Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 04-009

Highly Adaptive Lookup Systems for Peer-to-Peer Computing

Ewa Kusmierek, David Du, and James C. Beyer

February 20, 2004

Highly Adaptive Lookup Systems for Peer-to-Peer Computing

Ewa Kusmierek, David H.C. Du and James C. Beyer
Digital Technology Center and
Department of Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455
{kusmiere, du, beyer}@cs.umn.edu

***Abstract*— The performance of file sharing peer-to-peer systems depends to a large degree on the speed of lookup operation. A number of proposed solutions rely on distributed hashing techniques. Traditionally nodes are assigned fixed length identifiers which does not allow the table to expand or shrink with the increase or decrease in the node count. With the fixed length identifiers, the performance deteriorates when the number of nodes reaches a high value. On the other hand, the overhead of maintaining per-node state (i.e., information of all adjacent nodes) can be unnecessarily large if the number of nodes is small. The focus of our study is to propose a way to combine the *distributed* and *dynamic* nature of the system in a way that allows large and unpredictable changes in both the number and the distribution of nodes while providing scalability and good performance. Our approach is based on dynamic distributed hashing techniques; node identifier length varies with the number of nodes in the system. Hence, the system can adapt to the changing conditions and maintain good performance. In this paper, we describe the operations of the proposed adaptive system and verify its performance through simulations.**

Keywords: System design

I. Introduction

Peer-to-Peer (P2P) networks have recently gained a lot of interest and are a target of numerous research. They are instrumental in deployment of a number of large-scale distributed applications such as file sharing, distributed computing and collaborative environments. The requirements for P2P networks include good performance, self-organization, good scalability and robustness. There are additional requirements depending on the focused application. One of the major criteria for end-system multicast is the performance penalty compared to network layer multicast, e.g. duplicate packets on physical links and increased delay [1]. The file-sharing overlay performance evaluation is based on the path length measured by the number of nodes traversed to reach a node possessing the requested file. Path length determines the speed of lookup operation performed to retrieve the file. The cost in both overlay types is related to the per-node state [2].

This work is partially supported by NSF Grant EIA-0224424

The speed of a lookup operation in file sharing overlays depends on the method for determining the location of a requested object and on the routing used to reach that location. In Gnutella [3] flooding is used to search for the object. Napster [4] uses a central server that stores location of the available objects. Content addressable overlays [5–9] determine objects location by applying a hash function to the object's key, e.g. file name. They model the overlay network as a Distributed Hash Table (DHT). Object address cannot point directly to a node since the group of overlay nodes is not static. Instead, a typical design maps both nodes and objects to a common address space. The association between a node and an object is based on the numerical relation between node's and object's hash identifiers (addresses).

DHT-based systems are designed to handle changes in the number of nodes by applying the idea of consistent hashing [10]. The dynamic aspect of the system refers, however, not only to the number of nodes, but also to the *size of the hash table*. Having a fixed length identifier assigned to each node, limits the number of nodes in the system because the table cannot expand. In Chord [6], for example, the number of nodes must be small enough to make the probability of two nodes hashing to the same address negligible.

Content Addressable Network (CAN) [5] differs from other systems in that it does not use a hash function to generate node identifiers but rather chooses them randomly. The number of nodes can increase arbitrarily, however, as it increases the relative number of nodes known to any given node becomes smaller. As a result the number of

messages exchanged by the nodes during lookup increases and performance degrades. A lot of research effort is directed toward constructing an overlay whose performance scales well with the number of nodes [11, 12]. By allowing dynamic changes not only in the number of nodes but also in some parameters determining the topology of the overlay, such as dimensionality, we can achieve further performance improvement.

Most systems assume that the nodes are distributed *evenly* in the hash table. This may not always be the case. An uneven node distribution limits the number of nodes in the system even more since it increases the probability of collision. In systems such as CAN, where collisions do not take place, an uneven node distribution degrades the performance of a lookup process. The length of a path leading through a densely populated region is longer. The problem has been recognized mostly due to the potential load imbalance that it can cause [13,14]. The mechanisms developed to address uneven node distribution include the idea of virtual peers in Chord and assigning each peer several zones. Such a solution increases the per-node state considerably for all nodes in the overlay. CAN tries to place a new node in a less populated region of the hash table. Since, none of the overlay nodes has global knowledge, this solution can smooth node distribution only locally.

The focus of our study is to propose a way to combine the *distributed* and *dynamic* nature of the system in a way that allows large and unpredictable changes in both the *number* and the *distribution* of nodes while providing scalability and improved performance. We apply dynamic hashing techniques that allow the length of the identifier to vary. As the number of nodes increases, their identifiers become longer allowing the hash table to expand. The contraction takes place when the number of nodes decreases. In other words, the system is capable of adapting to the changing conditions in order to maintain good performance. The system benefits from having such adaptivity not only by being able to accommodate a wide range of the number of nodes without collisions but also by experiencing much slower performance degradation as the number of nodes increases.

We address a number of issues that arise when such a solution is applied in a distributed system. These include deciding when the identifier length should be changed, what the range of the change should be and who makes the decision. We describe modifications that have to be made to the per-node state, as well as to the routing process that allows nodes to route message not only without knowing the total number of nodes in the system, but also without knowing the exact length of nodes' identifiers. We evaluate how the additional dynamic characteristics of the system affects its performance. We address also the issue of uneven node distribution by allowing the length of the identifiers to vary throughout the table.

The remainder of the paper is organized as follows. We give a short summary of two DHT-based systems, CAN and Chord, in Section II. Since our scheme is based on dynamic hashing schemes, we also introduce dynamic hashing in Section II as related work. We describe how dynamic hashing can be applied to CAN in Section III. Section IV contains performance evaluation. The operations in the dynamic system are discussed in Section V. In Sections VI and VII we present some simulation results and offer some conclusions, respectively.

II. Related Work

Although we are addressing the general dynamic nature of P2P overlay networks, we briefly summarize CAN and Chord in this section and use these two systems to illustrate our concept. We provide also some background information on the dynamic hashing.

A. Distributed Lookup Systems

a) Content Addressable Network: The address space in CAN is represented by a virtual d-dimensional Cartesian space. Each point corresponds to a single location in the hash table. A node is mapped randomly into a point in space and assigned a zone in the vicinity of this point. An object's address is obtained by applying a hash function to the object's key. The hash function yields d coordinates in the virtual space. An inclusion relation determines the association between nodes and objects, i.e., a node whose zone contains object's address (point) stores the object.

