Peer-to-Peer Distributed Shared Memory?

Gabriel Antoniu, Luc Bougé, Mathieu Jan

Peer-to-Peer Distributed Shared Memory?

Gabriel Antoniu, Luc Bougé, Mathieu Jan
IRISA/INRIA and ENS Cachan Antenne de Bretagne
Campus de Beaulieu, 35042 Rennes, France
Contact: Gabriel.Antoniu@irisa.fr

Abstract

In this paper, we show that, although P2P systems *and* DSM systems *have been designed in rather different contexts, both can serve as major sources of inspiration for the design of a hybrid system, with intermediate hypotheses and properties. We propose the concept of* data sharing service *for grid computing, as a compromise between DSM systems and P2P systems. The main contribution of such a service is to decouple data management from grid computation, by providing* location transparency *as well as* data persistence *in a* dynamic environment. *To illustrate the proposed concept and validate its feasibility, we have implemented a software prototype: the* JUXMEM *platform.*

1. A peer-to-peer DSM?

Peer-to-peer [18, 16] (P2P) computing has recently known a growing interest within the distributed computing community. This is mainly due to the success of file sharing systems like Napster [36], Gnutella [19] or KaZaA [35], which have proven the adequacy of the peer-to-peer approach for data sharing on *highly dynamic, large-scale configurations* (millions of nodes). Several research projects on P2P systems are currently in progress, however most of them have focused on devising smart mechanisms for efficient sharing of *immutable, read-only* data at a very large scale. Only a few systems (e.g. Oceanstore [14], Ivy [17]) have started to address the issue of sharing mutable data, but the current solutions have proven efficient only in special cases: they generally assume few writers or few data modifications.

On the other hand, in today's scientific applications, data can generally be read, but also *modified* by multiple sites. To handle the consistency of replicated data, a lot of models and protocols have been proposed within the context of *DSM systems* (Distributed Shared Memory [21]). However, let us note that these systems have been designed by assuming a *static, small-scale* architecture (typically, a cluster of PC).

In this paper, we show that, although *P2P systems* and *DSM systems* have been designed in rather different contexts, both can serve as major sources of inspiration for the design of a hybrid system, with intermediate hypotheses and properties.

2. Why combine DSM systems and P2P systems?

2.1. Peer-to-peer systems: high scalability on highly dynamic configurations

The peer-to-peer model has been made popular by systems allowing music files to be shared among millions of intermittently connected users (Napster, Gnutella, etc.). The underlying model of these systems is simple and complementary to the client-server model: the relations between machines are symmetrical, each node can be client in a transaction and server in another. It has thus been proven that such a model scales very well without any need for a centralized storage server for the shared files: each client node is equally a server and can provide files to the other nodes. As an example, within the KaZaA network, 4,500,000 users simultaneously connected share 900,000,000 files containing 9 peta-bytes of data. This *high scalability* has drawn the attention of the distributed systems community, since it shows a way to make an important step forward in this field. Traditional distributed systems based on the client-server model have often shown limited scalability, generally due to bottlenecks generated by the use of centralized servers. By removing these bottlenecks, the peer-to-peer model not only enhances the system's scalability, but also improves its fault tolerance and availability despite the *high node volatility*. The system's activity is no longer dependent on the availability of a single server.

These important properties explain why the peer-to-peer model has attracted the interest of the scientific distributed systems community. Within this context, the research efforts mainly focused on devising efficient peer-to-peer localization and routing schemes [20, 25, 27, 29, 23, 12, 7],

based on the use of distributed hash tables (DHT). These schemes have been illustrated by systems like Chord [27], Tapestry [29] and Pastry [25], which serve as basic layers for higher-level data management systems, such as CFS [6], Oceanstore and PAST [26], respectively.

On the other hand, we can note that these systems focus on sharing immutable files: the shared data are *read-only* and can be replicated at ease, without any limit on the number of copies. It is easy to understand that, if the data were modifiable, a mechanism would be necessary to handle the consistency of the data copies. But, by guaranteeing consistency, the system would set a non-desired limit to scalability: the more the data copies, the higher the consistency overhead. Therefore, most peer-to-peer systems make a compromise: they favor scalability by sacrifying the data mutability.

