SWAM: A Family of Access Methods for Similarity-Search in Peer-to-Peer Data Networks

Farnoush Banaei-Kashani
Computer Science Department
University of Southern California
Los Angeles, California 90089
banaeika@usc.edu

Cyrus Shahabi
Computer Science Department
University of Southern California
Los Angeles, California 90089
shahabi@usc.edu

ABSTRACT

Peer-to-peer Data Networks (PDNs) are large-scale, self-organizing, distributed query processing systems. Familiar examples of PDN are peer-to-peer file-sharing networks, which support exact-match search queries to locate user-requested files. In this paper, we formalize the more general problem of *similarity-search* in PDNs, and propose a *family* of distributed access methods, termed *Small-World Access Methods (SWAM)*, for efficient execution of various similarity-search queries, namely exact-match, range, and k-nearest-neighbor queries. Unlike its predecessors, i.e., LH* and DHTs, SWAM does not control the assignment of data objects to PDN nodes; each node autonomously stores its own data. Besides, SWAM supports all similarity-search queries on multiple attributes. SWAM guarantees that the query object will be found (if it exists in the network) in average time logarithmically proportional to the network size. Moreover, once the query object is found, all the similar objects would be in its proximate network neighborhood and hence enabling efficient range and k-nearest-neighbor queries.

As a specific instance of SWAM, we propose *SWAM-V*, a Voronoi-based SWAM that indexes PDNs with multi-attribute data objects. For a PDN with N nodes SWAM-V has query time, communication cost, and computation cost of $O(\log N)$ for exact-match queries, and $O(\log N + \mathbf{s}N)$ and $O(\log N + \mathbf{k})$ for range queries (with selectivity $\mathbf{s}$) and $\mathbf{k}$NN queries, respectively. Our experiments show that SWAM-V consistently outperforms a similarity-search enabled version of CAN in query time and communication cost by a factor of 2 to 3.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems,Information networks* ; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process*; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Indexing methods*

General Terms

Design, Algorithms

CIKM'04, November 8–13, 2004, Washington, DC, USA.

Keywords

Peer-to-peer networks, similarity search, small-world, distributed hash table (DHT)

1. INTRODUCTION

Recently, Peer-to-peer Data Networks (PDNs) have emerged as a family of large-scale, self-organizing, distributed query processing systems. A PDN is a federation of a dynamic set of peer, autonomous nodes communicating through a transient-topology interconnection. Data is naturally distributed among the PDN nodes in fine grain, where a few data items are dynamically created, collected, and/or stored at each node. Therefore, the network scales linearly to the size of the dataset. With a dynamic dataset, node-set, and topology, PDNs should be considered as the new generation of distributed database systems with significantly less constraining assumptions as compared to their ancestors. Peer-to-peer file-sharing networks such as Freenet [9] and Kazaa [13] are familiar (but simplistic) examples of PDN.

We abstract out the semantic of each data object within a PDN by a d-dimensional vector (or, a tuple with d attributes); hence, the contents of the PDN comprise a horizontally fragmented and distributed relation with d attributes. For instance, the attributes can be the characteristic attributes of music files (e.g., artist-name, album-year) shared in a peer-to-peer file-sharing network, or they can be the amount of resources (e.g., CPU, memory, storage, bandwidth) that a node can share in a PDN for distributed computing such as SETI@home [20]. Subsequently, we focus on the challenging problem of efficient execution of similarity-search queries (i.e., exact-match, range, and $\mathbf{k}$-nearest neighbor queries) in such PDNs. With the former example, user can issue a range query to find all albums of an artist released within a particular time interval, and with the latter example, one can use a $\mathbf{k}$NN query to locate nodes with enough resources to run a particular sub-task. Traditional distributed databases employ hierarchical distributed directories as access methods to evaluate similarity queries. These access methods often assume small-scale networks with a static set of nodes that maintain the object directory. Besides, they fail to distribute the query execution load fairly among peer nodes. This is because with hierarchical index topology of the network, nodes at the higher levels of the hierarchy (e.g., root of the tree) inevitably receive more queries to process. Moreover, hierarchical network topologies are loop-free and intolerant to failures and/or autonomous presence of the PDN nodes. On the other hand, hash-based distributed data structures such as LH* [15], and more recently, DHTs [18, 21] assume large-scale

networks and construct distributed index structures with non-hierarchical topologies. However, these access methods enforce the location of the content within the network and hinder natural replication of the objects, which are both in conflict with the autonomy of the PDN nodes in maintaining their own content. Besides, since object assignment to the nodes is performed irrespective of the object distribution, these access methods fail to adapt to the distribution of the objects and may suffer from unbalanced query load. Most importantly, they are designed to support only exact-match queries on a single attribute.

The first contribution of this paper is a formalization of the problem of similarity-search in PDNs by 1) modeling the problem, 2) defining a set of metrics to evaluate the efficiency of the PDN access methods, and 3) introducing a basic access method to set a lower efficiency bound for similarity-search (see Section 2).

Next, learning from the principles of the small-world models, which are proposed to explain the phenomenon of efficient communication in social networks [22, 14], in this paper we define a family of access methods, termed *Small-World Access Methods (SWAM)*, for efficient similarity-search in PDNs. Intuitively, with SWAM the topology of the PDN is structured (in a self-organized manner) based on the content of the PDN nodes such that on the network topology 1) the average network distance between any pair of nodes scales logarithmically with the network size, and 2) nodes with similar content are clustered together; hence, enabling efficient location of the query object and thereafter, the objects similar to the query object, respectively (see Section 3.1 for a detailed discussion of this intuition). In this context, the PDN topology can be considered as a distributed index structure that, analogously to the typical index structures for databases, organizes the nodes and therefore, the data content of the PDN nodes for efficient similarity-search. It is important to note that with SWAM this topology is generated and maintained in a distributed manner by nodes executing straightforward algorithms as they join and leave the network and/or their content is updated (these algorithms are formally explained in Section 3.3). Thus, the efficient SWAM index structures are also scalable and practically adoptable.

To define the SWAM family formally, we formalize the intuitive properties discussed above and define three necessary and sufficient properties that characterize an access method as a member of the SWAM family (see Section 3.2). These properties ensure that the generated index structure 1) accurately partitions the data space for exact resolution of the location of the query object, 2) properly interconnects the nodes for fast reach to the query object, and 3) closely co-locates (on network) the nodes with similar content for batch retrieval of the similar content when the location of the query point is resolved. The first property ensures that we monotonically approach the query object as we traverse the network, and that we always find the query object if it exists in the network. As we explain in Section 4.1, surprisingly some DHTs (e.g., CAN) cannot guarantee even such a basic requirement of similarity-search. The second property ensures that the time and communication cost of resolving the query scales logarithmically to the size of the network. Finally, the third property enables efficient range and kNN queries. To the best of our knowledge, no current DHTs can collectively support these properties on PDNs with multi-attribute content.

Finally, as a proof of concept we introduce a Voronoi-based instance of SWAM, termed *SWAM-V*, which satisfies all the three properties of SWAM (see Section 3.3). For a PDN with N nodes SWAMP-V has query time, communication cost, and computation cost of $O(\log N)$ for exact-match queries, and $O(\log N + \mathbf{s}N)$ and $O(\log N + \mathbf{k})$ for range queries (with selectivity $\mathbf{s}$) and $\mathbf{k}$NN queries, respectively. SWAM-V proposes a non-hierarchical distributed index structure that indexes PDNs with multi-attribute objects and resolves queries exactly (i.e., without false dismissal). SWAM-V also respects the autonomy of the PDN nodes and self-configures the topology of the PDN based on the nodes own content. Consequently, it avoids unnecessary content migration and replacement, supports object replication, and adapts to the object distribution as new nodes join the PDN with the new content.

For more intuition about the SWAM-V properties mentioned above, here we briefly discuss the organization of the SWAM-V topology (more details in Section 3.3). For accurate partitioning of the data space (and as a result, proper clustering of the nodes with similar content), SWAM-V constructs a d-dimensional Voronoi diagram on the data space of the PDN, with the data objects available in PDN as the vertices/generators of the Voronoi diagram. The dual Delaunay graph of this Voronoi diagram defines the basis of the SWAM-V topology, where the PDN nodes that store data objects with neighboring Voronoi cells are neighbors on the topology. Starting from a querier node, SWAM-V directs a query toward its required object by forwarding the query from one node to a neighbor node that stores object(s) more similar to the query object. We prove that with Voronoi-based partitioning the query monotonically approaches the query object and eventually finds the object (if it exists). Moreover, the SWAM-V topology has shortcut random links to far-away nodes that allow for long jumps on the Voronoi diagram, similar to the ladders in the snakes-and-ladders board-game which enable fast traversal of the underlying grid.