The lookup operation is performed by first obtaining object's location through a hash function and then routing a request message toward this location in virtual space. Each node is aware only of the nodes in its neighborhood, i.e., nodes whose zones are adjacent to its own. Thus, the size of the routing table is bounded by $\mathcal{O}(d)$, where d is the dimensionality of the space. The next hop in the routing process is selected by choosing a neighbor whose zone is closest to the destination point, where the distance is measured along a straight line. The number of hops traversed to reach the destination is bounded by $\mathcal{O}(d\sqrt[d]{n})$, where n is the number of nodes.

The virtual space is always partitioned into a number of zones equal to the number of nodes (peers). Each zone can be interpreted as a bucket in a hash table with the capacity to hold one node. When a new node joins the

system, it "collides" with one of the existing peers since the point the new node picks at random belongs to some peer's zone. Thus, in order to accommodate the new node, the number of buckets has to be increased by splitting the "collision" zone in half. When a node leaves the system, the empty bucket is absorbed by one of the neighbors. Clearly, there is no upper bound on the number of nodes. The zones can be split indefinitely. However, as the number of nodes increases and their zone volumes become smaller, the performance deteriorates. Traveling a short distance in the virtual space may require traversing a large number of nodes.

The path length depends not only on the number of nodes but also on the number of dimensions, which determines the degree of connectivity among nodes. It is recommended to use a high number of dimensions to limit the expected path length if the number of nodes is also high. However, it is impossible to choose an optimal dimensionality if the number of nodes varies widely over time. Therefore, the number of dimensions should also change dynamically as a function of the number of nodes.

b) Chord: The address space in Chord contains a fixed number of m-bit identifiers. A node location is determined by applying a hash function to the node's IP address, while an object location is obtained by applying a second hash function to the object's key. The association between nodes and objects is based on the numerical comparison between their identifiers. An object is assigned to the node whose identifier is equal to or follows the object's identifier in the circular m-bit space. The space can be geometrically represented as a ring consisting of a set of numbers partitioned dynamically among a set of nodes.

Similarly to CAN, the lookup process consists of two steps: obtaining object's location and forwarding a request message toward that location. Each node maintains information about $\mathcal{O}(logn)$ nodes in the system. A node passes the request to a node whose identifier is the closest to the given object identifier among the nodes it is aware of. The path length is bounded also by $\mathcal{O}(logn)$.

When a new node joins the system, it splits the set of objects with its predecessor on the ring. The size of the ring is fixed, which means that it must be large enough to accommodate all nodes, i.e., no two nodes can map to the same location. Again it is impossible to choose an optimal length when the number of nodes varies significantly over time. Therefore, the length of identifiers should be a function of the number of nodes. An uneven distribution of nodes in the identifier space limits the number of nodes in the system even further by increasing the probability of two nodes hashing to the same entry.

B. Uneven Node Distribution

The possibility of uneven node distribution has been recognized with respect to both CAN and Chord. Given the volume of space in CAN or the ring length in Chord, denoted by V, and n nodes in the system, the even node distribution should result in n zones of equal volume. The probability that k of these zones are empty is $(1 - \frac{k}{n})^n$. The uneven distribution has two negative effects: load imbalance and extended path length bound. We consider the latter a negative effect even though the average path length may be shortened. The mechanism designed to address the uneven node distribution, usually target load balance under the assumption that the objects are distributed evenly and that they are equally popular. We concentrate on the effect of uneven distribution on the path length.

In order to minimize the probability of uneven distribution Chord utilizes virtual peers. Each node maintains a number of virtual nodes associated with it and is assigned to multiple zones. Intuitively such a solution provides a more uniform coverage of the address space. The per-node state is increased r times, where r is the number of virtual nodes per real node. Notice that all nodes in the system are affected by this increase. Our solution increases the per-node state only in the area of high node concentration.

CAN, on the other hand, directs *join* requests to the largest zone in the neighborhood of the node which received the request. If we imagine the height of a zone in a 2-d system to be inversely proportional to its volume, we can say that the request rolls toward the low-elevation large-volume zone. However, the request can easily get trapped in a local minimum. Hence, this solution can smooth the node distribution only locally. It increases also the time needed for a node to join the system.

C. Distributed Dynamic Hashing

A number of techniques have been proposed for implementing dynamic hashing [15,16]. Most of these schemes were designed to cope with the dynamic growth or shrinkage of a database. In order to allow a hash table (directory) to grow and shrink, the length of the identifiers must be adjusted dynamically. The size of the hash table determines the number of records that can be stored in it and is equal to the capacity of a single bucket multiplied by the number of buckets. When a new item hashes to a bucket whose capacity has already been reached, a collision occurs. Dynamic hashing provides a way to resolve the collision and accommodate the new item. In the following description of the dynamic hashing techniques, we assume that the bucket capacity is equal to one. Hence, a collision occurs when two items hash to the same entry in

the table. Recall that in dynamic lookup systems such as CAN or Chord, only one node can hash to a single entry.

1) Dynamic Hashing Methods: The two schemes designed for dynamic hashing are directory-based [17] and directory-less [18].

a) Directory-based Approach: The entries in a hash table are usually used to store the items. In the directory scheme a distinction is made between the entries in a table (directory) and the buckets. The scheme uses pointers to provide a level of indirection that allows fewer buckets than directory entries to exist. Assume that the directory has 2^d entries and therefore d bits are used for addressing. A bucket with more than one pointer can be addressed with a number of bits smaller than d. Its depth is smaller than the global directory depth d. A bucket with just one pointer has depth equal to the global depth. When a collision occurs in a bucket whose local depth is smaller than the global depth, a new bucket is created and the pointers are "split" between these two buckets. The depth of the old bucket is incremented and the same value is assigned as a local depth to the new one. A collision in a bucket whose local depth is equal to the global depth is handled by doubling the size of the directory (i.e., by adding one more bit to the directory address space). The identifier generated for a given key has a large but fixed number of bits. The number of bits used to retrieve an item can be smaller than the identifier length and is equal to the global depth. As the global depth changes so does the number of bits.

b) Directory-less Approach: In a directory-less approach the entries in the table are the buckets. The table size is increased by splitting one entry at a time when a collision occurs. Splitting an entry is realized by appending a new entry to the table. The new entry has the same address as the split one with a '1' prepended to it. The identifier of an old entry is also extended by prepending 0 to it. The entries are split in a round-robin fashion. Hence, the entry that is split is not necessarily the one involved in the collision.

