G-Grid: A Class of Scalable and Self-Organizing Data Structures for Multi-dimensional Querying and Content Routing in P2P Networks

Aris M. Ouksel[1]** and Gianluca Moro[2]

[1] 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

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

Abstract. Peer-to-Peer (P2P) technologies promise to provide efficient distribution, sharing and management of resources, such as storage, processing, routing and other sundry service capabilities, over autonomous and heterogeneous peers. Yet, most current P2P systems only support rudimentary query and content routing over a single data attribute, such as the file-sharing applications popularized in Napster, Gnutella and so forth. Full-fledged applications in distributed data management and grid computing demand more complex functionality, including querying and content routing over multiple attributes. In this paper, we present a class of scalable and self-organizing multi-dimensional distributed data structures able to efficiently perform range queries in totally decentralized dynamic P2P environments. These structures are not imposed a priori over the network of peers. Rather, they emerge from the independent interactions of autonomous peers. They are also adaptive to unanticipated changes in the network topology. This robustness property expands their range of usefulness to many application areas such as mobile ad-hoc networks.

1 Introduction

Peer-to-Peer (P2P) networks are emerging as a new computing paradigm for locating and managing contents distributed over a large number of autonomous peers. Autonomy implies that peers are not subject to central coordination. Each peer plays at least three roles, either as (i) a server of data and services, (ii) a client of data and services; and/or (iii) a router for network messages. P2P systems offer the prospect of realizing several desirable properties of emergent systems, including self-organization, which provides the ability to self-administer,

** Research partially supported by NSF grant IIS-0326284

[2] Technical Report no. DEIS-LIA-002-04, February 2004, Univ. of Bologna, Univ. of Illinois at Chicago

scalability, which enables support large number of users and resources without performance degradation, and robustness, which makes the system fault-tolerant in the event of peer failure or a peer leaving the distributed system [1].

Content in P2P systems can be conceptually represented as a single relational table, with multiple data attributes, horizontally partitioned among peers. Just as in distributed databases, the location of data partitions is transparent to users. Many popular and currently-deployed P2P systems actually follow this model in organizing shared files on the web. Some of the most well-known are Gnutella [2, 3] and its descendants such as Kazaa, Morpheous, WinMX, Emule. This first generation of P2P systems provided a valued service for many users, but they suffered from several drawbacks including, the inefficiency of the routing mechanism and the poor query expressibility. Routing is based on message flooding, with a likely implication of increasing network congestion in data-intensive applications. Queries are limited to single-attribute lookup operations, restricting thus the range of possible of applications in a P2P environment. These problems along with the demands of web users spurred research for alternative structures. Building on previous work in uni-dimensional distributed data structures, such as RP*, LH* [4, 5], DRT [6], [7], several new approaches were proposed. Among the most noteworthy are Chord [8], tapestry [9], Pastry [10], P-Grid [11], PeerDB [12]. These new systems did indeed improve performance and extended the flexibility of search by allowing querying by content. However, their limitation to single-attribute queries have continue to stymie efforts to expand the range of applications within acceptable performance results.

Many P2P applications demand richer query semantics over several attributes, comparable to those available in centralized relational DBMSs. Consider for example distributed data mining in a health application. In several practical scenarios, the data is likely to be distributed across a large number of hospitals, where each hospital can be viewed as an autonomous peer. The discovery of patient clusters with similar characteristics, such as gravity of disease, age, residence, sex and so on, requires executing partial range queries on multiple-attribute data distributed across several hospitals. Processing such queries would involve running a combination of individual single attribute queries, followed by an intersection step to filters out unqualified patients. Clearly, this operation will be cost-prohibitive in a distributed environment. This is only one example. Many similar applications will require efficient solutions to complex distributed range queries.

Multi-dimensional structures have been extensively investigated over the last 20 years where the main goal is to support efficiently complex range queries over multiple attributes. A literature survey in this area as well as two specific structures, IBGF and NIBGF, can be found in [13, 14]. These structures have been designed for environments where both control and data are centralized, and significant performance improvements have been achieved for both partial and complete range queries. Yet, their adoption in commercial relational DBMSs has remained very limited. We believe distributed environments may actually provide more compelling justifications for their acceptance. While, in centralized

systems, range queries over a set of attributes may be processed using single-attribute structures with acceptable performance, despite the large number of local accesses to disk, the same queries in P2P systems will be singularly cost-prohibitive without a multi-dimensional structure, as each local data access will now give rise to several network messages. Dynamic pure P2P networks will naturally amplify the severity of the costs because of continuous changes in the content, its distribution, and the underlying network topology.

Because of autonomy of peers, P2P networks are analogous to complex dynamic organisms in their behavior. For example, local changes in the molecular structure of a chemical compound may aggregate to yield altogether a new compound. A global property emerges generally from a series of simple local interactions. The same phenomenon may also occur in the distribution of content as autonomous peers interact independently with each other in a P2P system. Thus, in addition to scalability, our goal is to seek structures for P2P systems which exhibit emergence and self-organization properties characteristic of complex systems, where local interactions and self-organization of peers lead to a global organizational structure with excellent performance characteristics.

