

Network-Embedded Databases

Han Kiliççöte, Yaron Rachlin, Cem Ünsal, Ergin Güney, and Pradeep K. Khosla

Institute for Complex Engineered Systems

Carnegie Mellon University

Pittsburgh, PA 15213 - USA

{kiliccote | yrachlin | unsal | guney | khosla}@cmu.edu

Abstract. In this paper, we investigate the problem of embedding a large dynamic database in a large network. We assume that the network already exists and its topology cannot be changed. We also assume that the size of each data item in the database is fairly constant and each node in the network has equal storage capability. We propose an architecture that imposes an interconnection network on an existing network. Since this method relies on the topology of an existing network, the superimposed topology is called a *virtual interconnection network*. Such interconnection networks have potential applications ranging from replacing centralized domain name servers with distributed databases to creating the communication backbones of very large global ad-hoc networks. Because such interconnection networks can provide high fault tolerance, they can also be used to provide the infrastructure of distributed databases such as distributed public key infrastructures and fully distributed file systems. We call such databases *network-embedded databases*. In this paper, we describe network-embedded databases and the generalization we propose to de Bruijn networks that allows the database to scale gracefully.

Keywords: Interconnection networks, de Bruijn, fault tolerance, routing, large databases

1. Introduction

In this paper, we investigate an approach that allows users of a large network to share a large database. Even through the proposed solution is feasible for small networks ($< 10^3$ nodes) it's best if the number of nodes is very large ($10^6 - 10^{15}$ nodes). With small networks existing solutions are simpler to implement. We also assume that the database itself is either large or dynamic. Otherwise classical solutions such as replicating the information at each node could prove simpler to implement. We also assume that the number of data items is larger than the number of nodes in the network. We also assume that the size of each data item in the database is fairly constant and each node in the network has equal storage capability.

To discuss the ideas in this paper we set out the following definitions. D denotes a set called the data space. A data item, $d \in D$, is an atomic piece of information. We define a database, DB , as a triplet (K, D_s, q) where $K \subseteq D$ is set of keys, $D_s \subseteq D$ is a set of data items, $q:K \rightarrow D_s$ is a function that maps a key to a data item. We define a distributed database, DDB , as a triplet (V, DB, h_K) where V is a set of nodes, DB is a database and $h_K:K \rightarrow P_s(V)$ is a function that maps a key to a set of nodes, where

$P_s(V)$ is the power set of V , i.e., $P_s(V) = \{x \mid x \subseteq V\}$. In a distributed database, a node, $v \in V$, stores a copy of the key - data item pair, (k, d) , if $v \in h_K(k)$. Other data structures, such as distributed btrees, can be easily implemented by representing the instances of the data structure, such as the nodes of a btree, as data items.

Several commonly used approaches allow users to access information distributed in a network. In one approach, each machine maintains a local copy of the shared information. In a fully replicated database, $h_K = \{(k, V) \mid k \in K\}$, i.e., for all $k \in K$, $h_K(k) = V$. Even though this method has excellent performance for some applications, for practical reasons it can only be used with smaller networks or fairly static information. In another approach, information is stored and replicated at certain selected machines in the network called servers. In selected servers model, $h_K = \{(k, V_s) \mid k \in K\}$ where $V_s \subseteq V$ is the set of the selected servers. These machines are deemed more important than others, creating an imbalance in the load on the network and central points of failure. Bringing these servers down destroys the database and even the network. For example, current solution adopted by the Internet is to dynamically distribute the information to locally managed servers and maintain a limited number of root domain name servers to store all the domain names on the Internet and the IP address of the associated domain name servers. If all the root domain name servers were attacked simultaneously, the whole Internet would go down. In yet another approach, data items can be embedded in a graph so that the information in each vertex and its adjacent vertices can be combined to

recreate the data item [14; 15]. In such approaches, updating a single data item requires a message to all vertices, which limit the applicability to mostly static data. Distributed databases can be also embedded as trees in the network [12]. This structure suffers from root bottlenecks and single points of failure at the higher levels of the tree. Approaches that try to rely on distributing data and indexing data item locations [11] rely on centralized means to index the data, once again introducing bottlenecks and central points of failure.

