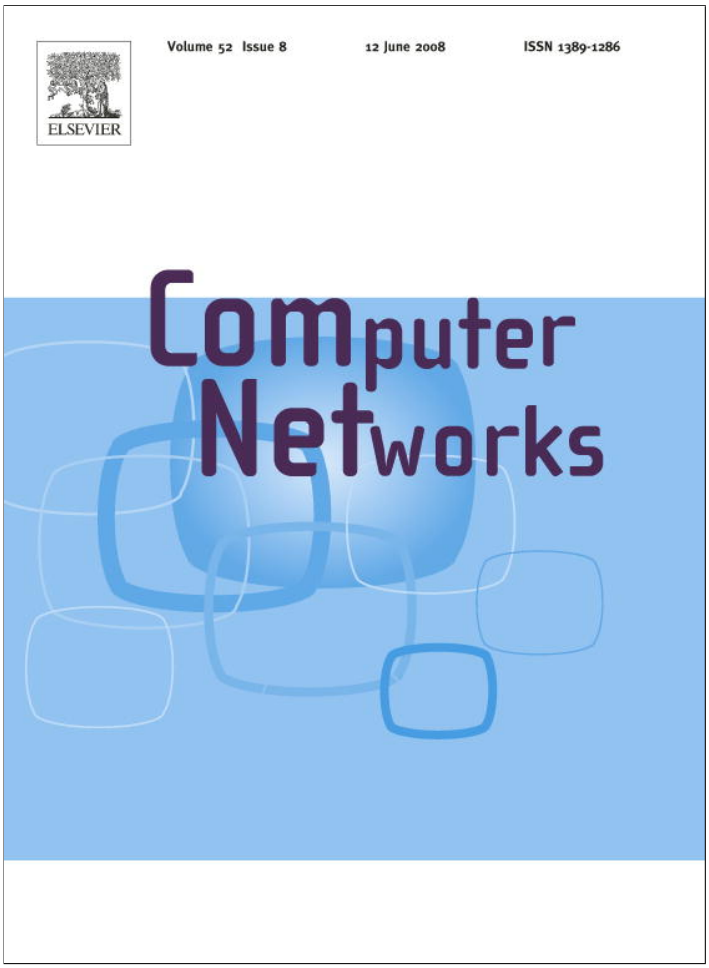

Available online at www.sciencedirect.com
ScienceDirect
Computer Networks 52 (2008) 1617–1633
Computer Networks
www.elsevier.com/locate/comnet

Broker-placement in latency-aware peer-to-peer networks

Paweł Garbacki [a,*], Dick H.J. Epema [a], Maarten van Steen [b]

[a] Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands
[b] Department of Computer Science, Vrije Universiteit Amsterdam, 1081 HV Amsterdam, The Netherlands

Received 9 April 2007; received in revised form 3 February 2008; accepted 12 February 2008
Available online 19 February 2008

Responsible Editor: Dr. I.F. Akyildiz

Abstract

In large peer-to-peer (P2P) overlay networks, nodes usually share resources to support all kinds of applications. In such networks, a subset of the nodes may assume the role of broker in order to act as intermediaries for finding the shared resources. When some notion of distance between nodes such as the internode latency is defined, a brokers may be responsible for maintaining information about resources shared by a group of nodes that are close to each other, with the set of nodes assigned to a broker being determined by the broker's location. In this paper, we present a broker-placement algorithm that finds a suitable location for a new broker when some broker is overloaded in such a way that some of the nodes are reassigned from the overloaded to the new broker. With latency as a metric, an overlay network can be embedded in an Euclidean space $\mathbb{R}^d$, and our algorithm amounts to an optimization problem of selecting a suitable region in $\mathbb{R}^d$ for broker-placement, where a region represents equivalent broker locations. Our algorithm guarantees that if suitable regions exist, one of them will be found. The worst-case complexity of the algorithm is $O(n^{d+1})$ with n the number of nodes that may be assigned to the new broker, which is optimal up to a linear factor in n. We further show a simple optimization that brings down the complexity of the algorithm to a linear function of n in most of the cases. The linear complexity of the broker-placement algorithm is confirmed in a series of experiments on a real dataset. In addition, the performance of our broker-placement algorithm is compared to the performance of a naive approach, and it turns out that in a system with one million servers and one hundred brokers, our broker-placement algorithm is roughly 150 times more efficient.

Keywords: Peer-to-peer; Resource brokerage; Broker-placement; Content delivery network

1. Introduction

With the advent of peer-to-peer overlay networks, we are gradually seeing a new generation of fully decentralized Internet applications that rely on resources spread across the Internet and owned by different organizations and users. In principle, these

* Corresponding author. Tel.: +31 15 2784571; fax: +31 15 2786632.
E-mail addresses: p.j.garbacki@tudelft.nl (P. Garbacki), d.h.j.epema@tudelft.nl (D.H.J. Epema), steen@cs.vu.nl (M. van Steen).

1389-1286/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2008.02.003

networks may consist of tens of thousands to millions of nodes, and so finding and sharing the resources that are suitable for a specific application may turn out to be a formidable task. A standard solution to this problem is to make use of special nodes called *brokers*. A broker collects resource information from a set of nodes so that it can assist other nodes in finding and obtaining resources. Nodes submit resource location queries to their broker and receive results from it. In this paper, we assume that resource brokering is just an extra functionality of a set of selected nodes. These nodes, in addition to their original role as resource providers, cooperate in sharing resources in a peer-to-peer overlay network. Any node that satisfies particular requirements such as high availability or processing power may become a broker. From this perspective, brokers may be seen as super-peers [38].

A broker can be viewed as a representative of a collection of nodes that share resources, and should therefore be able to handle a certain load of requests for resources. Clearly, having only a single broker for an entire overlay network may easily lead to a performance and availability bottleneck. On the other hand, if a broker represents only a few nodes, it can hardly be more effective than when resource requests are sent to each node individually. In conclusion, a broker should represent a group of nodes that is large enough to be effective, but at the same time it should be prevented from becoming a bottleneck because it is representing too many nodes.

In addition to this size requirement, a broker should preferably also represent a group of nodes that are in each other's proximity. In Internet-scale systems, network proximity is important for performance reasons. Not only should clients be serviced from nearby nodes to minimize latency, exploiting proximity is also important for keeping network traffic as local as possible. This locality, in turn, will generally lead to reduced link stress, that is, the number of packets from an overlay network that cross the same link in the underlying physical network. Achieving low link stress is an important design objective for many overlay networks.

In this paper, the broker-selection process is performed in two phases. First a node that will act as a broker is selected. The goal of the second step is to assign nodes that share resources to the newly selected broker. In this paper, we concentrate on the second phase, assuming that the broker node has already been selected. This paper was inspired by our work on Globule [25,26], in which brokers are used for selecting good locations for placing replicas of Web documents.

For the second phase, we take internode latency as our metric for proximity. The second phase then essentially reduces to identifying an appropriately sized group of nodes that are in each others vicinity. To this end, we place nodes in a d-dimensional Euclidean space in which the distance between two nodes is an estimate of the actual latency between those nodes. The viability and practical applicability of this approach has been extensively researched [6,14,15,21,23,33,34]. It has been also shown that in the Internet environment a value of d equal to 6 is sufficiently large to accurately approximate the internode latency [33]. Each node is then assigned to a broker, such that nodes that have been assigned to the same broker are guaranteed to be close to each other.

In this paper, we present an algorithm for placing (selecting a logical location for) a new broker when some broker gets overloaded. This algorithm is of worst-case complexity $O(n^{d+1})$ with n the number of servers that may be assigned to the new broker, which we show to be optimal up to a linear factor in n. However, a simple optimization allows us to reduce the complexity of the algorithm to a linear function of n in most of the cases. The main idea of our algorithm is to identify suitable regions for placing a new broker by intersecting at most d spheres in $\mathbb{R}^d$ centered at the nodes' locations. The radius of such a sphere is the distance between the server at the center and its current broker.

The rest of this paper is organized as follows. Section 2 discusses related work. In Section 3, we present our problem statement, give some definitions, and prove basic facts. Section 4 contains our broker-placement algorithm, and Section 5 discusses its complexity and proposes improvements. The broker-placement algorithm is evaluated in Section 6. We conclude in Section 7 with a brief summary and some remarks on the deployment of our algorithm in a real environment.

2. Related work

A form of resource brokerage is represented by super-peer networks [17–19,38]. Super-peers, which are equivalents of brokers, are selected from among the higher capacity peers. Each super-peer represents a group of peers, which are called weak peers in the context of a super-peer network. The role of a super-peer depends on the type of the P2P network.

In file-sharing networks such as Kazaa [1], Gnutella with ultrapeers [31], and Chord with super-peers [17], super-peers are responsible for locating content on behalf of their weak peers. Skype [3] and Tribler [28] select super-peers from high-capacity nodes and use them as system access points for bootstrapping new peers. The properties of weak peers can be exploited while assigning them to the super-peer nodes. For instance, the self organizing super-peer network based on two-level caching [7,8] exploits the semantic similarity between peer interests by grouping peers with similar interests under the same super-peer. To the best of our knowledge, none of the existing super-peer networks use inter-node latency as criterion for grouping weak peers.