This paper is organized as follows: section 2 introduces the G-Grid structure and its principal features; section 3 illustrates G-Grid at work in P2P environments; section 4 discusses briefly performance issues; section 5 highlights synthetically the robustness, and finally, section 6 summarizes the main ideas presented in this paper.

2 G-Grid Definition

The G-Grid is a distributed multidimensional data structure, which organizes a set of objects across any number of peers in a network. It is a novel structure developed from these works [13–16] related to multidimensional data structures for centralized systems . Each dimension represents one attribute of the objects. For example, location of an object may be one possible attribute. The G-Grid partitions the space of objects, based on the attribute values, into regions and structures these regions into a tree as follows (see the example of Figure 1): a node of the tree represents a region in the multidimensional data space and an edge links two regions, where one, called the child region, is properly nested in the other, called the father region. The root node represents the whole object space. One or more regions are assigned to one peer, and as result one or more nodes of the G-Grid tree structure are associated with a peer.

Formally, consider a relation table $\mathbf{T}$ with attributes A_0, A_1, ..., $A_{(d-1)}$, also called *keys*, taking their values from domains D_0, D_1, ..., $D_{(d-1)}$, respectively. In G-Grid a relation is viewed as a bounded d-dimensional hypercube
$[\min D_0, \max D_0)x...x[\min D_{d-1}, \max D_{d-1})$
where each attribute is represented by one dimension of the space. For the sake of generality, let us assume for now that this hypercube is normalized to the unit cube. In other words, each attribute value is mapped to a rational number in the half-open interval $\boldsymbol{U}$=[0,1) by simple interpolation, assuming the attribute

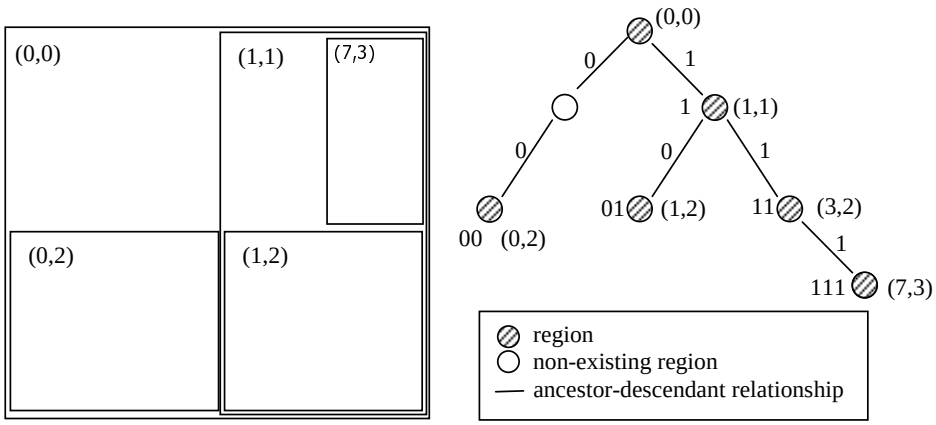

Fig. 1. an example of a partitioned 2-dimensional space and its counterpart tree

domains are bounded but not necessarily finite. In practice, this is not necessary. The entire data space is $\boldsymbol{U}^d$, where each tuple of the relation is attached to a point in $\boldsymbol{U}^d$, which is precisely the point whose coordinates along each dimension represent the scaled values obtained by interpolation from the tuple. Given a record $k = (k_0, k_1, \ldots, k_{(d-1)})$ of a relation R, each record k is represented by a point $k^0 = (k_0^0, k_1^0, \ldots, k_{(d-1)}^0)$ in $\boldsymbol{U}^d$.

The data space $\boldsymbol{U}^d$ is decomposed into subspaces called regions, as in the example of Figure 1. A *region* R is a d-dimensional hyper-rectangular subspace of $\boldsymbol{U}^d$ delineated by the pair $I = (x^{\min}, x^{\max})$, where $x^{\min}$ and $x^{\max}$ are d-dimensional vectors representing the coordinates of the minimum and the maximum points in the subspace, respectively. Let x be a point in $\boldsymbol{U}^{\mathrm{d}}$. Then a region R can be defined formally as: R= $\{x / x^{\min} \leq x < x^{\max}\}$. We refer to I as the identifier of region R. Initially, the data space consists only of one region R=$\boldsymbol{U}^d$ whose identifier is $I = \{x^{min} = (0,0), x^{max} = (1,1)\}$. Any point in $\boldsymbol{U}^d$ can be viewed as the identifier of a region with a measure 0. The records that fall within a region R are stored in a bucket of fixed size B. R is split when the number of records in it exceeds B. The decomposition of R is carried out by carving out a region R' from the overflowing region. In the ideal case, R and R' will split the records in B in half. This split mechanism has been first introduced in [14] and subsequently reutilized in another novel structure, the Generalized Grid File (GGF), designed for cluster P2P computing [17]. Other researchers have also adopted it, but for the single dimensional case, as for example in [11]. In [14], it is shown that this split mechanism guarantees a storage utilization of no less than one-third of the total size of a bucket. As a result, degenerate space partitioning is avoided with positive consequence on the information retrieval capability.