Recently, some mechanisms for sharing mutable data in a peer-to-peer environment have been proposed by systems like OceanStore, Ivy and P-Grid [8]. In OceanStore, for each data, only a small set of primary replicas, called the *inner ring* agrees, serializes and applies updates. Updates are then multicast down a dissemination tree to all other cached copies of the data, called *secondary replicas*. However, OceanStore uses a versioning mechanism which has not proven to be efficient at large scales, as published measurements [24] on the performance of updates assume a single writer per data block. The Ivy system has one main limitation: applications have to repair conflicting writes, thus the number of writers per data is equally very limited. P-Grid proposes a flooding-based algorithm for updating data, but assumes no conflicting writes. Besides, no experimental results have been published so far for this system. We can clearly conclude that handling consistency is a serious problem for peer-to-peer systems: the preliminary solutions tentatively proposed as of today have the significant drawback of limiting the system scalability, which is the main property which makes peer-to-peer systems interesting.

2.2. Distributed Shared Memory: consistency and transparency

The problem of sharing mutable data in distributed environments has been intensively studied during the past fifteen years within the context of Distributed Shared Memory (DSM) systems [15, 21, 2]. These systems provide transparent data sharing, via a unique address space accessible to physically distributed machines. As in the case of peer-to-peer systems, reading data on multiple nodes may result in data replication. But the DSM nodes can also modify the data, and this results in triggering some consistency action (e.g. invalidation or update), according to some consistency protocol which implements a given semantics stated by some consistency model. A large variety of DSM consistency models and protocols [21, 10, 5, 4, 11, 30] have been defined [22], in order to provide different compromises between the strength of the consistency guarantees and the efficiency of the consistency actions. These efforts have been carried out within the context of research on high performance parallel computing, often with the goal of providing maximum transparency at a minimum cost.

A central feature of DSM systems is *transparency*. First, these systems provide *transparent access* to data: all nodes can read and write any shared data in a uniform way, should the data be local or remote. The DSM system internally checks for data locality and takes the appropriate action in order to satisfy the access. Second, DSM systems also provide *transparent localization*: if the program accesses remote data, it is the responsibility of the DSM system to localize, transfer or replicate it locally, according to the corresponding consistency protocol.

However, existing DSM systems have generally shown satisfactory efficiency (i.e. near-linear speedups) only on small-scale configurations (in practice, up to a few tens of nodes [22]). This is often due to the intrinsic lack of scalability of the algorithms used to handle data consistency. These algorithms have often been designed by assuming a small number of copies per shared data. For instance, Multiple-Reader-Single-Writer algorithms [15] clearly cannot perform well at large scale, since any write operation on some data results in an expensive invalidation of *all* existing data copies. In the same way, Home-Based, Multiple-Writer algorithms [30] also rely on having the home node *centralize* and merge data modifications from *all* writers. On the other hand, an overwhelming majority of protocols assume a static configuration where nodes do not disconnect nor fail: the unique writer of a given data is not supposed to go down, nor is the home node in a home-based DSM. Only a few DSM systems have integrated some mechanisms for fault tolerance [28, 13]. However, nodes failures are supposed to be infrequent and are considered as an *exceptional behavior*. This is to be contrasted with the basic hypotheses of peer-to-peer systems, in which nodes are assumed to join and leave the network at any time, as a *regular behavior*. Therefore, we can conclude that getting DSM *highly scalable* and adaptive to *dynamic configurations* is a real challenge, since it conflicts with the founding properties of traditional DSM systems.

