Merging G-Grid P2P Systems While Preserving Their Autonomy

Gianluca Moro[1], Gabriele Monti[1], and Aris M. Ouksel[2]

[1] Department of Electronics, Computer Science and Systems
DEIS, University of Bologna
Via Venezia, 52, I-47023 Cesena (FC), Italy
gmoro@deis.unibo.it gmonti@deis.unibo.it

[2] Department of Information and Decision Sciences
University of Illinois at Chicago
2402 University Hall, 601 South Morgan Street
M/C 294 Chicago, IL 60607-7124, USA
aris@uic.edu

Abstract. Peer-to-Peer (P2P) systems share and manage huge amounts of data and resources distributed across a large number of machines. In pure P2P environments the control is totally decentralized in peers, which are autonomous, perform the same roles and do not have a global view of the entire system. Several P2P solutions have been deployed and proposed in literature managing data within a single P2P system with a variety of different features and performances. But in all cases a single P2P system conceptually corresponds to a single database containing a unique relational table and all functions, such as querying, routing and content location, are defined to properly work just within such a single system. Existing solutions cannot support several databases, which originate and grow separately, or simply just one database with several tables, because it would be necessary many distinct P2P systems. In this paper we present a solution to merge G-Grid P2P systems, which are multi-dimensional distributed structures for P2P data management, preserving their own autonomy and identity. Thanks to this users and applications can view the merged systems both as a single environment and also as a set of distinct systems. As P2P systems merge, all their functions, such as complete and partial range querying, content routing and so on, adapt naturally to efficiently work in the multi-grid P2P system.

1 Introduction

Though the P2P developments are still at the beginning we can already distinguish between two P2P system generations [1]. The first one regards the unstructured deployed P2P systems, like Gnutella [2, 3] and its descendants, in which the contents is not organized and the message flooding is the only way

[2] Research partially supported by NSF grant IIS-0326284

to locate data. In this kind of systems it is not guaranteed to find searched data, in fact they have to limit the number of hops since search costs, in term of messages, are exponential. The second generation is formed by proposals and prototypes which distribute data, in particular data record keys, across a P2P network according to predefined logical structure, some examples are Chord [4], CAN [5], Pastry [6], P-Grid [7], PeerDB [8]. In all these solutions the number of messages/hops to locate a record key is in general logarithmic in the number of peers.

They distinguish by employing different data structure, network topologies, levels of robustness and so on, but all of them support a single P2P system corresponding conceptually to a database containing a unique relational table with only one search index[3]; all functions, such as querying, routing and content location, are defined to properly work just within such a system. In particular, P2P data searches occur thanks to that index and each query must always involve all the indexed attributes. In other words, except CAN, the mentioned above solutions cannot support arbitrary partial range queries just because are based on mono-dimensional data structure.

A second limitation, on which this paper is mainly focused, regards the impossibility to support several databases originated and grown separately with distinct P2P systems, which at a given moment must start working together as an unique system.

In this paper we present an original solution to merge G-Grid P2P systems [9], which are multi-dimensional distributed data structure for data management in a P2P system, preserving their own autonomy and identity. Thanks to the preservation of the system identities users and applications can view a single resulting environment and/or an environment of distinct systems. Moreover, preserving the autonomy of merged systems means that each must be able to work singularly as it did before merging, namely its functioning must not be compromised by failures or changes in other systems. As distinct P2P systems merge all their functions, such as complete and partial range queries, content routing and so on, adapt naturally to efficiently work in the resulting multi-grid P2P system.

The merging operation is useful to self-integrate different P2P systems (i.e. databases) within an organization, but also among distinct organizations managing data with their own P2P systems. For instance, in business-to-business e-commerce companies need to integrate parts of their information systems preserving their identity and autonomy; the same holds also among scientific institutions interested in sharing some data, such as hospitals and research centers. For instance, preserving the system identities allows users and applications to decide if some operations, such as some queries, must be restricted within a single hospital or a subset of them.

We highlight that the merging it is important also in case of failures which partition logically the P2P network originating several isolated systems. In fact, the merging can restore the network integrity by self-reintegrating the com-

[3] In most systems the index can be defined by only one attribute

ponents. In this paper we study the basic merging operation for P2P systems managing databases with the same data schema.