Every region, and therefore any records, can also be identified by a pair of integer number (π, l), where l means the number of times that the starting region

(labelled $(0,0)$) has been split, while the binary conversion of π represents precisely the path from the root to the node representing the region (see Figure 1). Value l represents also the size of the region, i.e., $1/2^l$.

It is important to note that the splitting of any region is totally a local operation, and thus does not require any global information of the space. Looking at the Figure 1 it is easy to understand that l also represents the length of the binary conversion of π. When a new region is generated through a split we will take the π of the direct parent region, calculate the value *bin(π)*, adding in case some "0" on the left to reach the length l, attach a "0" in the beginning of the string for the left child or a "1" for the right one and then convert it again into decimal value. For instance, the left child region of $(1,2)$ will be π = *"0" + bin(1) = "001" = 1* and the right one π = *"1" + bin(1) = "101" = 5*.

This splitting approach leads to regions that are either disjoint or properly contain each other. The binary representing these regions differ in at least one bit, in the case, or one is the prefix of the other in the second. Figure 1 illustrates this relationship between regions. All regions at the same level cover disjoint subspaces of the space, we refer to this as the *spatial property*; whereas regions on the same are contained in each other, we refer to this as the *cover property*. As a consequence, the identifier of the deepest region where a record might be located can be computed prior to starting the search using the current maximum number of splits. If the target region does not exist, then the record will be located in one of its parent along the same path in the binary tree. As a result of the balanced load achieved by the split mechanism, and the fact that search is done along on the same path of the tree, the record search cost is logarithmic in the number of messages between regions. Next Section shows that thanks to a full *learning capability*, which is another fundamental feature we have introduced in G-Grid, search costs are less than logarithmic and can also be constant in some realistic scenarios independently on the number of peers in the overlay network.

3 G-Grid in P2P Environments

The G-Grid tree structure is embedded in the network of peers. Not all peers in a network are necessarily part of the G-Grid. Thus an edge in the tree structure may actually correspond to a path in the network.

The network consists of two kinds of peers: (i) s-peers, or structure-peers, are those that do manage at least one region of the G-Grid; (ii) c-peers, or client-peers, are those that do not manage any region. Both s-peers and c-peers may issue operations (object search requests, insertions, and possibly deletions) to other s-peers, and in addition s-peers provide routing tables to the operations. Initially the G-Grid may consist of only one s-peer. In the process of performing search operations, peers learn their routing table by learning new edges as shown in Figure 2. Progressively, they build an internal map of the whole object space across all s-peers in the G-Grid. This information is eventually exploited during subsequent operations to find more efficient routes to the desired objects. The goal is to minimize the number of hops in dynamic P2P networks where the

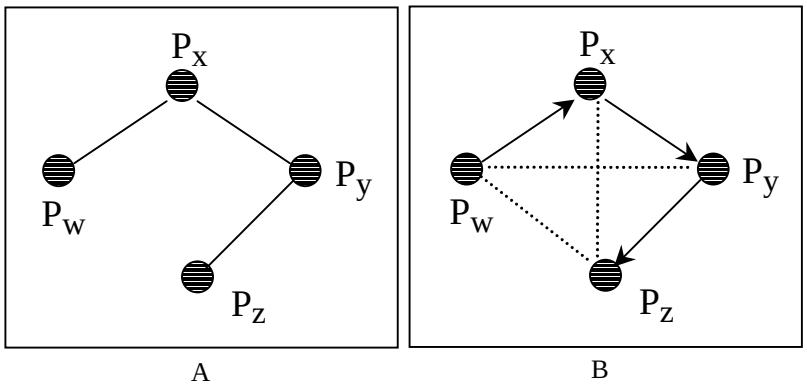

Fig. 2. A) Peers involved by an operation issued by Peer P_w routed towards P_z; B) Learning of new (dot line) links among them after completing the operation.

structure grows and the interaction between peers increases. In addition the learning mechanism contributes to distribute the workload among peers, even if they are hierarchical structured according to a tree. We report in the next section some experiments and theoretical analyses that explain the crucial variables governing the complex behaviours of the system.

Two interacting peers may decide to distribute control of their objects through a partitioning of their respective object spaces. What does trigger the distribution of control ? When the set of objects in an s-peer grows in a way that the s-peer becomes a bottleneck, an s-peer may spawn new nested regions, which are then handed over to other interacting peers (s-peers or c-peers, which then evolve to become s-peers) for control. Alternatively, the hand over may occur through direct solicitation of other available peers.