We perform a comparative study via simulation to verify the efficiency of SWAM-V versus our basic access method, as well as a version of CAN [18] that we extended over its original DHT to support range and $\mathbf{k}$NN queries. Our experiments show that unlike the basic access method, SWAM-V achieves logarithmic query time with limited resource usage, and despite assuming natural data distribution in the PDN, SWAM-V consistently outperforms CAN in query time and communication cost. For example, in a typical case SWAM-V improves the communication cost of $\mathbf{k}$NN queries at least 300% and the query time of range queries up to 200%. A very interesting observation is that as the number of dimensions (data attributes) increases, the query cost with SWAM-V remains almost constant. Although at first this property seems counter-intuitive, we argue that it is enabled by the accurate Voronoi-based partitioning of the data space with SWAM-V. Intuitively, with accurate partitioning despite more uniformity of the distance between data objects in the data space with higher dimensionality, SWAM-V can still distinguish the difference in distances and properly route the query through the nodes closer to the query object (hence, better performance and less query cost). This benefit is compensated by higher cost of the index construction, but since index construction occurs in a distributed fashion and incrementally as nodes join and leave the network, the higher construction cost hides. Also, note that "curse of dimensionality" does not happen in our case because PDN applications often assume only a handful of attributes per object.

The key contributions of our work can thus be summarized

as follows:

- We formalize the problem of similarity-search in PDNs and define a model, a set of performance measures, and a benchmark for PDN access methods.
- We introduce a set of properties that characterize SWAM, a family of potentially efficient access methods for similarity search in PDNs, which respect the natural data distribution of PDNs as a requirement.
- Based on the proposed properties, we introduce the SWAM-V access method for PDNs. To the best of our knowledge, SWAM-V is the first PDN access method that can guarantee the resolution of exact-match, range, and **k**NN queries in PDNs with multi-attribute data objects and with cost and response-time logarithmic to the size of the network.

We believe this work initiates a new and interesting research direction to extend the massive similarity-search database literature to support efficient search in PDNs as the new generation of distributed database systems. The remainder of this paper is organized as follows. In Section 2, we formally define the problem of similarity-search in PDNs. Section 3 elaborately describes the SWAM family of PDN access methods, and specifies SWAM-V as a particular member of the SWAM family. In Section 4, we explain the results of our experimental evaluation of SWAM-V. Section 5 covers the related work. Finally, Section 6 concludes the paper and discusses the future directions of this research.

2. FORMAL DEFINITION OF THE PROBLEM

In this section, we state and model the problem of similarity-search in PDN. We also describe a naive access method, which essentially scans the PDN to resolve the similarity queries, as the basic similarity-search mechanism that sets the lower performance bound for more efficient access methods.

2.1 Model

We assume a relational data model for the content of a PDN. A set of (maybe duplicate) tuples with the same schema are distributed among the nodes of the PDN (for multi-schema PDNs, we rely on schema reconciliation techniques such as [8]). Tuples are uniquely identified by a set of d attributes, the key of the schema. Hereafter, we use the terms tuple and key interchangeably wherever the meaning is clear. A similarity query is originated at a PDN node and is answered by locating at least one replica of all the tuple(s) with key *similar* to the query key. A PDN access method is a mechanism that defines 1) how to organize the PDN topology (interconnection) to an index-like structure, and 2) how to use the index structure to process the similarity queries. We are interested in the access methods for efficient processing of similarity queries in PDNs.

We model the PDN key space as a Hilbert space $(V, \mathcal{L}_p)$. $V = V_1 \times V_2 \times ... \times V_d$ is a d-dimensional vector space, where V_i, the domain of the attribute a_i for the key $\overrightarrow{k} = \langle a_1, a_2, ..., a_d \rangle$ in V, is a contiguous and finite interval of $\mathbb{R}$. The $\mathcal{L}_p$ norm with $p \in \mathbb{Z}^+$ is the distance function to measure the dissimilarity (or equivalently *similarity*) between two keys $\overrightarrow{k_1}$ and $\overrightarrow{k_2}$ as $\mathcal{L}_p(\overrightarrow{k_1} - \overrightarrow{k_2})$, where $\mathcal{L}_p(\overrightarrow{x}) = \left(\sum_{i=1}^{d} |x_i|^p\right)^{\frac{1}{p}}$.

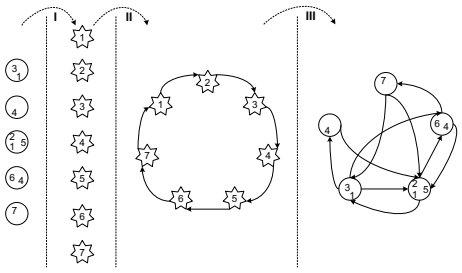


Figure 1: Reducing the general PDN model

We are interested in content-based access methods, i.e., access methods that organize the PDN topology based on the content of the PDN nodes. In general each PDN node may include more than one tuple. For better explanation of our content-based access methods, without loss of generality, we find it simple to assume a PDN model where each node stores *one and only one* tuple. To justify this assumption, here we show how to reduce the general PDN model to our assumed PDN model. Consider K as the set of keys (tuples) available in PDN and N as the set of PDN nodes. Assuming a general PDN model, we define a one-to-many mapping $\mathcal{M} : N \rightarrow K$ that maps each PDN node to the set of keys stored at the node[1] (Figure 1, Step I). Each key is considered as a *virtual* node embedded in V. Note that since tuples are replicated, there might be several virtual nodes with the same key. A content-based access method defines how to organize the set of virtual nodes corresponding to all nodes in N to a *virtual* PDN with particular topology and how to process the queries in the virtual PDN (Figure 1, Step II). Finally, the topology of the actual PDN is deduced by inverse mapping from the topology of the virtual PDN: a PDN node n is connected to a node m if and only if at the virtual PDN some virtual node in $\mathcal{M}(n)$ is connected to some other virtual node in $\mathcal{M}(m)$ (Figure 1, Step III). Also, the semantic of the query processing at the actual PDN nodes is defined by the query processing semantic at the corresponding virtual nodes such that the flow of the query at the actual PDN is logically identical to that of the virtual PDN. With this approach, the mapping and inverse mapping steps (Steps I and III) are independent of the access method used in Step II, and each access method for virtual PDNs (which is a PDN with only one tuple per node) defines an access method with similar characteristics for general PDNs. Hereafter, we assume the reduced model for PDNs and characterize the primitives of an access method to construct the topology/index and process the queries in such a PDN.

The topology of a PDN can be modelled as a directed graph $G(N, E)$, where the edge $e(n, m) \in E$ represents an asymmetric neighborhood relationship in which node m is a neighbor of node n. Schematically, we depict this relationship by drawing an arrow from node n to node m. $\mathcal{A}(n)$ is the set of neighbors for the node n. To achieve scalability, a node only maintains a limited amount of information about its neighbors, which includes the key of the tuples maintained at the neighbors and the physical addresses of the neighbors. A node can directly communicate with its neighbors. To construct the PDN index, an access method defines the *join* primitive[2] (similar to the *insert* operation

[1]Depending on the PDN application, if some of the data objects within a node are closely similar, then alternatively $\mathcal{M}$ can map a node to the centroid of the similar objects. Without loss of generality, we focus on the general case where the objects within a node are not closely similar.

[2]This *join* is different from the *join* operation in the rela-

with the traditional database access methods), which is used by the new node n to delineate $\mathcal{A}(n)$ as it joins the existing PDN. We assume that at least the physical address of one node in the existing PDN (if any) is available to n as it joins the PDN. As the new nodes join the PDN, its topology incrementally converges to the intended index structure. Similarly an access method defines the *leave* operation (equivalent to the delete primitive with the traditional access methods).

We are interested in the following types of similarity queries:

- **Exact-Match Query**: Given the query key $\overrightarrow{q}$, return the tuple t with key $\overrightarrow{k}$ such that $\overrightarrow{k} = \overrightarrow{q}$.
- **Range Query**: Given the query key $\overrightarrow{q}$ and the range $\mathbf{r}$, return all tuples t with key $\overrightarrow{k}$ such that $\mathcal{L}_p(\overrightarrow{k} - \overrightarrow{q}) \leq \mathbf{r}$.
- **k-Nearest-Neighbor (kNN) Query**: Given the query key $\overrightarrow{q}$ and the number $\mathbf{k}$, return the $\mathbf{k}$-ary $(t_1, t_2, ..., t_{\mathbf{k}})$ such that $\overrightarrow{k_i}$, key of t_i, is the i-th nearest neighbor of the key $\overrightarrow{q}$.