Two features make the directory scheme more suitable for our purpose. It allows entries with various local depths to co-exist in the system at the same time, while the directory-less scheme limits the length of the identifiers to only two values at any time. The directory scheme is more flexible in this way. The scheme resolves the collision by splitting the bucket involved in the collision without a need for a temporary solution to accommodate the new item.

2) Distributed Hashing: Distributed versions of both dynamic hashing schemes have been proposed in [19–21]. Since the centralized structure does not exist in the distributed system, the table is split among multiple servers. Each server as well as client have some knowledge about the structure. This knowledge is updated based on the information exchanged among servers, but it may be incomplete or obsolete. Therefore, addressing errors may occur, corrected by forwarding a request to another server. In EH* [21] (directory distributed scheme) the directory is collapsed into several cache tables to eliminate multiple pointers pointing to the same entry. Each server is responsible for storing one cache table corresponding to one bucket and each server maintains a replica of the directory containing pointers to buckets.

We are dealing with lookup systems that are already distributed and are applying a dynamic aspect to their design. We do not want to introduce additional complexity to the distributed design by assuming that the nodes maintain a replica of the central pointer structure. Our goal is to retain the scalability feature while allowing the node table to expand and shrink.

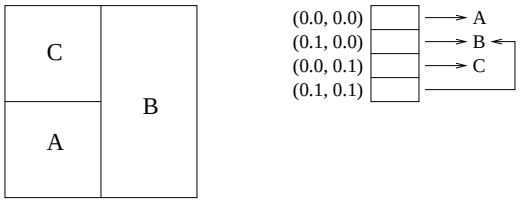
III. Dynamic Dimensionality

In the rest of the paper, for the convenience of presentation we focus on applying dynamic distributed hashing to CAN to address the dynamic aspect of the system. We examine the "addressing" scheme in terms of the length of the node identifiers and point out how it can be enhanced. Then we discuss how this enhancement affects the operations of the system and its performance.

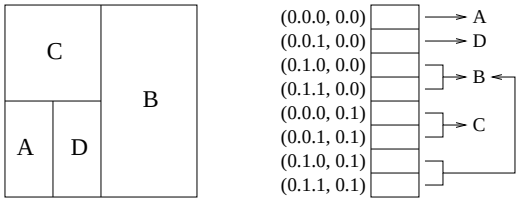
A. Dynamic Hashing in CAN

A node in CAN is assigned to a single zone in a d-dimensional space. Hence, its identifier consists of d coordinates. More precisely, there is a range for each coordinate specifying a width of the zone along a given dimension. This range is represented by a binary string and an identifier consists of d such strings. For example $(0.1.1, 0.1)$ is an identifier of a 2-dimensional space node. A string consisting of a single bit 0 corresponds to the maximum width along a given dimension. When a zone is split between two nodes, the node that gets the lower half appends a 0 to the binary string for a dimension along which the split takes place, the other node appends a 1. This additional bit is needed to distinguish between the identifier of a new node and the existing one.

The division of the virtual space can be represented by a directory structure as illustrated in Figure 1(a). The depth is determined by the length of the binary string for each coordinate. For example node B can be addressed along y-dimension with only 1-bit string: 0, while the other two nodes require two bits. The size of the directory is doubled whenever the smallest zone along a given dimension



(a) Before split



(b) After split

Fig. 1. Directory-based representation of the division of space

is split. Such an event corresponds to splitting an entry whose local depth is equal to the global depth. If node A splits its zone along x-dimension the size of the directory doubles (Figure 1(b)). The directory itself is an abstract structure and as such is not physically maintained by any node.

The current scheme for splitting zones allows adding nodes indefinitely. The number of entries in the directory is unlimited, but the volume of the corresponding space is fixed. As the number of zones increases and their volumes become smaller, the performance worsens. Traveling a short distance in the virtual space may require a large number of hops. In order to control the path length and to improve the lookup time, we extended dynamic hashing onto the *number of dimensions*. Our approach permits dynamic changes of the volume by varying the dimensionality of the system.

When the partition of space becomes too fine, we increase the volume of the space by adding another dimension. The identifier of each node is extended by adding another binary string as shown in Figure 2. An increase in dimensionality results in the higher node connectivity by increasing the number of neighbors. Recall that routing table size is bounded by $\mathcal{O}(d)$. Consequently, the path increases slower with the increase in the number of nodes.

B. Node Density

We proceed now to answer the following two questions: when a new dimension should be added and who makes that decision. The number of nodes in the system is a good

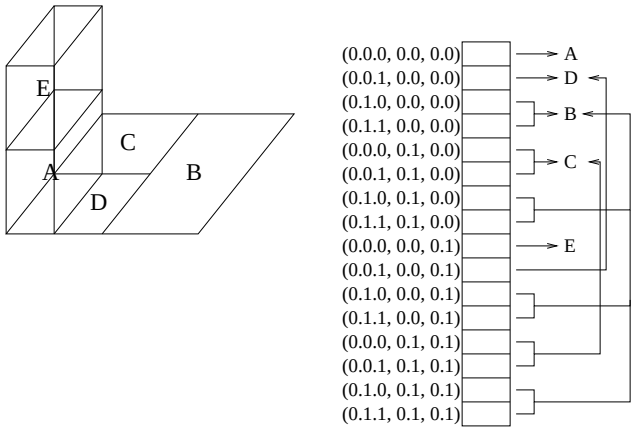


Fig. 2. "Extended" dynamic hashing

indicator on how fine the partitioning of space is. Therefore, a threshold on the number of nodes is used to control the dimensionality. Let n_0 denote the threshold. Then, when the number of nodes reaches n_0^d, the dimensionality is increased from d to $d+1$.

Since the system is fully distributed a decision on dimensionality adjustment has to be made in a distributed manner. Each node computes an estimate of the total number of nodes by calculating first a *local node density*, i.e., the number of nodes per unit of volume. The calculation is based on the knowledge of the system a node already has: knowledge of its neighborhood. Then the total number of nodes is equal to the local node density multiplied by the volume of the whole space.