Spawning may also be triggered by application-specific considerations. For example, let us assume that objects represented in the G-Grid structure are mobile, and one of the attribute is location. If the distance between the location of a peer and the location of its objects goes over a pre-specified threshold, a new region may be spawned into an appropriate available c-peer or merged into an available s-peer in a way which reduces the distance between peers and the objects in the region. Thus, the objects in peers migrate from one peer to another to bring them closer to the actual objects they represent. The *proximity concept* applies also in the case of mobile s-peers. As an s-peer moves away from the location of objects it manages, these objects are handed over to an s-peer that is closer to the objects. While the notion of proximity is used here in the context of geographic distance, it can be readily extended to other types of attributes.

In summary the main features of the G-Grid are the following:

– distributed. The objects in the G-Grid are distributed across autonomous peers.

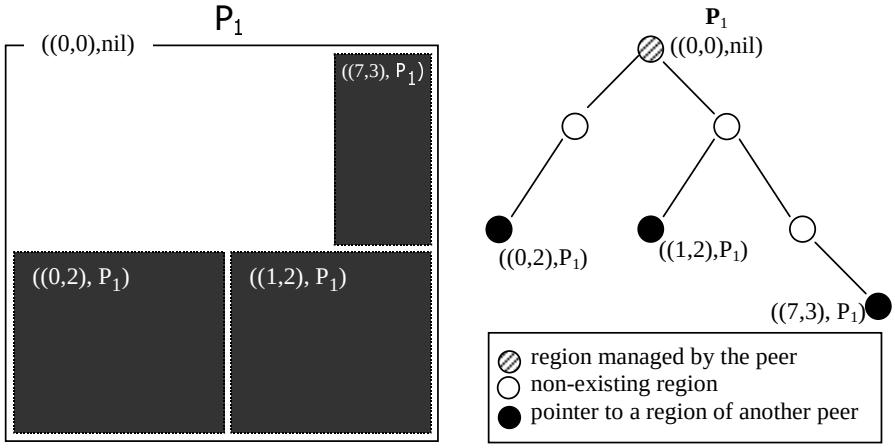

Fig. 3. A partitioned 2-dim space locally at Peer P_1 and its counterpart tree

- emerging. The structure is not imposed a-priori on the set objects in the distributed environment. Rather, the structure is built incrementally and emerges dynamically as peers interact with each other and learn each other's content.
- self-organizing. The decision for two peers to participate in the G-Grid structure and to distribute control among them is not imposed externally.
- scalable. The G-Grid is scalable in that its performance does not deteriorate as the number of peer increases.

3.1 G-Grid Split Rules

Peers are autonomous, and each peer views itself initially in control of its own whole object space (0,0), but at any one time it contains data located in only one region of the partitioned space. The remainder of the space is represented by index entries to other data regions contained in other peers. Initially, the data region in a peer coincides exactly with space (0,0). Each peer maintains a portion of the overall index, i.e., routing table, learned through its direct interactions with other peers.

As two peers interact and voluntarily decide to participate in the G-Grid, their spaces are partitioned into two nested regions with each peer taking control of one of the regions and keeping a pointer to the other peer and the region it holds. In general one of the two nested region may be collapsed in its father region. For instance, Figure 3 depicts a structure partitioned in three regions distributed across three peers. In particular the Figure depicts the physical partitioning at Peer 1, which stores locally the region (0,0) and three pointers to Peers managing the black-colored regions. The interacting peers will also keep track of the region descriptor describing each other's assigned region. These

latter descriptors become part of a local content-based routing table. The partitioning policy is flexible and may be driven by application and performance considerations.

Here we formally introduce the split rules giving also an example.

Let us introduce the following notations:

- let (r, λ) denote the identifier of the region assigned to a peer P, denoted also $P(r, \lambda)$ in case of ambiguity with peers managing a region with the same identifier; region (r, λ) contains both data elements and index entries and, when there is no ambiguity, it will also represent its content.
- Let $+$, $*$, and - represent the set union, intersection, and difference operators.
- Let $\overline{(r, \lambda)}$ denote the complement of (r, λ), namely $(r, \lambda) + \overline{(r, \lambda)} = (0, 0)$, that is the whole space, and $(r, \lambda) * \overline{(r, \lambda)} = \emptyset$. Note that $\overline{(r, \lambda)}$ is a concave space and therefore does not satisfy the definition of a region, moreover it contains only pointers to regions, which may enclose (r, λ) or be disjoint with it. Pointers are learnt through interactions with other peers.
- Let $((r^{'}, \lambda^{'}), P')$ denote an index entry in P indicating that $(r^{'}, \lambda^{'})$ is located in peer P'; in case of ambiguity it is denoted $P((r^{'}, \lambda^{'}), P')$. Note that when $P^{'}$=nil, then the content of the region is local.
- Let $(r^{'}, \lambda^{'})_H$ denote a special pointer in P, which represents a placeholder for region $(r^{'}, \lambda^{'})$ whose elements are those not in $P(r, \lambda)$. At some step during the lifecycle of the structure, $(r^{'}, \lambda^{'})$ was the region contained in P and then was later reduced to region (r, λ). The contents in $(r^{'}, \lambda^{'}) - (r, \lambda)$ were transferred to P' from P during the split operation.