A similarity query can originate from any PDN node at T_0-th time slot ($\forall\ T_0 \in \mathbb{Z}$), assuming a discrete wall-clock time with fixed time unit. A node that originates a query or receives the query from other nodes at the $(T_0 + i)$-th time slot ($\forall\ i \in \mathbb{Z}^+ \cup \{0\}$), can process the query locally and/or forward zero or more processed replicas of the query to its immediate neighbors at the $(T_0 + i + 1)$-th time slot. The collective processing of the query by the PDN nodes is completed when all expected tuples in the relevant result set of the query are visited by at least one of the replicas of the query. Besides the join and leave primitives, an access method defines the *forward* primitive for query processing based on the constructed PDN index. The forward primitive can only use the information at the local node to process the query and to make forwarding decisions. During query processing, the $\mathcal{L}_p$ distance between the query key $\overrightarrow{q}$ and the local key is computed to verify if the local tuple satisfies the query condition. Also, with content-based access methods the forward primitive may measure the $\mathcal{L}_p$ distances between the query key $\overrightarrow{q}$ and the neighbor keys to guide the query.

Finally, we define metrics to evaluate the efficiency of a PDN access method. An access method can be evaluated based on its construction cost, and/or based on its query processing cost and performance. Unless the set of nodes participating in PDN is extremely dynamic, the computation (CPU time) and communication costs of constructing and maintaining the index structure are negligible as compared to those of the query processing. On the other hand, the space required to construct the distributed index is proportional to:

$$\mathbf{S} = \sum_{\forall n \in N} |\mathcal{A}(n)|$$

Assuming a peer-to-peer model for a PDN, a uniform distribution of the space $\mathbf{S}$ (or equivalently, uniform distribution of the node connectivity) among all the nodes is favorable. We define three other metrics to measure the efficiency of a PDN access method for query processing. The first two metrics evaluate the cost of query processing in terms of the required system resources, whereas the last one measures the system performance from the user perspective:

1. *Communication cost* ($\mathbf{C}_1$): Average number of query replicas forwarded to complete the processing of a query.
2. *Computation cost* ($\mathbf{C}_2$): Average number of $\mathcal{L}_p$ distance computations to complete the processing of a query.
3. *Query time* ($\mathbf{T}$): Average response-time of a query. If processing of a query starts at time slot T_0 and completes at time slot T_1, the response-time of the query is equal to $T_1 - T_0$.

2.2 Basic Access Method

To set a lower bound for the efficiency of the PDN access methods, we consider an access method that naively scans the PDN nodes to resolve the similarity queries. Optimally, with scanning $\mathbf{C}_1$ and $\mathbf{C}_2$ are $\Theta(|N|)$, and $\mathbf{T}$ is $\Theta(1)$. With PDNs, since queries can be replicated, scanning is not necessarily sequential; hence, $\mathbf{T}$ can in fact be independent of $|N|$. With any connected topology, simple flooding of the query ensures a complete scan. However, various topologies balance the efficiency metrics differently. A star topology is theoretically optimal, but star is not a realistic topology for PDNs because it fails to distribute $\mathbf{S}$ uniformly. Other examples of the topologies that allow more uniform distribution of $\mathbf{S}$ are ring, spanning tree, and random graph, which all are near optimal in terms of $\mathbf{C}_1$ and $\mathbf{C}_2$ metrics, and $O(|N|)$, $O(\log|N|)$, $O(\log|N|)$ in terms of $\mathbf{T}$, respectively.

We consider an access method with a random graph index topology and flood-based query processing as the benchmark. The random graph $G_{N,p}$ is a graph with $|N|$ nodes, where each pair of nodes are connected with probability p. Such a graph has Poisson connectivity distribution with average connectivity $p\ |N|$. Thus, the space requirement $\mathbf{S}$ of the index is fairly distributed among the nodes. Also, it is shown that for $p > \frac{\log|N|}{|N|}$ the graph is connected with high probability [5]. Such a graph has $|E| = |N|\log|N|$ edges and the average distance between any two nodes is $O(\log|N|)$; hence, $\mathbf{C}_1$ is $O(|N|\log|N|)$, $\mathbf{C}_2$ is $O(|N|)$, and $\mathbf{T}$ is $O(\log|N|)$. Random graphs are frequently used to model large networks (such as the Internet), and since they are defined probabilistically, as compared to loop-free spanning trees with fixed number of edges $|N| - 1$, they are more appropriate for modelling the dynamic autonomous PDNs.

3. SWAM: A FAMILY OF SMALL-WORLD-BASED ACCESS METHODS

We define a family of efficient access methods for PDNs, termed *Small-World Access Methods (SWAM)*, which is designed based on the principles borrowed from the small-world models. Here, after a general overview of the useful properties of the small-world model, we define the SWAM family and characterize its properties. Also, as an example we introduce SWAM-V, a Voronoi-based instance of SWAM, which satisfies SWAM properties and achieves query time, communication cost, and computation cost logarithmic to the size of the network for all types of similarity queries.

3.1 Small-World Model as an Index Structure

The small-world model is a network topology proposed to explain the small-world phenomenon, the fact that two individuals in a society (i.e., a social network) can efficiently locate each other through a short chain of acquaintances logarithmic to the size of the network [22, 14]. The small-world graph is a hybrid graph, a superimposition of a regular grid and a dilute random graph ($p \ll 1$), inheriting

tional algebra.

both their properties (see Figure 2-a). It inherits average node-to-node path length $O(\log |N|)$ from the random graph component, and high *clustering* property from the grid. A graph is clustered if the neighbors of a node are more probably the neighbors of each other rather than the neighbors of the other nodes in the network. For a node n clustering is measured by the clustering coefficient $C(n)$, which is the realized fraction of all possible edges among the neighbors of n:

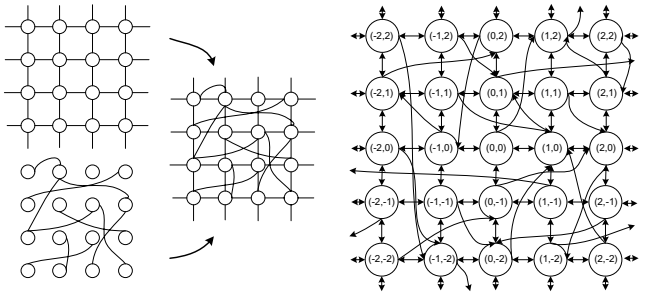
$$C(n) = l \Big/ \begin{pmatrix} |\mathcal{A}(n)| \\ 2 \end{pmatrix} \quad (1)$$

where l is the number of existing edges among the neighbors of n. The clustering coefficient of a graph is the average of the clustering coefficients of its nodes. For a complete graph, a grid, and a dilute random graph $G_{N,p}$, the clustering coefficients are 1, $\simeq \frac{3}{4}$, and $p \ll 1$, respectively.

To demonstrate a direct application of the small-world graph as an index structure for a PDN, we consider the following simple PDN. Assume the key space V is a subspace of $\mathbb{Z}^d$ rather than $\mathbb{R}^d$, and also assume all possible keys in V are available within the PDN, one key per PDN node. We can organize the topology of this PDN based on a small-world graph with a d-dimensional underlying grid as follows:

1. Grid component: The node storing the key $\overrightarrow{k} = \langle a_1, a_2, ..., a_d \rangle$ is a neighbor of all nodes with keys $\overrightarrow{k'}$ where $\mathcal{L}_p(\overrightarrow{k} - \overrightarrow{k'}) \leq b$ ($b \in \mathbb{Z}^+$); and
2. Random graph component: The node n_k storing the key $\overrightarrow{k} = \langle a_1, a_2, ..., a_d \rangle$ is a neighbor of one other node $n_{k'}$ with key $\overrightarrow{k'}$ selected probabilistically such that if $\mathcal{L}_p(\overrightarrow{k} - \overrightarrow{k'}) = x$, the probability of selecting n'_k as the neighbor of n_k is proportional to x^{-d} (i.e., a power-law distribution).