2.3. Hybrid approach: a data sharing service for scientific computing

Although *P2P systems* and *DSM systems* have been designed in rather different contexts, we think both can serve as major sources of inspiration for the design of a *hybrid* data sharing system. If DSM systems can usually handle configurations of *tens or hundreds* of nodes, corresponding

	DSM	Grid data service	P2P
Scale	10^1–10^2	10^3–10^4	10^5–10^6
Topology	Flat	Hierarchical	Flat
Resource control and trust degree	High	Medium	Null
Dynamicity	Null	Medium	High
Resource homogeneity	Homogeneous (clusters)	Rather heterogeneous (clusters of clusters)	Heterogeneous (Internet)
Data type	Mutable	Mutable	Immutable
Application complexity	Complex	Complex	Simple
Typical applications	Scientific computation	Scientific computation and data storage	File sharing and storage

Table 1. A grid data sharing service as a compromise between DSM and P2P systems.

to *cluster computing*, peer-to-peer systems generally target configurations of *millions* of nodes, corresponding to the scale of *Internet*. The hybrid data sharing system we propose targets configurations of *thousands and tens of thousands* of nodes, which corresponds precisely to the scale of *grid computing* [9].

Therefore, we think the adequate approach for the design of such a system is not to build a *peer-to-peer DSM*, nor a *shared-memory peer-to-peer system*, but rather a *data sharing service for grid computing*. Such a service has to address the problem of managing *mutable data* on *dynamic, large-scale configurations*. The approach we propose benefits both from DSM systems (transparent access to data, consistency protocols) and from P2P systems (scalability, support for resource volatility and dynamicity).

These two classes of systems have been designed and studied in very different contexts. In DSM systems, the nodes are generally under the control of a single administration, and the resources are trusted. In contrast, P2P systems aggregate resources located at the edge of the Internet, with no trust guarantee, and loose control. Moreover these numerous resources are essentially heterogeneous in terms of processors, operating systems and network links, as opposed to DSM systems, where nodes are generally homogeneous. Finally, DSM systems are typically used to support complex numerical simulation applications, where data are accessed in parallel by multiple nodes. In contrast, P2P systems generally serve as a support for storing and sharing immutable files. These antagonist features are summarized in the first and third columns of Table 1.

A data sharing service targets physical architectures with *intermediate* features between those of DSM and P2P systems. It addresses scales of the order of thousands or tens of thousands of nodes, organized as a federation of clusters, say tens or hundreds of hundred-node clusters. At a global level, the resources are thus rather heterogeneous, while they can probably be considered as homogeneous within the individual clusters. The control degree and the trust degree are also intermediate, since the clusters may belong to different administrations, which set up agreements on the sharing protocol. Finally, the service targets numerical applications like heavy simulations, made by coupling individual codes. These simulations process large amounts of data, with significant requirements in terms of data storage and sharing. These intermediate features are illustrated in the second column of Table 1.

The main contribution of such a service is to decouple data management from grid computation, by providing *location transparency* as well as *data persistence* in a *dynamic environment*. As explained in the scenarios described below, such a service can prove helpful for heavy numerical simulations, based on code coupling, with significant requirements in terms of data storage and sharing.

3. A data sharing service for the grid: sample scenarios

Persistence. Since grid applications can handle large masses of data, data transfer among sites can be costly, in terms of both latency and bandwidth. In order to limit these data exchanges, the data sharing service has to rely on strategies able to 1) reuse previously produced data; 2) trigger "smart" pre-fetching actions to anticipate future accesses and 3) provide useful information on data location to the task scheduler, in order to optimize the global execution cost.

Let us consider the following scenario, which illustrates the first point mentioned above. A client submits a computation $C = f_1(A, B)$ to the grid infrastructure. The execution is scheduled on server $S1$. To run this computation, the client needs to submit A and B (which may be large matrices) to $S1$. At the end of the computation, C is transferred from $S1$ to the client. Let us now assume that the

client has to execute a second computation $D = f_2(B, C)$ on the same server. To do this, the client would have to resubmit B and C to $S1$. To avoid such unnecessary transfers, the data sharing service has to provide *persistence* by allowing to reuse the matrices already present on the storage infrastructure. The client should be able to characterize the persistence guarantees for any data stored by the service.

Transparency. Another desirable feature for a data sharing service is *transparency with respect to data localization*: the service user should not explicitly handle data transfers between storage servers, but rather leave this to the service.