1) Node Density Threshold: In order to eliminate one step for the above procedure, it is convenient to express the threshold on the number of nodes in terms of node density. For simplicity of presentation and without loss of generality, we assume that the range of each dimension in space is the same and equal to Δx. Then the threshold on the number of nodes n_0 translates into the following density threshold $\rho_0 = \frac{n_0}{\Delta x}$. In the system with d dimensions ρ_0^d is used as a density threshold.

Each node evaluates the local density and when its value reaches ρ_0^d, the node increases the number of dimensions. The operation is triggered by a change in the node's neighborhood. Notice that if the distribution of nodes is uniform, all nodes will have similar estimates and the system will quickly converge to higher dimensionality. Otherwise, an increase in the number of dimensions will be only local, affecting a small volume of space. Therefore, the number of dimensions may vary throughout the space. Dimensionality may be higher in the more dense areas. This property is very useful in improving the performance of the system with uneven node distribution. It allows us to increase the connectivity only locally without imposing an unnecessary overhead in the whole space.

2) Density Evaluation: Depending on the size of the area used, we consider two possibilities for the local density assessment. First, a node can calculate the density based only on its own zone. In this case, the local node density is equal to the inverse of the zone's volume. Second, a node can use information about its neighborhood. The density of node i's neighborhood is defined as the number of neighbors (including the node itself) divided by the total volume of the neighborhood. The density assessment in this case is less localized. A node's zone volume reaching a threshold is no longer a sufficient condition for dimensionality increase. On the other hand, such an increase may occur even if the zone volume has not reached the threshold but the neighborhood's density is sufficiently high.

3) Density Evaluation with Various Dimensionalities: Given the fact that nodes of various dimensionalities may exist in the system, we now describe density assessment procedure in more details. The first step in assessing node density is the equalization of the number of dimensions throughout the neighborhood. If node i has d_i dimensions, it projects any zone in its neighborhood with higher dimensionality into d_i dimensions, and extends any zone with lower number of dimensions into d_i-dimensional space. A zone is extended by assuming that it covers the whole range in each of the additional dimensions.

Figure 3 illustrates this concept. The dimensionality of node i is 2 but it has neighbors with dimensionalities 1 and 3. By using projections and extensions node i obtains a 2-dimensional view of its neighborhood. Notice that two of i's 3-dimensional neighbors map to the same 2-dimensional zone but both of them are included in the node count.

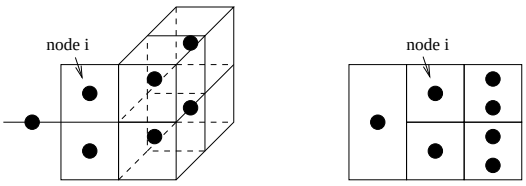


Fig. 3. Calculating density in multiple dimensionality case

C. Path Length and Node Distribution

Dimensionality change affects the distribution of nodes throughout the virtual space. Thus, we examine now how the performance depends on that distribution.

We distinguish two types of distances in the system. The first one is a distance between two nodes as measured by the Euclidean metric. The second is the distance measured by the number of hops in the overlay network between two nodes. This distance, which we refer to as path length in overlay network, is inversely proportional to the Euclidean distance between two adjacent nodes. In other words, a short Euclidean distance between adjacent nodes implies that a large number of hops have to be traversed to get from one point in space to another. Therefore, in order to minimize the maximum path length, we should maximize the minimum Euclidean distance between adjacent nodes. The minimum distance is maximized by equalizing the distances between adjacent nodes, which leads to an even distribution of nodes throughout the space. We now analyze more formally the dependence of the path length bound on the node distribution.

1) Path Length Bound for Even Distribution of Nodes: The maximum path length is bounded in the following way. We first find a bound on the path length that can be traveled in each of the dimensions. An even distribution of nodes implies that the number of nodes along each dimension is the same and equal to $\sqrt[d]{n}$. Therefore, the path length along a single dimension is bounded by $\mathcal{O}(\sqrt[d]{n})$. The maximum length path in d dimension has d such components, therefore its length is bounded by:

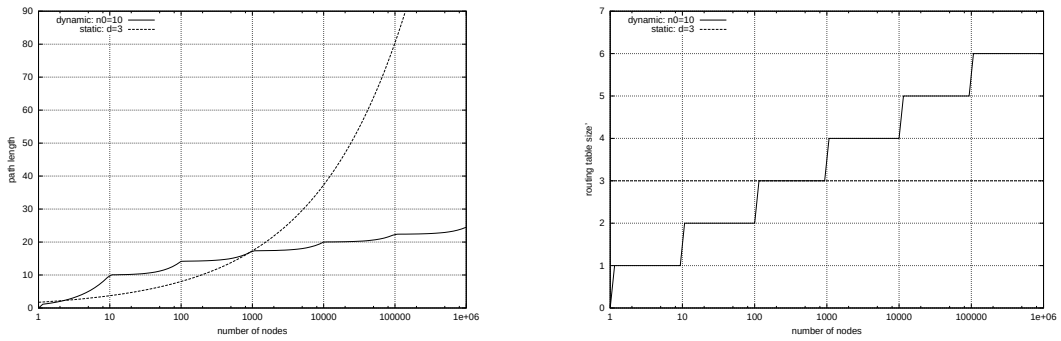
$$P_s(n,d) = \sqrt{\sum_{i=1}^{d} (\sqrt[d]{n})^2} = \sqrt{d}\sqrt[d]{n} \qquad (1)$$

2) Path Length Bound for Uneven Distribution of Nodes: If the nodes are not distributed evenly throughout the space, but the distribution along each dimension is even, the upper bound on the path length can be obtained in a similar way. Let n_i be the number of nodes along the ith dimension, where $n = \prod_{i=1}^{d} n_i$. Then the path length through all dimensions can be bounded by: $\sqrt{\sum_{i=1}^{d} n_i^2}$. Notice that in this case the upper bound is worse than for the even distribution case.

An uneven distribution of nodes along a single dimension produces an even longer worst case path. The maximum length of a path traveled along a single dimension in this case is equal to the maximum number of nodes contained in a segment of length equal to the half of the range of that dimension. This is the result of the fact that the space is a d-dimensional torus. The upper bound on path length can still be expressed as $\mathcal{O}(\sqrt{\sum_{i=1}^{d} n_i^2})$. The difference is in the constant, which is larger in the second case.