The latency between network nodes determines the service quality when the P2P overlay provides an infrastructure for delivery of relatively small content items such as Web pages. Content Delivery Networks (CDNs) [25,36] based on P2P architectures such as the Globule network [26] replicate content over peers hosting mirrored Web servers. The replicas of Web content are strategically placed closer (in terms of the latency) to the requesters [9,34]. To alleviate the process of selecting a suitable server for the Web content replica, the servers as well as end users issuing the requests for the content are embedded in an Euclidean space where the distance between points approximates the latency between the corresponding nodes [21,24,34]. The abstraction of the latency space makes it possible to compute the concentration points of end users and place replicas on the servers closest to those concentration points [34]. Resource brokers representing groups of servers improve the performance and robustness of locating servers for the same reasons why super-peers improve the performance of locating content.

In this paper, we assume that a broker represents a group of peers (e.g., Web servers) located close to each other in the latency space. Hence, a server situated in the desired area of the latency space can be located by consulting the broker responsible for that area. The concept of dividing the search space into disjoint areas which are maintained by the nodes of the overlay network has been addressed in the design of the content addressable network (CAN) [29,30]. However, CAN requires that the area maintained by a node has a shape of a hyper cuboid (multi-dimensional box). Therefore, CAN cannot be used to directly locate a server closest to a desired point in the Euclidean latency space.

3. The problem of broker-placement

In this section, we formulate the broker-placement problem. First we present the model of the system where distributed resources are located through a brokering layer. Then we introduce the terminology and establish some basic facts used in the rest of the paper. Finally, we formally define the broker-placement problem with the help of the introduced terminology and present an illustrative example of a problem instance and a corresponding solution.

3.1. System model

We consider a network of nodes sharing resources in a peer-to-peer fashion. Examples of these resources are disk space, network link bandwidth, and CPU cycles. The resources are selected based on the latency characteristics of the node where these resources reside. A typical example of a latency sensitive distributed infrastructures are content delivery networks [32]. Nodes in such a network are traditionally called *servers*. Adopting this naming convention, we shall refer to the nodes in our resource sharing network also as servers.

Each server is assigned a point in the *latency space*, which is a d-dimensional Euclidean space where distance between any two nodes approximates the latency between these nodes [34]. Positions in the latency space are used to identify nodes with certain latency characteristics. Taking a content delivery network as an example, it may be required to find a suitable server to host a replica of a web page close to the clients that access this web page. A server closest to the positions of the clients in the latency space can then be selected [34].

As the system size grows to reach millions of servers, efficient location of resources with certain characteristics becomes a scalability issue. A common approach to distributed resource management introduces a bridge between resource requesters and resource providers in the form of a brokering layer. A *broker* maintains information on the resources shared by a set of servers.

With server latency as the primary resource selection criterion, it is reasonable to assume that brokers are assigned servers with similar latency properties, i.e., located close to each other in the latency space. We define a simple rule of assigning servers to brokers (*server-to-broker assignment rule*): a server is assigned to the closest broker in terms of

the distance in the latency space. The area of the latency space composed of the points located closer to a particular broker than to any other broker in the system is called the *broker's responsibility region* (see Fig. 1). This way of defining the server-to-broker assignment has several advantages. First, the correlation between the latency characteristics of the broker and its servers leads to an easy solution of the resource location problem. A resource location request can be simply redirected to the broker closest to the desired location of the resource [10,4,12]. The target broker will then select a resource located at one of the available servers assigned to it. Second, the server-to-broker assignment rule guarantees that each server always has a well-defined broker also in the presence of broker node failures. Namely, when a broker unexpectedly leaves the system, all its servers are automatically reassigned to their second-closest brokers.

Resource brokers are not dedicated machines, but instead are selected from the population of servers. In this respect resource brokerage is just an additional functionality assigned to servers with additional capacities such as more processing power, higher availability, or more bandwidth. The concept of a broker relates to the notion of a super-peer [38]. Since brokers are not directly involved in the interaction between servers and their clients, the latency properties of the brokers do not have influence on the system performance and can thus be ignored in the broker-selection process. To keep the properties of the latency-driven server-to-broker assignment rule while having the flexibility of assigning servers to brokers based on non-latency factors, we allow each broker to freely select its *logical location* in the latency space. The logical location is independent of the broker's *latency-based location* and is used exclusively to identify the set of servers assigned to the broker. The server-to-broker assignment rule can be redefined using the logical location as follows: a server is assigned to the broker with the closest logical location in the latency space. The notion of a broker's logical location is illustrated in Fig. 1. In the rest of the paper, unless explicitly stated otherwise, the term "broker's location" refers to a broker's logical location.

The responsibility of the brokers is to constantly monitor the resources shared by their servers. Monitoring includes keeping track of relatively static information such as available software libraries, but may also include highly dynamic information such as the current disk space usage, CPU utilization, memory consumption, etc. The server monitoring activities combined with the handling of resource location requests can impose a significant load on a broker node. To prevent a broker from becoming overloaded, a mechanism to decrease the load imposed on a broker reaching its capacity limit is required. The flexibility in defining a broker's location suggests a natural solution to this problem. In order to offload an overloaded broker, a new broker can be logically placed in the proximity of (the logical location of) the overloaded broker. As a consequence of the server-to-broker assignment rule, some of the servers currently assigned to the overloaded broker will be reassigned to the new broker. The rest of this paper describes the method of selecting the (logical) location in the latency space of the new broker.

3.2. Formal background

The broker-placement problem can be translated into a geometric problem in Euclidean space where

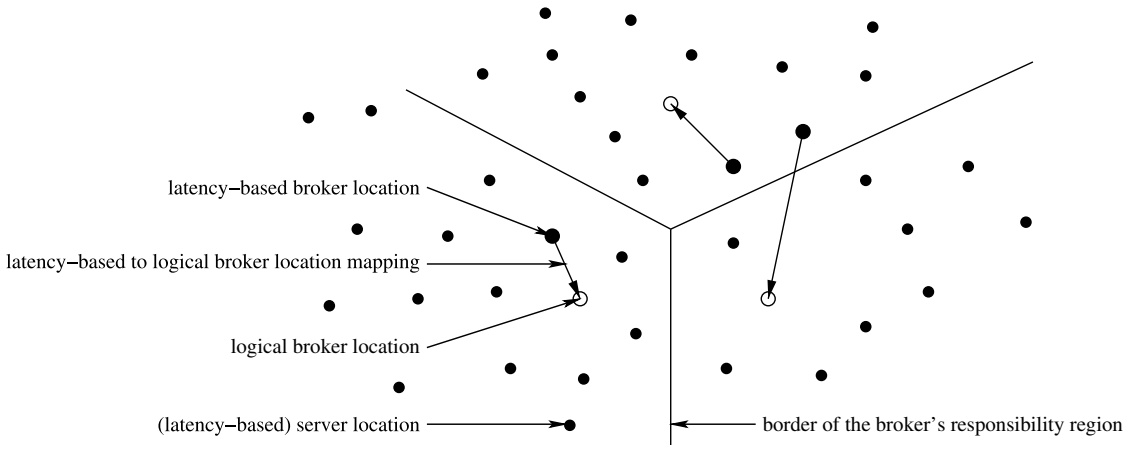

Fig. 1. Node dependencies in two-dimensional latency space.

the servers and the brokers are represented as points. Based on the dependencies between those points, we can identify areas of the space that represent equivalent broker locations. To formally define the translation of the broker-placement problem into a geometric problem, we need to introduce some terminology and establish some basic facts.

We will consider the d-dimensional Euclidean space $\mathbb{R}^d$ with the distance between two points x, y given by $\text{dist}(x,y) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$. The d-dimensional *closed* (*open*) *ball* $\overline{B}_r(p)$ $(B_r(p))$ with radius $r \geqslant 0$ and center p is defined as $\overline{B}_r(p) = \{x : \text{dist}(x,p) \leqslant r\}$ $(B_r(p) = \{x : \text{dist}(x,p) < r\})$. In $\mathbb{R}^1$, a closed ball is a closed interval; in $\mathbb{R}^2$ it is a circle with its interior. A k-dimensional ball in $\mathbb{R}^d$ with $k < d$ is defined as a ball in a k-dimensional linear subspace of $\mathbb{R}^d$. (In this paper, a linear subspace does not have to include the origin.) When we refer to a ball without specifying what kind, we assume that it is a closed ball.

The $(d-1)$-*sphere* $S_r(p)$ with radius $r \geqslant 0$ and center p in $\mathbb{R}^d$ is defined as $S_r(p) = \{x : \text{dist}(x,p) = r\}$. A k-sphere with $k < d-1$ in $\mathbb{R}^d$ is a k-sphere in a $(k+1)$-dimensional linear subspace of $\mathbb{R}^d$. A 2-sphere is an "ordinary sphere" or a single point, a 1-sphere is a circle or a single point, and a 0-sphere is a pair of points or a single point. A $(d-1)$-sphere in $\mathbb{R}^d$ is also called a *hypersphere*.

A set $\{\overline{B}_i : i = 1, \ldots, n\}$ of d-dimensional closed balls in $\mathbb{R}^d$ divides the space into *regions*; a region is defined as $\bigcap_{i=1}^{n} A_i$, where A_i is either $\overline{B}_i$ or B_i^c, with B_i^c the complement of the open ball B_i. So regions do include their boundaries. Now let's assume that each ball $\overline{B}_i$ has a weight denoted by $W(\overline{B}_i)$. We define the *weight* of a region as the sum of the weights of the balls that contain it. That is, for a region $R = \bigcap_{i=1}^{n} A_i$, its weight $W(R)$ is given by $W(R) = \sum_{A_i = \overline{B}_i} W(A_i)$.

