Spatial partitioning algorithms evaluation in multidimensional peer-to-peer networks

Panagiotis Ainalis

29-8-2009

Contents

List of Figures

Abstract

Over the Internet today, there has been great interest in rising Peer-to-Peer network overlays as they provide several benefits in comparison to the traditional client-server model. A major drawback of current p2p protocols is the uneven distribution of load balance among peers which results in a lower performance of the network itself. Our research focuses on ameliorating this problem by studying algorithms, used throughout the construction of the network, which aim towards superior data balance. We anticipate that these algorithms will result in an enhanced distribution of load, with better overall network performance. The protocol of choice for our experiments is P-Grid whose main characteristics are desirable. Our results highlight the importance of efficient network construction algorithms which lead to the fulfilment of certain criteria and metrics that related research has introduced.

Chapter 1

Introduction

1.1 Introduction

Throughout the last years, there has been a scientific turn towards distributed models and especially p2p technologies. A pure P2P network does not have the notion of clients or servers but only equal peer nodes that simultaneously function as both "clients" and "servers" to the other nodes on the network. This model of network arrangement differs from the client-server model where communication is usually to and from a central server. Such networks are useful for many purposes. Sharing data of any content (such as audio, video), realtime data (such as telephony traffic) and many other types of resources. Each node provides access to its computing and networking resources in exchange to participating to the p2p community.

The advantages of p2p networks are numerous. All peers provide resources, including bandwidth, storage space, and computing power. As the number of peers increases, the total capacity of the system also increases in contrast with the client-server architecture where a fixed set of servers exists and adding more clients results in slower data transfer for all users. The distributed nature of P2P networks also increases robustness in case of failures by replicating data over multiple peers, or by enabling peers to find the data without relying on a centralized index server. In the latter case, there is no single point of failure in the system.

1.2 History of peer-to-peer networks

Peer-to-peer technologies started development due to the file-sharing needs of Internet users. Therefore, early research was evolved around structures and algorithms designed to satisfy these needs.

1.2.1 First P2P-generation: Server-client

The first generation of peer-to-peer networks resembles the traditional client-server model regarding its organization. The system is responsible for controlling traffic among peers. Ths servers use a centralized database to store directories of the shared files of users and are required to update them accordingly. In this model, the procedure of searching involves sending queries to the server which replies back with a list of peers fulfilling his criteria and facilitating the connection in order to start downloading the files. The Server-client system is simple and efficient because the central directory is constantly updated and all users have to register to use the program. On the downside, there is only a single point of entry to the system, which might result in the collapse of the network, in the case of an attack for example. Another drawback noticed in these networks is the need of powerful servers and network bandwidth, in order to be able to manage and satisfy the queries of a massive number of users.

Famous peer-to-peer server-client-protocols are:

- Audiogalaxy[1] - Service ended in 2002.
- DirectConnect++[2]
- Napster[3] - The first famous p2p network, now changed to a fee-based service.
- Soulseek[4]
- TinyP2P[5] - The World's Smallest P2P Application, it is written in 15 lines of Python code.

[1] http://www.audiogalaxy.com/
[2] http://dcplusplus.sourceforge.net/
[3] http://free.napster.com/
[4] http://www.slsknet.org/
[5] http://www.freedom-to-tinker.com/tinyp2p.html

1.2.2 Second P2P-Generation: Decentralization

After the legal implications Napster had due to its content, the community focused on a network without a cetral index server and Gnutella[6] was born. Gnutella's attempt to invent the 'all-equal' scheme experienced severe problems from bottlenecks as the network grew from disappointed Napster users. FastTrack[7] solved this problem by giving some users the role of super nodes. By selecting some more powerful nodes to be indexing nodes, with lower capacity ndoes branching off from them, FastTrack allowed for a network that could scale to a bigger network size. Gnutella and any new peer-to-peer network adopted this technique as it allows for large and efficient networks without centralization.

The second generation includes distributed hash tables (DHTs), which help solve the scalability problem by selecting several nodes to index certain hashes (which in turn are used to identify files), allowing for fast and efficient searching for all instances of a file on the network. The main drawback of this technology is that it cannot support keyword searching, as opposed to exact-match searching.

1.2.3 Third P2P-Generation: Indirect and encrypted

The third generation of peer-to-peer networks involves the addition of anonymity features built in. The FSF (Free Software Foundation) created GNUnet[8], a promising peer-to-peer network which aims to provide maximum security and anonymity in contrast to most popular networks. Other attempts of such networks are Freenet[9], Entropy[10], ANts P2P[11].

A technique to maintain anonymity is to forward all packets through other users' clients in order to make it more difficult to identify who is downloading or who is offering files. In Addition, most of these programs use strong encryption to avoid packet sniffing. Another technique involves the "friend-to-friend" networks which only allow trusted users to connect to the user's computer, then each node can forward requests and files between its own friends providing extra anonymity.

[6] http://gnufu.net/
[7] http://developer.berlios.de/projects/gift-fasttrack/
[8] http://gnunet.org/
[9] http://freenetproject.org/
[10] http://www.anonymous-p2p.org/entropy.html
[11] http://antsp2p.sourceforge.net/

Third generation networks have not reached mass usage due to their complicated implementations requiring too much overhead in their anonymity features, making them too slow or hard to use. For example, in GNUnet's implementation there are no points of entry into the network available in public locations as opposed to other networks. The procedure an entering node requires to make is, scanning the Internet using heuristics and statistical analysis tools in order to "hit" a host already inside the GNUnet network which helps to initiate the bootstrap process. This implementation obviously aims to superior anonymity of the GNUnet users but this extreme overhead needed looks less appealing to the vast proportion of peer-to-peer users nowadays.

1.2.4 The fourth P2P-Generation: Streams over P2P

Apart from the traditional file sharing there are services that send streams instead of files over a P2P network. Therefore, one can hear radio and watch television without any server involved (the streaming media is distributed over a P2P network). It is important that instead of a tree-like network structure, a swarming technology known from BitTorrent[12] is used. Best examples are IPTV[13], Miro[14], Joost[15].

1.3 Classification

Based on how the nodes in the overlay[16] network are linked to each other we classify the p2p networks as unstructured and structured:

Unstructured: An unstructured P2P network is formed when the overlay links are established arbitrarily. Such networks can be easily constructed as a new peer that wants to join the network can copy existing

[12] http://www.bittorrent.com/

[13] http://www.itu.int/ITU-T/IPTV/

[14] http://www.getmiro.com/

[15] http://www.joost.com/

[16] An overlay network is a computer network which is built on top of another network. Nodes in the overlay can be thought of as being connected by virtual or logical links, each of which corresponds to a path, perhaps through many physical links, in the underlying network. For example, many peer-to-peer networks are overlay networks because they run on top of the Internet. Dial-up Internet is an overlay upon the telephone network.

links of another node and then form its own links over time. In an unstructured P2P network, if a peer wants to find a desired piece of data in the network, the query has to be flooded through the network to find as many peers as possible that share the data. The main disadvantage with such networks is that the queries may not always be resolved. Popular content is likely to be available at several peers and any peer searching for it is likely to find the same thing. But if a peer is looking for rare data shared by only a few other peers, then it is highly unlikely that search will be successful. Since there is no correlation between a peer and the content managed by it, there is no guarantee that flooding will find a peer that has the desired data. Flooding also causes a high amount of signaling traffic in the network and hence such networks typically have very poor search efficiency. Most of the popular P2P networks such as Gnutella and FastTrack are unstructured.

Structured: Structured P2P network employ a globally consistent protocol to ensure that any node can efficiently route a search to some peer that has the desired file, even if the file is extremely rare. Such a guarantee necessitates a more structured pattern of overlay links. By far the most common type of structured P2P network is the distributed hash table (DHT), in which a variant of consistent hashing is used to assign ownership of each file to a particular peer, in a way analogous to a traditional hash table's assignment of each key to a particular array slot. Some well known DHTs are P-Grid, Pastry, Tapestry, CAN, and Chord

1.4 Definition of the problem and our approach

Blanas et al.[1], present a verbose comparison of the most promising peer-to-peer protocols regarding range queries; that is, P-Grid, Can, Murk, VBI-trees. In two dimensions, P-Grid shows the best maximum throughput[17] results as the network size scales. A major drawback in the P-Grid-Z topology, which the authors implemented, is that volume-balanced selection results in uneven data distribution, with 10% of the peers storing 70–85% of the dataset. As might be expected, uneven data balancing induces uneven

[17]Maximum throughput definition is explained in Chapter 3

distribution of query accesses (that is, the percentage of query processes in which a peer participates).

The purpose of this work is to propose and study algorithms that lead to a more "fair" data distribution among peers while evaluating their performance using metrics like maximum throughput or latency. We anticipate that a more uniform distribution of datakeys in peers will result in increased network performance and scalability overall, especially during range queries. Our analysis occurs in two phases; a) during network construction, and, b) during range queries conducted after the construction is completed.

DHTs have shown great performance in equality searches, but the scientific work in range queries, especially in multiple dimensions, is far from complete. Many peer-to-peer protocols have been proposed in literature to solve the aforementioned challenges with ambiguous results. P-Grid's desirable features and, additionally, the encouraging results from related work, make it an ideal candidate to study the problem.

We studied the proposed algorithms and the results are interesting. None of the proposed solutions is the best in all aspects, each one has pros and cons depending mainly on the uniformity of datakeys distribution in the volume space. Some of our algorithms (random-walks) perform extremely well in uniform environments, whereas the cause of poor performance they present in non-uniform environments is not yet understood.

Chapter 2

Related work

2.1 Distributed Hash Tables

The design of Gnutella was the first attempt to create a decentralized indexing scheme using a peer-to-peer network. Each node of this network maintains a list of neighbors which uses in order to route messages across the peers. When a query is instantiated at a peer, the query is forwarded to every node in the neighbor list, recursively in the subsequent nodes, which requires a linear bandwidth to the number of total network peers. Additionally, the search requests have high probability to be dropped before the whole network has been contacted, therefore the results cannot be reliable. Unfortunately, this flooding scheme is the only valid way to locate data in networks with such infrastructure.

Extensive research has led to to structured networks which employ a globally consistent protocol to ensure that any node can efficiently route a search to some peer that has the desired data. Such a guarantee necessitates a more structured pattern of overlay links. The most common type of such networks are DHTs (Distributed Hash Tables) which provide a mapping of keys onto values on extremely large, Internet-scale systems. DHTs are a class of decentralized distributed systems that provide a lookup service similar to a hash table: pairs are stored in the DHT, and any participating node can efficiently retrieve the value associated with a given name. [1]

[1] It can be observed that the DHT interface follows the conventions used for retrieving and storing data in regular hash tables and that the distribution of data to the appropriate peers is completely transparent to the application.

Responsibility for maintaining the mapping from names to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows DHTs to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures. DHTs form an infrastructure that can be used to build more complex services, such as distributed file systems, peer-to-peer file sharing and content distribution systems, cooperative web caching, multicast, anycast, domain name services, and instant messaging. Notable distributed networks that use DHTs include BitTorrent, Yacy[2] or CCDN[3] (Coral Content Distribution Network). Distributed hash tables use a more structured key based routing in order to attain both the decentralization of Gnutella and Freenet, and the efficiency and guaranteed results of Napster. One drawback is that, like Freenet, DHTs only directly support exact-match search, rather than keyword search, although that functionality can be layered on top of a DHT.

DHTs characteristically emphasize the following properties:

Decentralization: the nodes collectivelly form the system without any central coordination.

Scalability: the system should function efficiently even with thousands or millions of nodes.

Fault-tolerance: the system should be reliable even with nodes continuously joining, leaving, and failing.

2.1.1 CAN

In [4], Ratnasamy introduces the Content Addressable Network (CAN) which is a distributed, decentralized P2P infrastructure that provides hash table functionality on an Internet-like scale. CAN was one of the original four distributed hash table proposals, introduced concurrently with Chord, Pastry, and Tapestry.

Like other distributed hash tables, CAN is designed to be scalable, fault tolerant, and self-organizing. The architectural design is a virtual multi-dimensional Cartesian coordinate space on a multi-torus. This d-dimensional

[2]http://yacy.net/

[3]http://www.coralcdn.org/

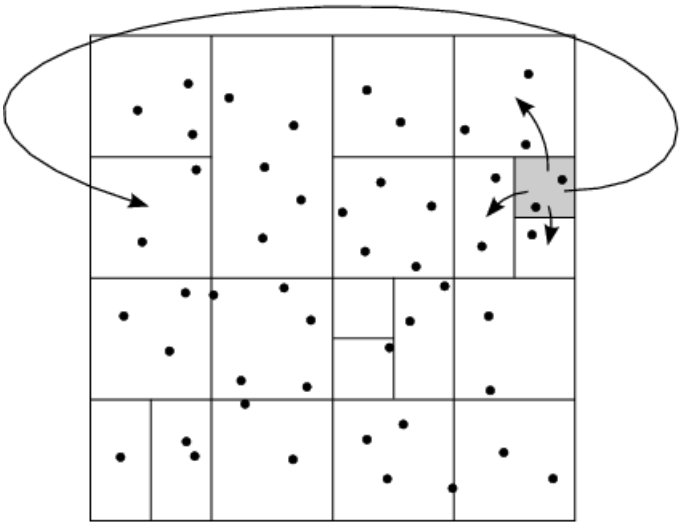


Figure 2.1: Simple Can topology. Source: [1]

coordinate space is a virtual logical address, completely independent of the physical location and physical connectivity of the peers. The entire coordinate space is dynamically partitioned among all the peers (N number of peers) in the system such that every peer possesses its individual, distinct zone within the overall space. A CAN peer maintains a routing table that holds the IP address and virtual coordinate zone of each of its neighbor coordinates. A peer routes a message towards its destination using a simple greedy forwarding to the neighbor peer that is closest to the destination coordinates.