The placeholder has no impact on the logical organization of the structure, the search procedure, except perhaps to increase the number of elements in a peer. It acts simply as a routing element necessary during transient states of the structure. This indirection in the search is necessary for those peers interacting with P and requesting elements in $(r^{'}, \lambda^{'}) - (r, \lambda)$. After the first interaction, these peers will have learned the new path and therefore it is no longer necessary to go through P, and instead go directly to P'. In other words, the organization will adapt naturally and incrementally to the new structure.

The split is always local to the two peers being split. There is no propagation to other peers and the completeness defined in [17] is preserved in G-Grid by construction.

Let us assume that two peers P_1 and P_2 meet and that they manage (r_1, λ_1) and (r_2, λ_2) respectively. There are two cases:

A. $(r_1, \lambda_1) * (r_2, \lambda_2) = \emptyset$, namely they manage two disjoint regions
B. $(r_1, \lambda_1) \subseteq (r_2, \lambda_2)$ or vice versa

which correspond to the two following rules:

A. $(r_1, \lambda_1) * (r_2, \lambda_2) = \emptyset$
   - $(r_1^{'}, \lambda_1^{'}) = (r_1, \lambda_1) + \overline{(r_2, \lambda_2)} / (r_1, \lambda_1)$;
     $(r_2, \lambda_2) = (r_2, \lambda_2) + (r_1, \lambda_1) / (r_2, \lambda_2)$;

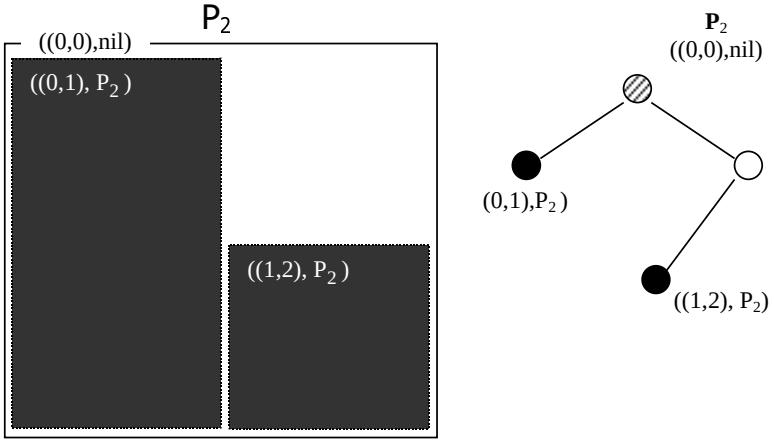

Fig. 4. A partitioned 2-dim space locally at Peer P_2 and its counterpart tree

- $\overline{(r_1^{'}, \lambda_1^{'})}$=$\overline{(r_1, \lambda_1)}$−$\overline{(r_1, \lambda_1)}$/$(r_2, \lambda_2)$+ $((r_2, \lambda_2), P_2)$;
- $\overline{(r_2^{'}, \lambda_2^{'})}$=$\overline{(r_2, \lambda_2)}$−$\overline{(r_2, \lambda_2)}$/$(r_1, \lambda_1)$+ $((r_1, \lambda_1), P_1)$;

B. (r_2, λ_2)∗$(r_1, \lambda_1) \neq \emptyset$.

- Let $(r_1, \lambda_1) = (r_1, \lambda_1)$+$(r_2, \lambda_2)$;
- Split (r_1, λ_1) into $(r_1^{'}, \lambda_1^{'})$ and $(r_2^{'}, \lambda_2^{'})$
- $\overline{(r_1^{'}, \lambda_1^{'})} = \overline{(r_1, \lambda_1)}$− $((r_2, \lambda_2), P_2)$-$((r_1, \lambda_1)$,nil)+$(r_1^{'}, \lambda_1^{'}), nil))$ +$((r_2^{'}, \lambda_2^{'}), P_2)$;
- $\overline{(r_2^{'}, \lambda_2^{'})} = \overline{(r_2, \lambda_2)}$− $((r_1, \lambda_1), P_1)$-$((r_2, \lambda_2)$,nil)+$((r_2^{'}, \lambda_2^{'}), nil)$+ $((r_1^{'}, \lambda_1^{'}), P_1)$;
- **If** $r_2^{'} \subset r_2$ **Then** $\overline{(r_2, \lambda_2)}$=$\overline{(r_2, \lambda_2)}$+$((r_2, \lambda_2), P_1)_H$
  Else if $r_2 \subset r_2^{'}$ **Then**
  - $\overline{(r_2^{'}, \lambda_2^{'})}$=$\overline{(r_2^{'}, \lambda_2^{'})}$−$\overline{(r_2, \lambda_2)}$/ $((r_2^{'}, \lambda_2^{'})$-$(r_2, \lambda_2))$ and
  - (r_2, λ_2)=(r_2, λ_2) + $\overline{(r_2, \lambda_2)}$/ $((r_2^{'}, \lambda_2^{'})$-$(r_2, \lambda_2))$.