See Figure 2-b for an example with 2-dimensional key space, $\mathcal{L}_1$ as the distance measure, and neighborhood boundary parameter $b = 1$. In [14], it is shown that with a greedy forwarding primitive, on average an exact-match query is resolved with $\mathbf{T}$, $\mathbf{C}_1$, and $\mathbf{C}_2$ all in $O(\log |N|)$. With the greedy forwarding, node n forwards a query $\overrightarrow{q}$ only to one of its neighbors with key $\overrightarrow{k}$ such that $\mathcal{L}_p(\overrightarrow{k} - \overrightarrow{q})$ is minimum among all neighbors in $\mathcal{A}(n)$, i.e., the neighbor with the most similar key to the query key $\overrightarrow{q}$ is selected to receive the query. It is easy to see the underlying grid topology ensures that when a node with key $\overrightarrow{k}$ receives a query $\overrightarrow{q}$, always either $\overrightarrow{k} = \overrightarrow{q}$ or the node has at least one neighbor with the key $\overrightarrow{k'}$ such that $\mathcal{L}_p(\overrightarrow{k'} - \overrightarrow{q}) < \mathcal{L}_p(\overrightarrow{k} - \overrightarrow{q})$.



a. Hybrid small-world graph b. Small-world as PDN index

Figure 2: The small-world model

Therefore, along the forwarding path of the query, the distance between the key at the current node and the target key $\overrightarrow{q}$ is monotonically decreasing as the query is forwarded. Besides, the probabilistically selected neighbors act as long jumps that ensure exponential decrease of this distance on average. Thus, the average forwarding path length is logarithmic to the size of the network.

The way we defined the neighborhood relationship between the PDN nodes based on the distance between their keys, together with the clustering property of the resulting small-world topology allow for the effective execution of other types of similarity queries as well. On one hand, we defined the neighborhood relationship such that neighbors of a node have keys closely similar to the key of the node, and consequently, similar to each other. On the other hand, due to the clustering property of the generated small-world graph, neighbors of a node are closely connected in terms of the hop-count in the network (i.e., number of the edges on the path between each pair of nodes). Therefore, a *locality* of tightly connected nodes with closely similar keys is created at the neighborhood of each node in the network. With a topology constructed out of such localities, range and **k**NN queries can be executed efficiently in two phases, first, by an exact-match query to locate the locality of the query key $\overrightarrow{q}$, and second, by flooding the query throughout the locality of $\overrightarrow{q}$. With a localized topology, flooding at the locality of the query key is efficient. We can locate all the keys relevant to the range and **k**NN queries in a limited number of hops h away from $\overrightarrow{q}$, where h is independent of the size of the network $|N|$. With our simple PDN example, for range and **k**NN queries all the relevant keys (and almost only relevant keys) are visited within $h = O(r)$ and $h = O(\lceil \mathbf{k}^{\frac{1}{d}} \rceil)$ hops from $\overrightarrow{q}$, respectively. Therefore, for both types of queries, $\mathbf{T}$ is $O(\log |N| + h)$, $\mathbf{C}_1$ is $O(\log |N| + h^d)$, and $\mathbf{C}_2$ is $O(d \log |N| + h^d)$.

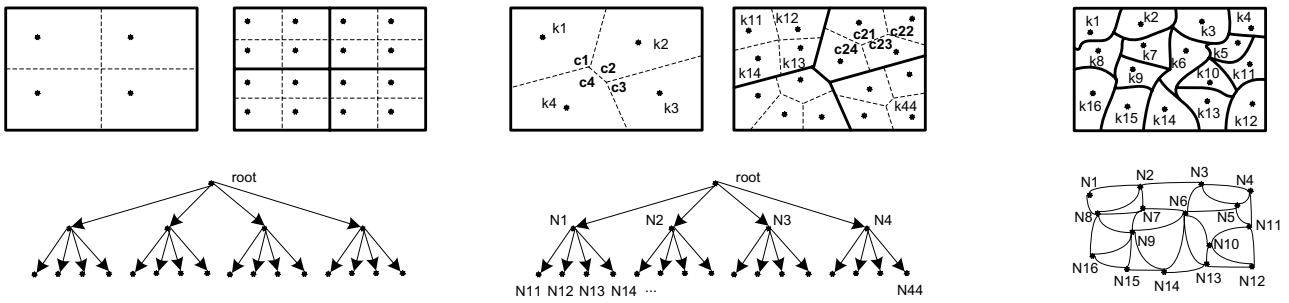
With an inclusive key space $V \subset \mathbb{Z}^d$, the simple PDN example considered here is only of illustrative significance. We, however, use the same properties to develop SWAM that applies to more general PDN models.

3.2 SWAM Family

Almost all the traditional access methods for database systems are based on one core idea to reduce the search space for efficient access (see the unified model in [7]). They recursively partition the key space into a set of disjoint *similarity* classes[3]. An index is then constructed as a hierarchy of the class representatives at successive levels (see Figure 3-a). The hierarchical index allows filtering out (i.e., to dismiss without inspection) the irrelevant/dissimilar classes while query is directed from the root of the hierarchy toward the similarity class of the query key. The average query time is logarithmic to the size of the database.

By mapping each node of the hierarchy to a PDN node, the same idea can be directly applied to index PDNs, although as we show later the resulting distributed hierarchical index structure is not appropriate for PDNs. Consider K as the set of keys available in a PDN. Any similarity-based relation can be used to partition the key space. For example, in Figure 3-b, V is recursively partitioned based on the GNA approach [6]. Starting from V as the global similarity class, at each level the parent similarity class c with the class representative $\overrightarrow{k} \in K$ is partitioned into a set

[3]The generic mathematical term for *similarity* class is *equivalence* class. Here, the equivalence relations that partition the space are based on the distance (or *similarity*) between the keys.



a. Recursive partitioning b. Recursive partitioning example: GNA c. Flat partitioning

Figure 3: Partitioning of the key space

of h disjoint subclasses c_i with representative keys $\overrightarrow{k_i} \in K$ ($i \in I_h = [1..h]$) such that $c_i = \{\overrightarrow{k'} \in V | \mathcal{L}_p(\overrightarrow{k'} - \overrightarrow{k_i}) < \mathcal{L}_p(\overrightarrow{k'} - \overrightarrow{k_j}), \forall j \neq i\}$. Considering that in a PDN each key $\overrightarrow{k}$ resides at a PDN node n_k, the GNA-tree corresponding to such a space partitioning is a *distributed* GNA-tree in which $\mathcal{A}(n_k) = \{n \in N | n = n_{k_i}, i \in I_h\}$. Query processing with such a distributed index tree is similar to that of its corresponding centralized counterpart, with query actually traversing a physically constructed tree rather than a tree structure in memory. Although this indexing approach may seem appealing, due to the lack of a balance load among its nodes, is inappropriate for PDNs. The unbalance load is evident by observing that nodes which represent larger similarity classes (i.e., nodes at the higher levels of the hierarchy) receive more queries to process. In the extreme case, the root of the hierarchy processes all queries. Besides, as mentioned in Section 2.2 hierarchical structures are loop-free and intolerant to failures and/or autonomous behaviors of the PDN nodes.

SWAM also employs the space partitioning idea; however, to avoid the problems with hierarchies, instead of recursive partitioning assumes a flat partitioning (see Figure 3-c). Each key $\overrightarrow{k} \in K$ (or $n_k \in N$) represents its own similarity class $c_k \subseteq V$ and the set of $|K|$ similarity classes are collectively exhaustive $V = \bigcup_{k \in K} c_k$ and mutually exclusive $c_k \cap c_{k'} = \varnothing, \overrightarrow{k} \neq \overrightarrow{k'}$. An uncharacteristic case is where two or more nodes store replicas of the same key $\overrightarrow{k}$. We assume all such nodes represent the same class c_k redundantly. Such a partitioning scheme can potentially balance the query processing load among PDN nodes. With hierarchies, neighborhood relationship between a pair of nodes is directly derived from parent-child relationship between their corresponding similarity classes to reflect the similarity between their classes. Similarly, with flat partitioning we define the neighborhood relationship based on the adjacency relationship between the similarity classes $\mathcal{A}(n_k) = \{n_{k'} \in N | c_k \textit{ and } c_{k'} \textit{ are adjacent}, k' \in K\}$. The resulting index structure is a graph instead of a loop-free tree. Besides, processing of the query can start from any node (e.g., the actual query originator) rather than exclusively from a unique node, the root.