Typical CAN topology: the shaded region is the responsibility area of the peer and the arrows indicate the peers which the shaded peer knows.

2.1.2 Chord

Using the Chord lookup protocol, node keys are arranged in a circle. The circle cannot have more than 2^m nodes. The ring can have ids/keys ranging from 0 to $2^m - 1$.

Fig. 1 : Typical Chord topology: the shaded region is the responsibility area of the shaded peer and the arrows indicate the entries in the finger table.

IDs and keys are assigned an m-bit identifier using what is known as

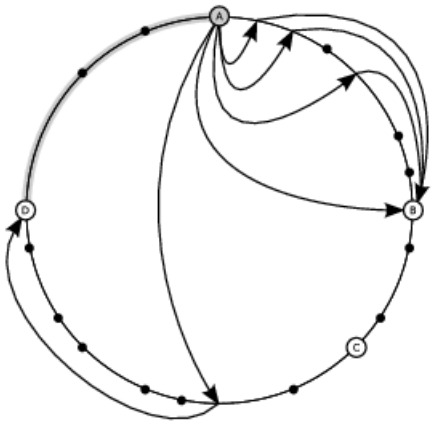


Figure 2.2: Simple Chord topology. Source: [1]

consistent hashing. The SHA-1 algorithm is the base hashing function for consistent hashing. The consistent hashing is integral to the probability of the robustness and performance because both keys and IDs (IP addresses) are uniformly distributed and in the same identifier space. Consistent hashing is also necessary to let nodes join and leave the network without disrupting the network.

Each node has a successor and a predecessor. The successor to a node or key is the next node in the identifier circle when you move clockwise. The predecessor of a node or key is the next node in the id circle when you move counter-clockwise. If there is a node for each possible ID, the successor of node 2 is node 3, and the predecessor of node 1 is node 0; however, normally there are holes in the sequence, so, for example, the successor of node 153 may be node 167 (and nodes from 154 to 166 will not exist); in this case, the predecessor of node 167 will be node 153. Since the successor (or predecessor) node may disappear from the network (because of failure or departure), each node records a whole segment of the circle adjacent to it, i.e. the K nodes preceding it and the K nodes following it. One successor and predecessor are kept in a list to maintain a high probability that the successor and predecessor pointers actually point to the correct nodes after possible failure or departure of the initial successor or predecessor.

2.1.3 Pastry

Although the distributed hash table functionality of Pastry is almost identical to other DHTs, what sets it apart is the routing overlay network built on top of the DHT concept. This allows Pastry to realize the scalability and fault tolerance of other networks, while reducing the overall cost of routing a packet from one node to another by avoiding the need to flood packets. Because the routing metric is supplied by an external program based on the IP address of the target node, the metric can be easily switched to shortest hop count, lowest latency, highest bandwidth, or even a general combination of metrics.

The hash table's keyspace is taken to be circular, like the keyspace in the Chord system, and node IDs are 128-bit unsigned integers representing position in the circular keyspace. Node IDs are chosen randomly and uniformly so peers who are adjacent in node ID are geographically diverse. The routing overlay network is formed on top of the hash table by each peer discovering and exchanging state information consisting of a list of leaf nodes, a neighborhood list, and a routing table. The leaf node list consists of the $\frac{L}{2}$ closest peers by node ID in each direction around the circle.

In addition to the leaf nodes there is also the neighborhood list. This represents the M closest peers in terms of the routing metric. Although it is not used directly in the routing algorithm, the neighborhood list is used for maintaining locality principals in the routing table.

Finally there is the routing table itself. It contains one entry for each address block assigned to it. To form the address blocks, the 128-bit key is divided up into digits with each digit being b bits long, yielding a numbering system with base 2^b. This partitions the addresses into distinct levels from the viewpoint of the client, with level 0 representing a zero-digit common prefix between two addresses, level 1 a one-digit common prefix, and so on. The routing table contains the address of the closest known peer for each possible digit at each address level, except for the digit that belongs to the peer itself at that particular level.

2.1.4 Tapestry

The first generation of peer-to-peer applications, including Napster and Gnutella, had restricting limitations such as a central directory for Napster and scoped broadcast queries for Gnutella limiting scalability. To address these problems a second generation of P2P applications were developed including Tapestry.

This overlay implements a basic key-based routing mechanism. This allows for deterministic routing of messages and adaptation to node failures in the overlay network. Tapestry is an extensible infrastructure that provides decentralized object location and routing focusing on efficiency and minimizing message latency. This is achieved since Tapestry constructs locally optimal routing tables from initialization and maintains them in order to reduce routing stretch. Furthermore, Tapestry allows object distribution determination according to the needs of a given application. Similarly Tapestry allows applications to implement multicasting in the overlay network.

Each node is assigned a unique nodeID uniformly distributed in a large identifier space. Tapestry uses SHA-1 to produce a 160-bit identifier space represented by a 40 digit hex key. Application specific endpoints GUID's are similarly assigned unique identifiers. NodeID's and GUID's are roughly evenly distributed in the overlay network with each node storing several different ID's. From experiments it is shown that Tapestry efficiency increases with network size so multiple applications sharing the same overlay network increases efficiency. To differentiate between applications a unique application identifier is used.

2.1.5 P-Grid

Our work focuses on the P-Grid protocol, a structured DHT which was introduced by Karl Aberer. P-Grid is a self-organizing structured peer-to-peer system, which can accommodate arbitrary key distributions (and hence support lexicographic key ordering and range queries), still providing storage load-balancing and efficient search by using randomized routing. Its salient features are:

- Good storage load-balancing despite arbitrary load-distribution over the key-space.
- Range queries can be naturally supported and efficiently processed on P-Grid because of P-Grid abstracts a trie-structure, and supports a rather arbitrary distribution of keys as observed in real life scenarios.
- A gossip primitive based update mechanism for keeping replicated content up-to-date.
- Easy merger of multiple P-Grids, and hence decentralized boostrapping of the P-Grid network.

- Query-adaptive caching is easy to realize on P-Grid to provice query load-balancing where peers have restricted capacity.

Overview: P-Grid abstracts a trie and resolves queries based on prefix matching. The actual topology has no hierarchy. Queries are resolved by matching prefixes. This also determines the choice of routing table entries. Each peer, for each level of the trie, maintains autonomously routing entries chosed randomly from the complementary sub-trees. In fact, multiple entries are maintained for each level at each peer to provide fault-tolerance (as well as potentially for query-load management). For diverse reasons including fault-tolerance andn load-balancing, multiple peers are responsible for each leaf node in the P-Grid tree. These are called replicas. The replica peers maintain an independent replica sub-network and uses gossip based communication to keep the replica group up-to-date. The redundancy in both the replication of key-space partitions as well as the routing network together is called structural replication.

In P-Grid, the data items hash to m-bit identifiers. Each peer is assigned all identifiers which begin with a given prefix, in such a way that each peer is responsible for a partition of the entire data space. For routing purposes, each peer maintains a link to a peer in the other side of the virtual binary tree, for every bit of its prefix. Lookup messages are forwarded to the peer which has the longest common prefix with the destination.

2.2 Range queries

Most P2P systems support only simple lookup queries. However, many new applications, such as photo sharing or massive multiplayer games, would greatly benefit from support for multidimensional range queries. Important research has been done in the field of range queries, also known as orthogonal range search queries. According to definition, a range query is supposed to return all keys in an interval defined by 2 bounds. Range queries are non-trivial search predicates for structured overlay networks. The Distributed Hash Tables we have described fulfil several desirable criteria such as time efficience in key searching, due to the logarithmic number of messages routed in order to retrieve information. Nevertheless, they also yield a major disadvantage. The uniform hashing functions they use to achieve probabilistically

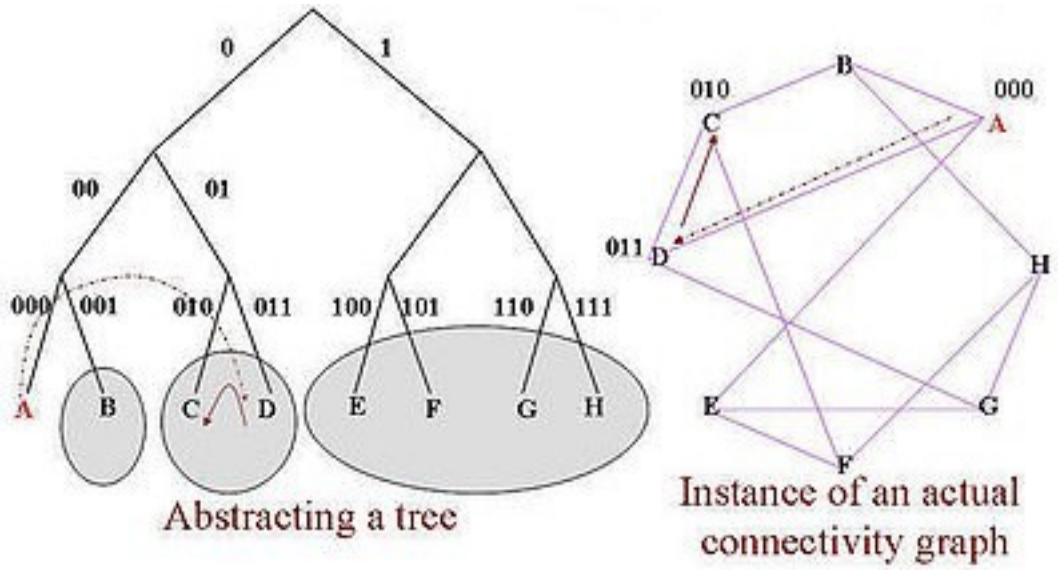


Figure 2.3: A simple P-Grid trie structure and its graph. Source: Wikipedia

good load balance is disastrous for range queries. Range queries are based on spatial locality which is however destroyed when keys are uniformly hashed before entering the network. As a results, DHTs cannot handle natively this kind of queries.

Numerous solutions have been proposed to solve this problem. These solutions tend to keep the good characteristics that DHTs have established while experimenting on novel network infrastructures and algorithms which can handle the additional complexity of range queries.

Most notable are illustrated in following sections.

2.2.1 Space-filling curves

Space-filling curves or Peano curves are curves whose ranges contain the entire 2-dimensional unit square, or the entire 3-dimensional unit cube, or for more dimensions the entire k-dimensional unit hypercube. Intuitively, a "continuous curve" in the 2-dimensional plane or in the 3-dimensional space can be thought of as the "path of a continuously moving point". A curve (with endpoints) is a continuous function whose domain is the unit interval [0,1]. Space-filling curves are a valuable tool for mapping multi-dimensional data down to a single dimension.

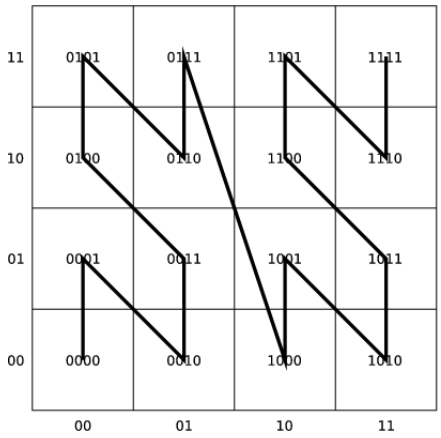


Figure 2.4: Z-order space filling curve applied over a two-dimensional unit square. Source: [1]

Z-ordering: One of the most often used space-filling curves in computer science is z-ordering. The z-value of a point in multidimensions is simply calculated by interleaving the binary representations of its coordinate values. Once the data are sorted into this ordering, any one-dimensional data structure can be used such as binary search trees, B-trees, skip lists or (with low significant bits truncated) hash tables.

Hilbert-curve: A major drawback of z-order curves is that it does not preserve spatial locality. A more satisfying curve on this matter is the hilbert curve shown in the figure below. On the downside, the algorithmic complexity of mapping K-dimensional points to 1 dimension using the Hilbert curve makes z-ordering usually a more desirable choice.