A set $P \subset \mathbb{R}^d$ is a *set of characteristic points* of a set of regions R if it contains a point in every region in R.

An example of a set of regions with weights and characteristic points is shown in Fig. 2. Four two-dimensional balls with weights 2, 4, 8 and 16 divide the space into ten regions denoted by $\mathsf{I}, \mathsf{II}, \ldots, \mathsf{X}$. The black dots represent the characteristic points.

In the following sections we make use of some basic facts derived from the definitions above.

Theorem 1. *When the intersection of at least two different spheres in* $\mathbb{R}^d$ $(d > 1)$ *is not empty, it is a k-sphere with* $k < d - 1$.

Proof. Direct consequence of [13], Eq. (3.25). □

Theorem 2. *If S is the non-empty intersection of d* $(d-1)$*-spheres, then S is a* 0*-sphere or we can select* $d - 1$ *out of these spheres such that their intersection is still S.*

Proof. Let $S_1, \ldots, S_n$ be a set of $(d-1)$-spheres. According to Theorem 1, the intersection of S_1 and S_2 is either S_1 (if S_1 and S_2 are the same) or a sphere S_1' of dimension lower than $d-1$. In the first case, we can omit sphere S_1 and the theorem holds. In the second case, consider the intersection of S_1' and S_3. Reasoning similarly, we can either omit S_3 or we end up with a sphere S_2' of dimension lower than the dimension of S_1'. If we did not omit any sphere in the first $d-1$ steps, then the dimension of $S_{d-1}' = S$ is 0. □

Theorem 3. *If R is the set of regions defined by a set of balls, then for every region in R, there exists an intersection of a number of the spheres corresponding to those balls that is contained in this region, or that is a* 0*-sphere that has a point in this region.*

Proof. Let's denote the set of server balls by B, the set of spheres of balls in B by U, and the set of intersections of any number of spheres in U by S. Let for any region $r \in R$, s_r be a sphere in S of smallest dimension that has a point in common with r. We claim that if the dimension of s_r is higher than 0,

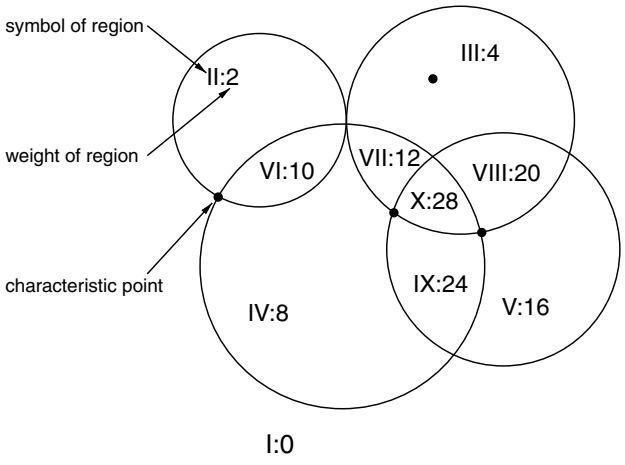

Fig. 2. An example of the regions, the region weights, and a set of characteristic points for a set of four balls in $\mathbb{R}^2$.

then $s_r \subset r$. So suppose this claim is not true; then we can find a region $r \in R$ such that $s_r \not\subset r$. Then sphere s_r contains at least one point inside region r and at least one point outside region r, which means that s_r intersects with the border of r. The border of region r is defined as the intersection of r with spheres in U. In other words, the intersection of s_r with one of the spheres in U has a point on the border of region r. According to Theorem 1, this intersection is a sphere of dimension lower than the dimension of s_r, which is in contradiction with the way sphere s_r is selected. □

3.3. Problem statement

We consider a distributed system consisting of two types of entities: brokers and servers. Each server is assigned to exactly one broker, and each broker is responsible for zero or more servers. Each entity (server or broker) is placed in $\mathbb{R}^d$, where the distance between server nodes is an estimate for the latency. Server locations are actual locations, they cannot be changed. Broker locations are logical locations, and they can be chosen freely. Of course, brokers are nodes so they also have actual locations, but these are irrelevant to our problem.

The server-to-broker assignment is described by a simple rule: each server is assigned to the broker closest to it (taking into account the logical locations of the brokers). Using terminology from computational geometry, we can rephrase this rule by stating that a broker is responsible for all servers situated inside its Voronoi region [22]. The Voronoi region of a broker is defined as the set of points that are closer to this broker than to any other broker in the system.

Each server in the system imposes some *demand* on the broker responsible for it due to requests for resources shared by this server. A broker can handle only a limited number of such requests in a predefined time interval, which we define as the *capacity* of the broker. Server demands and broker capacities are expressed by positive real numbers. We require that the sum of the demands of the servers assigned to a broker does not exceed its capacity.

Now assume that some broker b is overloaded and that a new broker $\hat{b}$ is available to take over the responsibility for some of b's servers. The problem that we then need to solve is to compute coordinates of $\hat{b}$ such that not too few but also not too many servers are assigned to it. More precisely, a solution to this problem has to satisfy two conditions:

1. the sum of the demands of the servers assigned to $\hat{b}$ may not exceed its capacity;
2. the sum of the demands of the servers re-assigned from b to $\hat{b}$ must at least be equal to the excess load of b.

Of course, the new load of $\hat{b}$ may consist of more than only the load shifted from b to $\hat{b}$, as also parts of the loads of other brokers may be shifted to $\hat{b}$. Depending on $\hat{b}$'s capacity, a solution to this problem may or may not exist.

In order to state the problem in more detail, we give the following definitions. The *server ball* $B(s)$ of server s is the ball centered at the location of s with radius equal to the distance between s and its broker, and the *server sphere* of server s is the boundary of $B(s)$. The weight of $B(s)$ equals the demand of server s. We define R as the set of regions defined by all server balls in the system, and $R(b)$ as the set of regions defined by the server balls of all servers assigned to broker b. Because $R(b)$ is defined by a subset of the balls that define R, every region in R is entirely contained in exactly one of the regions in $R(b)$. Each region in R and $R(b)$ is assigned a weight as defined in Section 3.2. Server s will be re-assigned to $\hat{b}$ if and only if $\hat{b}$ is placed inside $B(s)$. If after placing a new broker a server is at an equal distance from the new broker as from its old broker, it is not re-assigned to the new broker.

A region in R represents equivalent placements for the new broker $\hat{b}$ in the sense that the set of servers assigned to $\hat{b}$ depends only on the region but not on the specific location of $\hat{b}$ in it. If we place broker $\hat{b}$ inside some region in R, then the load produced by the servers assigned to $\hat{b}$ equals the weight of this region. Similarly, a region in $R(b)$ represents equivalent placements for the new broker $\hat{b}$ in the sense that the set of servers re-assigned from b to $\hat{b}$ depends only on the region but not on the specific location of $\hat{b}$ in it. If we place $\hat{b}$ inside a region in $R(b)$ then the load produced by the servers removed from b equals the weight of this region. Therefore, an acceptable location of $\hat{b}$ is a point that belongs both to a region in R with a weight at most equal to $\hat{b}$'s capacity (which satisfies the first condition for a solution), and to a region in $R(b)$ with a weight at least equal to b's excess load (the second condition is satisfied).

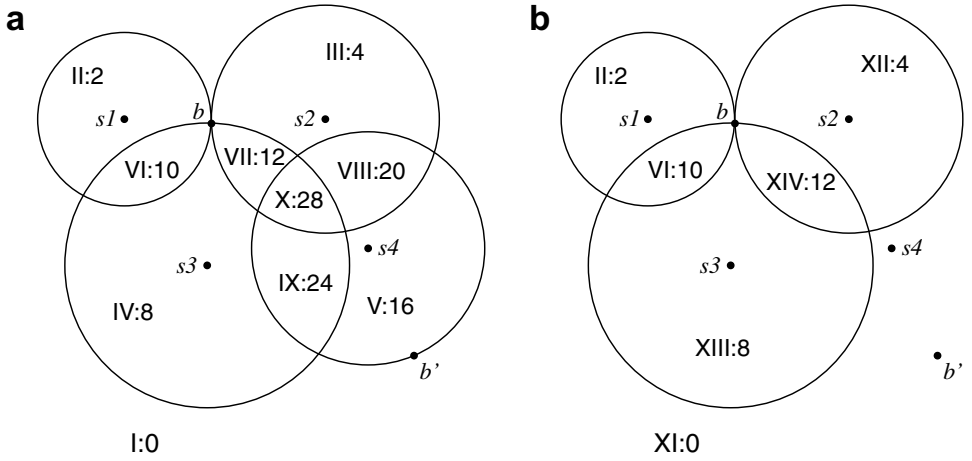

Fig. 3. Division of the space into regions (a) R and (b) $R(b)$.

3.4. An example

As an example, Fig. 3 presents a two-dimensional system that consists of two brokers b and b' with capacities 9 and 19, and four servers $s1, s2, s3, s4$ with demands 2, 4, 8 and 16, respectively. The broker of $s1$, $s2$ and $s3$ is b while $s4$ is assigned to b'. Fig. 3a shows the set of regions R with their weights, which consists of the ten regions (I to X) comprising the intersections of balls $B(s1)$ to $B(s4)$ and their complements. Fig. 3b shows the set $R(b)$ of regions created by the intersections of $B(s1)$, $B(s2)$ and $B(s3)$ (the balls of the servers assigned to b). Note that regions II and VI exist in both R and $R(b)$.