Section 2 introduces some basic G-Grid principles. Section 3 and 4 present the merging algorithm and the routing mechanism. Section 5 reports some theoretical results about the completeness of the merging, efficacy, efficiency and costs, including two examples of merging scenarios. Section 6 discusses related works and Section 7 concludes the paper.

2 G-Grid Distributed Data Structure: Basic Principles

G-Grid [9] is a grid file data structure and algorithms for data management in a dynamic P2P environment where autonomous machines, connecting/disconnecting arbitrarily, have only a limited view of the entire resulting system. G-Grid is a novel structure developed from preceding works on multi-dimensional data structure for centralized systems [10, 11], which extends another novel P2P structure designed for cluster P2P computing called the Generalized Grid File (GGF) [12]. A survey on centralized multi-dimensional data structures is in [13].

G-Grid operates in a d-dimensional space where each record is represented as a point with d coordinates in such a space. The point coordinates correspond to the record key attributes on which searches are performed. Each key attribute is bounded into its own domain range [min,max], but without loss of generality all of them can always be scaled in the same interval, for instance [0,1]. In this way any record can be seen as a point of the hypercube $U^d = [0,1]^d$ which forms the whole data space.

As points fill the space it is split into hyper-rectangular sub-volumes of U^d called regions, which store a maximum number b of records (bucket size); it is also contemplated a minimum number of records, namely $\frac{b}{3}$. Each peer manages one or more of these regions. At the beginning a single region covers the whole space U^d (see the region (0,0) in Figure 1); as record are inserted the bucket size is reached and the region is split into half along one of the d dimensions. The split generates temporarily two nested regions, but then the one with more records is collapsed into its parent region. With reference to Figure 1, the region (0,0) has been split in (0,1) and (1,1), and then (0,1) has been collapsed in (0,0).

Of course, if the two generated regions do not satisfy the min or max bucket constrains the split continues until it finds a more nested solution[4]. This split mechanism has been introduced in [11] and subsequently reutilized in GGF. Other researchers have also adopted it, but for the single dimensional case, as for example in [7]. The split mechanism in [11] did not need to collapse the region with more records in its father region because they all remained always in the same centralized machine.

In general all the algorithms of the structure in [11], which was designed for a centralized system, assume both a centralized control on the structure and the existence of a global knowledge about the data structure itself. For instance

[4] the splitting mechanism always converges to a solution because the overflowing region is recursively partitioned into sub-regions geometrically smaller.

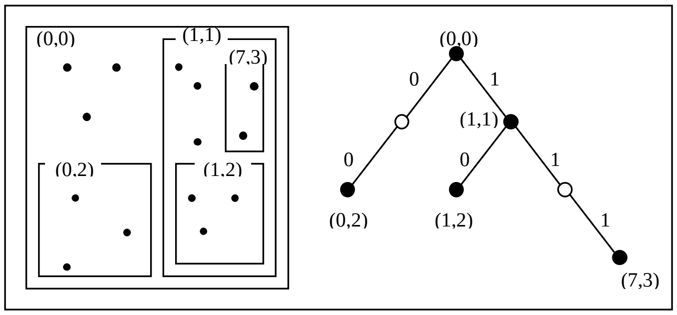

Fig. 1. Example of a 2-dimensional space partitioning and its correspondent tree

they make use of global variables to store several critical information about the topology of the data structure itself and when the structure topology changes these information there can be immediately updated. This corresponds to say that the system assumes to know completely and perfectly all its parts.

Viceversa in a dynamic distributed environment, like a pure P2P networks of autonomous participants, no central coordination can be assumed, no global or complete knowledge can be kept. This means that all algorithms to manage and querying the structure in P2P networks are different. In fact they have to take into account several issues in addition: for instance unreliability of peers, network costs and contents location, which, among other things, changes continuously due to unpredictable connections and disconnections of peers.

Moreover, in P2P environment the entire split procedure must be executed locally within the peer managing the overflowing region in order to avoid any network communication. Net traffic is only generated when some of the records are moved to another peer, which becomes responsible for the new nested region.