2.2.2 SCRAP-MURK

Ganesan et al.[3], present two popular spatial-database solutions, and compare them experimentally in a peer-to-peer environment. The first solution, named SCRAP, uses space-filling curves with range partitioning. The second solution, named MURK, uses kd-trees.

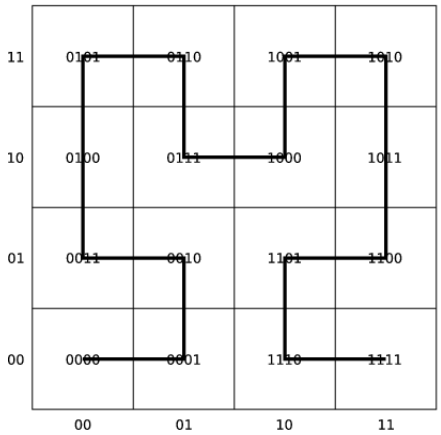


Figure 2.5: The Hilbert curve. Source: [1]

2.2.2.1 SCRAP

The SCRAP approach uses two steps to partition the data:

a) All data is usually mapped down into a single dimension using a space-filling curve, either z-ordering or the Hilbert curve.[4]

b) The single-dimensional data is then ranged-partitioned across the available nodes. To make things clearer, let's suppose each data gets converted into a z-value through z-ordering. After the range-partitioning each node manages data in one contiguous range of z values. It is easy to maintain the range-partitioning through churn: When a new node enters the network, it splits the range of an existing node; when a node leaves, one of its neighbors takes over its range[5] .

This approach has shown positive results in experiments such as good locality and load balancing between the participant nodes. However, a major problem of space-filling curves is that its 1-dimensional range may actually map to peers irrelevant to the initial query. A desirable characteristic would be to confine the number of peers only to those which contribute to the initial search query, and SCRAP does not fulfil this criteria efficiently.

[4]It should be noted that this mapping is bijective: every 2-dimensional point maps to a unique single-dimensional value and vice versa.

[5]In the real world nodes leave the network randomly. In order to avoid data loss, replication of data may need to be implemented.

2.2.2.2 MURK

MURK partitions the data directly in the high-dimensional space, breaking up the data space into hypercubes (or "rectangles"), with each peer managing one hypercube. This is achieved using a kd-tree structure whose leaves correspond to a hypercube stored by a peer. This kind of partitioning looks similar to an existing peer-to-peer system called CAN[4] with one decisive difference: In CAN, a new node that enters the network splits an existing node's data space equally, whereas, in MURK, the data load is splitted equally.

In the MURK network, each peer knows his neighboring peers in all dimensions, that is, the peers that share a boundary with this peer. For greater efficience, skip pointers have been used which is a random list of "extra" neighbors per peer obtained from techniques such as random walks. When a query is initialized at peer P, the following procedure will happen: the query is sent to the neighboring peer that is minimizing the distance to the query range centroid, where the distance from a peer P to a hypercube R is defined as the minimum Manhattan distance from any point in P's area to any point in R. Once the query reaches a relevant peer, the query is flooded to all relevant neighbors recursively.

The experiments, performed in [1], show that the MURK topology, although maintaining good results regarding the rate of responsiveness in queries, does not scale well as network size increases while exhibiting reduced maximum throughput in contrast to P-Grid.

2.2.3 Range queries on P-Grid

P-Grid partitions the key-space in a granularity adaptive to the road at that part of the key-space. Consequently, it is possible to realize a P-Grid overlay network where each peer has similar storage load even for non-uniform load distributions. This network provably provides as efficient search of keys as traditional Distributed Hash Tables do. In contrast to P-Grid, DHTs work efficiently only for uniform load-distributions. Hence we can use a lexicographic order preserving function, i.e., $\bigvee s_1,s_2 : s_1 < s_2 \Rightarrow h(s_1) < h(s_2)$, instead of uniform hashing to generate the keys, and still realize a load-balanced P-Grid network which supports efficient search of exact keys. Moreover, because of the preservation of lexicographic ordering, range queries can be done efficiently and precisely on P-Grid. The trie structure of P-Grid allows different range query strategies, processed serially or parallely, trading

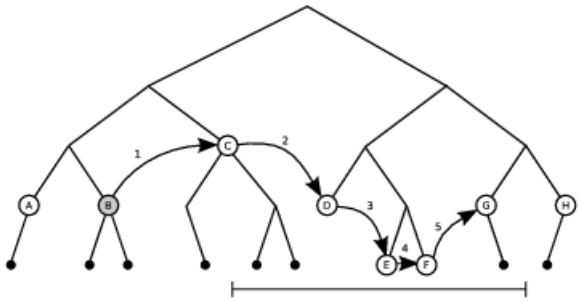


Figure 2.6: Min-Max traversal algorithm. Source: [1]

off message overheads and query resolution latency.

Datta et al.[2]describe two classes of algorithms for queries: the min-max traversal algorithm which is sequential and the shower algorithm which parallelizes the execution of range queries. In our protocol we have used the shower algorithm.

In the **Min-Max traversal algorithm**, queries can be enacted sequentially by starting at the node holding data items belonging to the one bound of the range and forwarding the query to a peer responsible for the next partition of the key space, until a peer responsible for the other bound of the range is encountered. There are two problems using this approach. First of all, the underlying network does not always have the information belonging to the next neighboring range partition. The second problem is that the sequential nature of the algorithm creates a possible point of failure in case one of the intermediate peers fails.

In the **Shower algorithm**, the initial query is forwarded in parallel to peers who have a part of the query result. and then this query is forwarded recursively to the other partitions in the interval using each peer's routing table.In the course of forwarding, it is possible that the query is forwarded to a peer responsible for keys outside the range. However, it is guaranteed that this peer will forward the range query back to a key-space partition within the range. This technique gives faster results and less susceptible to node failures but it requires more message exchanges than the min-max algorithm in average.

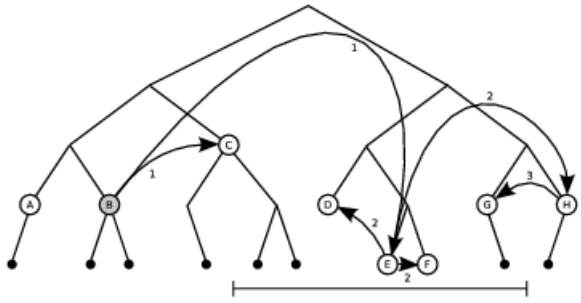


Figure 2.7: Shower algorithm. Source: [1]

2.3 Peer-to-peer network simulators

Most peer-to-peer networks are consisted of thousands of heterogeneous computers, each one with different bandwidth, computing power and typically peers join and leave randomly. There exist global research platforms that support the development of new network technologies such as PlanetLab. PlanetLab is a group of computers throughout the Internet available as a testbed for computer networking and distributed systems research but it has some disadvantages. First of all, it is composed of (at best) a few hundred nodes , a number not large enough to test the scalability of a protocol and secondly, the nodes are rather powerful and homogeneous with good network connectivity which is not representative of the Internet community.

Consequently, experiments and evaluation of networks and algorithms can be a tedious and error-prone process. The techniques used to avoid this, include analytical solutions, simulators and experiments with the actual system. There is no single widely accepted framework for performing experiments on peer-to-peer networks; each individual solution has its own strengths and weaknesses and often focuses on different aspects of the problem.

2.3.1 DHTsim

DHTSim[6] is a discrete event simulator for structured overlays, specifically DHTs. It is intended as a basis for teaching the implementation of DHT

[6]http://www.informatics.sussex.ac.uk/users/ianw/teach/dist-sys/dht-sim-0.3.tgz

protocols, and as such it does not include much functionality for extracting statistics. RPC is implemented as discrete event based message passing within the JVM. Identifiers are assigned randomly. It does not support distributed simulation, nor does it allow for nodes to fail. Simulator scenarios are specified using a simple script file. Churn can be simulated with two script commands which allow a number of nodes to join over a period of time or a number of randomly selected nodes to leave over a period of time. It is written in Java.

2.3.2 P2PSim

P2PSim[7] is a discrete event packet level simulator that can simulate structured overlays only. It contains implementations of six candidate protocols: Chord, Accordion, Koorde, Kelips, Tapestry and Kademlia.

Event scripts can be setup to simulate churn but neither the churn nor the node failure statistics are exhaustive. P2PSim can simulate node failures and both iterative and recursive lookups are supported. Node IDs are generated by consistent 160-bit SHA-1 hashing. Distributed simulation, cross traffic and massive fluctuations of bandwidth are not supported. The C++ API documentation is poor, but implementation of other protocols can be built by extending certain base classes. Custom event generators can also be implemented by extending a base class. The P2PSim code suggests support for a wide range of underlying network topologies such as end-to-end time graph, G2 graph, GT-ITM, random graph and Euclidean graph, which is the most commonly used. P2PSim developers have tested its scalability with a 3,000- node Euclidean ConstantFailureModel topology.

2.3.3 PeerSim

PeerSim is an event-based P2P simulator written in Java, partly developed in the BISON project[8] and released under the GPL open source licence. It is designed specifically for epidemic protocols with very high scalability and support for dynamicity. It can be used to simulate both structured and unstructured overlays. Since it is exclusively focused on extreme performance simulation at the network overlay level of abstraction, it does not regard

[7] http://pdos.csail.mit.edu/p2psim/
[8] http://www.cs.unibo.it/bison

any overheads and details of the underlying communication network, such as TCP/IP stack, latencies, etc. Its extendable and pluggable component characteristics allow almost all predefined entities in PeerSim to be customised or replaced with user-defined entities. For flexible configuration, it uses a plain ASCII file composed of key-value pairs.

Rangesim++: Rangesim++ is a simulator based on PeerSim that supports cycle-driven and event-based simulations. It is implemented by V. Samoladas in C++ , which greatly improves the time of experiments (about 100 times faster than its predecessor which is written in Java). We have used this simulator to get our experimental results for our proposed algorithms. Rangesim++ is described more thoroughly in the next chapter.

Chapter 3

Evaluating Protocols for load balancing

In order to evaluate the performance of our proposed algorithms, we reimplemented some parts of the Rangesim simulator in order to satisfy our needs. In this chapter, we will describe the input data for the simulations, the structure of Rangesim and the alterations we had to make and lastly an extensive discussion on our proposed algorithms.

3.1 Datasets/Querysets

Uniform: This dataset is composed of points distributed uniformly in the whole volume space.

Hypersphere: This dataset contains points distributed on a hypersphere.

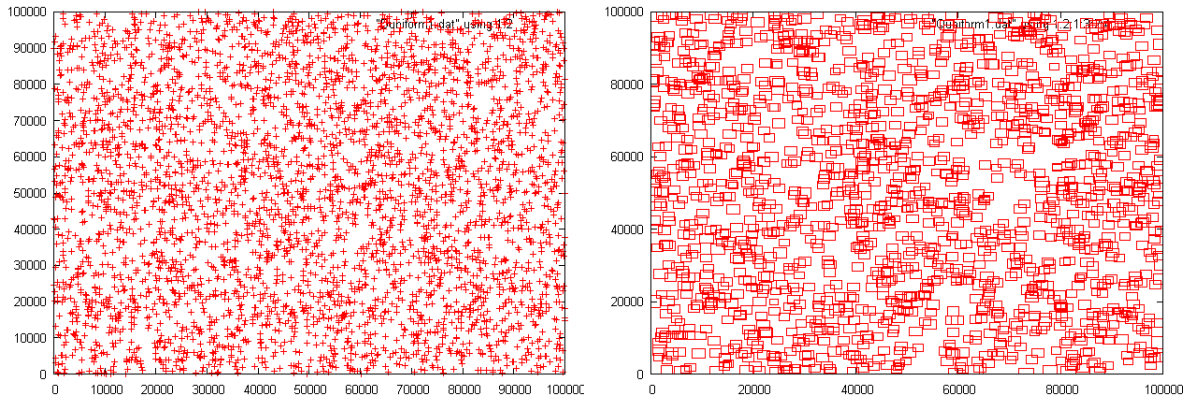
Greece: This dataset was generated from real geographic data retrieved from R-treeportal[1]. It contains random points distributed throughout the road network of Greece.