For space reasons we limit the description of the rules by giving a single example related to the meeting of Peer 1 and 2 represented in Figure 3 and Figure 4 respectively. In each Figure is depicted both the spatial structure and its correspondent tree. The two peers manage the same region (0,0), therefore must be applied the rule B. First of all the two structures of Figure 3 and Figure 4 are conceptually merged by simply superimposing the two trees (see Figure 5A); a region identifier may be associated with a list of peers containing data located in the same spatial region.

Then, as illustrated in Figure 5B the rule performs a buddy split of the region (0,0) generating two regions: (0,1) and (1,1). Finally, the merged structure is divided between the two peers as depicted in Figure 6. We highlight that the region (0,1) has been collapsed in (0,0) of Peer 1, while Peer 2 has introduced a placeholder towards Peer 1.

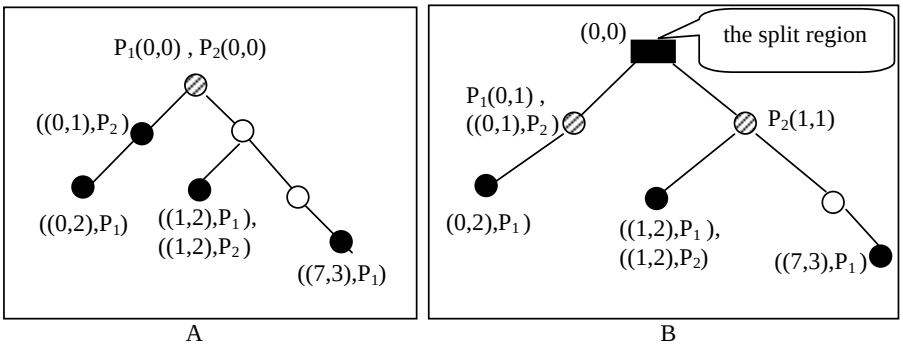

Fig. 5. A) Merged tree by superimposing the one of Fig. 3 and Fig. 4; B) Merged tree after the split of the region (0,0) in (0,1) and (1,1)

4 Performance Analyses

For space reasons we present here only some of the empirical results, which we have conducted by implementing a simulation of G-Grid, and some theoretical results confirming these experiments. The simulation manages exact match queries and record insertions and incorporates both the region splitting mechanism and the learning capability. Moreover it can be configured with some parameters, such as the region bucket size b and the rate of insertions with respect to queries $\frac{insertions}{queries}$. In the experiments the structure evolves and grows in a dynamic fashion starting from one peers and by generating operations randomly. On this basis G-Grid is a stochastic system with complex behaviours where each state of the system depends on the preceding one, but the set itself of the states evolves

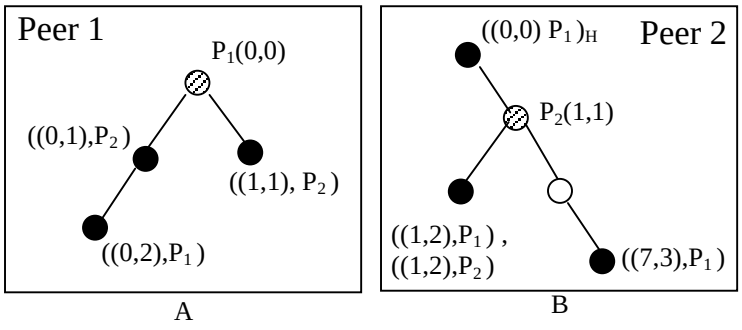

Fig. 6. Physical configurations A) at Peer 1 by collapsing (0,1) in (0,0), and B) at Peer 2 with a placeholder to (0,0) of Peer 1

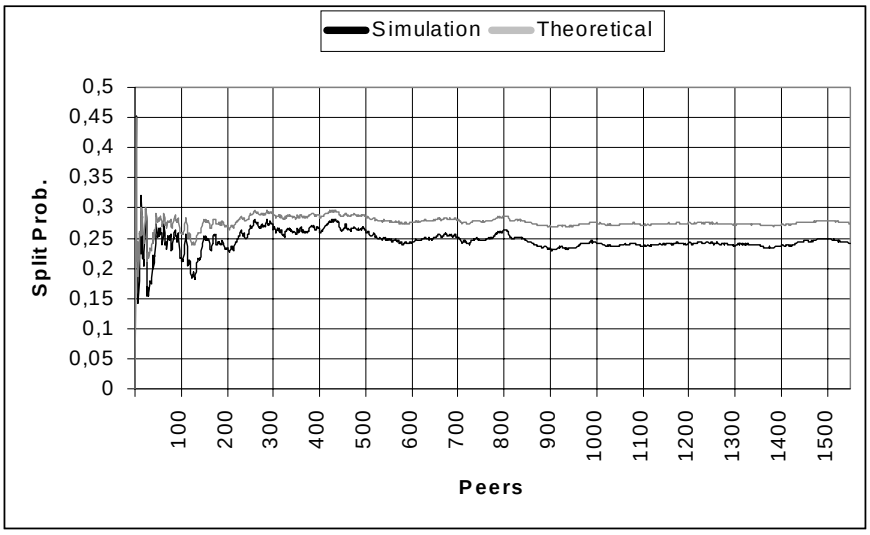

Fig. 7. The approximated theoretical split prob. compared with a simulation

dynamically over the time growing very quickly and making hard any analysis based on Markov chains.