As depicted in Figure 1 the multi-dimensional space can be mapped into a binary search tree, where each node represents a region and each link (labeled with a bit zero or one) is a relationship between a region and its nested child region. The identifier of each region, which is directly achieved by the tree, is composed by a pair of integers (π, l) in which π is the decimal conversion of the juxtaposition of bits along the path in the binary tree to get to the region, while l indicates how many times the space U^d has been split to obtain that region (see Figure 1).

By comparing the binary conversion of π it is possible to determine if a region r is an ancestor of another region r'. For instance the region (1,1) in Figure 1, which corresponds to 1, is an ancestor for (1,2), which is 01, because they share the post-fix 1. Differently (0,2), which corresponds to 00, is not an ancestor for (7,3), which corresponds to 111, because they don't share any post-fix.

G-Grid gives rise to data structures which preserve the two following properties:

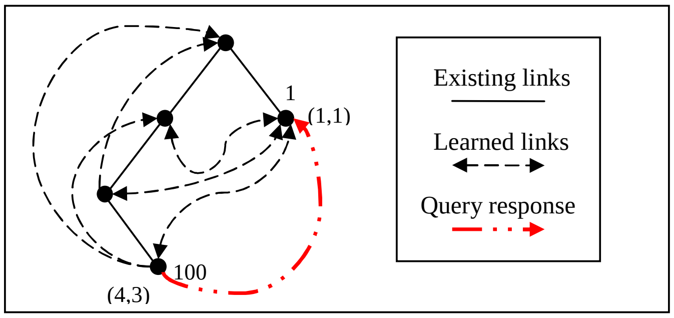

Fig. 2. Learning of links due to a query issued by the peer of region (1,1) for a record in the peer of region (4,3).

- *Spatial property.* Regions in the same level of the tree do not intersect each other;
- *Coverage property.* Each region is always nested in another one, its parent or in some ancestor, except for the root region which is nested in itself.

Each G-Grid peer maintains a local routing table containing couples [region,peer], namely links towards its parent/ancestor peer (i.e. parent region) and its children/descendant peers (i.e. nested regions). The routing table of a peer contains also all peers that it discovers by querying and inserting data in the structure.

In G-Grid, operations like queries, insertions, deletions and updates occur in two steps. Here for space reasons we refer to exact match queries and range queries.

At the first step the peer calculates locally the region identifier π of the potential smallest region containing the searched records. This occurs according to any typical function which maps a multi-dimensional space to a straight line, namely to a mono-dimensional space. For an exact match query the function takes in input all the d attribute values of the searched record, which represent its coordinates, and returns the corresponding π. A range query is resolved simply calculating in the same way the corners of the hyper-rectangle determined by the intervals specified by the query; the region that contains all the corners is the target one and the records inside the hyper-rectangle represent the result set. The target region is determined locally by simply taking from all the corners the maximum common post-fix among them. At the second step p sends a query message to the peer that manages the target region, if p knows it, otherwise to the closest relative of the target peer that p knows. In the worst case the query follows a path along the tree by traversing parent-child links (see Figure 1), so the cost is at most logarithmic.

However the worst case is almost unrealistic, in fact each peer learns new couples [region,peer] which adds to its knowledge, namely its own routing table, on the basis of the following self-learning capability. Each peer traversed by a query, or by any other operation, adds to the message its reference (i.e. the couple [region,peer[5]]); of course the first reference appended is the one of the peer issuing the query.

If the path followed by the query is $p_1, ., p_h, .., p_t$, where p_1 and p_t are the issuing peer and the target one respectively, the number of hops is $H = t - 1$ and the new couples/links globally learnt by the peers in the path are $\binom{H}{2}$. This allows peers to increase their knowledge about the topology of the overall structure guaranteeing better search costs. In a dynamic situation, where the structure and the number of peers grow, analyses show [9] that any record is located on average with less than 2 hops if the rate $\frac{insertions}{queries}$ ranges like $\Theta(N^{-2})$ (N is the number of peers); for greater rate the average no. of hops grows logarithmically. The routing remains efficient even if each peer has millions of references[6] in its routing table, in fact it can be maintained in RAM and indexed with an usual tree structure that guarantees logarithmic search costs. The learning capability is another new feature not present in [11].