In our example of Fig. 3, broker b is overloaded by an amount of 5 because the total demand of $s1$, $s2$ and $s3$ is equal to 14. Suppose that the new broker $\hat{b}$ has capacity 11. Then because of the first condition in our problem statement that says that the new load of $\hat{b}$ must not exceed its capacity, we see from Fig. 3a that $\hat{b}$ should be placed inside one of the regions I, II, III, IV or VI. In order to satisfy the second condition that says that enough load should be moved away from b, we see from Fig. 3b that $\hat{b}$ should be placed in one of the regions VI, XIII, or XIV. So the proper locations for $\hat{b}$ are all points in regions IV (which is a subset of XIII) and VI.

4. The broker-placement algorithm

In this section, we present our broker-placement algorithm. After a short outline of the algorithm (Section 4.1), we present the algorithm's pseudocode (Section 4.2) and a detailed explanation of the essential element that significantly reduces the complexity of the algorithm (Section 4.3).

4.1. Solution outline

The general idea of the algorithm is simple: when b is the overloaded broker, consider the regions r in R one by one, find for each such r the region r_b in $R(b)$ that contains it, and check whether the weights of r and r_b satisfy the two conditions for a solution. When the first such pair (r, r_b) is found, the algorithm can be terminated, and the new broker $\hat{b}$ can be located at any point in r. Because for a region in R it is easy to find the corresponding region in $R(b)$ that contains it, we are going to explain only the method for finding an element of R with an appropriate weight.

The main problem that needs to be solved is how to efficiently enumerate the elements of R. A naive approach is to consider all possible subsets of servers and check whether their balls intersect to create a region. As with n servers there are 2^n such subsets, this approach has exponential complexity, rendering it infeasible.

Our solution is based on the observation that in order to enumerate the regions in R, it is sufficient to find a set of characteristic points for R (see Section 3.2). An advantage of representing regions with points is that for a particular point it is relatively easy to find the regions that contain this point. Note that a point located on the boundary of a region may belong to more than one region. Our algorithm selects a set of characteristic points by taking suitable points on the intersections of at most d server spheres. Because there is $O(n^d)$ of such intersections and from each intersection we select at most two points, the size of the set of characteristic points found by our algorithm is of order $O(n^d)$, with n denoting the number of servers, which will turn

out to reduce the worst-case complexity of our algorithm to $O(n^{d+1})$.

4.2. The algorithm

The high-level pseudocode of the broker-placement algorithm is presented in Fig. 4. The algorithm is iterative (line 1). Each iteration starts with selecting a subset $\widehat{U}$ of the set U of all server spheres with not more than d elements (line 2). If all possible sets $\widehat{U}$ have already been examined, the algorithm terminates without providing a solution (lines 3–5). In line 6 we compute the intersection $\hat{s}$ of all spheres in $\widehat{U}$. According to Theorem 1 in Section 3.2, $\hat{s}$ is either empty or is a sphere. We continue the current iteration of the algorithm only if $\hat{s}$ is non-empty (line 7).

In lines 8–10 we construct set $\widehat{P}$ by selecting some points from $\hat{s}$. Keeping in mind that every 0-sphere

```
Input:
    b            // the overloaded broker
    b̂            // the new broker to place
    excess_load  // the excess load that needs to be removed from
                 // broker b
    capacity     // the capacity of the new broker b̂
    d            // the dimension of the space
    U            // the set of server spheres
    R            // the set of regions defined by spheres in U
Output:
    p            // a location for broker b̂, or NULL if no suitable
                 // location exists

1  while (p == NULL) do
2      Û := a set of at most d spheres from U which was not considered in any
       of the previous iterations;
3      if (all possibilities for Û are exhausted) then
4          p := NULL;
5          return;
       end
6      ŝ := the intersection of the spheres in Û;
7      if (ŝ ≠ ∅) then
8          if (ŝ is a 0-sphere) then
9              P̂ := the set of all points in ŝ;
           else
10             P̂ := the one-element set containing any point of ŝ;
           end
11         forall (r ∈ R with r ∩ P̂ ≠ ∅) do
12             w := the weight of r;
13             w_b := the weight of the region in R(b) which contains r;
14             if (w ≤ capacity and w_b ≥ excess_load) then
15                 p := any point in the interior of r;
16                 return;
               end
           end
       end
   end
```

Fig. 4. High-level pseudocode of the broker-placement algorithm.

contains at most two points, the size of set $\widehat{P}$ equals either one or two. In line 11 we iterate over the regions that contain one of the points in $\widehat{P}$ (a point in $\widehat{P}$ is a characteristic point of the regions selected in line 11). We will show later in this section that each set in R will be eventually considered in one of the iterations of the algorithm (the union of the sets $\widehat{P}$ across all iterations constitutes a set of characteristic points of the set of regions R). The method of finding regions in R with a point in $\widehat{P}$ as well as computation of the weights of those regions (lines 12 and 13) is described in Section 4.3. The weights are used to determine if a region is a suitable location for the new broker (line 14). If the region satisfies the capacity and excess load requirements, then any point inside that region is a suitable location for the new broker (line 15).

Different executions of the broker-placement algorithm with the same input may result in different solutions. The nondeterminism is caused by the freedom of choice of $\widehat{U}$ in line 2 and the selection of $\widehat{P}$ in line 10. We will show that this nondeterminism does not affect the correctness and the completeness of our algorithm.

The *correctness* of our algorithm – the point p selected in line 15 is a suitable location for broker $\hat{b}$ – is a simple consequence of the conditions in line 14. However, the *completeness* of our algorithm, which means that it does find a solution if one exists, requires explanation. In order to prove completeness, it is enough to show that if the algorithm terminates without finding a solution, every region in R has been considered in lines 12–16. In turn, it is sufficient to show that the points in the union of the sets $\widehat{P}$ selected in all iterations in lines 8–10 forms a set of characteristic points.

Let's take a region $r \in R$. According to Theorem 3 in Section 3.2, there exists an intersection of a number of spheres in U that is contained in r or that is a 0-sphere with a point in r. Theorem 2 adds that we can limit ourselves to the intersections of at most d spheres, and this is precisely what we do when selecting $\widehat{U}$ in line 2. Let's assume that $\hat{s}$ computed in line 6 is the intersection mentioned in Theorem 3. There are two possibilities: $\hat{s}$ is either (a) fully contained in r, or (b) is a 0-sphere that has a point in r. If (a) is the case, then regardless of the point selected in lines 8–10, the region r is considered in lines 12–16. On the other hand, if $\hat{s}$ is a 0-sphere (case (b)), then in lines 12–16 we consider all regions that have a point in common with $\hat{s}$ (r is one of those regions). Consequently, if the algorithm terminates without finding a solution we can be sure that all regions in R have been considered, and so no solution exists.

4.3. The computation of the region weights

In this section, we describe in detail a method for finding the weights of the regions in R containing a specific point p that is on the boundary of all regions it belongs to. Note that indeed all points in $\widehat{P}$ in the algorithm satisfy this condition. The idea presented here is crucial for the efficient implementation of line 15 of the broker-placement algorithm presented in the previous section. The idea amounts to finding which regions we can enter when going from p into all possible directions in a neighborhood of p.

Let's denote by R_p the set of all regions in R to which point p belongs. Let B_1 be the set of server balls that contain p in their interior, and let B_2 be the set of server balls that contain p on their boundary. Because a ball in B_1 contains an environment of p, such a ball contains also all regions in R_p. Now the weight of any $r \in R_p$ can be expressed as $x_1 + x_2$, with x_1 the sum of the weights of the balls in B_1 and x_2 the sum of weights of some of the balls in B_2, respectively. Which balls add to x_2 will be explained below.

To illustrate the meaning of x_1 and x_2, let in Fig. 3a p be the point of the intersection of the server spheres of $s3$ and $s4$ that is located inside $B(s2)$. In this case, R_p contains the regions III, VII, VIII and X, B_1 contains only $B(s2)$, and B_2 contains the balls $B(s3)$ and $B(s4)$. Therefore, the value of x_1 for all regions in R_p is equal to the weight of ball $B(s2)$, which is 4. As both balls in B_2 contain region X, the value of x_2 for region X is 24, which is the sum of the weights of balls $B(s3)$ and $B(s4)$. The values of x_2 for regions III, VII, and VIII can be found in a similar way to be 0 (no balls in B_2 enter in the computation of x_2), and 8 and 16 (only one ball in B_2 adds to x_2).

In general, the elements of B_1 and the value x_1, which is independent of the particular region in R_p, can of course be determined in a simple way.

We describe the method of finding the values x_2 for the regions in R_p in two steps:

1. We define a criterion that allows us to decide for a subset $B_3 \subset B_2$ whether there exists a region in R_p which is contained in all balls in B_3, but not in any of the balls in B_2 which are not in B_3. The

value of x_2 of this region is then the sum of the weights of the balls in B_3.
2. Rather than having to consider in step 1 all subsets of B_3 of B_2, we show how to limit the number of sets B_3 that need to be considered.