For each of these datasets, three querysets were created respectively containing 30000 queries. Each query returns approximately 50 keys as an answer. A query is a rectangle, centered around a randomly chosen point of the

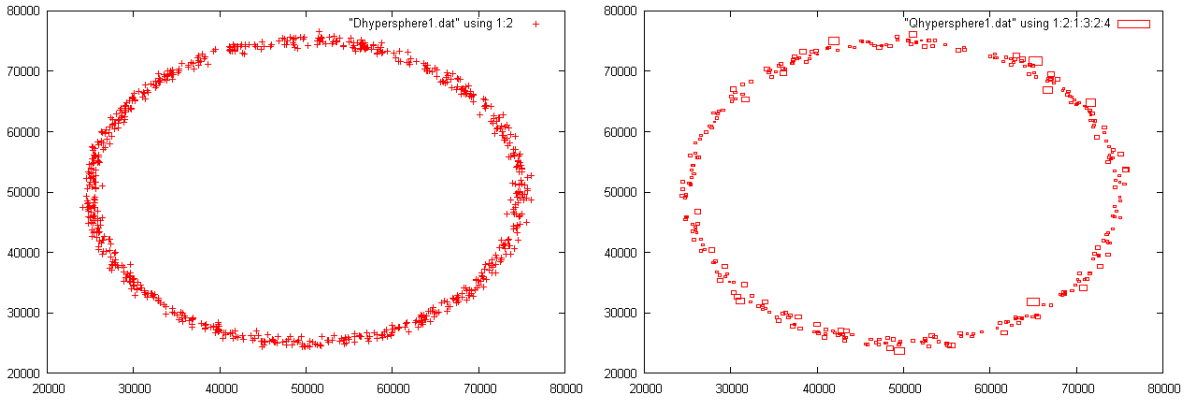
[1]http://www.rtreeportal.org/

dataset. Thus, the distribution of query ranges is similar to the distribution of datasets. Graphical representations (generated using gnuplot) of the datasets and querysets we used are shown below:

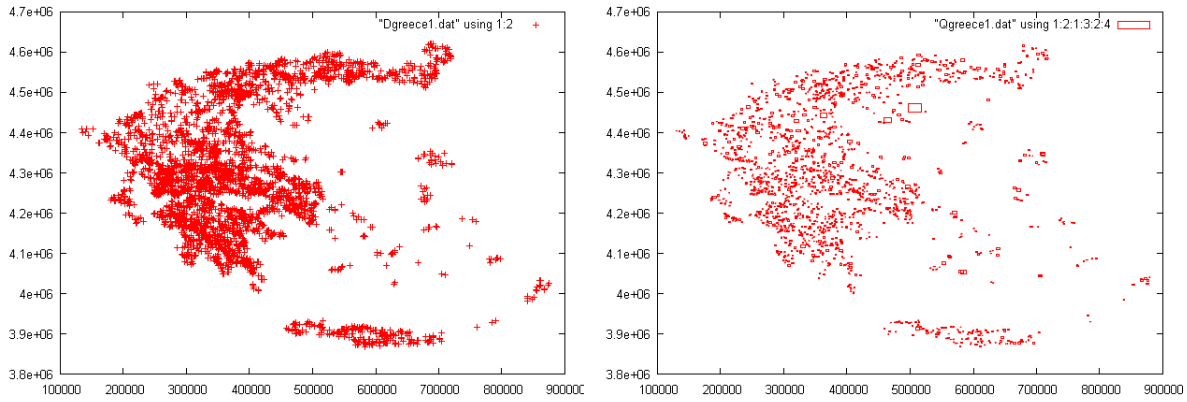
3.1.1 Dataset and Queryset - Uniform distribution



3.1.2 Dataset and Queryset - Hypersphere



3.1.3 Dataset and Queryset - Greece



3.2 Rangesim++ Framework

Rangesim++ is the C++ ported version of Rangesim[1], which is respectively based on PeerSim. It supports some DHTs, including a version of the P-Grid protocol which we used as a basis in our experiments. This framework offers a valuable API with features which are illustrated below:

- A command line parser to feed input to the simulator
- A convenient logging library to export information and results.
- Data point, Range and Queryset classes for Multi Dimensional Range Search.
- Statistical and random number generation facilities.

3.2.1 Cycle-based vs Event-driven simulations

In synchronous analysis, we consider that all peers of the network run independently and operate in cycles (unit time) using a global clock . A process can be enacted from any peer in the network and at each cycle, peers may receive messages from other peers as part of this process. After processing an incoming message, the peer produces a number of outgoing messages to continue the process. In synchronous analysis, the processing capabilities

of every peer are considered homogeneous and contention between peers for resources is not taken into account. This quite simple model is studied by cycle-based simulations. Intuitively, when simulating queries, we keep the network in a fixed state (nodes cannot join or leave the network) and run the processes sequentially. It is a valuable tool to extract information and evaluate algorithms in a straightfoward manner avoiding a more complex environment.

On the other hand, asynchronous analysis takes into account the congestive and competetive nature of realistic networks. In real peer-to-peer networks, peers can have heterogeneous networking and processing capabilities. Thus, their behavior becomes much harder to study and usually requires a fully detailed event-driven simulation. In our experiments we used the cycle-driven approach, as it was adequate for evaluating the performance metrics of our algorithms.

3.2.2 Rangesim++ design structure

The Rangesim++ framework makes use of abstract factories to instantiate the protocol wanted for simulation. A simple configuration file is the argument to the simulator, which takes cares of all input needed to process the experiments. Through this factory design pattern and the flexibility it offers, we can avoid hard coding and thus, benefit from lower recompilation time and more elegant code structure.

The configuration file expects the following input:

- Network protocol to be initialized.
- Definition of datasets/querysets.
- Network size (total number of peers joined before queries start). Multiple network sizes can be defined.
- Number of repetition for the experiment.
- Network construction algorithm and topology.
- Reports path to extract the results.
- Type of results.

3.2.3 Classes & Description

Simulation: The Simulation class contains all information needed to control each experiment. Its data structures contain:

- One vector of Datakeys and one vector of QuerySets.
- A vector named data_index to define a permutation over the Datakeys vector which takes place during network construction.[2]
- A pointer to the root node of the trie.
- A vector containing pointers to all peers of the network.
- Miscellaneous information on network size, topology and statistical tools.

TrieNode: The TrieNode class instantiates nodes in a P-Grid trie. Each node in a trie has information on:

- A bitstring denoting its path from the root.
- A Range corresponding to the volume space this node occupies.
- Pointers to parent and child nodes.
- Number of peers and Datakeys the node's subtree contains (valuable for extracting statistics).

Peer: Each peer is associated with a leaf ot the network trie.It contains:

- A pointer to the corresponding node.
- The uid.
- A vector of pointers to neighbors (routing table).
- Statistical variables regarding received messages, sent messages during build construction and the number of times this peer has been chosen as boostrap peer (diagnostic purposes).

[2]The keys coresponding to each node n of the trie are exactly the keys "pointed at" by range [n.dlow, n.dhigh) of data_index. The author of the framework decided not to sort the Datakeys vector directly because it is too expensive computationally to swap Datakeys.

Process: A Process object is responsible for executing the P-Grid-Z protocol (P-Grid with z-order space-filling curves) of search for a query, starting from the initial peer. It contains a queue of Message objects used for communication throughout the query process.

Message: A Message is sent in the context of a process. It contains a recipient peer, a receipt time and an lprefix (used by the shower protocol).

JoinProcess: A JoinProcess object is enacted each time a new peer is trying to join the virtual network. JoinProcesses are responsible for the construction of the network by routing messages through peers using the construction algorithms we propose. It contains information of the ID of the bootstrap peer and the target which the entering peer is trying to locate.

JoinMessage: A JoinMessage is sent in the context of a JoinProcess. Like a query message, it contains a recipient peer, a receipt time and an lprefix.

Statistics: Statistics collection and reporting facilities.

3.2.4 Simulation process

The original implementation of the P-Grid protocol in Rangesim++ was consisted of three parts:

1. Network construction: First of all, the whole trie starts constructing until meeting the requirements of the experiment (splitting the trie until the required number of nodes is reached), using either volume-balance selection or data-balance selection. In the next step, the peers were added to the network and the routing tables for each peer were generated. When the network construction is finished, the algorithm for permutating the Datakeys in the $data\_index$ vector is executed. The network construction described, is executed in an automatic way.

2. Queries execution: All search queries get executed sequentially, each one enacted at a randomly selected peer. For each query, a process is instantiated which opens the statistic files and takes care of the search and feeds the logger classes.

3. Reports extraction: In this final stage, all information regarding the performance and metrics of the search queries are exported to the equivalent files and the simulation is finalized.

This implementation focuses on the stable state of the system, not on the trasient state where new peers join or existing peers leave the network. This work focuses on the transient state which involves the network construction. In order to evaluate the construction algorithms, we had to reimplement the network construction phase using features of the P-Grid protocol and make the needed alterations on the reporting facilities, which resulted at the so-called PGridJoin protocol. In PGridJoin the steps needed to construct the network are the following:

1. We made the assumption that the network starts with two peers splitting the whole data space in half, each peer having the other one in its routing table.

2. Peers start joining sequentially using the same procedure. When a new peer wants to join, a split point is chosen (depending on the spatial-partioning scheme).

3. A random peer already in the network is chosen to initialize the JoinProcess.

4. JoinMessages are exchanged between peers, using the P-Grid routing algorithms, until the peer containing the split point is found.

5. When this peer is found, we make use of the techniques proposed in this thesis to reach the final peer(called mate).

6. The trienode is now splitted, the mate peer becomes the left child and the new peer becomes the right child.

7. They alter their volume space ranges, with each one containing half the original space.

8. The new peer copies the mate's routing table and each one adds the other one as a new neighbor at the last level.

9. Data_index is updated appropriately.

In the 8th step of this procedure, a problem is observed. When a new peer joins the network, he copies the mate's routing table, therefore, the oldest peers in the network are likely to appear more often in the routing tables of peers. Experiments prove that these peers suffer great load during the network construction and even bigger load during searches, thereupon, we considered it meaningless to include these results in our thesis. P-Grid has a way around these, using randomized algorithms. Every time two peers meet (we do not care why these peers may meet at this point), they exchange their neighbor lists, therefore randomizing, in some scale, the routing tables.

We have implemented an exchange algorithm that takes place in the construction phase when peers exchange JoinMessages. Initially, both peers (sender and recipient) add their neighbors in a common list. In accordance to P-Grid protocol rules, we check sequentially whether a peer is capable of being a neighbor at each bit-level of the sender and we create a second list of candidate peers for each level. The server chooses randomly a peer from this list and substitutes it into the respective level of this routing table. The same procedure occurs to the receiver. We anticipate that, even though the simulations now demand more time to complete (about 7 more hours for each topology experiment), our approach is algorithmically fast, accurate and more importantly it is algorithmically trivial to be applied to a real P-Grid network. In the subsequent experiments, shown in chapter 4, we have seen remarkable improvement concerning our metrics.

3.3 Network construction algorithms with load balancing

We used the Rangesim simulator to run experiments on 6 different network contstruction algorithms. The logic behind these proposed algorithms is straightforward. Each entering peer wants to locate and split a heavily loaded peer while keeping the cost as low as possible. We measure the cost in terms of latency, maximum throughput and message traffic.

Below, we include some terminology and a brief description of the six algorithms we implemented:

- Boostrap: Bootstrap is the initial peer known to the entering node which initiates the JoinProcess. The bootstrap peer is used to pick at random a point in the volume space (as the traditional volume-

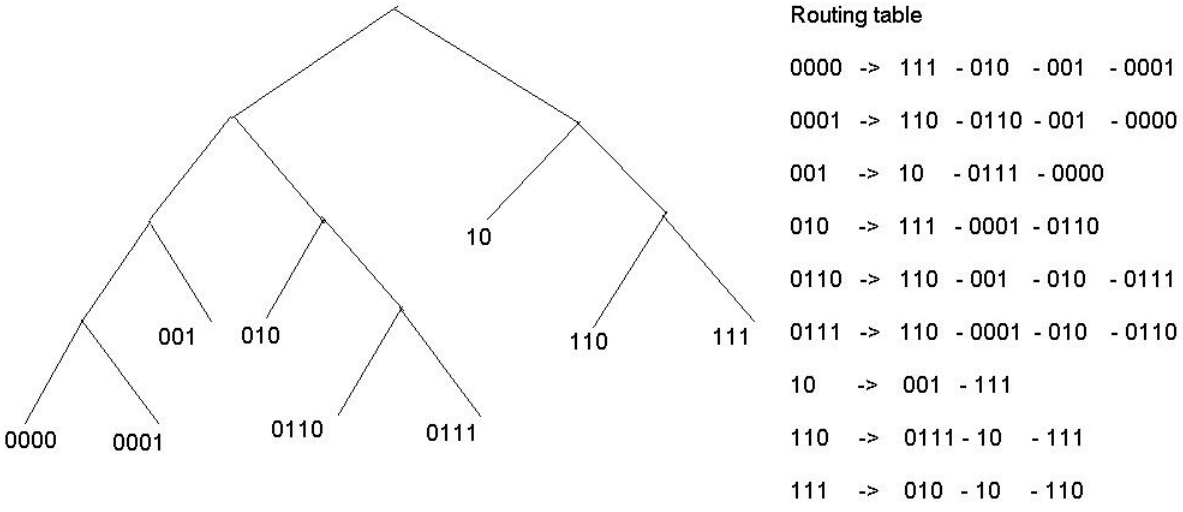


Figure 3.1: An example P-Grid structure and its routing table

balancing protocol does) and then routes JoinMessages until the peer responsible for this point is discovered.

