end of a reading request execution;

– Transitions **begin_writing** and **end_writing** (subclass of transactions: R2_): beginning/end of a transaction treatment;

– Transactions **begin_up** and **end_update**: these transitions model the beginning/end of updates of a metadata copy.

5.1. Model with strong consistency

This model (see figure 2) describes the functioning of a catalog distributed on a peer-to-peer network managed in *JuxMem*. It essentially simulates the synchronization for the common access to shared data in *JuxMem*. In this model, we consider that all the peers (TMs) want to access the same catalogu's granule. Every peer can send a reading query or writing. The queries are executed according to their arrival order. If a peer sends a writing query on a data for the catalog (place $pairs$ of the model), it is executed according to its arrival order (Arr until $Exit_queue$). In that case, the data is locked (consumption of the token of the place $Synch$ of the model and the entry to the writing processor modelled by the place $writing$) by the beginning of the execution until its end. On the other hand, if it is about a reading, it releases the other readings which follow it and block writings until the end of its execution (consumption of the token of the place $Synch$ and entry to the reading processor modelled by the place $reading$).

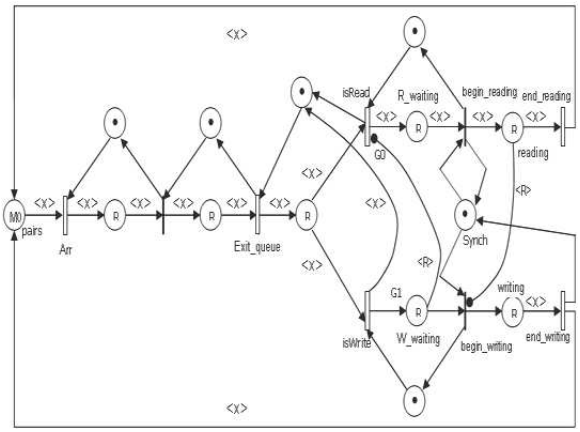

Figure 2. *Model with strong consistency*

5.2. Model with weak consistency

This model (see figure 3) improves the previous by integrating weak consistency for the readings. It functions in the same way. However in this model, the readings are not any more executed in the same unity of treatment (processor) as writings. Indeed, writings on one granule are executed by the processor of our master catalog (place $writing$ of the model) managed by *JuxMem*. On the other hand, the readings on same granule (really a copy) are executed by the processor of $R\_catalog$ (replica of the catalog in *JuxMem*). Therefore, writings are going to jam between them only (exclusive locking) and the readings will be released. Hence a parallelism of treatment between the readings and writings is noticed.

The weak consistency is then controlled by a degree of tolerated staleness (modelled by the parameter *N* on the inhibitor arc connecting the place Log with the transition Deb_Lec) which depends on the current request. This degree of freshness is measured with regard to the master catalog. It corresponds to a number of writings or transactions made on the master catalog and not yet reproduced on the slave ($R\_catalog$). So, if

a reading request on the data *d1* wants to access the processor of $R\_catalog$, we look if the number of writings executed by the processor of the catalog on *d1* and not yet reproduced on $R\_catalog$ (number of tokens in the place Log) did not reach the degree tolerated by the request (N). If it is not the case, the request will be treated without problem. Otherwise, it will be put on hold and we proceed to an update of the metadata of $R\_catalog$ by the update processor (place $update$ of the model). During the update phase of the data, the unities of treatment of the catalog and the $R\_catalog$ are blocked to avoid a possible modification of the data during their reconciliation.

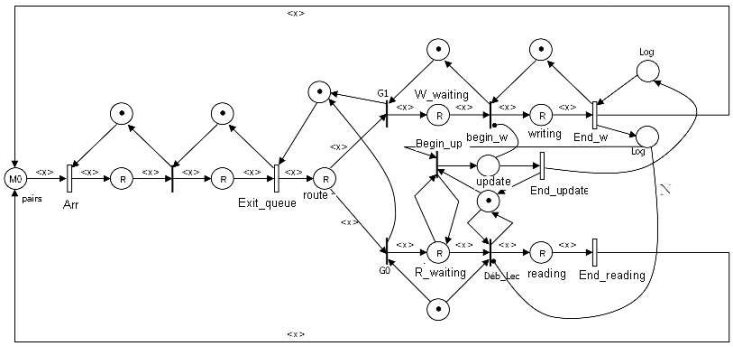

Figure 3. *Model with weak consistency*

5.3. Analysis of our two models

Now, let us recall how to determine the performance indices which are in interest (response time, throughput) in the context of SWN for our system. Let us take transition Arr which models the requests sent by the clients (place *pairs*), it's mean throughput for $(x) \in R$ (i.e request type x) is given by:

$$\overline{\chi}(Arr, x) = \sum_{\widehat{(}\mathbf{m})\in \mathbf{TSRS}, \widehat{\mathbf{m}}[\mathbf{Arr(x)}>} w(Arr, x).\pi(\widehat{\mathbf{m}}) \tag{1}$$