3) Maintaining Even Distribution: Since an even distribution of nodes throughout the space is optimal for the path length, we want to maintain this property. Recall that in the static system an ordered list of dimensions is used for zone splitting, i.e., each node splits its zone first along dimension x, then y and so on in a round-robin fashion.

The first step in achieving an even distribution in a



(a) Path length (b) Table size

Fig. 4. Performance comparison for $n_0 = 10$

dynamic system is to keep the number of nodes along each dimension roughly the same. Therefore, when a new dimension is added, the subsequent zone splits are performed only along that new dimension in order to decrease the difference between number of nodes along "old" and "new" dimensions. The threshold n_0 can be interpreted as a threshold on the number of nodes along a single dimension.

IV. Performance Comparison

In order to evaluate the performance of the dynamic system, we compare it against the performance of the static system with the same number of nodes. We assume that a central mechanism is used to keep track of the number of nodes and control dimensionality. Hence, the results presented here are based on the asymptotic behavior of the dynamic system. We compare two aspects of the system's performance. One of them is the path length, and the other is the size of the routing table. We use two terms: number of neighbors and size of the routing table, interchangeably.

A. Path Length Comparison

Let N_1 be the initial number of nodes increasing with time to N_2. Let d_2 be the number of dimensions optimal for N_2 nodes in terms of both path length and size of the routing table. In the dynamic system, d_1 is the initial number of dimensions, and the dimensionality increases with the number of nodes using n_0 as a threshold. Once the number of nodes reaches N_2, the number of dimensions is equal to $d_3 = \lceil \log_{n_0} N_2 \rceil$. The optimal number of dimensions d_2 is reached only when $n_0 = \sqrt[d_2]{N_2}$. The path length bound for a dynamic system is given by:

$$P_d(n, n_0) = \sqrt{(d(n) - 1)n_0^2 + \left(\frac{n}{n_0^{d(n)-1}}\right)^2} \quad (2)$$

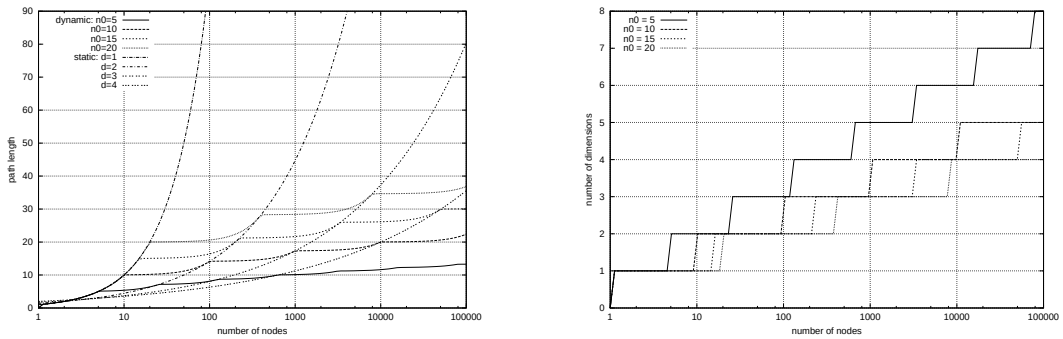
where n is the number of nodes and $d(n)$ is the corresponding number of dimensions:

$$d(n) = \lceil \log_{n_0} n \rceil \quad (3)$$

Figure 4(a) presents the comparison of the path length bounds for static and dynamic system with threshold $n_0 = 10$. The number of dimensions $d_2 = 3$ in the static system is assumed optimal for $N_2 = 1000$. The dynamic system uses smaller number of dimensions for $n < N_2$ and, therefore, path length is potentially longer than path length in the static system. The size of the routing table maintained by each node is smaller though. Since the selected threshold n_0 is equal to $\sqrt[d_2]{N_2}$, both system have the same path length bound for $n = N_2$. Beyond that point the path length bound for a dynamic system is smaller since the dimensionality keeps increasing. Figure 4(b) shows the comparison of the routing table size between the dynamic system and the static systems. The table size in the dynamic system increases with the number of nodes but only linearly. Therefore, using a number of dimensions smaller than d_2 for the number of nodes smaller than N_2 is not justified. The gain in terms of the routing table size is rather insignificant. Hence, it is reasonable for the dynamic system to have an initial number of dimensions $d_2 = 3$ and start increasing it only after the number of nodes exceeds $N_2 = 1000$. The path length of such a system would follow the curve for the static system with $d = 3$ up to $n = N_2$ in Figure 4(a) and then switch to follow the dynamic system curve.

B. Threshold Influence on Path Length

We examine now how the value of threshold n_0 affects system performance starting with some observations. Figure 5(a) presents the influence of the threshold n_0 on the path length bound. The dynamic system curves have a



(a) Path length

(b) Dimensionality

Fig. 5. Threshold influence on performance

wave-like shape. The path length increases with the number of nodes until it reaches the point where dimensionality is increased. For a given value of threshold n_0, this change occurs at $n = n_0^b$, where $b = 1, 2, \ldots$, and is shown as crest of the "wave". The rate of path length change also increases. Once the change of dimensionality occurs, the rate is slowed down resulting in a wave-like pattern. We notice also that each wave crest lies on the curve for the static system path length for some value of d. All dynamic curves follow initially static curve for $d = 1$, which is linear $P(n) = n$ (notice log-scale on x-axis). Next they intersect static curve for $d = 2$ at $n = n_0^2$, for $d = 3$ at $n = n_0^3$ and so on. It is not surprising since Equation (2) reduces to Equation (1) for $n = n_0^b$, $b = 1, 2, \ldots$ We observe then that each next wave on a dynamic curve corresponds to a higher by one number of dimensions starting with one dimension. For a given number of dimensions the rate of path length increase follows the same pattern independent of the threshold value n_0. The difference is in how long this pattern is allowed to continue. The "cut-off" point for the next wave occurs earlier for a lower value of n_0 which results in a lower path length bound. The dimensionality of the system is adjusted more frequently as presented in Figure 5(b).

C. Routing Table Size

There are two main differences between the static and the dynamic system with respect to the routing table size. In the static system the routing table size does not depend on the number of nodes and is determined by dimensionality. In the dynamic system the number of neighbors changes with the number of nodes and is affected by the dimensionality of the node's neighbors. Consider a d-dimensional node, which has neighbors with higher dimensionality. The number of neighbors of d-dimensional node is higher than indicated by its dimensionality. On the other hand, a lower-dimensionality neighbor generally does not have the opposite effect on the number of neighbors. Dynamic system has nodes with various dimensionalities present in the system since the change of dimensionality occurs gradually. We conclude that the average routing size table is higher than indicated by the average dimensionality.