The challenge is to define the similarity-based partitioning relation such that the resulting graph-based index structure bears indexing characteristics similar to those of the hierarchical index structures. Particularly, it should allow filtering of (i.e., avoid visiting) the irrelevant classes effectively as query is directed from a query originator toward the similarity class of the query key. Moreover, to support range and **k**NN similarity queries effectively, alike hierarchical index structures similar classes should be in proximity of each other in terms of the hop-count in the index topology. Finally, the $O(\log N)$ expected query time achieved by the hierarchies is also desirable with the graph-based index structure. As outlined in Section 3.1, these requirements are addressed by the properties of a basic small-world graph. A SWAM index structure is a general graph-based index structure that satisfies a generalization of the same properties as follows:

Property 1 : Monotonic approach toward query key When a node with key $\overrightarrow{k}$ receives a query $\overrightarrow{q}$, always either $\overrightarrow{q} \in c_k$, or the node has at least one neighbor with a key $\overrightarrow{k'}$ such that $\mathcal{L}_p(\overrightarrow{k'} - \overrightarrow{q}) < \mathcal{L}_p(\overrightarrow{k} - \overrightarrow{q})$. Consequently, if the node n_k receives the query $\overrightarrow{q}$, it is guaranteed that for all $\overrightarrow{k''} \in \{\overrightarrow{j} \in K | \mathcal{L}_p(\overrightarrow{j} - \overrightarrow{q}) \geq \mathcal{L}_p(\overrightarrow{k} - \overrightarrow{q})\}$ the node $n_{k''}$ will never be visited in future during the greedy forwarding, and the similarity class $c_{k''}$ is filtered.

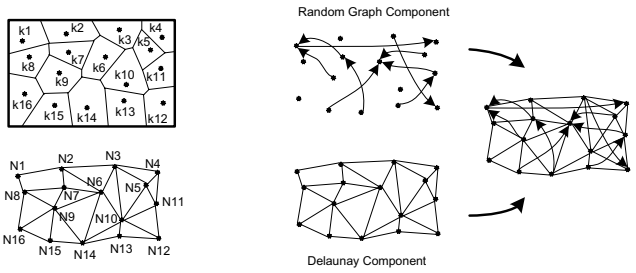
Property 2 : Localized index topology With a localized index, for each node n_k the set of nodes at its neighborhood $\mathcal{A}(n_k)$ are tightly connected and store keys closely similar to $\overrightarrow{k}$. We measure these two characteristics with the two metrics Clustering Coefficient (CC) and Neighbor Distance Distribution (NDD), respectively. For a node n, $\text{CC}_n = C(n)$ is defined by Equation 1. For a graph G, $\text{CC}_G = \frac{1}{|N|} \sum_{\forall n \in N} \text{CC}_n$. Also, NDD is the probability distribution function of the random variable $\overline{X} = \mathcal{L}_p(\overrightarrow{k'} - \overrightarrow{k})$, $\forall n_k \in N \; \forall n_{k'} \in \mathcal{A}(n_k)$. As we discussed in Section 3.1, a localized topology allows efficient processing of the range and **k**NN similarity queries.

Property 3 : Logarithmic forwarding-path length For an exact-match query (processed by greedy forwarding), on average $\mathbf{T} = O(\log N)$.

Any graph-based index structure that maintains these SWAM properties is a member of the SWAM family. In Section 3.3, we introduce an example SWAM index structure.

3.3 SWAM-V: A Voronoi-based SWAM

SWAM-V partitions the key space V to a Voronoi diagram [17] (see Figure 4-a). For each key $\overrightarrow{k_i} \in K$ ($i \in I_{|K|}$), $n_{k_i} \in N$ represents the similarity class $c_{k_i} = \{\overrightarrow{k} \in V | \mathcal{L}_p(\overrightarrow{k} - \overrightarrow{k_i}) < \mathcal{L}_p(\overrightarrow{k} - \overrightarrow{k_j}), \forall j \neq i\}$, which is the *Voronoi cell* of n_{k_i}. Accordingly, the neighborhood of the node n_{k_i} is defined as $\mathcal{A}(n_{k_i}) = \{n_{k_j} \in N | c_{k_i} \textit{ and } c_{k_j} \textit{ are adjacent}, \forall j \in I_{|K|}\}$.



a. Voronoi diagram and dual Delaunay graph

b. SWAM-V topology

Figure 4: SWAM-V index structure

```
// Join Delaunay graph
n_{k_i} ←Exact-Match(k_h);
n_next ← n_{k_i};
repeat {
      m ← n_next;
      n_next ← Update(m);
      A(n_{k_h}) ← A(n_{k_h}) ∪ {m};
} until (n_next = n_{k_i});

// Join random graph
A(n_{k_h}) ← A(n_{k_h}) ∪ {n_random};
```

Figure 5: SWAM-V join primitive

Nodes that store replicas of the same key share the same neighborhood; i.e., if $\overrightarrow{k_i} = \overrightarrow{k_j}$, $\mathcal{A}(n_{k_i}) = \mathcal{A}(n_{k_j})$. The resulting graph is the dual Delaunay graph of the Voronoi diagram and is unique for each diagram (see Figure 4-a). Since the neighborhood relationship is symmetric, the Delaunary graph is depicted as an undirected graph. The SWAM-V topology consists of a random graph component (identical to that of the small-world graph) that is superimposed over the Delaunay graph (see Figure 4-b).

THEOREM 1. *The SWAM-V index structure satisfies the SWAM Property 1.*

PROOF. The (extended) boundary between the cells of two neighboring nodes $\overrightarrow{k_i}$ and $\overrightarrow{k_j}$ is defined as $B(\overrightarrow{k_i}, \overrightarrow{k_j}) = \{\overrightarrow{k} \in V | \mathcal{L}_p(\overrightarrow{k} - \overrightarrow{k_i}) = \mathcal{L}_p(\overrightarrow{k} - \overrightarrow{k_j})\}$. The boundary $B(\overrightarrow{k_i}, \overrightarrow{k_j})$ bisects the space into two half-spaces, where $H(\overrightarrow{k_i}, \overrightarrow{k_j}) = \{\overrightarrow{k} \in V | \mathcal{L}_p(\overrightarrow{k} - \overrightarrow{k_i}) < \mathcal{L}_p(\overrightarrow{k} - \overrightarrow{k_j})\}$ is the *similarity-dominance space* of $\overrightarrow{k_i}$ over $\overrightarrow{k_j}$, and vice versa. The similarity class of $\overrightarrow{k_i}$ is alternatively defined as $c_{k_i} = \bigcap_{j \in I_{|K|} \setminus \{i\}} H(\overrightarrow{k_i}, \overrightarrow{k_j})$.

Assume node n_{k_i} receives a query $\overrightarrow{q}$. If $\overrightarrow{q} \notin c_{k_i}$, then $\overrightarrow{q} \notin \bigcap_{j \in I_{|K|} \setminus \{i\}} H(\overrightarrow{k_i}, \overrightarrow{k_j})$. Hence, $\overrightarrow{q} \in \bigcup_{j \in I_{|K|} \setminus \{i\}} H(\overrightarrow{k_j}, \overrightarrow{k_i})$. Therefore, $\exists j \in I_{|K|} \setminus \{i\}, \mathcal{L}_p(\overrightarrow{q} - \overrightarrow{k_j}) < \mathcal{L}_p(\overrightarrow{q} - \overrightarrow{k_i})$. □

The SWAM-V index structure also satisfies Properties 2 and 3. In Section 4, we verify Property 2 by measuring the clustering coefficients NDD and CC. Also, Property 3 follows from the same property of the small-world graph [14]. Due to the lack of space, here we omit the proof for this claim. We include the proof in an extended version of this paper. Below, we describe the primitives of the SWAM-V access method.

3.3.1 Index Construction

As PDN nodes join the network, the SWAM-V index structure is incrementally constructed. Consider a PDN of $h-1$ nodes n_{k_1} to $n_{k_{h-1}}$. The new node n_{k_h} executes the join primitive shown in Figure 5 to join the two components of the SWAM-V index structure, i.e., the Delaunay graph and the random graph. We assume that n_{k_h} has access to at least one of the nodes n_{k_1} to $n_{k_{h-1}}$, say n_{k_a}. Through n_{k_a}, n_{k_h} issues an exact-match query[4] (see Section 3.3.2.1) for key $\overrightarrow{q} = \overrightarrow{k_h}$ to locate the node n_{k_i} such that $\overrightarrow{k_h} \in c_{k_i}$. Thereafter, n_{k_h} constructs its cell c_{k_h} one border at a time, starting from the border with n_{k_i}. Again through n_{k_a}, n_{k_h} sends an *update* request to n_{k_i}. The update request is forwarded like an exact match query with $\overrightarrow{q} = \overrightarrow{k_i}$ to reach n_{k_i}. Upon receiving the update request, n_{k_i} calculates the bisector $B(\overrightarrow{k_i}, \overrightarrow{k_h})$, and updates its neighborhood $\mathcal{A}(n_{k_i})$ based on its new divided cell. In response, n_{k_i} sends the address of its neighbor n_{next} to n_{k_h} (via n_{k_a}), where n_{next} is a neighbor of n_{k_i} such that $B(\overrightarrow{k_i}, \overrightarrow{k_h}) \cap B(\overrightarrow{k_i}, \overrightarrow{k_{next}}) \neq \varnothing$. Since a Voronoi cell is a convex hull, there are at least two such neighbors for n_{k_i}. After receiving the update response, the new node n_{k_h} updates its neighborhood by $\mathcal{A}(n_{k_h}) \leftarrow \mathcal{A}(n_{k_h}) \cup \{n_{k_i}\}$ and repeats the same update procedure with n_{next}. The update is completed when all borders of the Voronoi cell c_{k_h} are found, i.e., when n_{k_h} receives n_{k_i} as the response of an update request. An exception to the procedure described above is when $\overrightarrow{k_h} = \overrightarrow{k_i}$. In such a case, the new key is a new replica of an existing key and $c_{k_h} = c_{k_i}$.