- Mate: The node that the entering peer will finally select and split its area.
- Intermediary: The Bootstrap peer is used to pick at random a point in the volume space; the peer responsible for this point will be called Intermediary. In volume-balanced and data-balanced selection the Intermediary node is the Mate. Throughout the rest protocols, we will use the Intermediary to expand the JoinProcess and acquire information on other peers' data load. In the final step, a Mate will be chosen, one that contains the maximum number of datakeys, and he will be splitted.

Volume-balance: An entering node picks uniformly at random a point in the multidimensional search space, and then selects the node already in the network which is responsible for this point. Practically, the Boostrap peer picks randomly this point and through JoinMessages the node that contains this point is finally found, becomes the Mate and gets splitted. Thus, existing nodes are selected for splitting with

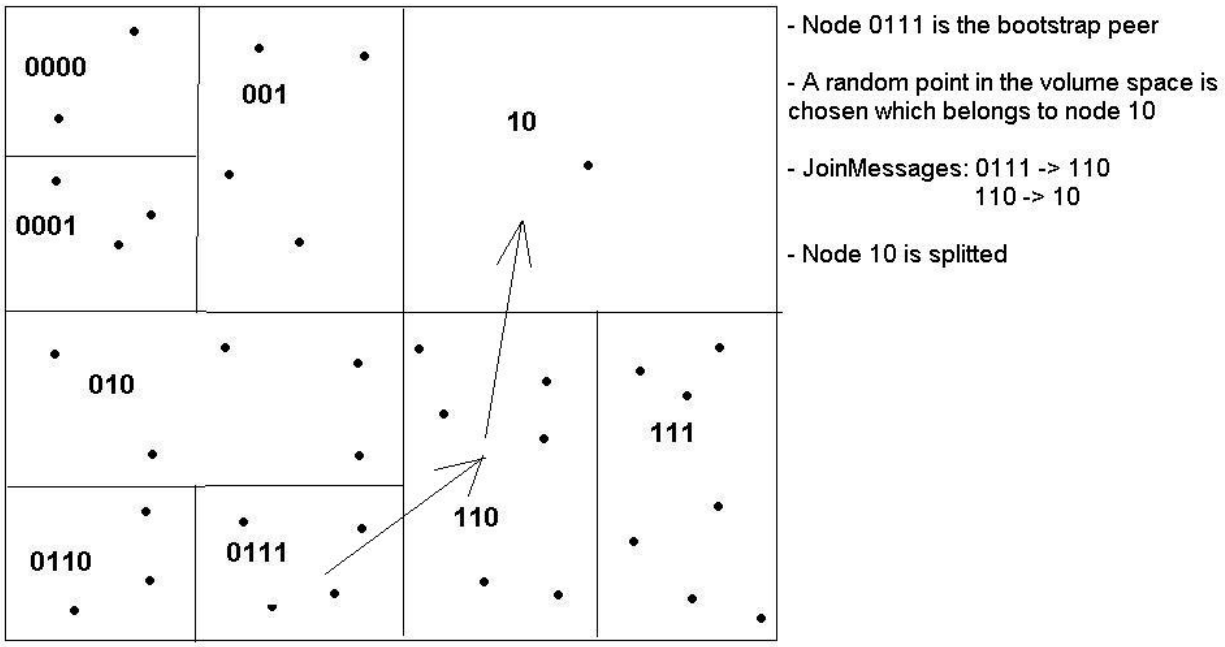


Figure 3.2: Volume balance algorithm

probability proportional to the volume of their assigned space partition. This strategy tends to equalize the volume of space assigned to nodes, but possibly not the load among peers.

Data-balance: An entering node chooses uniformly at random a point from the **indexed dataset** and then selects the node responsible for this point. Practically, the Boostrap peer picks randomly this point and through JoinMessages the node that contains this point is finally found, becomes the Mate and gets splitted. Thus, existing nodes are selected for splitting with probability proportional to the amount of data they store. This strategy tends to equalize the amount of data assigned to nodes, but is up to inquiries whether it equalizes load as well. Implementing this strategy in practice is not straightforward, as the whole indexed dataset (or its distribution) is not accessible to the boostrap peer.

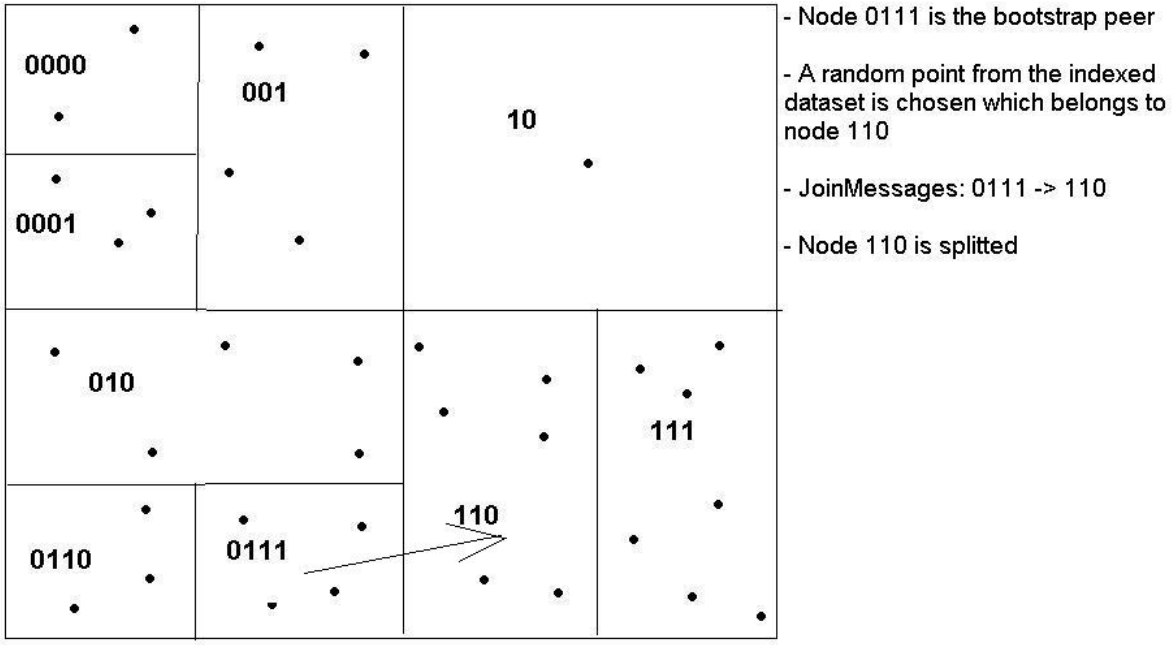


Figure 3.3: Data balance algorithm

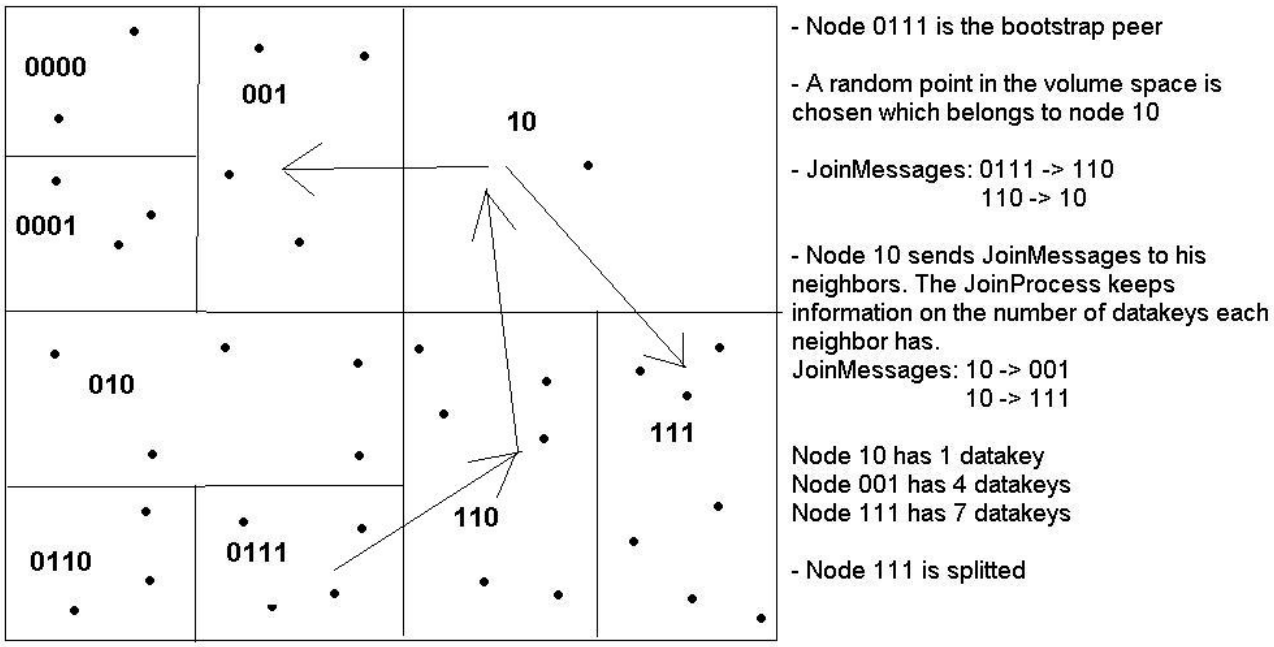


Figure 3.4: Best neighbor algorithm

Best-neighbor: An entering node picks uniformly at random a point in the multidimensional search space, and then selects the node responsible for this point. Practically, the Boostrap peer picks randomly this point and routes the new peer towards the node that contains the point, the Intermediary. This Intermediary node will exchange JoinMessages with all neighbors and will discover the most data loaded neighbor, which becomes the Mate and will get splitted accordingly. The reason we implemented this algorithm is to locate peers that exist frequently on other peers' routing tables and, therefore, receive a high number of messages. Although there is no proven correlation between the number of datakeys in a node and how frequently this node appears in other peers' routing tables, we anticipate that this method will ameliorate the load on these peers. The advantage of this method is that it improves the volume-balanced algorithm by adding the cost of an extra hop in terms of latency (all neighbors receive messages in parallel).

Until-target: An entering node picks uniformly at random a point in the

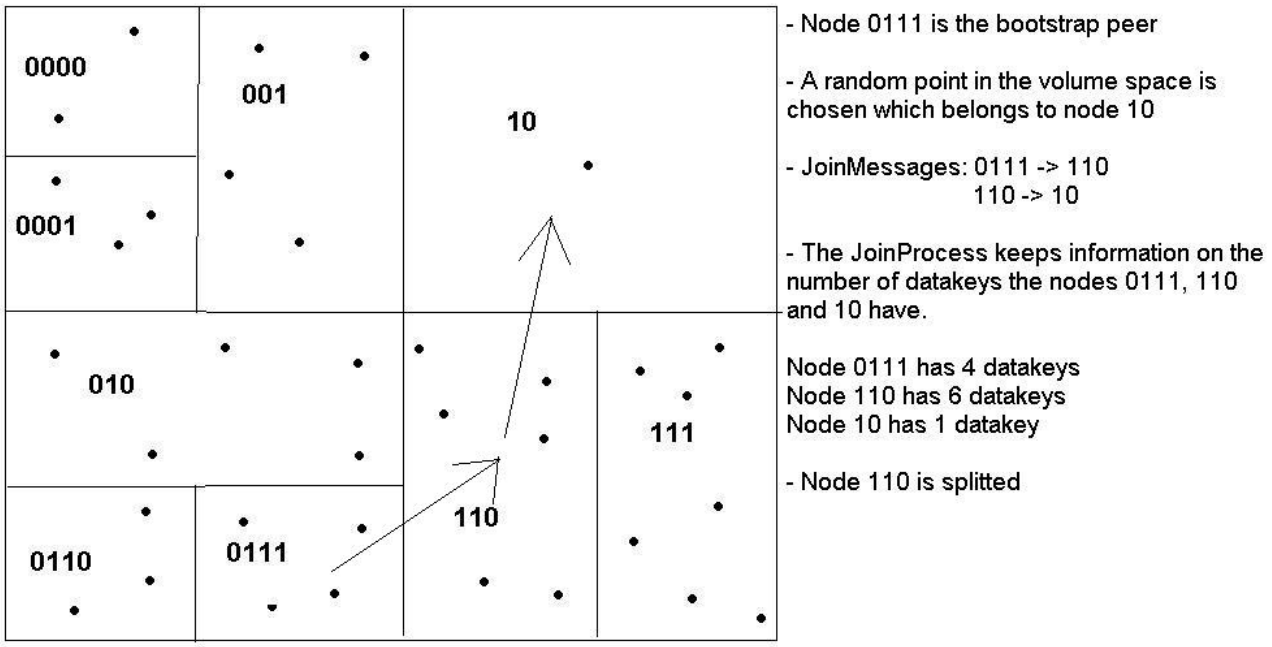


Figure 3.5: Until target algorithm

multidimensional search space, and then selects the node responsible for this point. Practically, the Boostrap peer picks randomly this point and routes the new peer towards the node that contains the point, the Intermediary. During this routing procedure, the JoinProcess will keep track of intersected peers and keep information on the data keys on each peer. Therefore, the Intermediary peer will have knowledge of the most data loaded peer during the routing process. This peer becomes the Mate and will get splitted accordingly. The logic behind this algorithm is simple: we utilize the messages (which are exchanged anyway) during the routing phase in order to obtain information simultaneously, therefore eliminating any extra cost.