3 Self-Merging of Distinct G-Grid P2P Systems

In G-Grid when any peer connected to a physical network cannot contact any existing P2P systems to which join in, it sets itself as the root of the multi-dimensional structure. Local data of the peer are represented as points in its root region. The impossibility to contact any existing P2P system is high in the Internet because peers connect/disconnect arbitrarily changing also their IP address, and in any case the bootstrap of a system must be able to deal with this scenario.

In other situations it is reasonable that distinct P2P systems may have grown up intentionally separated, for instance internally to an organization with several departments or branches, like it happens with several autonomous databases.

The merging algorithm presented in this paper works in both cases as follows. Let us denote two existing structures G' and G'' with roots r' and r'' respectively. The two structures can be identified by using the IP address of r' and r'' together with their local time-stamp generated when they created the respective structures. The aim of this identifier is to distinguish the links stored in the routing table of every peer in such a way that each one is aware of links towards other P2P systems. This preserve the identity of the two P2P systems. When two peers p' and p'' meet they straight away realize if they belong to the same structure or not, in this last case they start merging.

The basic idea is to allow p' and p'' to exchange part of their knowledge (see Fig. 3), namely part of their routing tables. Then p' and p'' can propagate the merge by contacting other peers of G' and G'' according to the new knowledge just received, and in turn they can contact other peers and so on. The data in the

[5] for instance each peer can simply be addressed by its IP
[6] each reference is a couple [π,IP] of 8 bytes

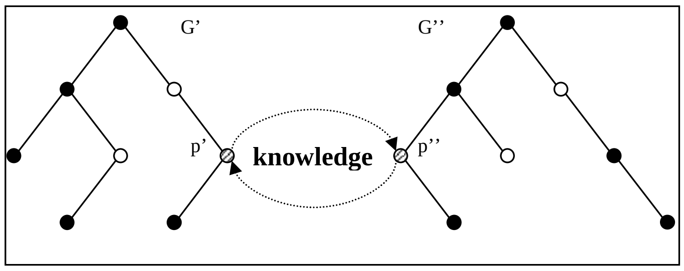

Fig. 3. First step of the merging algorithm.

two structures are not moved, moreover pre-existing links within each structure remain unchanged. This allows each system to keep its own autonomy, namely they can continue to work independently on each other even after the merging.

The two systems can regulate the grade/speed of their merge and hence the costs of their integration by simply limiting the number of propagations between them. This is useful to manage different situations with different requirements, from the case in which network costs have priority over the speed of the merge, such as in the Internet, to the opposite case of the full integration of systems in the shortest possible time, such as within an enterprise network.

As depicted in Figure 3 in the first step the peers p' and p'' will send each other a message containing a certain number L of links (of the kind $[p, (\pi, l)]$) taken randomly from their own routing table. Choosing them randomly we avoid to overload some parts of the tree, especially during the first moments after the merging when the number of links between the grids are at the lowest level.

At the second step (see Figure 4a) p' and p'' randomly choose, from their routing table, the I peers to which the received message will be forwarded. Once that the messages are delivered, the informed peers update their routing table and the structures will appear like in Figure 4b.

4 Routing and Querying Algorithms in Merged Systems

Basically the merging of two structures does not entail changes in the splitting procedure. The only difference involves the routing, in fact every time that a crossed peer p_a has links to another grid(s), apart from the one it belongs to, it will forward the message also to them. Obviously this will happen only if the message has not been already forwarded by another peer before in the path followed by the message. This is possible because when a peer forward a message to another grid it appends the identifier of that grid and the pair [peer, region] identifying the recipient node.