where $\widehat{m}$ is the symbolic tangible marking and $w(Arr, x)$ is the firing rate for static subclasses (x) of transition Arr; $TSRS$ is the tangible symbolic reachability set. Let us denote by P' the set of places visited by a request (x) starting from its generation until the arrival of the response, $\widetilde{\mathbf{m}}(p, x)$ the marking of place p for the static cartesian product of the tuple (x) in $\widehat{m}$ and $\pi(\widehat{\mathbf{m}})$ the steady state probability for the marking $\widehat{m}$. The mean number of requests $\overline{N}(x)$ is:

$$\overline{N}(x) = \sum_{p\in P'} \widetilde{\mathbf{m}}(p, x)\pi(\widehat{\mathbf{m}}) \tag{2}$$

and the mean response time is computed (using Little law) as:

$$\overline{R(x)} = \frac{\overline{\mathbf{N}}(x)}{\overline{\chi}(Arr, x)} \tag{3}$$

In this work, we are interested in the variation of the response time for the treatment of the reading requests according to the consistency and to the load of the system and the

impact of the management of the catalog (synchronization of replicas) at the treatment time of the requests. For the obtaining of these performances criteria, we modify the following parameters:

– The number of requests, to make vary the charge of the system;

– the freshness degree tolerated

– And the rates of transitions *isRead* and *isWrite* to have a uniform distribution of tokens in the simulation.

To make the qualitative and quantitative analysis we use the software WNSIM of GreatSPN; let us recall that this simulator is the only tool which permits symbolic simulation in SWN models. So as to do the comparison, we show several curves for the two models.

– **Results**

In this section, we present our experimental results.

Response time vs relaxation:

We subject both systems to a global charge of 5000 requests and release gradually the freshness degree at the level of our system.

5.3.1. Case where there are more readings than writings

We subdivided the requests for the experiment of the figure 4 into 80% of read requests and 20% of transactions. For instance in figure 5 we are 60% of read-only requests and 40% of transactions. We obtained the results below: according to the look of the curves of both figures, we notice that if the relaxation degree (N) increases, the response time in *JuxMem* remains constant. It is due to the fact that *JuxMem* tolerates no relaxation: the data are subjected to a strong consistency; independently of the relaxation the response time remains big and constant. The response time of our system, as for him, decreases gradually. This can be explained by the fact that our system allows the parallelism of the executions of reading requests with regard to those of writing. Let us note furthermore that for a charge of 60% of requests and 40% of transactions (figure 5) the response time in our system gets closer to that of *JuxMem*. This phenomenon is normal because the more we have transactions, the more we have updates of synchronization at the level of the replica thus more blockings of readings. We can say that globally the relaxation degree as well as the charge of network affects the speed of requests routing.

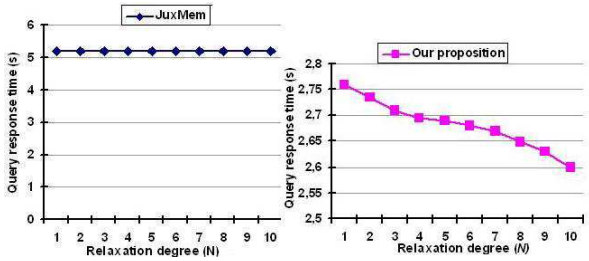

Figure 4. *Response time vs relaxation (*80% *of readings and* 20% *of writings)*

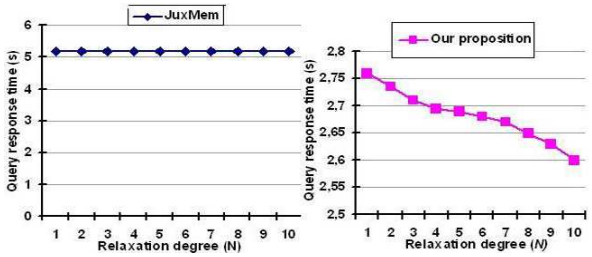

Figure 5. *Response time vs relaxation (*60% *of readings and* 40% *of writings)*

5.3.2. Case where there are fewer readings than writings

We subjected both systems to a charge of 20% of requests and 80% of transactions (see figure 6).

We notice the same looks as in the previous curves. On the other hand, the response time of *JuxMem* is better than ours. Indeed, our system aims to improve the response time of read-only requests treatment. However, as the execution of a request requires in most of the time a certain freshness in spite of the relaxation, the system is compelled to make synchronizations between the replicas. The frequency of these synchronizations is all the bigger as there is of writings operation. Therefore, the increase of the number of writings with regard to that of the readings has a negative impact on our model.