Random-walks: An entering node picks uniformly at random a point in the multidimensional search space, and then selects the node responsible for this point. Practically, the Boostrap peer picks randomly this point and routes the new peer towards the node that contains the point, the Intermediary. The search for an efficient Mate happens in two

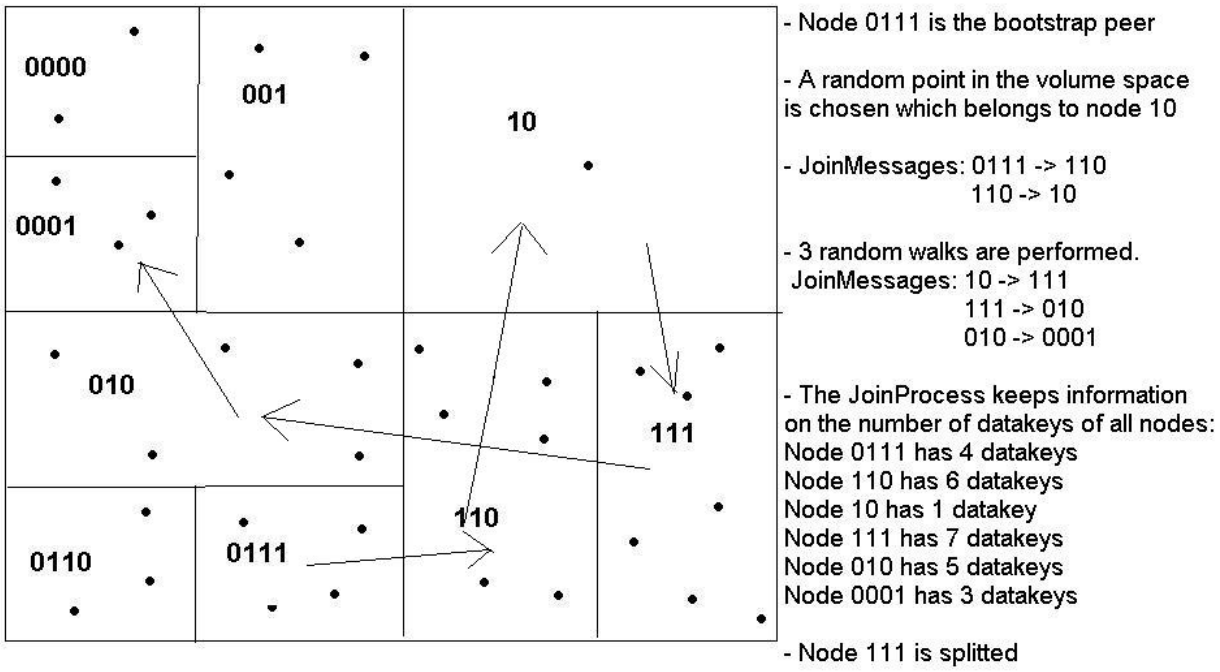


Figure 3.6: Random-walks algorithm

phases; Phase 1 involves the routing until the Intermediate peer is found and phase 2 involves the random walks using $\log_2 n$ number of steps. The JoinProcess keeps track of the data keys on each peer it traverses, during both of these phases. As soon as the random walks are finished, the most data loaded peer, the Mate, is chosen and splitted. The idea is to acquire extra information of the network while requiring approximately $2\log_2 n$ steps to complete the JoinProcess. We hope that the benefits of the information gained during the random walks will outweight the disadvantage of doubling the number of messages needed during the JoinProcess.

Pure-random: During our elementary experiments, the random-walks algorithm presented good results in uniform environments. Therefore our instinct has led us to research further the nature of this algorithm. An entering node picks uniformly at random a point in the multidimensional search space, and then selects the node responsible for this

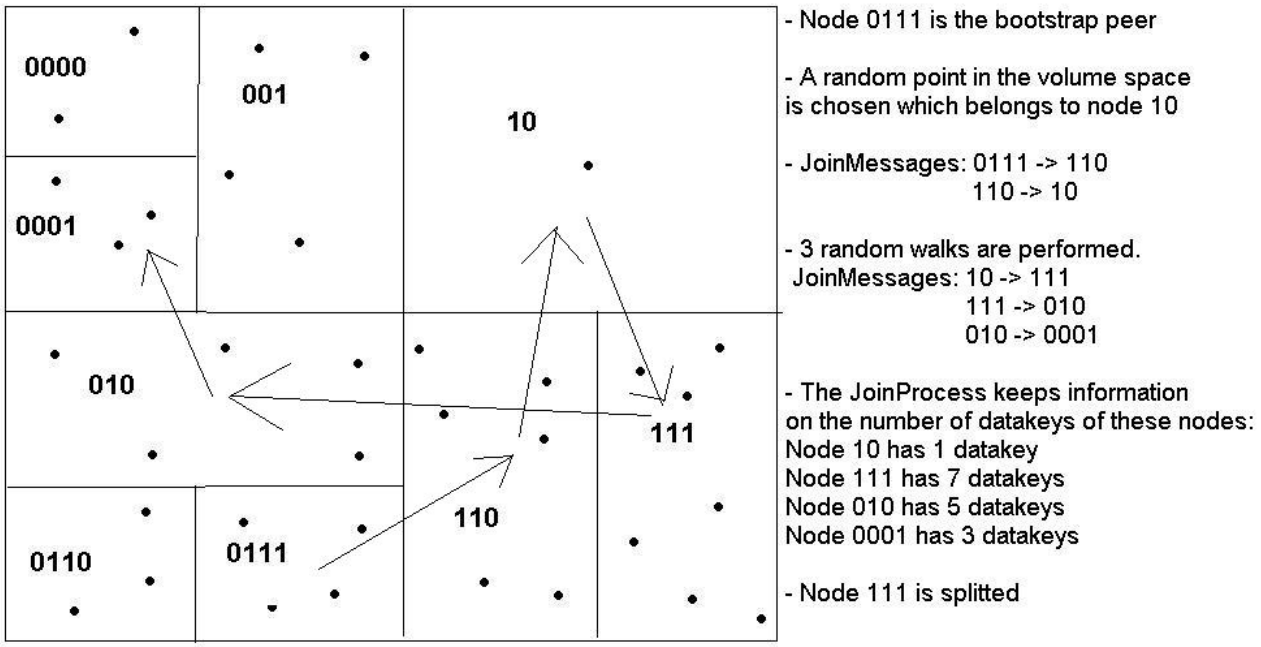


Figure 3.7: Random-walks algorithm

point. Practically, the Boostrap peer picks randomly this point and routes the new peer towards the node that contains the point, the Intermediary. From the Intermediary, a network traversal will start with log2n steps (n denotes the network size) using random walks. During this traversal, the JoinProcess will keep track of intersected peers and keep information on the data keys on each peer. When the random walks are completed the most data loaded peer, the Mate, is chosen and splitted.

Chapter 4

Analysis

In this chapter we will describe the metrics we used to evaluate the performance of our proposed algorithms. Next, we will show the graphs with the results of our experiments and our comments on these.

4.1 Metrics

1. Fairness Index[1]: The primary criteria which evalutes the efficiency of a network topology is the "amount of fairness" among the peers. The Fairness Index is a method to quantify the distribution of datakeys between the network peers and show a metric based on the fairness of it. Definition: f(x) = $\frac{(\sum_{i=1}^{n} x_i)^2}{n\sum_{i=1}^{n} x_i^2}$, $0 \leq f(x) \leq 1$. Equal distribution of datakeys among peers achieces index of 1. Intuitively, the higher the fairness index, the more fair the algorithm is.

2. Average number of JoinMessages per JoinProcess needed, in order to complete the entry of a new peer.

3. Maximum number of JoinMessages for a JoinProcess per network topology.

4. Λmax [1] during network construction.

5. Latency: Number of network hops performed by the search.

[1] http://www.cs.wustl.edu/~jain/papers/ftp/fairness.pdf

6. Λmax [1] during search, using messages received and sent from the most overloaded peer.

7. Average message traffic during search processes.

Blanas et al.[1], introduce a new metric called maximum throughput (Λmax) . Their experimental results show that network traffic is inadequate as a network performance criteria and their proposal is to replace it by this new metric. Maximum throughput is the maximum rate of queries that a P2P network can sustain without any peer becoming overloaded. An appealing feature of maximum throughput is that it only depends on the distribution of message traffic among peers, and does not require expensive computation (such as detailed event- driven simulation).

Definition: $\Lambda max = \frac{1}{max_j \mu_j}$, Intuitively, $\max_j \mu_j$ is the average per-query processing time of the "most loaded" peer in the network. In our implementation, JoinProcesses and Processes are considered queries and we computate the maximum throughput as: Λmax = Number of processes / Number of messages (received or sent) of the most overloaded peer.

4.2 Overall comparison

The following two tables provide to the reader an easier way to compare the results. The values are taken from the graphical results and correspond to a network of 50000 peers, which we feel that is a representative sample. There 3 values for each algorithm with correspond to each dataset: Greece, Hypersphere and Uniform respectively.

- Fairness index
- Average number of messages per JoinProcess
- Maximum number of messages per JoinProcess
- Λmax during network construction

	Fairness Index	Avg. Messages	Max. Messages	Λmax
Volume	0.07 - 0.06 - 0.65	6.94 - 6.94 - 6.94	16 - 15 - 16	67.9 - 62.9 - 61.34
Data	0.54 - 0.67 - 0.67	7.42 - 7.57 - 6.96	18 - 17 - 16	26.1 - 29.0 - 58.00
BestNeighbor	0.17 - 0.18 - 0.90	7.17 - 6.65 - 8.03	17 - 17 - 16	8.93 - 8.02 - 14.84
UntilTarget	0.26 - 0.34 - 0.87	5.20 - 4.49 - 7.02	16 - 17 - 15	13.9 - 11.8 - 63.93
Pure-Random	0.38 - 0.67 - 0.93	19.6 - 18.8 - 21.7	32 - 32 - 31	4.46 - 4.06 - 94.52
Random-walks	0.54 - 0.82 - 0.92	19.6 - 18.8 - 21.7	32 - 32 - 32	4.40 - 4.01 - 87.87

- Latency
- Λmax during queries based on received messages
- Λmax during queries based on sent messages
- Average number of messages per Query

	Latency	Λmax-received	Λmax-sent	Avg. Msg Traffic
Volume	9.20 - 9.30 - 10.20	61.5 - 74.5 - 129.8	59.1 - 68.0 - 118.9	9.89 - 9.87 - 15.19
Data	10.5 - 10.8 - 10.25	46.8 - 43.0 - 127.0	46.2 - 42.7 - 115.6	14.2 - 15.8 - 15.30
BestNeighbor	9.67 - 9.61 - 10.19	51.3 - 72.8 - 110.1	48.7 - 68.2 - 100.5	12.3 - 13.2 - 15.59
UntilTarget	10.1 - 10.4 - 10.22	52.1 - 58.4 - 140.6	49.5 - 56.6 - 126.9	13.5 - 15.1 - 15.58
Pure-random	10.1 - 11.0 - 10.27	55.6 - 50.3 - 426.5	55.2 - 49.8 - 346.5	14.1 - 15.5 - 15.61
Random-walks	10.6 - 10.7 - 10.27	56.3 - 47.4 - 463.7	56.0 - 47.3 - 380.1	14.2 - 15.5 - 15.61

The experiments present some very interesting results. First of all, we managed to improve the fairness of data distribution in our topologies. A remarkable fact is that the Random-walks algorithm has better fairness index than the data-balance algorithm, regarding all datasets.

Until-target has the lowest average number of messages during network construction and scales incredibly well. Volume-balance and best-neighbor algorithms come next in nearly identical results and data-balance is a bit higher. Random-walks algorithm uses approximately $\log_2 n$ more messages than the other topologies, as expected.Our third metric, maximum number of messages per joinprocess show respective results.

Volume-balance exhibits the best maximum throughput during network construction on greece and hypersphere datasets and data-balance comes second. The until-target algorithm comes third in non-uniform distributions and best-neighbor, random-walks algorithms perform much worse. It is quite interesting though that, until-target algorithm has greater results than the

data-balance in uniform distributions. Even more surprising are the results of random-walks and pure-random protocol which surpass the data-balance topology in uniform distributions.