To complete the construction of the SWAM-V index structure, in addition to the Delaunay graph the new node n_{k_h} must also join the random graph component by selecting a node n_{random} such that $\mathcal{L}_p(\overrightarrow{k_h} - \overrightarrow{k_{random}})$ follows the power-law distribution. We piggy-back this step to the previous step by having n_{k_h} select n_{random} among all the nodes that are previously visited by the update requests. This step completes the execution of the join primitive. The leave primitive implements the reverse procedure and is trivial. We omit description of the leave primitive due to lack of space.

3.3.2 Query Processing

Here, we describe three forwarding primitives to process various types of queries using the SWAM-V index structure.

3.3.2.1 Exact-Match Query.

An exact-match query is executed by greedy forwarding. When node n_k receives a query $\overrightarrow{q}$, if $\mathcal{L}_p(\overrightarrow{k} - \overrightarrow{q}) < \min_{\forall n_{k_i} \in \mathcal{A}(n_k)} \mathcal{L}_p(\overrightarrow{k_i} - \overrightarrow{q})$, then $\overrightarrow{q} \in c_k$. Therefore, either $\overrightarrow{q} = \overrightarrow{k}$ or $\overrightarrow{q} \notin K$, where in both cases query is terminated with the result sets $R = \{n_k\}$ and $R = \varnothing$, respectively. Otherwise, n_k continues forwarding the query by sending the query to one of its neighbors n_{k_m} such that $\mathcal{L}_p(\overrightarrow{k_m} - \overrightarrow{q}) = \min_{\forall n_{k_i} \in \mathcal{A}(n_k)} \mathcal{L}_p(\overrightarrow{k_i} - \overrightarrow{q})$. From Theorem 1, we know that in worst-case the greedy forwarding terminates in $|N|$ hops.

3.3.2.2 Range Query.

A range query with query key $\overrightarrow{q}$ and range $\mathbf{r}$ is executed in two successive phases: 1) to locate the *locality* (or similarity class) of the query key, and 2) to explore the nodes located within the query sphere with radius $\mathbf{r}$ centered at the locality of the query key (see Figure 6). At Phase 1, the query is interpreted as an exact-match query with query key $\overrightarrow{q}$. This

[4]In Section 2.1, we defined the query result as a tuple set. Here, equivalently we consider the address of the node(s) storing the resulting tuple(s) as the query result to explain the implementation of the access method primitives.

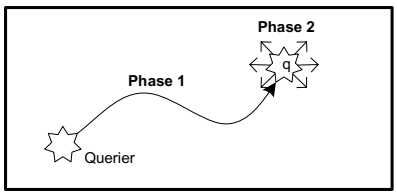


Figure 6: SWAM-V range query

phase terminates when the query reaches n_k such that $\overrightarrow{q} \in c_k$. At Phase 2, starting from n_k, each node n that receives the query for the first time, forwards it to all its neighbors $m_{k'} \in \mathcal{A}(n)$ if and only if $\mathcal{L}_p(\overrightarrow{k'} - \overrightarrow{q}) < \mathbf{r}$. Theorem 2 ensures that when the selective flooding is terminated, all nodes $n_{k''} \in N$ with $\mathcal{L}_p(\overrightarrow{k''} - \overrightarrow{q}) < \mathbf{r}$ are visited by some query replica.

THEOREM 2. *SWAM-V answers all range queries without any false dismissal.*

PROOF. Assume n_k is the nearest neighbor to the query key $\overrightarrow{q}$; i.e., $\overrightarrow{q} \in c_k$. We prove the theorem by contradiction. Suppose $\exists\ n_{k''} \in N$ such that $\mathcal{L}_p(\overrightarrow{k''} - \overrightarrow{q}) < \mathbf{r}$, but $n_{k''}$ is not visited. From Theorem 1, we know that between $n_{k''}$ and n_k there exist a path $\mathcal{P}_{k''}$, a sequence of neighboring nodes $n_{k_h=k''}, n_{k_{h-1}}, ..., n_{k_2}, n_{k_1=k}$, such that $\mathcal{L}_p(\overrightarrow{k_{i-1}} - \overrightarrow{q}) \leq \mathcal{L}_p(\overrightarrow{k_i} - \overrightarrow{q})\ \forall i \in [2..h]$. Therefore, $\forall i \in [1..h-1]$, $\mathcal{L}_p(\overrightarrow{k_i} - \overrightarrow{q}) < \mathbf{r}$. However, the node $n_{k=k_1}$ receives the query at the end of Phase 1. Thus, based on the condition for selective flooding $\forall i \in [2..h]$, n_{k_i} must also receive the query. Hence, $n_{k''=k_h}$ receives the query, contradicting our assumption. □

With Property 2, the flooding time is proportional to the range $\mathbf{r}$ and independent of the size of the network $|N|$ (see Section 3.3.3).

3.3.2.3 k*NN Query.*

Similar to range queries, a $\mathbf{k}$NN query is executed in two phases. As with range queries, Phase 1 is equivalent to an exact-match query for the query key $\overrightarrow{q}$ to locate n_k such that $\overrightarrow{q} \in c_k$. Since $\forall k' \in K \backslash \{k\}$, $\mathcal{L}_p(\overrightarrow{k} - \overrightarrow{q}) < \mathcal{L}_p(\overrightarrow{k'} - \overrightarrow{q})$, the node n_k is the 1st nearest neighbor $n_{k_{1NN}}$ of the query $\overrightarrow{q}$. At Phase 2, the rest of the $\mathbf{k}$ nearest neighbors, $n_{k_{2NN}}$, $n_{k_{3NN}}$, ..., $n_{k_{\mathbf{k}NN}}$, are located following Theorem 3.

THEOREM 3. *The $\mathbf{k}$-th nearest neighbor to $\overrightarrow{q}$ is a neighbor of one of the nearer neighbors of $\overrightarrow{q}$; i.e., $\forall\ \mathbf{k} \in [2..|K|]$, $n_{k_{\mathbf{k}NN}} \in \bigcup_{i \in [1..\mathbf{k}-1]} \mathcal{A}(n_{k_{iNN}})$.*

PROOF. We prove the theorem by contradiction. Suppose $n_{k_{\mathbf{k}NN}} \notin \bigcup_{i \in [1..\mathbf{k}-1]} \mathcal{A}(n_{k_{iNN}})$. From Theorem 1, we know $\exists\ m_{k'} \in \mathcal{A}(n_{k_{\mathbf{k}NN}})$ such that $\mathcal{L}_p(\overrightarrow{k'} - \overrightarrow{q}) < \mathcal{L}_p(\overrightarrow{k_{\mathbf{k}NN}} - \overrightarrow{q})$. Therefore, $m_{k'} \in \{n_{k_{iNN}} | i \in [1..\mathbf{k}-1]\}$. Hence, $n_{k_{\mathbf{k}NN}} \in \bigcup_{i \in [1..\mathbf{k}-1]} \mathcal{A}(n_{k_{iNN}})$. □

Thus, at Phase 2, starting from $n_{k_{1NN}} = n_k$, query is forwarded from $n_{k_{iNN}}$ to $n_{k_{(i+1)NN}}, i \in [1..\mathbf{k}-1]$ until it visits all $\mathbf{k}$ nearest neighbors of $\overrightarrow{q}$. When the i-th nearest neighbor $n_{k_{iNN}}$ receives the query, it locates the next node $n_{k_{(i+1)NN}}$ as follows. The $(i+1)$-th nearest neighbor is $n_{k_{(i+1)NN}} \in \bigcup_{j \in [1..i]} \mathcal{A}(n_{k_{jNN}})$ such that $\mathcal{L}_p(\overrightarrow{k_{(i+1)NN}} - \overrightarrow{q}) = \min_{\forall m_{k'} \in \bigcup_{j \in [1..i]} \mathcal{A}(n_{k_{jNN}})} \mathcal{L}_p(\overrightarrow{k'} - \overrightarrow{q})$. $n_{k_{iNN}}$ forwards the query, which includes the set $\bigcup_{j \in [1..i-1]} \mathcal{A}(n_{k_{jNN}})$ received from the previous node piggy-backed with its own neighborhood $\mathcal{A}(n_{k_{iNN}})$, to the next node $n_{k_{(i+1)NN}}$. Phase 2 terminates in $\mathbf{k}$ hops.