Once that more grids exist, if a new record must be inserted, it is possible to decide to store it only in one or in all of them. The second option has higher computational costs but improve the system robustness as creates as many copies

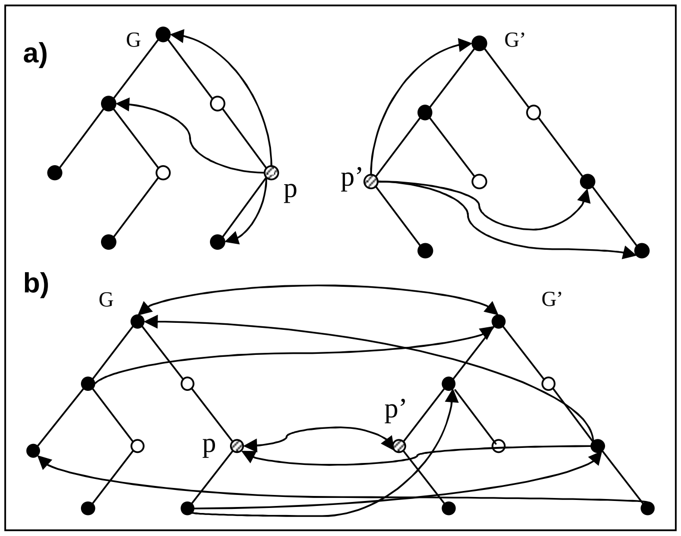

Fig. 4. a) Second step of the algorithm, b) new links between G and G' thanks to the merging algorithm

as the number of the existing grids. In case of data redundancy the consistency must be guaranteed by updating all the copies using the support of transaction.

5 Merging Efficacy and Efficiency: Theoretical Analysis

To measure the grade (i.e. the efficacy) of merging of two structures the approach we use is to calculate the probability that a generic operation (query, insertion, etc.) issued in one of them reaches the other one. Considering the structures G' and G'', right after the merging there are I peers belonging to G' that know/link at least one peer of G'' and vice versa; henceforth they are called *aware peers*. Supposing that the average number of hops necessary to perform the operation in G' is equal to H then $H+1$ peers are crossed by the message and the probability to explore G'' starting from G' is:

$$P(G' \rightarrow G'') = 1 - (1 - \frac{I}{N})^{H+1} \qquad (1)$$

where N is the number of peers and $\frac{I}{N}$ is the rate of aware peers in G'.The idea in the equation (1) is to compute the inverse probability, that is 1 minus the probability of $H+1$ unsuccessful tries.

From the (1), knowing the number of peers N, the average number of hops H and the desired probability $P(G' \rightarrow G'')$ of immediate merging, it is possible to calculate the number I of peers to which the merging message must be forwarded.

$$I = N(1 - \sqrt[H+1]{1 - P(G' \rightarrow G'')}) \qquad (2)$$

It may happen that p' does not know enough peers to reach that result. In this case it is possible to propagate to other peers the merging just described allowing them to propagate it in cascade for a given number of levels/times. This number must be the smallest necessary to achieve I aware peers and can ben computed in advance.

To measure how fast (i.e. merging efficiency) the knowledge spreads through peers and completes the merging, we have defined the function ΔI below which estimates the expected increase of aware peers of G' after executing any operation:

$$\Delta I = \sum_{h=1}^{H} h \cdot P(h) \tag{3}$$

Where $P(h)$ is defined as the probability that h unaware peers are crossed during the query, insertion, etc. It is obtained using a binomial distribution:

$$P(h) = \binom{H+1}{h}(1 - \frac{I}{N})^h (\frac{I}{N})^{H-h+1} \tag{4}$$

Where $\frac{I}{N}$ is the rate of aware peers in G'. The rationale of the formula (3) is that given an average number of hops H, the number of unaware peers crossed may vary between 0 and $H+1$. Obviously these two extremes are not considered when ΔI (h goes from 1 to H) is evaluated since no unaware peers are crossed. For the other cases, the probability of their occurrence depends on the number h of informed peers as shown in (4).

5.1 Performance Evaluations in Some Significative Scenarios

Theoretical analysis of the G-Grid behavior showed that in several realistic scenarios the structures presents on average features typical of regular graphs. Hence the average number of hops H from the number of peers N is approximately as follows:

$$H = log_T N \tag{5}$$

where $T = \frac{E}{N}$ is the number of links on average per peer while $E = N^{\frac{H+1}{H}}$ is the total number of links.