However we have found the split probability, which predicts the growth of the system and it is useful to find important theoretical results related to the *average path length* (APL) to deliver any message in the system. For space reasons we limit to present here the simplest version which well approximates the mentioned above probability for very low values of the bucket size.

Split Probability Definition: let t be any instant in the life of a G-Grid structure G and let us denote the following variables:

- N_t = records at the instant t randomly distributed in G
- M_t = regions/peers in G at the instant t
- b = the region bucket size

then the split probability is the probability that a record insertion at the instant t ends in a region already full with b records, namely:

$$P_s(N_t, M_t, b) = \frac{1}{k - j + 1} \cdot \sum_{i=j}^{k} \frac{i}{M_t} \tag{1}$$

where j and k, which are the minimum and maximum number of full regions respectively, are the following:

$$j = max(N_t - M_t \cdot (b-1), 0) \qquad k = floor\left(\frac{N_t - \frac{b}{3} \cdot M_t}{\frac{2}{3} \cdot b}\right) \tag{2}$$

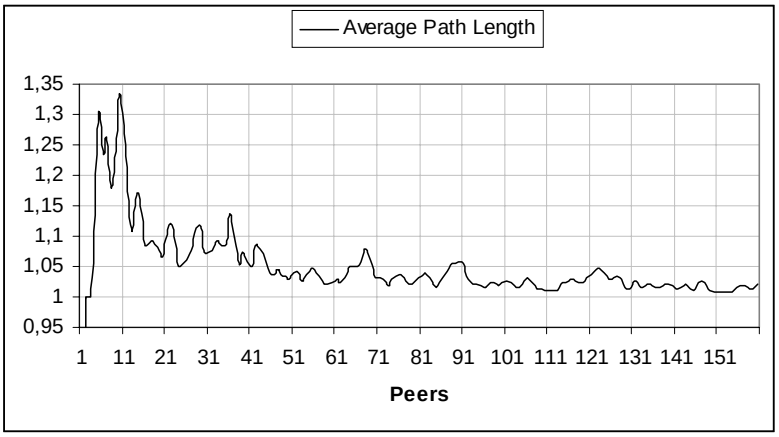

Fig. 8. When $\frac{insertions}{queries} \approx \frac{1}{M_t^2}$ the average path length to deliver messages tends to 1

Figure 7 illustrates a numeric comparison between the formula 1 and an experiment conducted with $b = 6$ where the system grows up to more than 1500 peers. The two split probabilities oscillate until there are less than 50 peers, then both of them stabilize with a difference around 2.5%; in all experiments the stabilization occurs always independently on the rate $\frac{insertions}{queries}$. Experiments have also confirmed an average *storage utilization* per region equals to $\frac{2b}{3}$, that is the number of records stored on average by regions.

The rate $\frac{insertions}{queries}$ instead is determining for the APL in the system, in fact if the rate remains constant the APL tends to grow, but less than logarithmically with respect to the number of peers. This effect is due to the learning capability which reduces the distances in the system by creating new links. Finally, we verified both theoretically and experimentally that if the rate $\frac{insertions}{queries}$ changes like $\Theta(\frac{1}{M_t^2})$ then the APL in the system tends quickly to 1 (see Figure 8). Intuitively this can explained by the fact that both queries and insertions creates new links in the system, but in addition insertions cause splits introducing new regions and new peers which increase the APL. In several realistic scenarios the number of queries is more than quadratic with respect to the number of users/machines, for instance in the World Wide Web each user access may originate in cascade many messages and queries. Also the execution of more traditional queries, such as joins and range queries, can generate that number of requests.

5 Robustness

The G-Grid allows peers to connect and disconnect autonomously from the structure. C-peers can connect and disconnect without an impact on the overall G-Grid structure, except perhaps that responses to their already initiated requests

will not have a return address. On the other hand, an non-anticipated disconnection of an s-peers may make the s-peer's objects and local routing table inaccessible. Thus, it is important that s-peer return local information to the system in non-catastrophic disconnection. An **orderly disconnection** is one where an s-peer hands over its content to another peer to preserve data accessibility and routing information. It can: either (i) merge its local information (both objects and routing table) to its father s-peer or a child s-peer., or, (ii) solicit a c-peer as a replacement. The choice is determined based on policies which will enhance the overall performance of the system.