3.3.3 *Analysis*

We evaluate the efficiency metrics for SWAM-V as follows:

- Space: From [19], we have:

$$\mathbf{S} = \begin{cases} \sum_{i=1}^{s} \frac{|N|}{i} \binom{|N|-i-1}{i-1} \binom{i}{2-i} & \text{d+1=2s} \\ \sum_{i=1}^{s} \frac{3}{i+1} \binom{|N|-i-1}{i} \binom{i+1}{2-i} & \text{d=2s} \end{cases}$$

  $\mathbf{S}$ is uniformly distributed among $n \in N$ and, e.g., with a 2-dimensional space $|\mathcal{A}(n)|_{average} = \mathbf{S}/|N| \simeq 6$.

- Query time: For the exact-match query, with Property 1 in the worst-case $\mathbf{T} = O(|N|)$, and with Property 3 on average $\mathbf{T} = O(\log|N|)$. Also, on average for the two-phase range queries (with selectivity $\mathbf{s}$) and $\mathbf{k}$NN queries, $\mathbf{T}$ is $O(\log|N| + \mathbf{s}N)$ and $O(\log|N| + \mathbf{k})$, respectively.

- Communication cost: For the exact-match query on average $\mathbf{C}_1 = O(\log|N|)$. Also, on average for range and $\mathbf{k}$NN queries, $\mathbf{C}_1$ is $O(\log|N| + \mathbf{s}N|\mathcal{A}(n)|_{average})$ and $O(\log|N| + \mathbf{k})$, respectively.

- Computation cost: Similarly, for the exact-match query on average $\mathbf{C}_2 = O(|\mathcal{A}(n)|_{average} \log|N|)$. Also, on average for range and $\mathbf{k}$NN queries, $\mathbf{C}_2$ is $O(|\mathcal{A}(n)|_{average}$ $(\log|N| + \mathbf{s}N))$ and $O(|\mathcal{A}(n)|_{average}(\log|N| + \mathbf{k}))$, respectively.

4. EXPERIMENTAL ANALYSIS

We conducted a set of experiments via simulation to *verify* the results of our study. Note that we have already shown analytically that SWAM-V has logarithmic query cost. We implemented a multi-thread simulator in C and used two Enterprise 250 Sun servers to preform the experiments. Based on the efficiency metrics introduced in Section 2.1, we compare SWAM-V versus our basic access method (see Section 2.2) as well as CAN [18], which is a multi-dimensional DHT.

4.1 Experimental Methodology

Our Monte Carlo simulation consists of a set of "runs". We setup each run by 1) generating a dataset, 2) distributing the dataset among a set of PDN nodes, 3) indexing the PDN with the three access methods SWAM-V, BASIC, and CAN, and 4) running 1000 queries (all of which either exact-match, range, or $\mathbf{k}$NN) and recording the average $\mathbf{T}$, $\mathbf{C}_1$, and $\mathbf{C}_2$ for each of the index structures as the result of the run. Each result data-point reported in Section 4.2 is the average result of 100 runs. The coefficient of variance across the runs was below 2.5% and hence show the stability of the result. Below, we explain the detail of each setup.

We consider the Hilbert spaces $(V, \mathcal{L}_1)$, $(V, \mathcal{L}_2)$, and $(V, \mathcal{L}_\infty)$ as the key/data space with $V = V_1 \times V_2 \times ... \times V_d$ as a d-dimensional hypercube, where $V_i = [-1, 1]$. We generate a dataset of $|K|$ keys by selecting each attribute $a_i \in V_i$ of a key $\overrightarrow{k} = \langle a_1, a_2, ..., a_d \rangle$ following either a uniform distribution, or a normal distribution[5] with expected value $\mu = 0$ and standard deviation $\sigma = 0.33$. We distribute the set of

[5] $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$

Access Methods	Exact-Match			Range ($\mathbf{s} = 1\%$)			$\mathbf{k}$NN ($\mathbf{k} = 5$)		
	$\mathbf{T}$	$\mathbf{C}_1$	$\mathbf{C}_2$	$\mathbf{T}$	$\mathbf{C}_1$	$\mathbf{C}_2$	$\mathbf{T}$	$\mathbf{C}_1$	$\mathbf{C}_2$
BASIC	3.77	16835.53	5000	3.77	16835.53	5000	3.77	16835.53	5000
SWAM-V	3.82	3.82	68.51	6.84	57.27	125.3	9.24	9.24	171.35
CAN	5.67	5.67	47.73	13.38	78.36	111.28	10.9	51.37	93.72

Table 1: Comparative study results with $d = 5$ and $N = 5000$

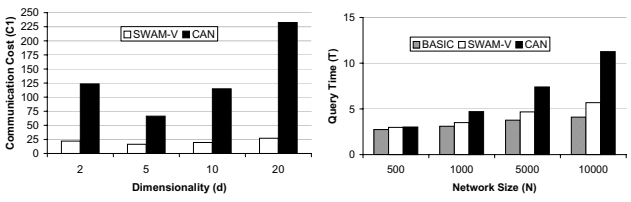
keys K among a set of nodes N, where $|N| = c^{-1}|K|$ with $c = 2$ as the data replication coefficient. With SWAM-V and BASIC, first keys are randomly assigned to the nodes and then the index structures (i.e., PDN topologies) are constructed based on the actual content of the nodes. For the comparison to be fair, we select p for the BASIC random graph $G_{N,p}$ such that $\mathbf{S}_{\texttt{BASIC}} = \mathbf{S}_{\texttt{SWAM-V}}$. On the other hand, with CAN keys cannot be assigned to the nodes randomly. The CAN index structure is first constructed based on the identifiers of the nodes (randomly selected from V) rather than their content. Each key is then assigned to a particular node with *similar* identifier.

We described query processing with SWAM-V and BASIC in Sections 3.3.2 and 2.2. The original CAN access method only supports exact-match queries via greedy forwarding. Even for the exact-match queries, since CAN does not satisfy the SWAM Property 1, during query forwarding the distance from the current node to the query key $\overrightarrow{q}$ may not be monotonically decreasing. Therefore, with greedy forwarding query may be trapped at a local optimal, i.e., at a node that does not have any neighbor closer than itself to $\overrightarrow{q}$, and consequently, query may never reach the target. To be fair to CAN, we excluded the CAN queries that result in false dismissal from our comparative study. Also, to implement range and $\mathbf{k}$NN queries with CAN we adopt the 2-phase query processing scheme from SWAM-V. However, since Theorems 2 and 3 do not hold for CAN, for both range and $\mathbf{k}$NN queries we use naive, non-selective flooding (similar to query processing with BASIC) in the second phase. We apply scope-limited flooding and only consider the time $\mathbf{T}$ and costs $\mathbf{C}_1$ and $\mathbf{C}_2$ required to complete the query.

For each query, we select the node that originates the query and the query key $\overrightarrow{q}$ randomly from N and K, respectively. For range queries, the range $\mathbf{r}$ is selected such that the selectivity $\mathbf{s}$ of the query varies from 0.01% to 1%. Finally, for $\mathbf{k}$NN queries we consider queries with $\mathbf{k} = 2, 5, 10$.