During queries, volume-balance has the best performance in terms of latency and best-neighbor comes next closely. Another interesting notice is that, data-balance is the worst topology regarding latency, exhibiting even worse results than the until-target, random-walks topologies. Additionally, volume-balanced selection maintains the lowest average message traffic, regardless of data distribution in volume space. The other topologies achieve higher average message traffic per query; neither topology shows any significant improvement over another one.

The maximum throughput of the network during queries has ambiguous results:

In non-uniform distributions, volume-balanced selection achieves again the best results with best-neighbor topology coming in the second place, while data-balanced selection and until-target have worse results. In random-walks and pure-random topology, a low amount of peers becomes even more disproportionally overloaded, in terms of received and sent messages during queries. This poor performace makes these topologies the worst choice on non-uniform distributions. We also notice an interesting fact: although best-neighbor and until-target topologies exhibit superior fairness index than the traditional volume-balance, they show worse performance regarding the other metrics, thus, rejecting our initial hypothesis that superior data distribution among peers would result in superior network performance as well, in non-uniform environments.

In uniform distributions, we get the opposite results. Volume-balance and best-neighbor have poor maximum throughput, while, data-balance and until-target show minor improvement, therefore in uniform environments our initial hypothesis is confirmed. The clear winner, in this criteria, are the random-walks and pure-random algorithms, as they approximately four times more efficiently than the traditional volume-balance and the theoritically ideal data-balance algorithm.

In order to explain the inefficient maximum throughput of "more fair" topologies in non-uniform distributions, we reimplemented the simulations to evaluate the following hypothesis: In non-uniform distributions, there is no correlation between volume space and data space. Therefore, we made the assumption that a peer occupying a large volume space, while containing a low number of datakeys (or no datakeys) would rarely be chosen as a mate

by our algorithms. Thus, his volume space remains large and as a result this peer has a high probability of getting picked as an Intermediate peer at the initialization of a JoinProcess, ergo more messages received and sent. In order to validate this hypothesis, we created some statistics on how many times each peer is chosen as boostrap. Unfortunately, the results reject our assumptions. It is up to further research to investigate the cause of this phenomena or present new techniques to overcome the drawbacks of these solutions.

4.3 Scalability

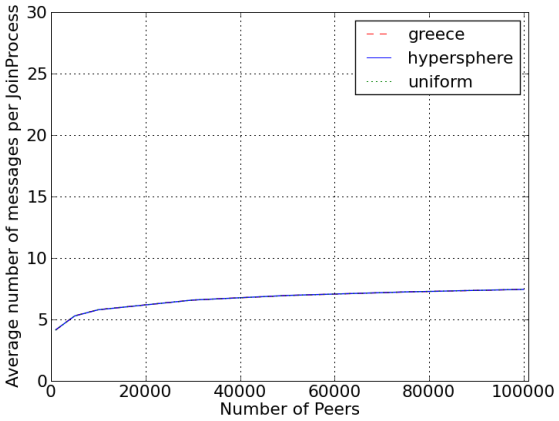
In this section, we the examine the effect of network size on the performance of the protocols we studied. Each topology is ordered in the following manner:

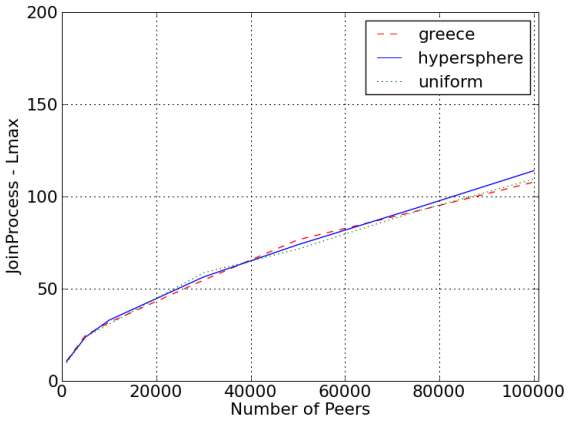
1. Average number of messages per JoinProcess.
2. Maximum throughput in network construction.
3. Maximum throughput during queries, computing sent and received messages.
4. Average number of messages per query.

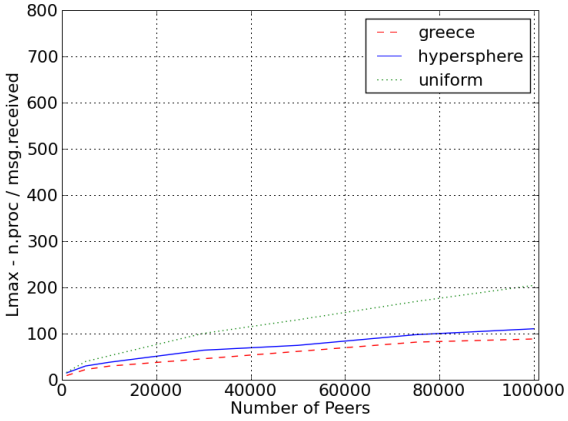
We don't include the plots of the following metrics for the reasons explained:

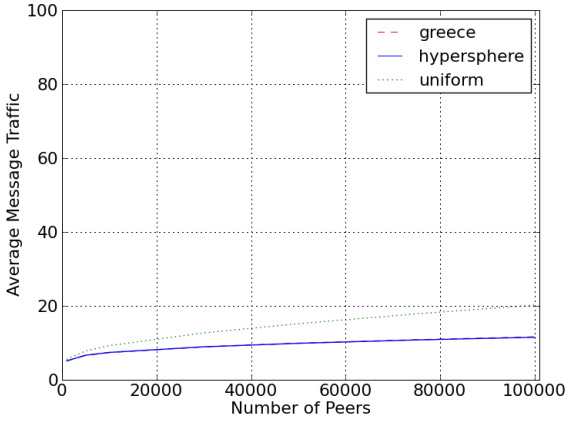
- Fairness Index: The "fairness" of data distribution among peers is not affected significantly by network size.
- Latency: P-Grid has the property of exhibiting nearly equivalent latency regardless of topology. Our experiments validate this property, therefore we omit these plots.
- Maximum number of messages per JoinProcess: Plots are omitted, because this metric can be computed through the maximum throughput for each network size.(Λmax = Number of JoinProcesses / Number of messages (received or sent) of the most overloaded peer)