V. System Operations

The fact that dimensionality varies throughout the space, changes the way in which some operations are performed. In this section we briefly describe these changes.

a) Neighborhood Relation: The definition of a neighbor is basically the same. Two nodes i and j are neighbors if their zones overlap along $d - 1$ dimensions and abut along one dimension. If two nodes have different dimensionalities, this criterion is applied with the smaller of the two numbers of dimensions: $d = min(d_i, d_j)$.

b) Object Identifiers: Recall that nodes and objects share the same address space. Therefore, a change of dimensionality affects also object identifiers. An object address is a point in space. Once a new dimension is added all points have to be "remapped" to a higher dimensionality space by adding another coordinate. In order to avoid the actual remapping, we assume that there is a maximum number of dimensions set for the system D_{max}. A hash function applied to an object's key returns D_{max} coordinates. If a smaller number of dimensions $d < D_{max}$ is used, then only the first d coordinates are considered. Notice that there is no need to store D_{max} coordinates for a node, since the value of each coordinate beyond d dimensions is by default 0. Note also that the cost of operations is related to d and not to D_{max}.

c) Dimensionality Change Operation: A decision about dimensionality change is always prompted by a node joining or leaving the system. All nodes in the affected neighborhood compute the density threshold when

a new node joins the system. The dimensionality of a new node is inherited from an existing node that is splitting its zone. The dimensionality of a node decreases when all of its neighbors have dimensionality lower than its own. The update of neighborhood information includes also dimensionality. The order of dimensions along which the zones are consolidated follows the reversed order of dimensions along which zones were split.

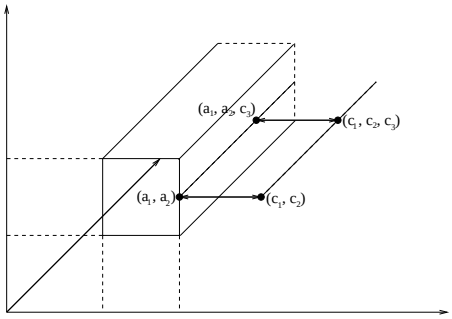


Fig. 6. Distance between a zone and a point

d) Routing: The routing follows a straight line through the Euclidean space. A node chooses the neighbor which is the closest to the destination as the next hop. In order to estimate the distance between a neighbor and the destination *c*, the node first chooses point a in the neighbor's zone that is the closest to the destination. Next it calculates distance between these two points, a and *c*, using standard Euclidean metric. Having variable dimensionality throughout the space poses the question how the distance should be calculated, i.e., what dimensionality should be assumed.

One approach to the problem is to use the maximum number of dimensions D_{max} by extending the zones with dimensionality lower than D_{max}. We show that we can reduce the computational cost of selecting the next hop by using *local* dimensionality, i.e., the dimensionality of each neighbor. In other words, the distance between a node and a point is the same whether we use D_{max} or the node's number of dimensions.

Assume that the neighbor node's dimensionality d is lower than D_{max}. We use d coordinates of point *c*, $(c_1, c_2, ..., c_d)$, to calculate the distance and compare it with the distance based on D_{max} coordinates. Let a be the point in the node's zone which is closest to c in d-dimensional space. The distance between a and c is then: $\sqrt{(a_1 - c_1)^2 + ... + (a_d - c_d)^2}$. When dimensionality is extended to D_{max}, we use all coordinates of point *c*: $(c_1, c_2, ..., c_d, ..., c_{D_{max}})$. Now the point $(a_1, a_2, ..., a_d, c_{d+1}, ..., c_{D_{max}})$ belongs to the zone and it is the closest point to *c* in that zone[1]. Hence, the distance is the same in both dimensionalities: $((a_1 - c_1)^2 + ... + (a_d - c_d)^2 + (c_{d+1} - c_{d+1})^2 + ... + (c_{D_{max}} - c_{D_{max}})^2)^{\frac{1}{2}} = ((a_1 - c_1)^2 + ... + (a_d - c_d)^2)^{\frac{1}{2}}$. Figure 6 shows the distance between a 2-dimensional zone and a point in 2- and 3-dimensional space.

[1] The statement can be proved by contradiction.

TABLE I
DIMENSIONALITY "DISTRIBUTION"

# of nodes/dim	1	2	3	4	5
10	80%	20%			
20	55%	45%			
50	1%	94%	3%		
100		81%	19%		
200		33%	67%		
400		3%	95%	2%	
800			77%	23%	
1600			32%	68%	
3200			4%	96%	
6400				95%	5%

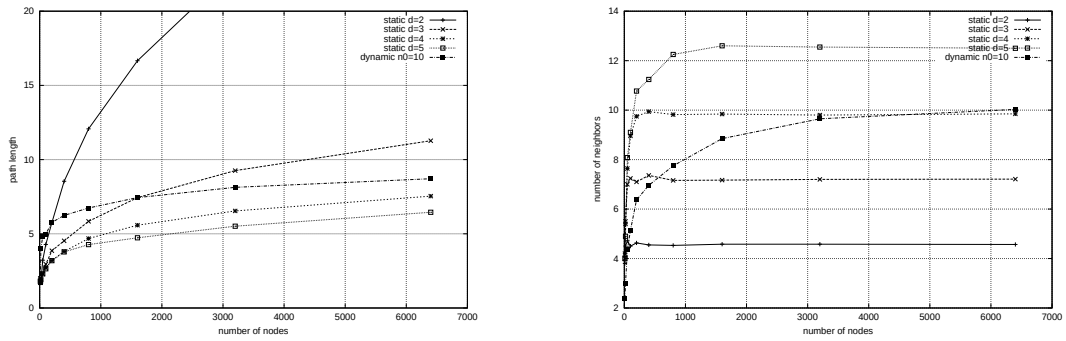
VI. Simulation Results

We conducted a number of simulations to verify the performance of the dynamic system. We examine how the performance changes with the number of nodes and how it compares to the performance of the static system. We show also how the size of the area used by each node for density evaluation and the value of the threshold n_0 affect the performance.