We are mostly interested in defining an efficient and decentralized h_K that does not contain any single point of failure. This issue is name translation from the key to a set of locations that collectively contains the information. Our decentralized approach combines direct translation of names to virtual locations with name translation for virtual locations to physical locations. This can be achieved by layering a virtual network on top of an underlying network infrastructure, such as the IP-based Internet. Such virtual networks have been successfully used in other domains, such as with PPP, the Mbone, and the emerging Abone. In each case, the virtual network uses an existing network to emulate a new network in which the links in the new network are paths (i.e., sequences of links) in the existing network.

Informally our idea can be explained using social networks as follows: Every person in the world is assigned a set of people called neighbors. The assignment of neighbors does not depend on whether the people are physically close or related. For example, the neighbor of a student, John, at CMU can be Pierre at Lyon. Every person is responsible of

maintaining the status of their neighbors, such as where they are, how to reach them, whether they are available etc. For example this may correspond to storing their telephone numbers and brief notes about their status. The number of neighbors that each person maintains is much smaller than the number of people in the world. The neighbor assignment is not random but done through a deterministic function such that any person can determine whether any two other people are neighbors or not. Furthermore if two people are not neighbors, any node can use this function to determine if there is a common neighbor between them without any communication. For example, *without asking any of his neighbors*, John can find out whether Suzy (John has no idea where Suzy is) is a neighbor of any of his neighbors or whether there is a neighbor of one of his neighbors who must be a neighbor of Suzy. Let's say John found that Pierre must have a neighbor (John doesn't know his or her name or telephone number, but he knows that he or she must exist) who is a neighbor of Suzy. John can call Pierre (John has Pierre's telephone number) and ask him to forward a message to Suzy. Pierre does not know who Suzy is but can apply the same procedure to find that Ali is a neighbor of Suzy. Pierre can call Ali (Pierre has Ali's phone number; John didn't know Ali or his phone number) and ask him to forward the message to Suzy. Ali knows Suzy and can call her directly. This way with very few messages ($\approx 3-4$) anyone can send a message to anyone else in the world without using any broadcast messages or a central database to find the telephone number of that person. This idea is similar to the logical hypercube (independently discovered) method presented at '99 SRDS. [19] Our

network embedded database idea goes one step further and assigns information to people through another deterministic function. For example, Suzy can be assigned to store information about SRDS2000. John without knowing who Suzy (or Ali) is, can find out that his neighbor, Pierre, must know a neighbor whose neighbor stores information about SRDS2000 *without any communication* with Pierre or anyone. Using a similar forwarding mechanism, John can access the information about SRDS2000. Also to allow the network to dynamically grow and guarantee location independence of the information we propose to use a generalization of the de Bruijn networks.

In the following section, we present some of the graph theoretic definitions we need to describe the method and its properties. In the third section, we introduce virtual interconnection networks. In the fourth section, we describe how virtual interconnection networks can be used to implement a distributed database without single point of failures. In the last section, we discuss our preliminary results.

2. Graph-Theoretic Definitions

To discuss the virtual interconnection approach several graph theoretic definitions must be introduced.

A *directed graph* $\mathbf{G} = (V, A, \mathbf{j})$ is a finite nonempty set V of vertices and a set A of arcs, and a function $\mathbf{j} : A \rightarrow V \times V$. The *origin* of the arc a is v_1 if and only if $\mathbf{j}(a) = (v_1, v_2)$ and the *terminus* of the arc a is v_2 if and only if $\mathbf{j}(a) = (v_1, v_2)$. The *order* of a graph, $|\mathbf{G}|$, is the number of vertices in \mathbf{G} i.e., $|\mathbf{G}| = |V|$.

A path $P(v_a, v_b)$ in a graph \mathbf{G} is a sequence of arcs, $[a_1, a_2, \dots, a_n]$, such that $\text{origin}(a_1) = v_a$, $\text{terminus}(a_n) = v_b$ and for every pair of consecutive arcs the terminus and the origin coincides, i.e., for $1 \leq i \leq n-1$, $\text{terminus}(a_i) = \text{origin}(a_{i+1})$. The order of P is n . A path may contain the same arc more than once or pass from the same vertex more than once. The *distance* between two vertices, $d(v_1, v_2)$ is the order of the shortest path between v_1 and v_2 . A graph \mathbf{G} is *connected* if there is a path between any two vertices in the graph. The *diameter* of a connected graph, $D(\mathbf{G})$, is the smallest integer such that for all vertices, v_i, v_j in \mathbf{G} , $d(v_i, v_j) \leq D(\mathbf{G})$.