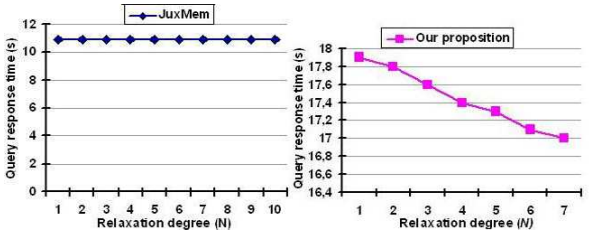

Figure 6. *Response time vs relaxation (*20% *of readings and* 80% *of writings)*

Impact of the management of the catalog:

The management of the catalog (of its replica) requires synchronizations during the refreshments asked by the reading requests. During these phases of synchronization, no request is executed to bring the data in an inconsistent state. This slows down the system and increases its response time. So, the execution time of a request with only-reading contains those of the writing requests that must be reproduced on the level of $R\_catalogue$ so that this last one reaches the degree of freshness asked by the reading request. The following experiment resumes the parameters of simulation of figure 4 but calculates the impact of the synchronizations at the treatment time of the requests with only-reading. We measure this impact through the waiting times of the requests during the synchronizations. The figure 7 shows the results which we obtained: We see that the more the relaxation degree is high, the more the impact due to the synchronization of the

replicas is small, what is completely logical. Let us note however that the more we relax the freshness, less the results of the requests with only reading will be reliable.

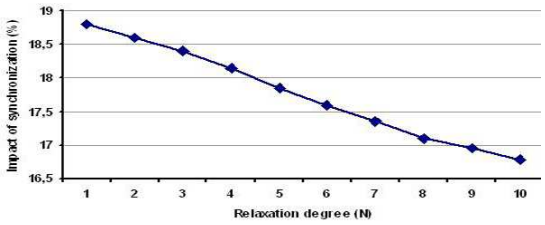

Figure 7. *Impact of the management of the catalogue (synchronization)*

6. Conclusion

We have proposed in this paper a new transactional distributed system with weak consistency of metadata managed in *JuxMem* and its requests routing algorithm and we have compared its performances criteria (response time, impact of the management of the catalog) to those obtained with strong consistency. The choice of an effective algorithm of fast requests routing is essential to guarantee, in the Web 2.0 applications, a good performance of requests execution.
Due to the high level of modelling of SWN, synchronization and parallelism between the read-only queries treatment and that of the transactions is taken into account in our study, this is an important result in our context. Refined performance indices depending on subclass cartesian product are computed via the symbolic simulator WNSIM, this is another important result of this article; the global performance indices presented on this work are obtained by merging those refined performances. The other main contribution of this work is how to use stochastic well formed Petri nets in complex systems such as distributed database with weak consistency of metadata.

In the future works, the existence of optimal degree of tolerated staleness and the correlations of this optimal degree with the parameters of the system (transactional load) will be investigated. This would allow us to see more the limits of our proposition to be able to bring it possible improvements.

7. References

[1] G. et O. Gardarin. *Le Client-Serveur*. Editions Eyrolles, 1997.

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

[3] Mathieu Jan. *JUXMEM : un service de partage transparent de données pour grilles de calcul fondé sur une approche pair-à-pair*. Thèse de doctorat, UNIVERSITÉ DE RENNES 1, France, 2006.

[4] eBay. http ://www.ebay.com.

[5] Flickr. http ://www.flickr.com.

[6] FaceBook. http ://www.facebook.com.

[7] A. Rowstron and P. Druschel. *Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems*. IFIP/ACM Int. Conf. on Distributed Systems Platforms (Middleware), 329-350, 2001.

[8] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. *Chord : A scalable peer-to-peer lookup service for internet applications*. In Proc. ACM SIGCOMM, Août 2001. http://citeseer.csail.mit.edu/469485.html.

[9] Dmitri Loguinov, Anuj Kumar, Vivek Rai, Sai Ganesh. *Graph-Theoretic Analysis of Structured Peer-to-Peer Systems: Routing Distances and Fault Resilience*. SIGCOMM 03, August 25-29, 2003, Karlsruhe, Germany. Copyright 2003 ACM 1-58113-735-4/03/0008.

[10] S. Gancarski, H. Naacke, E. Pacitti and P. Valduriez. *The Leganet System: Freshness-Aware Transaction Routing in a Database Cluster*. Information Systems Journal 32(2), pp. 320-343, 2006.

[11] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. *On Well formed colored nets and their symbolic reachability graph*. In Springer- Verlag, editor, Proc. of the 11th International Conference on Application and Theory of Petri Net, pages 373-396, Paris, France, 1990.

[12] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. *Stochastic well-formed colored nets and symmetric modeling applications*. IEEE TC 1993.

[13] S. Haddad, P. Moreaux, and M. Sene. *Performance Evaluation with SWN: a technical contribution*. R´eseaux et syst'emes r´epartis - CP Vol 13 N 6, 2001.

[14] G. Franceschinis, R. Gaeta, and C. Bertoncello. WNSIM: *manual*. PEG, Dipart.di Informatica, Univ. di Torino (Italy), 2001.

[15] G. Franceschinis, R. Gaeta, and C. Bertoncello. GreatSPN: *User's Manual (version 2.0.2)*. PEG, Dipart. diInformatica, Univ. di Torino (Italy), 2002.