Now we are able to define different scenario and for each of them to evaluate the performances of our merge algorithm and to estimate how quickly the merging gets complete We can also evaluate whether T is big enough for the network to achieve the desired $P(G' \rightarrow G'')$ and if the merging is propagated to other peers. In the two examples below we consider two grids/systems having both $N = 5000$ and different values for H or T. The difference between them is that the first one is a scenario where each peer has a lower number of neighbors and an higher average number of hops. Then we want to obtain in both examples a $P(G' \rightarrow G'') = 0.5$.

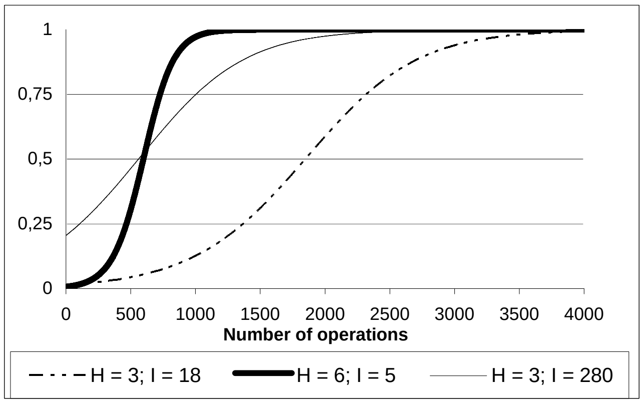

Fig. 5. Growth of the merge probability by executing standard operations on G' with $b = 3$ and $\frac{insertions}{queries} = 0.5$

- *Example 1:* $T = 4$. Supposing we limit the maximum number of links in the routing table to 4 per grid we obtain $H = log_4(5000) \approx 6$ and a required I:

$$I = 5000(1 - \sqrt[7]{1 - 0.5}) = 471$$

  In this situation it is not necessary to use the propagation to contact that number of peers since Figure 5 (thickest curve) shows that $P(G' \rightarrow G'')$ grows very quickly even starting with only 5 aware peers (4 plus p' originating the merging). In fact $P(G' \rightarrow G'') = 0.5$ with little more than 500 operations, which is very close to 471, namely the minimum number of required operations.
- *Example 2:* $H = 3$, this means that $T = 17$ while the required number of links I is 795. The dotted curve of Figure 5, which is the growth of the merging without the propagation, shows that $P(G' \rightarrow G'')$=0.5 after about 2000 operations. This results can be improved by propagating the merging as shown by the thin curve of Figure 5, in fact with only 2 levels of propagation $P(G' \rightarrow G'')$=0.5 after about 500 operations. The additional cost of the propagation in terms of messages is $17 + (17 \cdot 16) = 289$ and the total cost in term of operations is $\frac{289}{3} + 500 \ll 2000$. [7]

In general the features of G-Grid helps this kind of merging whatever is the structures topology. In fact, the more H is low, the more T is on average high and consequently the initial $P(G' \rightarrow G'')$ is high; vice versa if H is high, and therefore T is low, the merging of the two structures can grow rapidly by simply exploiting usual operations.

[7] division by 3 because H=3 means that each operation costs on average 3 messages

6 Related Works

The issue of merging different structures seems not having been considered yet, or at least it has not been treated explicitly. In fact the most significative works, like CAN [5], Chord [4], P-Grid [7], Pastry [6], PeerDB [8] and the k-dimensional search algorithm proposed in [14], do not deal with this feature. However it is not so unlikely that different peers give rise to more than one structure, especially in an unstable scenario like Internet and also in case of failures partitioning the net. Hence, merging can be also seen as a procedure to restore the network integrity. In general, the world is made of autonomous systems which evolves and successively need to be integrated/merged, as it happens for instance with database systems.