Ad 1: We claim that for a subset $B_3 \subset B_2$ there exists a region in R_p which is contained in all balls in B_3, but not in any of the balls in B_2 which are not in B_3, if and only if the centers of the balls in B_3 can be separated from the centers of the balls in $B_2 \setminus B_3$ with a hyperplane containing point p. A point p' located close to p lies inside a ball in B_3 centered at s if the angle between the vectors (p, p') and (p, s) is smaller than $\pi/2$. As a consequence, a point p' lies inside the intersection of the balls in B_3 if and only if their centers and p' lie on the same side of the hyperplane orthogonal to (p, p') and containing point p.

As an example, in Fig. 5 we show a detail of Fig. 3a around the point p, which is the intersection of the spheres of $s3$ and $s4$ that is located inside $B(s2)$. Point p' is located inside $B(s3)$ (respectively $B(s4)$) because the angle between vectors (p, p') and $(p, s3)$ (respectively $(p, s4)$) is smaller than $\pi/2$. Point p' lies inside region X because p', $s3$ and $s4$ lie on the same side of the hyperplane h' orthogonal to (p, p') and containing point p. Point p'' is located inside $B(s4)$ and outside $B(s3)$ because the angle between (p, p'') and $(p, s4)$ is smaller than $\pi/2$ and the angle between (p, p'') and $(p, s3)$ is larger than $\pi/2$. Point p'' lies inside region VIII because p'' and $s4$ are located on the same side and $s3$ on the opposite side of the hyperplane h'' orthogonal to (p, p'').

Ad 2: As we have just shown, if C is the set of the centers of the balls in B_2, then finding the regions in R_p amounts to finding partitioning of C into two subsets which can be separated by a hyperplane containing point p. All possible partitions can be found efficiently in the following way. Let's assume that $C \cup \{p\}$ contains at least d linearly independent points and let C_1 be any non-empty subset of C. If needed, we can add to $C \cup \{p\}$ a few linearly independent points and treat them as the centers of balls with zero weights. Then, if sets C_1 and $C \setminus C_1$ can be separated with a hyperplane containing point p, there exists a separating hyperplane that contains p and at least $d - 1$ points in C. This fact is a conclusion from [37], Corollary 4.2 (the k-set T mentioned in this corollary is our set $C_1 \cup \{p\}$, and p_0 is point p). Now we can consider in step 1 only subsets of B_2 of $d - 1$ elements instead of all possible subsets in order to compute all regions in R_p.

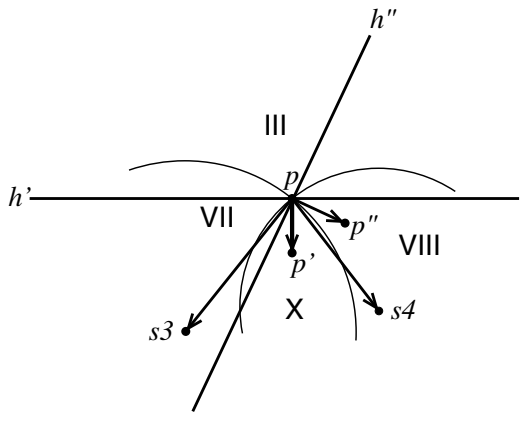

Fig. 5. An illustration of the criterion to determine the set of balls that contain a particular point.

5. Algorithm analysis and improvements

In this section, we assess the worst-case complexity and propose some optimizations of the basis broker-placement algorithm.

5.1. Worst-case complexity

In this section we present the worst-case complexity of our broker-placement algorithm as it depends on the number n of servers. We assume the dimension d of the space to be constant.

We estimate the execution cost of each line of the pseudocode separately. Line 2 requires constant time. We do not specify precisely how to select set $\widehat{U}$, but it is not difficult to provide an implementation that executes in constant time (e.g., by defining an order on U's subsets and considering these subsets in increasing order). Line 6 requires computing the intersection of the spheres in $\widehat{U}$, which can be done in constant time as the size of set $\widehat{U}$ is limited by d. The checks performed in lines 7 and 8 can be implemented as part of the process of computing $\hat{s}$. The operations performed in lines 12–16 require also constant time. The method of finding the weights w and w_b described in Section 4.3 requires time proportional to $n + |B_2|^d$, where $|B_2|$ is the size of set B_2 or, in other words the number of spheres that intersect in a point in $\widehat{P}$. Here, n is the cost of computing value x_1 and $|B_2|^d$ the cost of computing the values x_2 of the regions formed by intersections of balls in B_2. When choosing $\widehat{U}$ in any later iteration, we can skip any subset of the set of the spheres of B_2 because if $\widehat{U}$ is a subset of the set selected in

line 2 in one of the previous iterations, then as set $\widehat{P}$ we can take the set selected in lines 8–10 in that previous iteration. The conclusion is that the (amortized [35]) cost of lines 11–16 is $O(n)$.

We have shown that one iteration of the while loop costs $O(n)$. The number of iterations is limited by the number of possible sets $\widehat{U}$, which is $O(n^d)$ (the number of at-most-d-element subsets of an n-element set). The worst-case complexity of the broker-placement algorithm is therefore $O(n^{d+1})$.

According to [5], page 73, n hyperspheres divide $\mathbb{R}^d$ into $O(n^d)$ regions. Therefore, every algorithm that considers all regions is not more than $O(n)$ times faster than our solution.

5.2. Optimizations

Assuming that the dimensionality d of the latency space is a small constant, e.g., equal to 6 [34], the actual cost of the broker-placement algorithm is determined by the number n of servers. Now we will present a method which allows us to limit the number of servers considered during the execution of the algorithm.

It is intuitively clear that the new broker has to be placed in the proximity of the overloaded broker. Consequently, the servers which are located far from the overloaded broker will not be affected by adding the new broker. These servers can be thus ignored by the broker-placement algorithm. The precise formulation of criterion which servers can actually be ignored requires establishing some facts. Of course, in order for the placement to be effective, at least one server has to be reassigned from the overloaded to the new broker. This condition limits the area of the latency space $\mathbb{R}^d$ where the new broker can potentially be placed in a way described by the following observation.

Observation 1. *At least one of the servers of the overloaded broker b is reassigned to the new broker $\hat{b}$ only if* $\mathrm{dist}(b,\hat{b}) \leqslant 2g$, *where g is the distance between b and the farthest of b's servers.*

Proof. Let's assume that the placement of the new broker results in reassigning server s_1 from b to $\hat{b}$. The server-to-broker assignment rule (see Section 3.1) implies that $\mathrm{dist}(s_1,\hat{b}) \leqslant \mathrm{dist}(s_1,b)$. From the definition of g we get $\mathrm{dist}(s_1,b) \leqslant g$. Applying the triangle inequality [2] leads to the following reasoning $\mathrm{dist}(b,\hat{b}) \leqslant \mathrm{dist}(s_1,b) + \mathrm{dist}(s_1,\hat{b}) \leqslant 2\mathrm{dist}(s_1,b) \leqslant 2g$. □

Observation 1 implies that a server has to be considered by the broker-placement algorithm only if it is closer to the border of the ball with radius g centered at b than to its current broker. When the server and broker locations in the proximity of broker b are distributed evenly in $\mathbb{R}^d$, then the number of servers that need to be considered by the broker-placement algorithm is proportional to the number of servers assigned to a single broker. Under the assumption that the number of brokers in the system grows proportionably to the number of servers, the number of relevant servers that need to be considered in a single execution of the broker-placement algorithm remains constant.

One remaining problem is how to select efficiently the relevant servers. We propose that a server that joins the system registers not only at the closest broker, but also at the brokers that have to consider this server while placing a new broker. Having brokers keep information about the servers that can potentially be affected by a new broker-placement has the additional advantage that all the information required to place a new broker is already available locally at the overloaded broker.

6. Performance evaluation

In this section we assess the efficiency of the broker-placement algorithm presented in Section 4.2 with the optimizations introduced in Section 5.2 using as input a representative dataset of server locations. Our results show that the performance in that case is much better than the worst-case performance.

6.1. Experimental set-up

The basis of the data set used in our experiments contains 315, 373 node locations in a 6-dimensional space of Web clients that accessed one of five Web servers, four in the Netherlands and one in the USA, collected between June 1 and August 26, 2004. The node coordinates were produced by SCoLE [33,34], which essentially runs a GNP [21] instance in cooperation with a number of other hosts acting as landmarks. SCoLE was configured to cooperate with landmarks deployed on 19 different PlanetLab nodes [27].

Although the number of nodes in this set is considerable, it was not sufficient for a profound analysis of the broker-placement algorithm. According to Netcraft [20], an analysis and research company,

the number of Web servers found by its Web Server Survey in October 2004 exceeded 22 million. Our aim was to check the behavior of our broker-placement algorithm in a system of comparable size. Therefore, we employ the following method for expanding the SCoLE data set to the required size: we first pick randomly and uniformly a point from the SCoLE data set, and then draw a new server location from a normal distribution centered at that point. Using this method we can generate sets of server locations of any size while preserving the characteristics of the latency space such as the cumulation of points in certain areas and the average distance between points. The data points were generated for each experiment independently with different random seeds.

We have performed three series of experiments with a simulator that implements several variants of the broker-placement algorithm. To increase the reliability of the obtained results, each experiment has been repeated 10 times with different random seeds and the presented results show the averages across the repetitions. The simulator was deployed on a computer with a Dual-Core AMD processor running at 2.0 GHz with 16 GB of main memory.