Let us consider a scenario in which a distributed federation of 3 clusters, A_1, A_2 and A_3 co-operate together as shown on Figure 1. Each cluster is typically interconnected through a high-performance local-area network, whereas they are all coupled together through a regular wide-area network. Consider for instance a weather forecast simulation. Cluster A_1 may compute the forecast for a given day, then A_2 for the next day, and finally A_3 for the day after. Thus, A_3 uses data produced by A_2, which in turn uses data produced by A_1, as in a pipeline. To communicate data from A_1 to A_2, the usual approach [1] consists in writing data on a hard disk of A_1, then use some FTP-like tool to transfer them on a disk of A_2. This send-receive method requires an *explicit* participation of the applications. Besides, it obviously does not scale: if multiple servers get involved in the co-operation, the management of communication and synchronization grows quickly very complex. In contrast, we can easily imagine a programming model where applications can read/write data from/to a data sharing service which is in charge of *transparently* localizing and transfering data.

Automatic redistribution. Numerical grid applications usually manipulate *structured* data: matrices, meshes, etc., which can be distributed on multiple nodes. Descriptive informations about how data are structured and distributed and about the access patterns used by the applications can equally help the service to improve its performance, thanks to appropriate pre-fetching schemes. For example, when an element of some matrix distributed on a given cluster is accessed by a node in a second cluster, this could trigger the matrix transfer to the second cluster, with an automatic redistribution if necessary.

Let us consider again the pipeline scheme on the 3-cluster federation described in the previous scenario. Let us now assume that application A_1 uses a block data distribution, A_2 uses a cyclic distribution and A_3 uses a block-cyclic distribution. Communication strategies available in existing grid environments, based on explicit transfers, would clearly make the application code use very complex communication patterns. Here again, a data sharing service can make an extra step forward towards transparency by providing facilities for automatic data redistribution. The application code is then greatly simplified.

4. Preliminary validation: the JUXMEM prototype

4.1. Overview of the JUXMEM platform

In order to tackle the issues described above, we have defined an architecture proposal for a data sharing service. This architecture mirrors a federation of distributed clusters and is therefore *hierarchical* and is illustrated through a software platform called JUXMEM [33, 3] (for *Juxtaposed Memory*). A detailed description of this architecture is given in [3]. The architecture consists of a network of peer groups (`cluster` groups), each of which generally corresponds to a cluster at the physical level. All the groups are inside a wider group which includes all the peers which run the service (the `juxmem` group). Each `cluster` group consists of a set of nodes which provide memory for data storage (called *providers*). In each `cluster` group, a node manages the memory made available by the providers of the group (the *cluster manager*). Any node (including providers and cluster managers) can use the service to allocate, read or write to data as a *client*. All providers which host copies of the same data block make up a `data` group, to which is associated an ID. To read/write a data block, clients only need to specify this ID: the platform transparently locates the corresponding data block. Consistency of replicated blocks is also handled transparently (according to the sequential consistency model, in the current version). In order to tolerate the volatility of peers, a dynamic monitoring of the number of copies of data block is used and new copies are created when necessary, in order to maintain a given redundancy degree. Cluster manager roles are also replicated, to enhance cluster availability.

4.2. Preliminary evaluation

The JUXMEM prototype has been built using the JXTA [34] generic peer-to-peer framework, which provides basic building blocks for user-defined peer-to-peer services. For our preliminary experiments, we used a cluster of 450 MHz Pentium II nodes with 256 MB RAM, interconnected by a 100 Mb/s FastEthernet network. As a first evaluation, we have measured the influence of the volatility degree of providers on the duration of a sequence `lock-put-unlock` executed in a loop by a client. During the execution of this loop, a random provider hosting a copy of the data (out of a fixed number of providers hosting each one a copy) is killed every δ seconds, where δ is a parameter of the experiment. When the system detects these events it transparently triggers the dynamic creation of new copies