A **disorderly disconnection** of a s-peer occurs in catastrophic situations such as computer crashes or physical network problems. Objects in the s-peer become inaccessible and routing through the s-peer is no longer possible. Depending on the network topology, a disorderly disconnection could cause the G-Grid partitioning into two disjoint component. To enhance robustness of the system, one approach is through duplication of information in s-peers. Besides the associated information integrity problems, duplication does not eliminate the problem of G-Grid partitioning, it only reduces the likelihood of its occurrence. Our approach is to avoid altogether duplication and rely on the learning mechanism of the system, which establishes incrementally links between the various s-peers as the level of interaction increases, and thus provides multiple routes to get to s-peers. To what extent does the learning mechanism reduce the likelihood of G-Grid partitioning? Our preliminary experimental results show that the likelihood of partitioning is practically nil. What will be the effect on performance and availibility of data in combining both duplication and the learning mechanism? Answers to these questions require an extensive robustness analysis, which we intend to do in the future.

After a disorderly disconnection, an s-peer may rejoin the G-GRID either as a c-peer or as an s-peer. If it chooses the former approach, it will have to issue direct insertion requests for all its objects to the G-Grid system. In the latter, it will have to wait for an interaction with another s-peer and then integrate its content through the normal partitioning process.

As indicated earlier, an important concept in the G-Grid is the peers' ability to learn other peers' local routing tables during search operations. As a peer interacts with other peers, its local routing table grows and improves its capability to find the most efficient route to its target objects. Clearly, learning content-based routing tables is an emergent property in that the minimum path to a target peer is discovered without having to encode into the system a minimum path algorithm.

6 Conclusions

In this paper, a class of scalable self-organizing data structures for P2P networks, called the G-Grid, is introduced. These structures enable efficient multidimensional search based on partial range queries. We have also illustrated how peers can exploit the properties of these structures to learn dynamically both the dis-

tribution of content and the network topology, and thereby, provide algorithms for efficient processing of range queries. In the worst case, search costs for a single object, measured as the number of hops over peers, are logarithmic in the number of peers. But, for many realistic workloads of insertions of new objects and retrievals, such as those currently taking place on the web, the average is equal or less than 2 hops, independently on the wideness of the P2P network. We have also sketched out an aspect which is seldom treated in P2P literature, namely the possibility of merging independently constructed data structures. This is particularly important for two autonomous organization, which make the decision to share data between them for commercial or scientific reasons.

This work is a summary of ongoing work towards the idea of achieving virtual DBMSs from P2P systems as a set of emergent services, but which abide by the same desirable properties of centralized DBMSs, namely, data integrity and consistence, transaction processing and a complete SQL expressiveness.

References

1. Moro, G., Ouksel, A.M., Sartori, C.: Agents and peer-to-peer computing: a promising combination of paradigms. In: Proceedings of the First 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. W. Litwin, M. A. Neitmat, D.A.S.: RP*A Family of OrderedPreserving Scalable Distributed Data Structures. In: In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94), Santiago, Chile. (1994) 342–353
5. W. Litwin, M. A. Neitmat, D.A.S.: LH*Linear Hashing for Distributed Files. ACM Transactions on Database Systems **4** (1996) 480–525
6. B. Kröll, P.W.: Distributing a search tree among a growing number of processors. In: In Proceedings of the ACM International Conference on Management of Data (SIGMOD'94), Minneapolis, MN, USA, ACM Press (1994) 265–276
7. Pasquale, A.D., Nardelli, E.: Adst: An order preserving scalable distributed data structure with constant access costs. In Carey, M.J., Schneider, D.A., eds.: Proceedings of the 28th Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'01), Piestany, Slovak Republic. Volume 2234., Springer-Verlag (2001) 211–222
8. 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
9. Zhao, B.Y., Kubiatowicz, J., Joseph, A.: Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. In: Technical report, UCB/CSD-01-1141, University of California, Berkeley. (2001)
10. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science **2218** (2001) 329–340

11. Aberer, K., Cudr-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., Schmidt, R.: Improving data access in p2p systems. SIGMOD Record **2** (2003)
12. Ng, W.S., Ooi, B.C., Tan, K.L., Zhou, A.Y.: PeerDB: A P2P-based System for Distributed Data Sharing. In: International Conference on Data Engineering (ICDE). (2003) 633–644
13. Ouksel, A.M.: The interpolation-based grid file. In: Proceedings of the ACM SICACT-SIGMOD Symposium on Principles Of Data Base Systems, ACM (1985) 20–27
14. Ouksel, A.M., Mayer, O.: A robust and efficient spatial data structure: The nested interpolation-based grid file. Acta Informatica **29** (1992) 335–373
15. Ouksel, A.M., Kumar, V., Majumdar, C.: Management of concurrency in interpolation-based grid file organization and its performance. Information Sciences Journal **2** (1994) 129–158
16. Ouksel, A.M., Kammermeier, F.: The interpolation-based grid file revisited. Technical Report Progress Report, PhD dissertation, Computer Science Department, Kaiserslautern University (2002)
17. Ouksel, A.M., Moro, G., Litwin, W.: GGF: A Generalized Grid File for Distributed Environments. Technical Report UIC-IDS-CRIM/TECH-REPORT No.2002-05, University of Illinois at Chicago, DEIS University of Bologna (2002)