A. Path Length and Routing Table Size

In the first set of tests, each node in the dynamic system evaluates the density based on the volume of its own zone and uses $n_0 = 10$ as the threshold. Within the given range of the node count ($\leq$ 6400), the dynamic system starts with 1 dimension and reaches dimensionality 5. Therefore, we have included static systems with dimensionality 2, 3, 4 and 5 in the test. 1-dimensional system performance deteriorates very quickly and thus was excluded. Figure 7(a) presents the comparison between the average path lengths obtained in both types of systems. The average path length is calculated over a set of paths obtained by selecting a random destination for each node in the system.

We observe that as the number of nodes increases, the path length in the dynamic system outperforms static systems with consecutive dimensionalities. First, the path length becomes shorter than in the 2-dimensional (2-d) system, then it overtakes the 3-d system. The difference between the path length in the dynamic system and in the

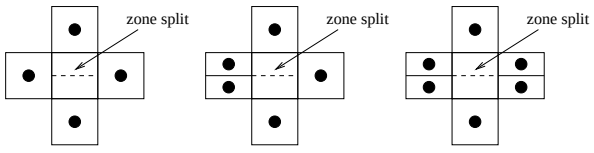


(a) Average path length (b) Average number of neighbors

Fig. 7. Comparison between static and dynamic system

4-d system decreases past the intersection point with the 3-d path length and we expect the dynamic system to overtake the 4-d system for a higher number of nodes. When the number of nodes reaches 6400 the dynamic system has mostly nodes with dimensionality 4 (Table I) but the average path length is longer than that in the 4-d system. This is the price we pay for the dynamic system's ability to adapt. When the threshold n_0 is equal to 10, we consider dimensionality 4 to be optimal for the number of nodes $10^3 \leq n < 10^4$. Hence, the performance of the dynamic system with respect to the path length is slightly below optimal.

Figure 7(b) presents the comparison of the average per-node state in the same settings. The average number of neighbors increases with the number of nodes for both systems. It increases also with dimensionality in the dynamic system. For the largest examined number of nodes (6400), it is close to the number of neighbors in the 4-d system, and the majority of nodes in the dynamic system have dimensionality 4 (Table I).



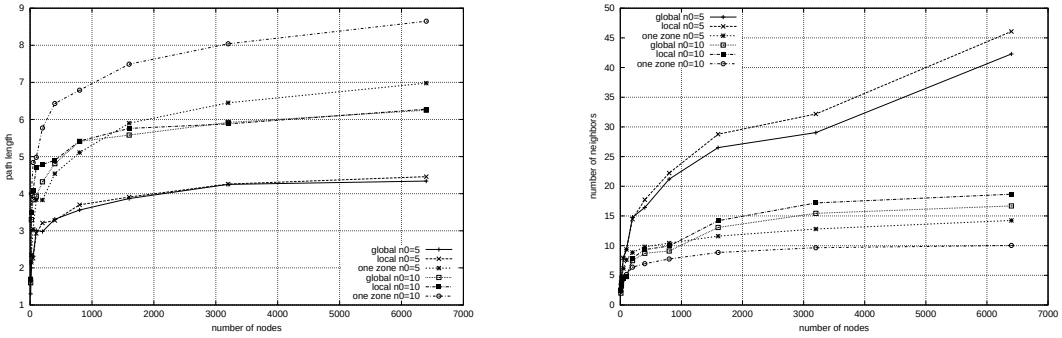
(a) change=+2 (b) change=0 (c) change=-2

Fig. 8. Influence of a node insertion on the number of neighbors

B. *Influence of Density Assessment Area on Performance*

We have also examined how the size of the area used by each node for density assessment affects the performance. We compare behavior of the system in which the number of nodes is tracked globally (and dimensionality is changed globally) with the behavior of the system in which the density assessment is done locally. The former is a theoretical case considered for the comparison purposes only. In the latter case we distinguish two subcases: a node uses either its neighborhood to assess density or only the volume of its own zone. In all cases we use two different thresholds $n_0 = 5$ and $n_0 = 10$. We comment first on the influence of the size of the area used for the density assessment. Figure 9(a) illustrates the effect it has on the path length. The first two cases, global and local neighborhood-based, do not differ significantly in performance which suggests that the node distribution is fairly even. The average path length obtained with a one-zone density assessment is longer. This statement is true independent of the threshold used.

In order to explain this behavior, we have examined the average number of neighbors in all three cases and Figure 9(b) presents the results. The per-node state in the one-zone density assessment case is smaller. Hence, the better performance in terms of the path length for global and local-neighborhood systems comes at a price. Both systems have higher degree of "connectivity". We believe that the reason has to do with the fact that every insertion of a node into the system affects the neighbors of the node which is splitting its zone. The total change in the number of neighbors for these nodes (including the one splitting its zone) can be positive as well as negative. Figure 8 shows an example of three types of change. Depending on the size of neighboring zones, the sum of the changes in the number of neighbors of all nodes affected is +2, 0 or -2. In the static system, where the dimension to split along is selected in the round robin fashion, the number of positive changes is roughly the same as the number of negative changes. Hence, the average number of neighbors remains fairly constant. In the dynamic system, zones are split along one dimension only, until a new dimension is



(a) Average path length

(b) Average number of neighbors

Fig. 9. Various scopes of dimensionality change

added. Once dimensionality is increased, most node insertions cause a positive change in the number of neighbors. If the next dimension is added before this effect can be balanced by an equal number of negative changes, the average number of neighbors remains elevated. In the dynamic system, in which the whole neighborhood is used for density assessment, the nodes affect each other more than in a single-zone assessment system. Hence, high density of the node's neighborhood may cause an increase in the node's dimensionality, preventing it from going through a zone split with a negative change and lowering its number of neighbors. Hence, the average number of neighbors is higher which of course results in a shorter path length.

The rate of increase in the average number of neighbors is higher than the corresponding rate of increase in the path length. The former is the price paid for the performance, therefore, the cost is increasing disproportionately to the gain in this case. Recall that in the static system the number of neighbors does not increase with the number of nodes. In the one-zone density assessment system the relation between price and gain is much better. We present also the average dimensionality as a function of the number of nodes in the systems with various density assessment areas (Figure 10). There is no significant difference in average dimensionality between neighborhood-based and one zone-based systems. We notice that the number of neighbors is higher than indicated by the average dimensionality in both cases and that it is significantly higher for neighborhood-based system. We attribute this effect again to the previously described node insertion process.