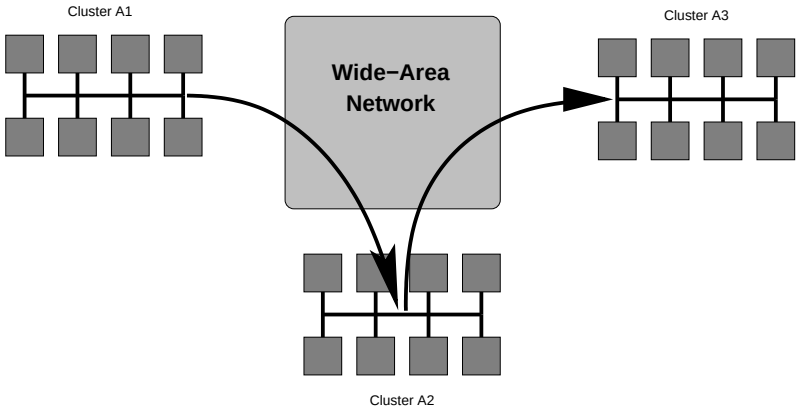

Figure 1. Numerical simulation for weather forecast using a pipeline communication scheme with 3 clusters.

of the data block on the remaining providers (out of the 16 providers at the beginning of the test) that do not already host one copy of the data block, in order to maintain a given redundancy degree. For realistic situations (e.g. $\delta \gg 80$ s), the reconfiguration overhead is less than 5%. Availability is thus enhanced despite node failures, *without significant overhead*.

5. Conclusion

We introduce the concept of *data sharing service* for grid computing, as a compromise between DSM systems and P2P systems. We show that such a system addresses an architecture with *intermediate* features between those of DSM and P2P systems. The implementation of a JXTA-based JUXMEM prototype has shown the feasibility of such a system. We plan to use JUXMEM as an experimental platform for various data consistency models protocols supporting peer volatility. We also plan to enable the platform to use high-performance networks (such as Myrinet or SCI) for data transfer. The final goal is to integrate this service into a large-scale computing environment, such as DIET [31], developed within the ReMaP [39] project. This will allow an extensive evaluation of the service, with realistic codes, using various data access schemes. These issues are currently subject to research within the GDS [32] (*Grid Data Service*, `http://www.irisa.fr/GDS/`) project, which gathers together the PARIS [37], ReMaP and REGAL [38] Research Groups of INRIA. GDS is a project of the ACI MD joint action (*Action Concertée Incitative Masses de Données*) supported the French Ministery of Research, INRIA and CNRS.

References

[1] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, and S. Tuecke. *GridFTP Protocol Specification*. GGF GridFTP Working Group Document, Sept. 2002.

[2] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. TreadMarks: Shared memory computing on networks of workstations. *IEEE Computer*, 29(2):18–28, Feb. 1996.

[3] G. Antoniu, L. Bougé, and M. Jan. JuxMem: An adpative supportive platform for data sharing on the grid. In *The 12th International Conference on Parallel Architectures and Compilation Techniques (PACT2003)*, New Orleans, Louisiana, Sept. 2003. To appear.

[4] B. N. Bershad, M. J. Zekauskas, and W. A. Sawdon. The Midway distributed shared memory system. In *Proceedings of the 38th IEEE International Computer Conference (COMPCON Spring'93)*, pages 528–537, Los Alamitos, CA, Feb. 1993.

[5] J. B. Carter, J. K. Bennett, and W. Zwaenepoel. Implementation and performance of Munin. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles (SOSP)*, pages 152–164, Pacific Grove, CA, Oct. 1991.

[6] F. Dabek, F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 202–215, Chateau Lake Louise, Banff, Alberta, Canada, Oct. 2001.

[7] F. Dabek, B. Zhao, P. Druschel, and I. Stoica. Towards a common API for structured peer-to-peer overlays. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, Feb. 2003. Springer.

[8] A. Datta, M. Hauswirth, and K. Aberer. Updates in highly unreliable, replicated peer-to-peer systems. In *23rd International Conference on Distributed Computing Systems (ICDCS 2003)*, pages 76–87, Providence, Rhode Island, USA, May 2003.

[9] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Supercomputer Applications*, 15(3):200–222, Mar. 2001.