The aim of the first series of experiments was to test the scalability of our broker-placement algorithm and its response to the system dynamics. In the scalability test, we start with a system consisting of 10,000 servers and one broker. Then we perform a series of 99 steps in each of which we first add to the system 10,000 servers and then one new broker. The experiment testing the behavior of the broker-placement algorithm in a dynamic setting is also split into 100 steps which are executed in a system with 1,000,000 brokers and 100 servers. In each step, we remove one randomly selected broker and subsequently add a new broker that is placed by our algorithm. The demands of the servers in all experiments are selected randomly and uniformly from the interval $[1, 10]$. The position of the new broker is determined by our broker-placement algorithm. The new broker is meant to offload the broker with the highest load among all the brokers currently in the system. If this highest load is l, we let the excess load in our broker-placement algorithm (*excess_load*) be equal to $0.25l$, and we take the maximal load that we allow to be assigned to the new broker (*capacity*) randomly and uniformly from the interval $[0.25l, 0.5l]$.

The values of the excess load of the overloaded broker and the maximal load of the new broker are motivated as follows. The fraction of the load removed from the overloaded broker should be significant. Adding a new broker to the system is usually expensive, not only because of the cost incurred by the placement algorithm, but also because of registering the new broker in the system, which requires updating routing information. We want to avoid a situation when a broker gets overloaded over and over again because the load removed by the placement algorithm is too low. A fraction of 0.25 of the load removed from the overloaded broker should be sufficient to prevent it from getting overloaded again very fast. On the other hand, the load produced by the servers assigned to the new broker should not be too high. Brokers keep some information about their servers, and this information has to be transferred when a server is reassigned to a new broker. To limit the number of server-to-broker reassignments, we bound the maximal load of the new broker by a fraction of 0.5 of the load of the overloaded broker.

The second series of experiments aims at comparing our broker-placement algorithm with an alternative approach, which relies on the observation that the new broker can usually be placed near the overloaded broker. In this approach we select potential new broker locations until an appropriate point has been found in the following way. A possible location for a new broker is determined by a normal distribution centered at the location of the overloaded broker. The variance of this distribution is equal to the distance between the overloaded broker and its furthest server multiplied by a constant scaling factor. The value of this constant was selected manually based on the observations of the algorithm's behavior on different data sets. This location-selection method adapts to the characteristics of the environment of the overloaded broker.

We run both algorithms on the same data set. In order to create an initial broker–server assignment we could not use any of the evaluated algorithms because this might give preference to one of them. Instead, we use a method widely deployed for load-balancing problems. Servers located close to each other are grouped together using a basic k-means clustering algorithm described in [16]. Brokers are positioned in the centers of the k clusters produced by this algorithm. After the positions of all brokers were computed in this way, we select randomly and uniformly one of them and try to offload it by placing a new broker. The values of the excess load of the selected broker and the maximal

load of the new broker are determined in the same way as in the first series of experiments.

In the third series of experiments we investigate the sensitivity of the broker-placement algorithm to the value of the excess load removed from the overloaded broker (*excess_load*) and the maximal load of the new broker (*capacity*). We modify each of those values while keeping the other constant at the level of $0.25l$ (in the case of the *excess_load*) and $0.75l$ (for the *capacity*), where l is the load of the overloaded broker.

6.2. Performance results

It is theoretically possible that for a particular problem instance no suitable position of a new broker exists. In our experiments however, we did not observe such a situation.

The system evolution in the course of the first series of experiments is visualized in Fig. 6. To enable visualization in two dimensions, we used the multi-dimensional scaling (MDS) [11] technique to reduce the dimensionality of the latency space.

Fig. 7 depicts how the execution time required by our algorithm to place a new broker varies with the system size when we add 10,000 servers and one broker at a time. The presented execution time includes the overhead imposed by the optimization described in Section 5.2. The optimization requires that each server joining the system registers at the brokers with locations that satisfy the condition defined in Observation 1. The results of the experi-

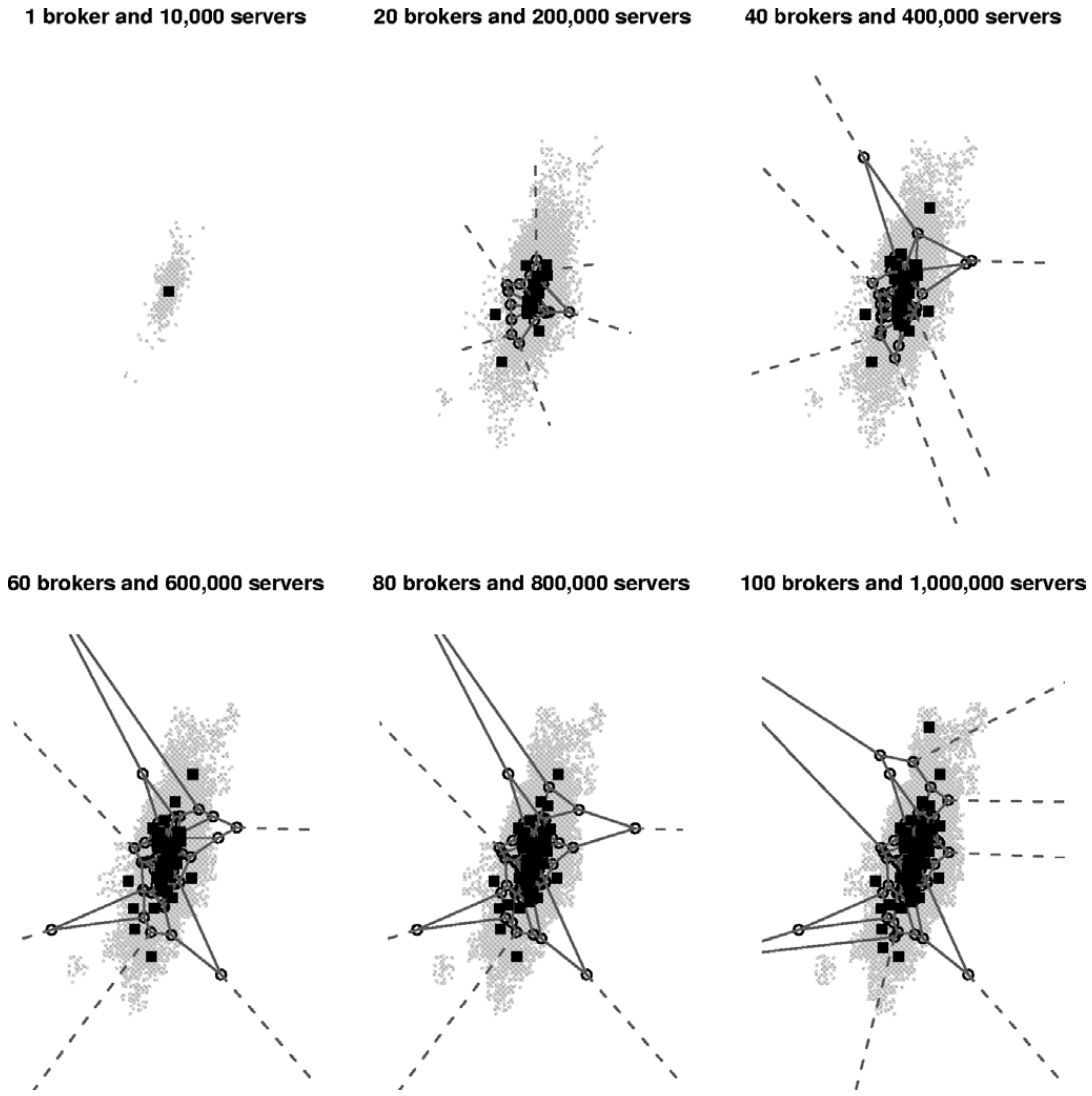

Fig. 6. The visualization in two dimensions of the first series of experiments. Brokers are represented with black squares, servers with small grey dots and borders of broker responsibility regions with lines. Points of intersection of broker responsibility region borders are marked with black circles. Striped lines indicate the borders of broker responsibility regions that stretch to infinity.

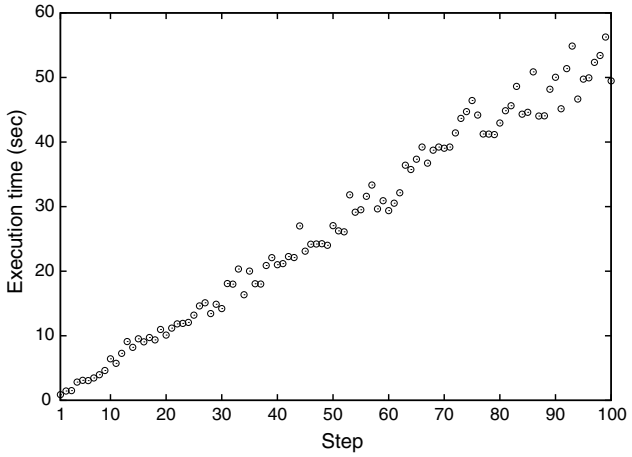

Fig. 7. The execution time of our broker-placement algorithm when adding 10,000 servers and one broker in every step.

ments show that our broker-placement algorithm scales well as the system size increases. We observe a linear correlation between the number of servers in the system and the time needed to compute the position of a new broker.