To compare our approach with existing proposals let us evaluate how they could merge two distinct P2P systems. In all of them it would be necessary (i) to elect one system as the resulting one and (ii) to move each peer from the second to the first system. We highlight that this implies to lose the autonomy and identify of each system involved in the merging. In any case the number of messages is as follows: each disconnecting peer must release its K keys to a neighbor peer in its home system; moreover when it enters in the new system it receives new K keys from some other peer and transmits its own record keys, which we assume for simplicity equals to K. Each insertions costs as a search, namely $LogN$ with N the number of peers. In short, to compare the result to our initial message exchange let us assume to move $\frac{N}{2}$ peers into a system with already N peers, then the cost in term of messages is the following:

$$C(N,K) = \frac{2 * N * K}{2} + K \cdot \sum_{i=N}^{N+\frac{N}{2}} Log\ i \approx \Theta(K(N + Log\ N!)) \qquad (6)$$

This value is much higher than the one required by the solution proposed in this work which is always lower than N and no record must be moved. What is also different is that in our solution, after this initial low cost to start the merge, then the merge increases spontaneously by exploiting the normal execution of operations such queries, insertions etc.

7 Conclusions

In this paper we introduced a solution for merging G-Grid P2P Systems, which are multi-dimensional data structure for dynamic P2P environments, preserving their own autonomy and identity, and giving to users and applications the view of a unique resulting system.

The main properties of a G-Grid structure, such as the capability to execute both complete and partial range queries, the self-organization of the structure which emerges from local P2P interactions, holds also in the resulting multi-grid system.

Concerning the scalability, the performances of merged systems does not deteriorate as the number of peer increases. The merging operation has been designed in such a way that all mechanisms, such as content routing, querying and data manipulation, adapt naturally as distinct P2P systems meet each other.

Merging only occurs at a logical level by creating links among distinct structures and once that some links are generated, peers incrementally improve their knowledge completing the merge spontaneously by means of usual operations.

The solution its simple and the network traffic generated can be adapted to the available bandwidth and to the desired merging rapidity.

References

1. Moro, G., Ouksel, A.M., Sartori, C.: Agents and peer-to-peer computing: a promising combination of paradigms. In: Proceedings of the 1st International Workshop on Agents and Peer-to-Peer Computing, Bologna, Italy, July 2002. Volume 2530., Springer (2003) 1–14
2. Jovanovic, M.A., Annexstein, F.S., Berman, K.A.: Scalability issues in large peer-to-peer networks - a case study of gnutella. Technical Report Technical Report, University of Cincinnati (2001)
3. Kan, G.: 8. In: Peer-to-Peer: Harnessing the Benefits of Disruptive Technologies. O'Reilly & Associates (2001) 94–122
4. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM, ACM Press (2001) 149–160
5. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. In: Proceedings of ACM SIGCOMM 2001. (2001) 161–172
6. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. LNCS **2218** (2001) 329–340
7. Aberer, K., Cudr-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., Schmidt, R.: P-grid: A self-organizing structured p2p system. SIGMOD Record **2** (2003)
8. Ng, W.S., Ooi, B.C., Tan, K.L., Zhou, A.Y.: PeerDB: A P2P-based System for Distributed Data Sharing. In: Proceedings of ICDE. (2003) 633–644
9. Ouksel, A.M., Moro, G.: G-Grid: A class of scalable and self-organizing data structures for multi-dimensional querying and content routing in p2p networks. In: Proceedings of the 2nd Internat. Workshop on Agents and Peer-to-Peer Computing, Melbourne, Australia, July 2003. Volume 2872., Springer (2004) 123–137
10. Ouksel, A.M.: The interpolation-based grid file. In: Proceedings of the ACM Symposium on Principles Of Data Base Systems, ACM (1985) 20–27
11. Ouksel, A.M., Mayer, O.: A robust and efficient spatial data structure: The nested interpolation-based grid file. Acta Informatica **29** (1992) 335–373
12. Ouksel, A.M., Moro, G., Litwin, W.: GGF: A Generalized Grid File for Distributed Environments. Technical report, DEIS Univ. of Bologna, Univ. of Illinois at Chicago (2002)
13. Gaede, V., Günther, O.: Multidimensional access methods. ACM Comput. Surv. **30** (1998) 170–231
14. Pasquale, A.D., Nardelli, E.: Distributed searching of k-dimensional data with almost constant cost. In: Proceedings of 4th East European Conference on Advances in Databases and Information Systems. Volume 1884., ADBIS (2000) 239–250