[10] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. In *17th International Symposium Computer Architecture (ISCA 1990)*, pages 15–26, Seattle, WA, June 1990.

[11] L. Iftode, J. P. Singh, and K. Li. Scope consistency: A bridge between release consistency and entry consistency. In *Proceedings of the 8th ACM Annual Symposium on Parallel Algorithms and Architectures (SPAA'96)*, pages 277–287, Padova, Italy, June 1996.

[12] F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal hash table. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, Feb. 2003. Springer.

[13] A.-M. Kermarrec, G. Cabillic, A. Geffhut, C. Morin, and I. Puaut. A recoverable distributed shared memory integrating coherence and recoverability. In *The 25th International Symposium on Fault-Tolerant Computing Systems (FTCS-25)*, pages 289–298, Pasadena, California, June 1995.

[14] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An architecture for global-scale persistent storage. In *Proceedings of the 9th International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS 2000)*, number 2218 in Lecture Notes in Computer Science, pages 190–201, Cambridge, MA, Nov. 2000. Springer.

[15] K. Li and P. Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 7(4):321–359, Nov. 1989.

[16] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical Report HPL-2002-57, HP Labs, Mar. 2002. `http://www.hpl.hp.com/techreports/2002/HPL-2002-57.pdf`.

[17] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen. Ivy: A read/write peer-to-peer fi le system. In *5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, Boston, MA, Dec. 2002.

[18] A. Oram. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.

[19] A. Oram. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, chapter Gnutella, pages 94–122. O'Reilly, May 2001.

[20] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '97)*, pages 311–320, Newport, Rhode Island, June 1997.

[21] J. Protic, M. Tomasevic, and V. Milutinovic. Distributed shared memory: concepts and systems. *IEEE Paralel and Distributed Technology*, pages 63–79, 1996.

[22] J. Protić, M. Tomasević, and V. Milutinović. *Distributed Shared Memory: Concepts and Systems*. IEEE, Aug. 1997.

[23] S. Ratnasamy, S. Shenker, and I. Stoica. Routing algorithms for DHTs: Some open questions. In *1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, number 2429 in Lecture Notes in Computer Science, pages 45–52, Cambridge, MA, Mar. 2002. Springer.

[24] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: the oceanstore prototype. In *2nd USENIX Conference on File and Storage Technologies (FAST '03)*, Californie, CA, USA, Mar. 2003.

[25] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware 2001)*, pages 329–350, Heidelberg, Germany, Nov. 2001.

[26] A. I. T. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 188–201, Chateau Lake Louise, Banff, Alberta, Canada, Oct. 2001.

[27] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM 2001 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM 2001)*, pages 149–160, San Diego, CA, Aug. 2001.

[28] F. Sultan, T. Nguyen, and L. Iftode. Scalable fault-tolerant distributed shared memory. In *The IEEE/ACM SuperComputing conference (SC'00)*, pages 54–55, Dallas, Texas, Nov. 2000.

[29] B. Y. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, Apr. 2001.

[30] Y. Zhou, L. Iftode, and K. Li. Performance evaluation of two home-based lazy release consistency protocols for shared memory virtual memory systems. In *Proceedings of the 2nd Symposium on Operating Systems Design and Implementation (OSDI'96)*, pages 75–88, Seattle, WA, Oct. 1996.

[31] DIET. `http://graal.ens-lyon.fr/~diet/`.

[32] The GDS Project. `http://www.irisa.fr/GDS/`.

[33] JuxMem. `http://www.irisa.fr/paris/Juxmem/welcome.htm`.

[34] The JXTA project. `http://www.jxta.org/`.

[35] KaZaA. `http://www.kazaa.com/`.

[36] Napster protocol specifi cation. `http://opennap.sourceforge.net/napster.txt`, Mar. 2001.

[37] The PARIS Research Group. `http://www.irisa.fr/paris/General/welcome.htm`.

[38] The REGAL Research Group. `http://www.inria.fr/recherche/equipes/regal.en.html`.

[39] The ReMaP Research Group. `http://www.inria.fr/recherche/equipes/remap.en.html`.