In Fig. 8 we show how the algorithm reacts to the dynamic changes in a system with 1,000,000 servers and 100 brokers. In each of the steps of the experiment we remove a randomly selected broker and subsequently add a new broker. The new broker is placed to offload the broker with the highest load. The offloaded broker does not have to be any of the neighbors of the removed broker, and so the locations of the removed broker and the new broker do not have to be in any way related. The execution time of the broker-placement algorithm varies between the steps, but the vast majority of the execution time values fits in the 20-second interval around the average of 58 s. Note that the average of 58 s is close to the execution time of step 100 of the experiment with results presented in Fig. 7, in which step the number of servers (respectively brokers) grows to 1,000,000 (respectively 100).

The results obtained in the second series of experiments are presented in Figs. 9 and 10. Those figures show the execution time of the two broker-placement algorithms run in a system with 100 brokers (note the different ranges of the vertical axes). The number of nodes varies from 100,000 to 1,000,000. Both algorithms scale pretty well. However, the random algorithm needs 20 min to place a new broker in a system with 100,000 servers, while our broker-placement algorithm solves the same problem in less than 10 s. For 1,000,000 servers the difference in execution time is measured in hours.

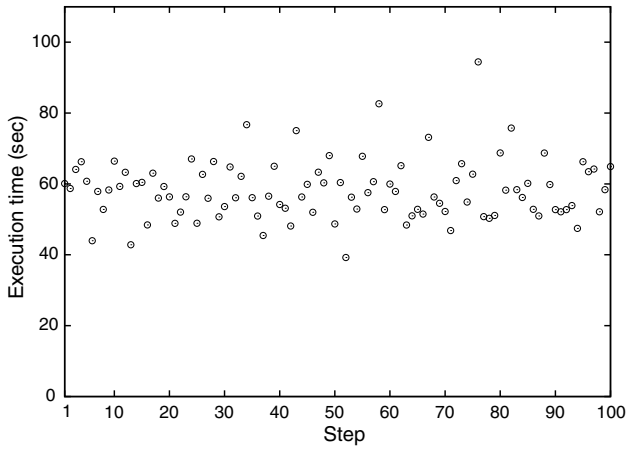

Fig. 8. The execution time of our broker-placement algorithm when one broker is removed and a new broker is added in every step.

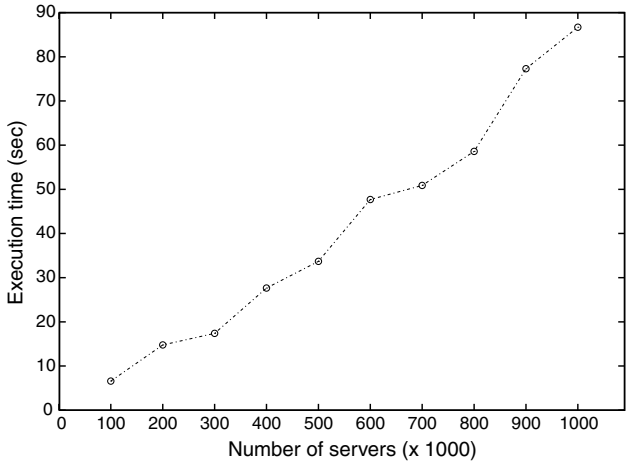

Fig. 9. The execution time of our broker-placement algorithm.

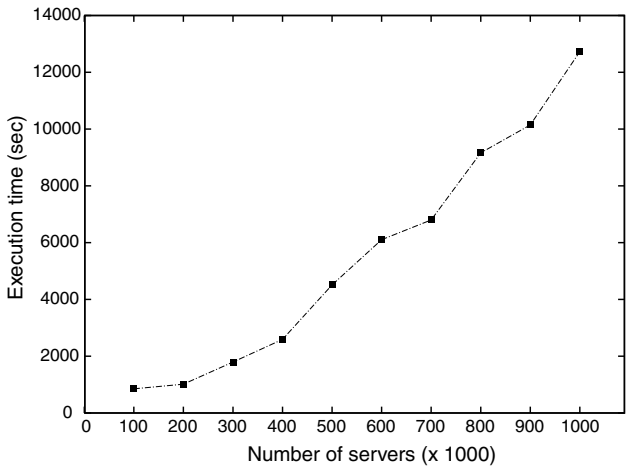

Fig. 10. The execution time of the random broker-placement algorithm.

In both series of experiments, our broker-placement algorithm proved to scale linearly with the system size. This observation is not completely unexpected. In each iteration of our broker-placement algorithm we select randomly a set of spheres and check whether any region that contains a point of their intersection is suitable for broker-placement. The algorithm stops after it finds the first solution. The number of iterations is thus determined by the ratio between the number of suitable regions and the number of all regions. Adding new servers produces new regions, but some of these are also appropriate for broker placement. Apparently, in the neighborhood of the overloaded broker the fraction of regions suitable for broker-placement is more or less constant regardless of the system size. To support this claim we analyze the results obtained for the random algorithm. The random approach selects a point and checks if it is suitable as a location for a new broker. This test costs time proportional to the number of servers in the system. So, because the execution time of the random algorithm grows linearly with the number of servers, the number of points probed by this algorithm is more or less constant. This means that the size of the area of suitable broker locations does not change very much as the system grows.

The results obtained in the third series of experiments are presented in Figs. 11 and 12. The results in these figures show the impact of the *excess_load* and the *capacity* parameter values on the algorithm execution time and the fraction of algorithm executions that do not lead to a solution (a location of the new broker that satisfies the constraints imposed by

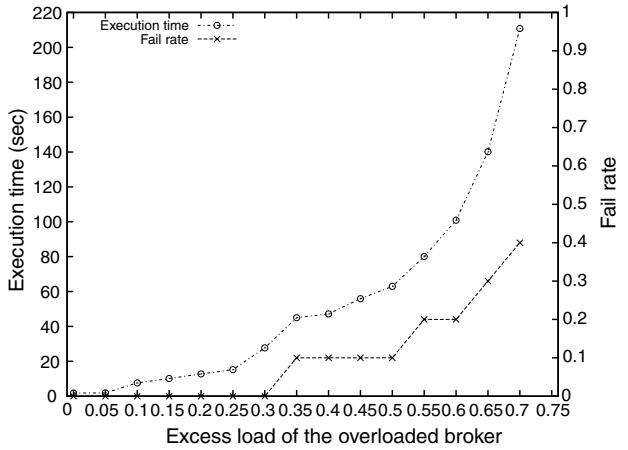

Fig. 11. The execution time and the fraction of problem instances without a solution (fail rate) as a function of the excess load of the overloaded broker. The capacity of the new broker is fixed at 0.75 of the overloaded broker's load.

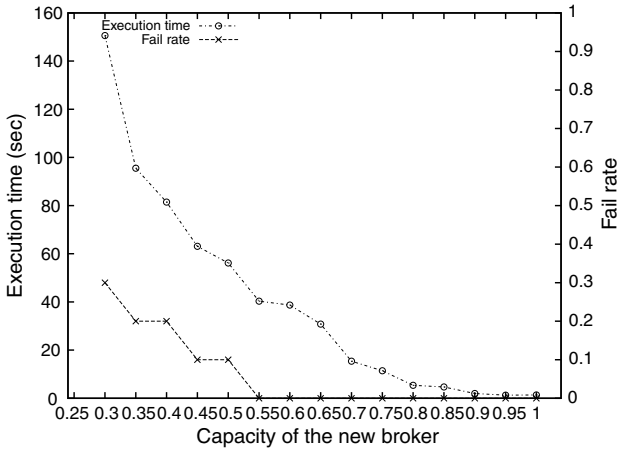

Fig. 12. The execution time and the fraction of problem instances without a solution (fail rate) as a function of the new broker's capacity. The excess load of the overloaded broker is fixed at 0.25 of the overloaded broker's load.

the *excess_load* and the *capacity* values). Note that the completeness property of our broker-placement algorithm proven in Section 4.2 implies that if the algorithm finished its execution without finding a solution, then no solution exists. The results in Fig. 11 have been obtained by running the algorithm for a value of the new broker's *capacity* fixed at 0.75 of the overloaded broker's load, and for a value of the *excess_load* parameter varying from a fraction of 0–0.7 of the overloaded broker's load. The results in Fig. 12 correspond to a symmetric experiment where the *excess_load* is fixed at 0.25 of the overloaded broker's load and the *capacity* ranges from a fraction of 0.3–1 of the overloaded brokers's load. The results presented in Figs. 11 and 12 suggest that both the algorithm performance and the existence of a solution is highly sensitive to the parameter settings. The algorithm execution time as well as the probability of finding a solution can be improved by decreasing the value of the *excess_load* of the overloaded broker or increasing the value of the *capacity* of the new broker.

7. Conclusions and discussion

In this paper we have proposed a solution for a load-balancing problem in P2P overlay networks that amounts to finding a suitable location for a new broker in $\mathbb{R}^d$. Our broker-placement algorithm has worst-case complexity $O(n^{d+1})$, with n the number of servers in the system, which we prove to be optimal up to a linear factor in n. With a simple optimization we have reduced the complexity of the algorithm to a linear function of n in most of

the cases. The results of our experiments performed on a representative dataset show that the worst-case complexity is practically never achieved. The time needed to place a broker in a system consisting of 1,000,000 nodes is less than 2 min. Furthermore, by relaxing the values of the algorithm parameters specifying the excess load removed from the overloaded broker and the capacity of the new broker, the execution time can be significantly reduced at the cost of the placement accuracy. Hence, our algorithm can be used for time-efficient approximation of the desired solution. If the original broker load-related objectives are not satisfied as a result of the inaccuracies in the approximated placement, more than one broker can be added to the system in a sequence of consecutive executions of our algorithm.