4.2 Experimental Results

The difference between the trend of the results for uniform and normal data distributions, as well as the results for various distance functions $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_\infty$, is insignificant. Here, we report the results for uniform data distribution with $\mathcal{L}_2$ as the distance function of the space. Table 1 shows a typical set of comparative results for a PDN with $d = 5$ dimensions and $N = 5000$ nodes. To achieve logarithmic query time, BASIC spends the communication and computation resources of the PDN unlimitedly. As illustrated, SWAM-V and CAN are both comparable to BASIC in terms of the query time, and invest the resources much more efficiently. SWAM-V consistently outperforms CAN in terms of both $\mathbf{T}$ and $\mathbf{C}_1$. We attribute this advantage to the SWAM three properties. Particularly, with Properties 1 and 2, the index structure accurately partitions the space and co-locates the similar content, enabling better filtering and less redundant query processing. Also, Property 3 allows more efficient traversal of the index structure. The cost of more accurate partitioning is higher connectivity of the nodes, and consequently, higher computation cost $\mathbf{C}_2$.



a. Dimensionality scaling b. Network size scaling

Figure 7: The scaling results

Figure 7 depicts the scaling properties of SWAM-V as compared to those of CAN. Figure 7-a illustrates how the communication cost of the $\mathbf{k}$NN queries with $\mathbf{k} = 10$ in a PDN of $N = 10000$ nodes scales as dimensionality of the data space increases. With SWAM-V, $\mathbf{C}_1$ remains almost unchanged at different dimensions, while CAN is less efficient at high-dimensional spaces. Theorem 3 ensures finding each next nearest-neighbor in one hop, irrespective of the dimensionality of the space. This property is enabled by the accurate Voronoi-based partitioning of the data space with SWAM-V. On the other hand, with inaccurate space partitioning CAN is forced to use blind flooding. With flooding, the communication cost of the query is proportional to the connectivity of nodes, which grows as the dimensionality of the space increases. Due to the "no-free-lunch" rule, with SWAM-V the extra complexity associated with higher dimensionality is shifted to the index construction time, but since index construction occurs in a distributed fashion and incrementally as nodes join and leave the network, the higher construction cost hides. Figure 7-b shows how the query time of the range queries with selectivity $\mathbf{s} = 0.1\%$ scales as PDN grows in size. The dimensionality of the data space is fixed to $d = 10$. The random graph component of SWAM-V satisfies Property 3, and similar to BASIC, enables SWAM-V to maintain the logarithmic query processing time as the network size scales. On the other hand, since CAN only consists of a grid-like component, the traversal of the index slows sublinearly with increase in the size of the network.

Finally, we calculated the locality measures CC_G and NDD for 1000 SWAM-V index structures with size and data dimensionality randomly selected from $\{500, 1000, 5000, 10000\}$ and $\{2, 5, 10, 20\}$, respectively. On average, we found $\text{CC}_G \simeq 0.54$, which is comparable to $\text{CC}_G = 0.75$ for the small-world graph. Also, NDD is a sharp normal distribution with standard deviation $\sigma = 0.02$, which ensures high content locality with the SWAM-V index topology.

5. RELATED WORK

Multi-dimensional access methods for database systems can be categorized to two classes [10]: hierarchical access methods and hash-based access methods. With the advent of the distributed and networked database systems, and most recently PDNs, access methods from each of these classes are extended to support efficient access in distributed environments. In [23], a 2-level hierarchy with a distributed root architecture is proposed to index a PDN. With two levels in the hierarchy, the filtering power of this access method is limited. In [1], prefix search tree is used as an index

structure for PDNs. Also, in [16] a Voronoi-based heuristic is applied to develop a search tree. As we discussed in Section 3.2, hierarchical index structures fail to distribute the query execution load fairly among PDN nodes. On the other hand, in [12] and [4] DHTs are assumed as the fundamental data access mechanism for in-network query processing with PDNs. Also, in [11] a range-caching scheme is developed based on the DHT indexing to support range queries. Distributed hash-based access methods such as LH* [15] and DHTs [18, 21] assume PDN as a distributed storage system, where users are indifferent to the placement of the data within the data network and only insist on data availability. Such a service model is inconsistent with the typical usage of PDNs as content-sharing federation of autonomous nodes that maintain their own content. SWAM-V respects autonomy of the PDN nodes.

6. CONCLUSION AND FUTURE WORK

In this paper, first we defined a formal framework to study the problem of similarity-search in PDNs. Subsequently, we proposed a set of properties to generate efficient index structures (i.e., PDN topologies) for processing similarity queries in PDNs. These properties are realized by a family of access methods, the *SWAM* family. We introduced *SWAM-V*, a member of the SWAM family, which supports exact-match, range, and **k**NN queries for PDNs with multi-attribute objects. Leveraging on the SWAM properties, SWAM-V achieves query time, communication cost, and computation cost logarithmic to the size of the network. We verified these results via both analysis and simulation. Moreover, since unlike DHTs, SWAM-V does not enforce the placement of the objects within the network, it avoids unnecessary content replacement, supports object replication, and adapts to the object distribution. For example, in a typical case SWAM-V improves the communication cost of **k**NN queries at least 300% and the query time of range queries up to 200%. Finally, we demonstrate that by distributed and incremental pre-computation of the accurate Voronoi-based partitioning, SWAM-V enjoys a query cost which is mainly independent of the dimensionality scale-up.

We intend to extend this study by investigating other members of the SWAM family that as compared to SWAM-V enforce less constraining assumptions and support PDN applications with specific restrictions/requirements. With some PDN applications, strict enforcement of the neighbor selection rules to construct the index structure is either impossible or inefficient. For instance, with a sensor network (i.e., another example of PDN) a node n_k beyond the radio range of node $n_{k'}$, can never be a neighbor to $n_{k'}$, irrespective of the similarity of the two keys k and k'. Currently, we are studying *SWAM-P*, an access method with probabilistic index topology and flexible neighbor selection policies that allow PDN nodes to exercise their *autonomy* in selecting their neighbors as they join the PDN. Our initial results show that as the PDN nodes exercise more autonomy, the efficiency of SWAM-P gracefully degrades from that of SWAM-V to the efficiency of the basic access method [3]. Also, with SWAM-V we assume queries are much more frequent than updates. However, with some PDN applications, datasets and/or node-sets are extremely dynamic such that the overhead of maintaining the index structure exceeds the benefit of using the access method. With such applications, efficient but simple scan-based access methods may outperform the access methods with complex index structures. In [2], we present initial results of our effort to develop such an access method.

Acknowledgments

We would like to thank Jigesh Vora, Gerran Ueyama, and Ranjit Raveendran for their assistance in conducting the experiments. This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC), IIS-0082826 (ITR) and IIS-0238560 (CAREER) and unrestricted cash gift from Microsoft. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

[1] K. Aberer, P. Cudr-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-grid: A self-organizing structured p2p system. *SIGMOD Record*, 32(2), 2003.

[2] F. Banaei-Kashani and C. Shahabi. Efficient flooding in power-law networks. In *Proceedings of Twenty-Second ACM Symposium on Principles of Distributed Computing (PODC'03)*, July 2003.

[3] F. Banaei-Kashani and C. Shahabi. Searchable querical data networks. In *Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing in conjunction with VLDB'03*, September 2003.

[4] M. Bawa, G. Manku, and P. Raghavan. SETS: Search enhanced by topic-segmentation. In *Proceedings of the 26th Annual International Conference on Research and Development in Informaion Retrieval (SIGIR'03)*, August 2003.

[5] B. Bollobás. *Random Graphs*. Academic Press, New York, 1985.

[6] S. Brin. Near neighbor search in large metric spaces. In *Proceedings of the 21th International Conferenceon Very Large Data Bases (VLDB'95)*, September 1995.

[7] E. Chavez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroquin. Searching in metric spaces. *ACM Computing Surveys*, 33(3), 2001.

[8] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proceedings of ACM International Conference on Management of Data (SIGMOD'01)*, November 2001.

[9] Freenet. Freenet project, 2004. http://freenet.sourceforge.net/.

[10] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2), 1997.

[11] A. Gupta, D. Agrawal, and A. El Abbadi. Approximate range selection queries in peer-to-peer systems. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research*, January 2003.

[12] R. Huebsch, N. Lanham, B. Loo, J. Hellerstein, S. Shenker, and I. Stoica. Querying the inernet with PIER. In *Proceedings of 29th International Conference on Very Large Data Bases (VLDB'03)*, September 2003.

[13] KaZaA. Sharman networks, 2004. http://www.kazaa.com/.

[14] J. Kleinberg. The small-world phenomenon: an algorithmic perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, May 2000.

[15] W. Litwin, M. Neimat, and D. Schneider. LH*: A scalable, distributed data structure. *ACM Transactions on Database Systems*, 21(4), 1996.

[16] G. Navarro. Searching in metric spaces by spatial approximation. *The Very Large Databases Journal (VLDBJ)*, 11(1), 2002.

[17] A. Okabe, B. Boots, K. Sugihara, and S. Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley, 2nd edition, 2000.

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM '01*, August 2001.

[19] R. Seidel. *Exact upper bounds for the number of faces in d-dimensional Voronoi diagrams, DIMACS Series*, volume 4. American Mathematical Society, 1991.

[20] SETI@home, 2004. http://setiathome.ssl.berkeley.edu//.

[21] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM '01*, August 2001.

[22] D. Watts and S. Strogatz. Collective dynamics of small world networks. *Nature*, (393):440–442, 1998.

[23] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proceedings of the 19th International Conference on Data Engineering (ICDE'03)*, March 2003.