4.3.1 Scalability of Volume-balancing protocol

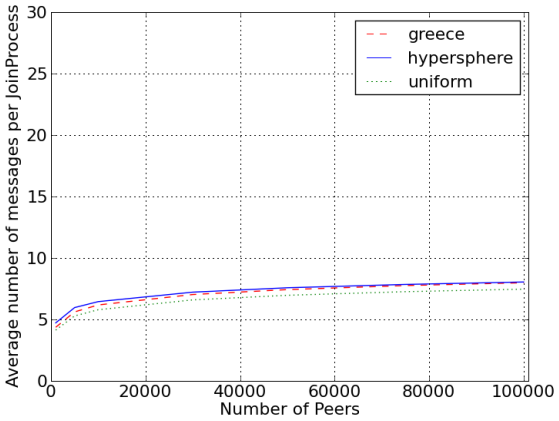


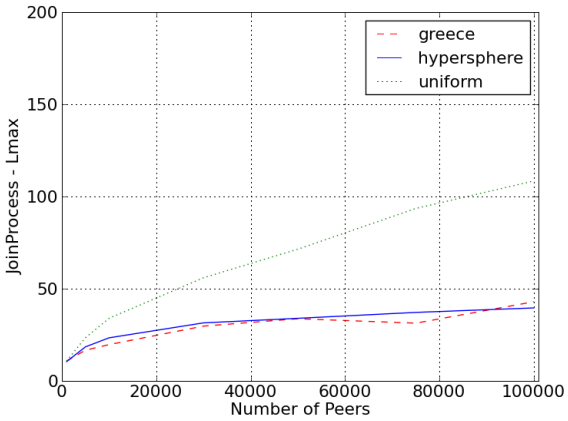


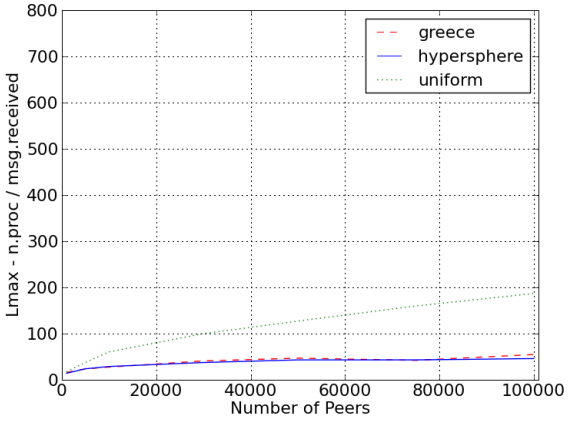


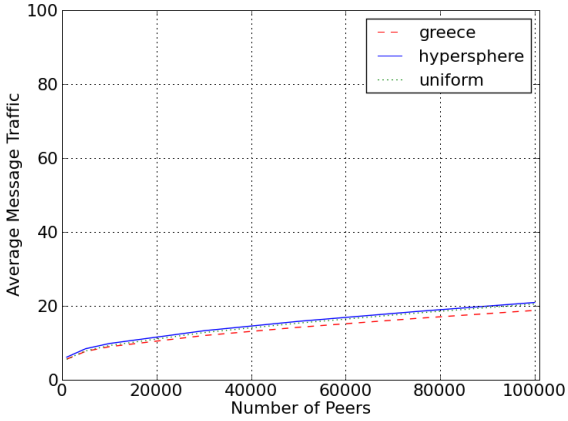


4.3.2 Scalability of Data-balancing protocol

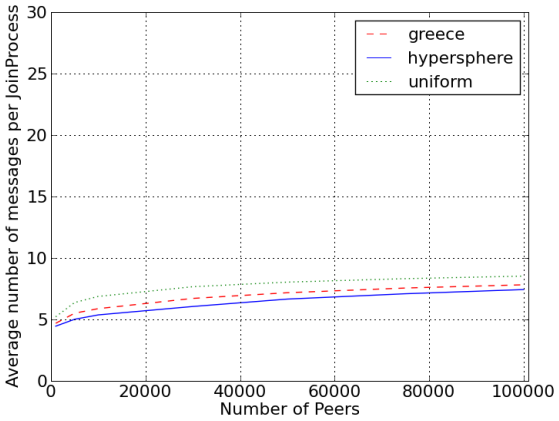


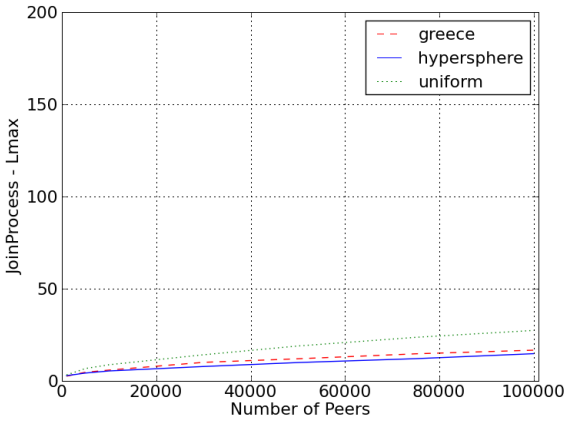


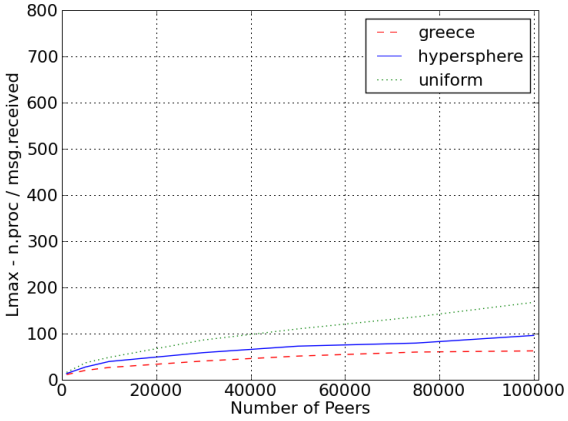


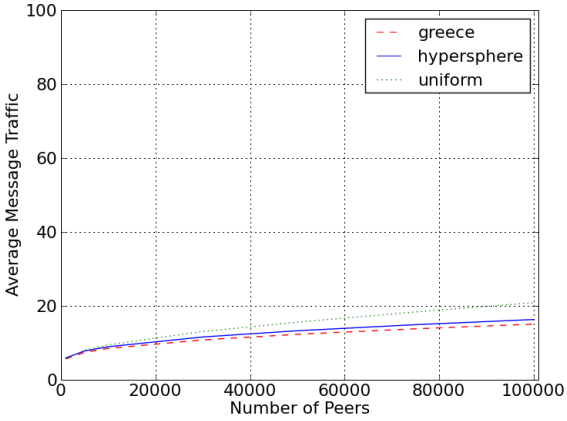


4.3.3 Scalability of Best-neighbor protocol

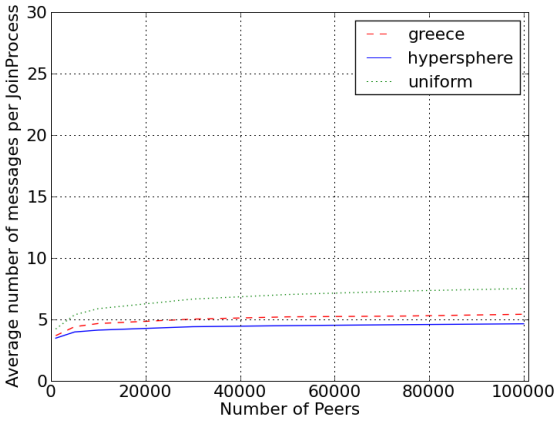


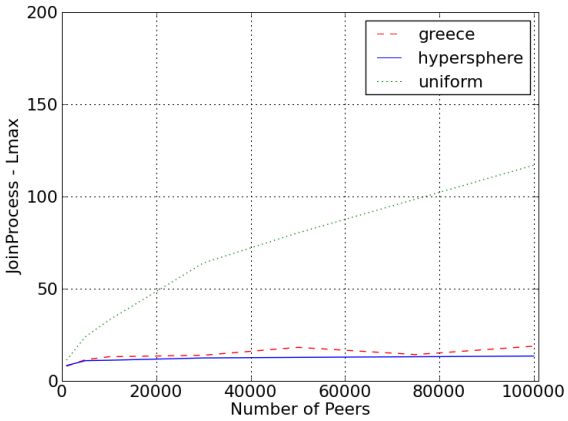


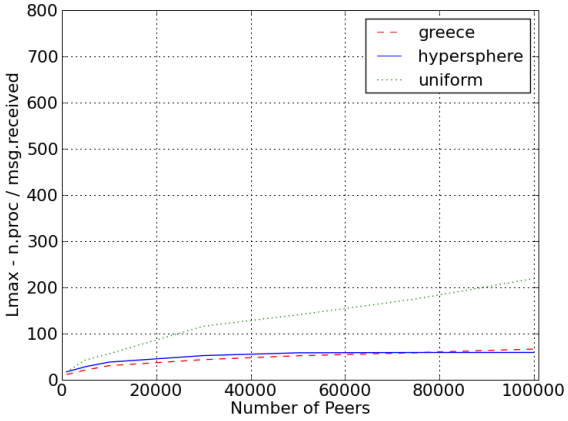


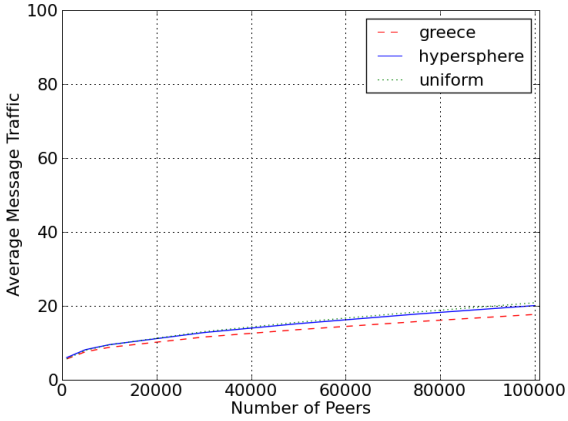


4.3.4 Scalability of Until-target protocol

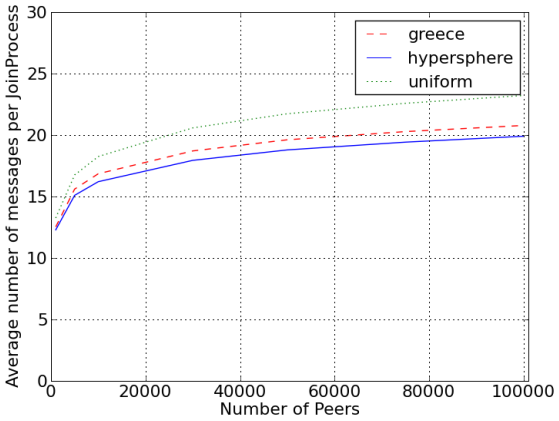


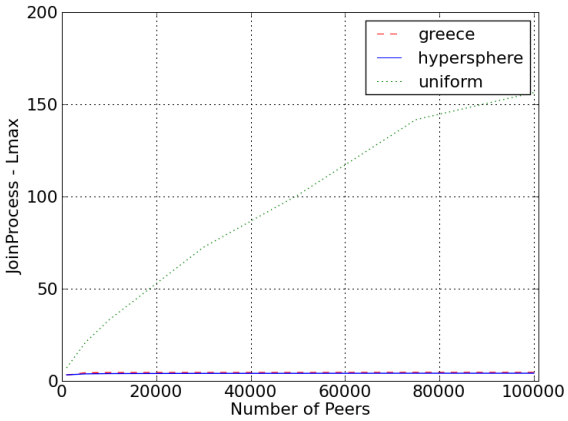


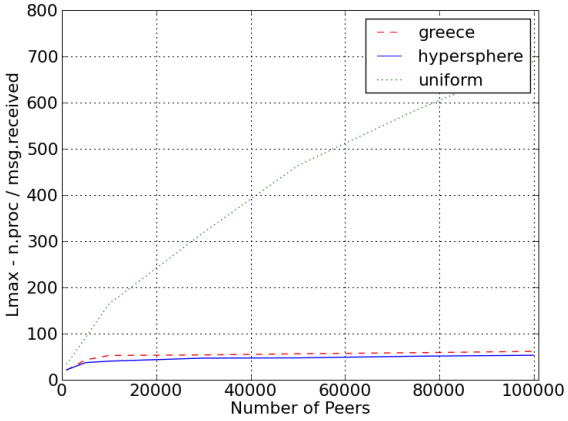


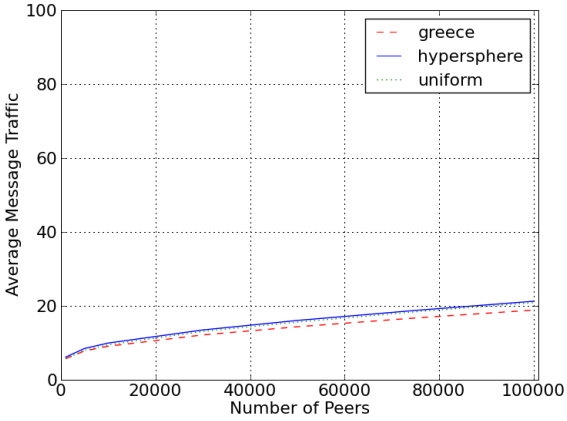


4.3.5 Scalability of Random-walks protocol

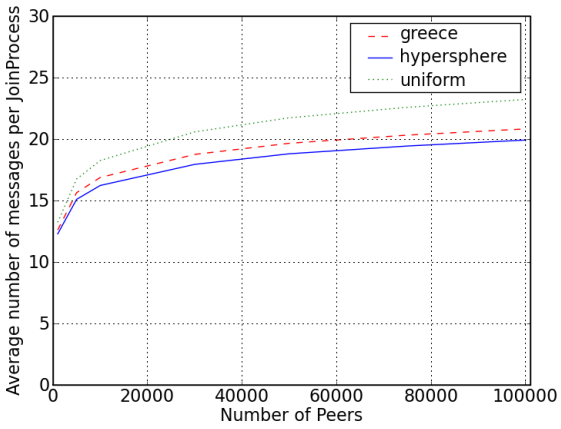


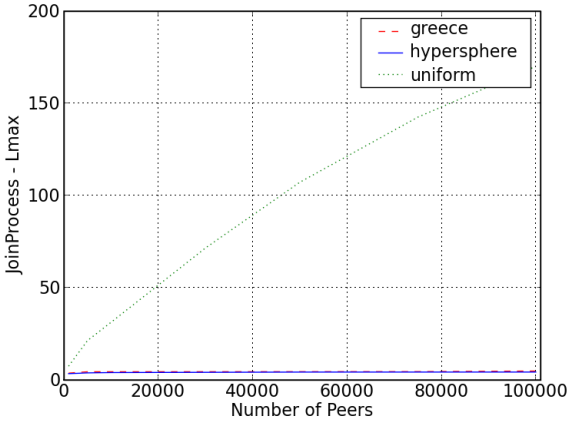


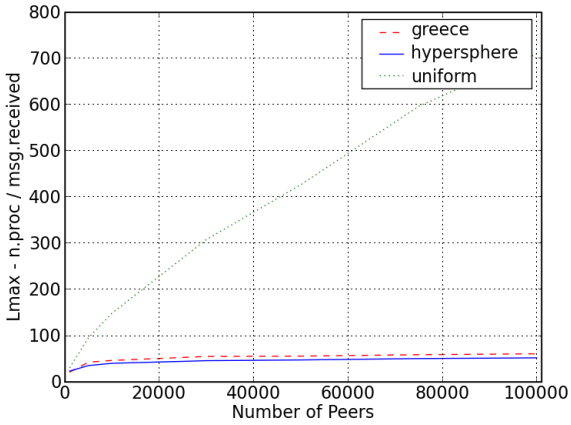


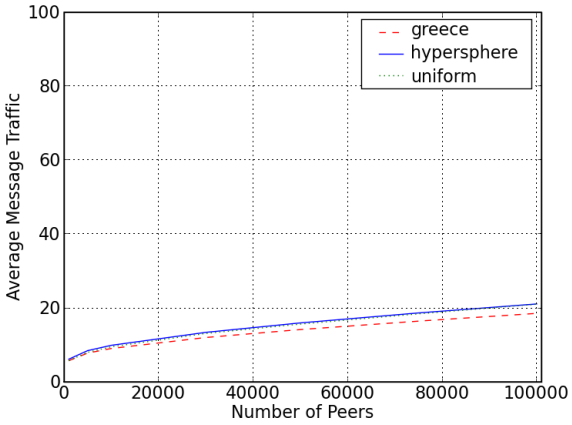


4.3.6 Scalability of Pure-random protocol









Chapter 5

Conclusion

We studied the proposed algorithms and the results are interesting. As discussed in the previous chapter, no proposed solution is best in all aspects. All topologies have pros and cons depending mainly on the uniformity of distribution in the volume space:

- Volume-balanced selection outperforms the other approaches regarding the number of exchanged messages during network construction, latency, average message traffic during queries. The load is distributed well in non-uniform distributions, nevertheless, in uniform distributions it is the worst choice.

- Data-balance selection while giving a theoretically ideal data load balance does not perform as well as the other approaches exhibiting only minor improvements in some metrics. Our initial hypothesis, that data balance may not be the most desirable feature in a relatively "good" network, is confirmed.

- Best-neighbor topology achieves low message trafic in both network construction and query phases of the experiments, and low latency. It also shows good results in non-uniform distributions, nearly as good as the volume-balance algorithm, whereas in uniform distributions it behaves poorly.

- Until-target topology performs adequately, on average, regarding all metrics, without showing significant improments over others on any specific criteria.

- Random-walk and pure-random topologies achieve the maximum fairness on data distribution among peers, even better than data-balanced selection. They perform extremely well under uniform data distributions even though they require more message traffic and latency on average than the other topologies. Nevertheless, the cause of poor performance they present in non-uniform environments is not yet understood.

The presented topologies can perform efficiently under specific datasets. Until-target shows good characteristics under all datasets while random-walks exhibit great improvement under uniform data distribution. The Volume-balancing algorithm, even though it has the worst "fairness" among the protocols, is performing better than all other protocols in non-uniform enviroments.

Considering the definition of the problem, there are several topics that are not covered in this thesis and future work will hopefully give a more complete view of this scientific area.

- Event-driven simulation.
- Workloads having more dimensions.
- Research to discover the cause of poor performance in non-uniform distributions.
- Applying similar techniques to other peer-to-peer protocols like CAN, Chord, MURK.

All in all, we managed to identify the fact that fairness of data distribution is not important in non-uniform distributions as opposed to uniform ones. We have proved that the P-Grid protocol can scale efficiently in uniform distributions. However, the biggest challenge remains to develop and experiment on novel techniques which distribute the load evenly in non-uniform distributions as well.

Bibliography

[1] Spyros Blanas and Vasilis Samoladas. Contention-based performance evaluation of multidimensional range search in peer-to-peer networks. In *InfoScale '07: Proceedings of the 2nd international conference on Scalable information systems*, pages 1–8, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[2] Anwitaman Datta, Manfred Hauswirth, Renault John, Roman Schmidt, and Karl Aberer. Range queries in trie-structured overlays. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 57–66. IEEE Computer Society, 2005.

[3] Prasanna Ganesan, Beverly Yang, and Hector Garcia-Molina. One torus to rule them all: Multidimensional queries in P2P systems. In Sihem Amer-Yahia and Luis Gravano, editors, *Proceedings of the Seventh International Workshop on the Web and Databases (WebDB)*, pages 19–24, June 17–18, 2004.

[4] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, pages 161–172. ACM Press, August 27–31, 2001.