We did not specify precisely which node executes the broker-placement algorithm. Because the overloaded broker has the data needed by the algorithm, it seems to be logical that it is the one that finds the location for the new broker. On the other hand, assigning computationally intensive tasks to a broker machine reaching its maximal throughput is not acceptable. Instead of waiting till the very last moment, we can start the offloading procedure much earlier, e.g., when the broker's load reaches a constant fraction of its maximal value. Another option is to send all the data needed by the algorithm (a compressed file with the positions of 1,000,000 servers has a size of 7 MB) to another broker and let it find the location of the new broker. The broker-placement algorithm in the form described in Section 4.2 uses a centralized approach. The iterations of the main loop are however independent of each other, so they can be executed in parallel on different nodes.

References

[1] Kazaa Project home page, <http://www.kazaa.com>.

[2] M. Abramowitz, I.A. Stegun (Eds.), Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables, Dover Publications, 1965.

[3] S.A. Baset, H. Schulzrinne, An analysis of the skype peer-to-peer internet telephony protocol, in: INFOCOM'06, Barcelona, Spain, 2006.

[4] P. Bose, P. Morin, A. Brodnik, S. Carlsson, E.D. Demaine, R. Fleischer, J.I. Munro, A. Lopez-Ortiz, Online routing in convex subdivisions, in: International Symposium on Algorithms and Computation, 2000.

[5] L. Comtet, Advanced Combinatorics: The Art of Finite and Infinite Expansions, Reidel, Dordrecht, 1974.

[6] F. Dabek, R. Cox, F. Kaashoek, R. Morris, Vivaldi: a decentralized network coordinate system, in: ACM SIGCOMM 2004, Portland, OR, 2004.

[7] P. Garbacki, D.H.J. Epema, M. Van Steen, Two-level semantic caching scheme for super-peer networks, in: Tenth IEEE International Workshop on Web Caching and Content Distribution (WCW2005), Sophia Antipolis, France, 2005.

[8] P. Garbacki, D.H.J. Epema, M. Van Steen, Optimizing peer relationships in a super-peer network, in: 27th International Conference on Distributed Computing Systems (ICDCS 2007), Toronto, Canada, 2007.

[9] B.J. Ko, D. Rubenstein, A greedy approach to replicated content placement using graph coloring, in: SPIE ITCom Conference on Scalability and Traffic Control in IP Networks II, Boston, MA, 2002.

[10] E. Kranakis, H. Singh, J. Urrutia, Compass routing on geometric networks, in: 11th Canadian Conference on Computational Geometry, Vancouver, 1999.

[11] J.B. Kruskal, M. Wish, Multidimensional Scaling, Sage Publications, 1977.

[12] F. Kuhn, R. Wattenhofer, Y. Zhang, A. Zollinger, Geometric ad-hoc routing: of theory and practice, in: 22nd ACM Symposium on Principles of Distributed Computing (PODC'03), Boston, 2003.

[13] H. Li, D. Hestenes, A. Rockwood, Spherical conformal geometry with geometric algebra, in: G. Sommer (Ed.), Geometric Computing with Clifford Algebras, Springer-Verlag Telos, 2001.

[14] H. Lim, J.C. Hou, Chong-Ho Choi, Constructing internet coordinate system based on delay measurement, IEEE/ACM Transactions on Networking 13 (3) (2005) 513–525.

[15] E.K. Lua, T. Griffin, M. Pias, H. Zheng, J. Crowcroft, On the accuracy of embeddings for internet coordinate systems, in: Internet Measurement Conference (IMC'05), Berkeley, CA, 2005.

[16] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: L.M.L. Cam, J. Neyman (Eds.), Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, University of California Press, Berkeley, California, 1967.

[17] A.T. Mizrak, Y.-C. Cheng, V. Kumar, S. Savage, Structured superpeers: Leveraging heterogeneity to provide constant-time lookup, in: IEEE Workshop on Internet Applications, San Jose, CA, 2003.

[18] W. Nejdl, W. Siberski, M. Wolpers, C. Schmitz, Routing and clustering in schema-based super peer networks, Technical Report, Learning Lab Lower Saxony, University of Hannover, Hannover, Germany (November 2002).

[19] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, A. Löser, Super-peer-based routing strategies for rdf-based peer-to-peer networks, Web Semantics 1 (2) (2004).

[20] The Netcraft Project, <http://www.netcraft.com>.

[21] T.E. Ng, H. Zhang, Predicting internet network distance with coordinates-based approaches, in: INFOCOM'02, New York, NY, 2002.

[22] A. Okabe, B. Boots, K. Sugihara, S.N. Chi, Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, second ed., Wiley, Chichester, 2000.

[23] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, S. Bhatti, Lighthouses for scalable distributed location, in: Second

International Workshop on Peer-to-Peer Systems, Springer-Verlag, Berlin, 2003.
[24] M. Pias, J. Crowcroft, S.R. Wilbur, T. Harris, S.N. Bhatti, Lighthouses for scalable distributed location, in: Second International Workshop on Peer-to-Peer Systems (IPTPS'03), Berkeley, CA, 2003.
[25] G. Pierre, M. Van Steen, Design and implementation of a user-centered content delivery network, in: Third IEEE Workshop on Internet Applications, San Jose, CA, 2003.
[26] G. Pierre, M. van Steen, Globule: a collaborative content delivery network, IEEE Communications Magazine 44 (8) (2006).
[27] The Planet Lab Project, <http://www.planet-lab.org>.
[28] J. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D.H.J. Epema, M. Reinders, M. Van Steen, H. Sips, Tribler: a social-based peer-to-peer system, Concurrency and Computation: Practice and Experience 20 (2008) 127–138.
[29] S. Ratnasamy, P. Francis, M. Handley, R. Karp, A scalable content-addressable network, in: ACM SIGCOMM, San Diego, CA, 2001.
[30] S. Ratnasamy, M. Handley, R. Karp, S. Shenker, Topologically-aware overlay construction and server selection, in: INFOCOM'02, New York, NY, 2002.
[31] A. Singla, C. Rohrs, Ultrapeers: Another step towards gnutella scalability, <http://www.limewire.com/developer/Ultrapeers.html>.
[32] S. Sivasubramanian, M. Szymaniak, G. Pierre, M. van Steen, Replication for web hosting systems, ACM Computing Surveys 36 (3) (2004) 291–334.
[33] M. Szymaniak, G. Pierre, M. van Steen, Scalable cooperative latency estimation, in: Tenth International Conference Parallel and Distributed Systems, IEEE Computer Society Press, Los Alamitos, CA, 2004.
[34] M. Szymaniak, G. Pierre, M. van Steen, Latency-driven replica placement, IPSJ Journal 47 (8) (2006).
[35] R.E. Tarjan, Amortized computational complexity, SIAM Journal on Algebraic Discrete Methods 6 (1985) 306–318.
[36] A. Vakali, G. Pallis, Content delivery networks: status and trends, IEEE Internet Computing 7 (6) (2003).
[37] U. Wagner, On the number of corner cuts, Advances in Applied Mathematics 29 (2002) 152–161.
[38] B. Yang, H. Garcia-Molina, Designing a super-peer network, in: 19th International Conference on Data Engineering (ICDE'03), Bangalore, India, 2003.

Pawel Garbacki is a Ph.D. candidate at Delft University of Technology, Delft, The Netherlands. He holds two MSc titles in Computer Science from Vrije Universiteit Amsterdam and Warsaw University, both obtained in 2003, and a BSc title in Mathematics from Warsaw University obtained in 2001. In the summer of 2005 and 2006 he pursued research internships at IBM T.J. Watson Research Center, Yorktown Heights, NY, and during the summer of 2007 he worked at Google in Zurich, Switzerland. His research interests include data distribution protocols for peer-to-peer networks and resource management in grid environments.

Dick H.J. Epema received the MSc and Ph.D. degrees in mathematics from Leiden University, Leiden, The Netherlands, in 1979 and 1983, respectively. From 1983 to 1984, he was with the Computer Science Department, Leiden University. Since 1984, he has been with the Department of Computer Science, Delft University of Technology, where he is currently an Associate Professor in the Parallel and Distributed Systems group. During the academic year 1987–1988, the fall of 1991, and the summer of 1998, he was also a visiting scientist at the IBM T.J. Watson Research Center, Yorktown Heights, NY. In the fall of 1992 he was a visiting professor at the Catholic University of Leuven, Belgium. His research interests are in the areas of performance analysis, distributed systems, peer-to-peer systems, and grids.

Maarten van Steen received a Masters degree in Applied Mathematics from Twente University, and a Ph.D. in Computer Science from Leiden University. He is currently a full professor at VU University in Amsterdam where he teaches systems-oriented courses and conducts research on large-scale distributed systems, with an emphasis on systems where nodes can take decisions on only locally available information. These include wireless systems (notably large-scale sensonets) and wireline systems (such as decentralized grid infrastructures and traditional peer-to-peer systems).