C. Density Threshold Influence on Performance

The influence of the threshold on the performance is more intuitive. Using a lower threshold makes the system more sensitive to changes in the number of nodes. The result is a shorter path but also much higher per-node state.

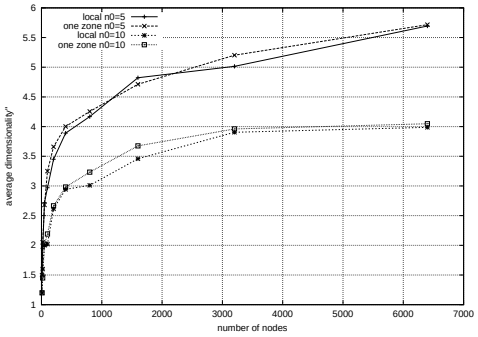


Fig. 10. Average dimensionality in dynamic systems

Dimensionality varies in the dynamic system in time as well as throughout the space. We have examined the latter aspect of the system behavior. Table 1 presents the percentage of nodes for each value of dimensionality present in the system for various numbers of nodes. With value 10 used as the node number threshold n_0, single-zone density assessment and node number limited to 6400, the maximum number of dimensions does not exceed 5. When the number of nodes is close to the power of n_0, the majority of nodes have dimensionality determined by Equation (3). Since the dimensionality change is localized, there are usually nodes of two or three different dimensionalities present in the system. The variation would be even more pronounced if the node distribution was less uniform.

VII. Conclusions and Future Work

We have presented how CAN can benefit from the idea of dynamic dimensionality. By allowing the dimensionality to vary with the number of nodes and throughout the space, the system can adapt to dynamical changes in the set of nodes. The size of the routing table maintained by each node is increased compared to the system with static dimensionality but the performance measured by the

path length is considerably improved. The technique used to maintain dynamic dimensionality is derived from dynamic hashing. It adapts the length of identifiers of both data items and nodes, allowing to control the access time.

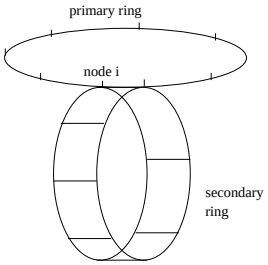


Fig. 11. Multiple ring levels in Chord

To support the claim of broad applicability of dynamic hashing to general distributed lookup systems, we briefly describe how Chord's performance can be improved using dynamic hashing. Assume that m-bit identifiers are used. Once the collision is encountered, the length of identifiers is doubles by appending another m bits. In the geometric interpretation, we create another ring and attach it to the the existing node, say i, participating in the collision. This node belongs now to two rings. The routing is done using m most significant bits of the identifier. If item's successor, node i, does not have the item, it passes the request onto the second ring for a similar routing procedure, this time using a second group of m bits. Notice that the secondary ring is 2-dimensional since node i is potentially responsible for a set of data items (Figure 11). These data items are now partitioned among nodes on the secondary ring. The identifiers of nodes on the secondary ring have the first m bits such that node i is their successor. The second group of m bits can have an arbitrary value. New nodes, whose successor is node i, are added to the secondary ring. The procedure of extending the identifier can be continued by adding more rings at different levels (dimensions). The details of maintaining the connection between rings as well as performance improvement that can be obtained are a subject of future work.

ACKNOWLEDGMENT

The authors would like to thank Sylvia Ratnasamy for making the implementation of CAN available for simulations.

REFERENCES

[1] Dejan Kostic and Amin Vahdat, "Latency versus cost optimizations in hierarchical overlay networks," Tech. Rep. CS-2001-04, Duke University, November 2001.

[2] Jun Xu, "On the fundamental tradeoffs between routing table size andnetwork diameter in peer-to-peer networks," in *INFOCOM*, 2003.

[3] "Gnutella," .

[4] "Napster," .

[5] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker, "A scalable content addressable network," in *Proceedings of ACM SIGCOMM*, 2001.

[6] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan, "Chord: A scalable Peer-To-Peer lookup service for internet applications," in *Proceedings of ACM SIGCOMM*, 2001.

[7] Antony Rowstron and Peter Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," *Lecture Notes in Computer Science*, vol. 2218, 2001.

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

[9] J. Kubiatowicz et al., "OceanStore: An architecture for global-scale persistent storage," in *9th international Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[10] D. Karger, E. Lehman, T. Leighton, M. Levibe, D. Lewin, and R. Panigrahy, "Consistent hashing and random trees: Distributed caching protocol for relieving hot spots on the world wide web," in *29th Annual ACM Symposium on Theory of Computing*, 1997, pp. 654–663.

[11] Dahlia Malkhi, Moni Naor, and David Ratajczak, "Viceroy: A scalable and dynamic emulation of the butterfly," in *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, 2002.

[12] Frans Kaashoek and David R. Karger, "Koorde: A simple degree-optimal hash table," in *2nd International Workshop on Peer-to-Peer Systems*, 2003.

[13] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, and Richard Karpand Ion Stoica, "Load balancing in structured p2p systems," in *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, USA, Mar. 2002.

[14] John Byers, Jeffrey Considine, and Michael Mitzenmacher, "Simple load balancing for distributed hash tables," in *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, USA, Mar. 2002.

[15] Per-Ake. Larson, "Dynamic hashing," *BIT*, vol. 18, pp. 184–201, 1978.

[16] R. Enbody and D. H-C. Du, "Dynamic hashing schemes," *ACM Computing Surveys*, vol. 20, no. 2, pp. 85–114, 1988.

[17] R. Fagin, J. Nievergelt, N. Pippenger, and H. Strong, "Extendible hashing - a fast access method for dynamic files," *ACM Transactions on Database Systems*, vol. 4, no. 3, pp. 315–344, 1979.

[18] W. Litwin, "Linear hashing: A new tool for file and table addressing," in *VLDB*, 1980, pp. 212–223.

[19] W. Litwin, M. Neimat, and D. Schneider, "LH* - linear hashing distributed files," in *ACM SIGMOD*, 1993.

[20] R. Devine, "Design and implementation of DDH: A distributed dynamic hashing algorithm," in *4th International Conference on Foundation of Data Organizations and Algorithms*, 1993.

[21] V. Hilford, F. Bastani, and B. Cukic, "EH*-extendible hashing in a distributed environment," in *Proceedings of the 21st Computer Software and Applications Conference*, 1997.