A *connectivity function* of a graph $\mathbf{G} = (V, A, \mathbf{j})$, is a function $F_C: V \times V \rightarrow \{0, 1\}$ such that for any two vertices, $v_i, v_j \in V$, $v_i \neq v_j$, $F_C(v_i, v_j) = 1$ if and only if $(v_i, v_j) \in A$. For a given graph, there may be more than one way to define the connectivity functions. An obvious connectivity function is through a lookup table that lists all the elements in A . Interesting connectivity functions are *analytical* in the sense that the definition of the function does not refer to a knowledge of the elements of A , but can evaluate connectivity based on the origin and terminus. Such functions are frequently used in defining the algebraic graphs used in multiprocessor computers. Examples of such graphs are Cayley graphs [7] including hypercube [5] and [8] and de Bruijn networks [13].

A *routing function* $F_R: V \times V \rightarrow P$ is a function that defines a path in a graph \mathbf{G} for any two vertices.

An *interconnection network*, I_N , is a triplet (\mathbf{G}, F_C, F_R) where \mathbf{G} is a graph, F_C is a connectivity function of \mathbf{G}

and F_R is the routing function that defines a path in \mathbf{G} for any two vertices. Most well investigated F_C and F_R require a restrictive vertex identification scheme. The order of the graph must be known in advance and the vertices must be assigned an ID composed from a preselected alphabet. However, given an arbitrary graph, it is usually not possible to convert it to an interconnection network because the functions F_R and F_C of an interconnection network, are not defined based on the arcs of the graph but rather the other way around, i.e., when designing an interconnection network, F_R and F_C are picked first, then the arcs between the vertices are created based on the value of F_C for the specific vertices. Unfortunately when imposing an interconnection network on an existing network, it is not possible to add new connections (i.e., arcs) between the vertices. To be able to solve this problem we will use a new structure called an arc-path mapping.

An *arc-path mapping*, M , is a quadruple $M = (\mathbf{G}_F, \mathbf{G}_T, P, \mathbf{f})$ where $\mathbf{G}_F = (V, A_F, \mathbf{j}_F)$ and $\mathbf{G}_T = (V, A_T, \mathbf{j}_T)$ are two graphs sharing the same set of vertices, P is a set of paths in \mathbf{G}_T , and \mathbf{f} is a function $\mathbf{f}: A_F \rightarrow P$ that maps an arc in \mathbf{G}_F to a path in \mathbf{G}_T . \mathbf{G}_F is a graph with the same vertices as \mathbf{G}_T that can be constructed from \mathbf{G}_T by connecting some pairs of vertices in \mathbf{G}_F with an arc if a path exists between them in \mathbf{G}_T .

An arc-path mapping allows someone to create a new graph using an existing graph \mathbf{G}_T . For example as shown in Figure 1, a graph with 4 vertices and 5 arcs can be used to create another graph with 3 arcs where each arc maps to a path in the first graph.

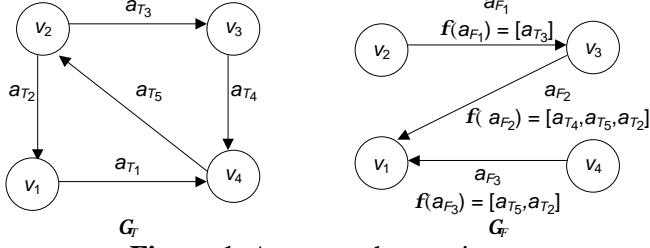


Figure 1. An arc-path mapping

An interesting property of arc-path mappings is that for *any* connected graph and connectivity function, it is possible to create an arc-path mapping satisfying the connectivity function.

Theorem 1. For any connected graph $\mathbf{G}_T = (V, A_T, \mathbf{j}_T)$ and any connectivity function $F_C: V \times V \rightarrow \{0,1\}$, there exists at least one set of paths P_N and a function \mathbf{f}_N such that $M_N = (\mathbf{G}_{F_N}, \mathbf{G}_T, P_N, \mathbf{f}_N)$ is an arc-path mapping where $\mathbf{G}_{F_N} = (V, A_{F_N}, \mathbf{j}_{F_N})$ and for every pair of vertices $v_i, v_j \in V$, $(v_i, v_j) \in A_F$ if and only if $F_C(v_i, v_j) = 1$.

Proof. To prove this, the desired arc-path mapping M_N will be constructed from an arbitrary graph \mathbf{G}_T and arbitrary connectivity function F_C . Let $\mathbf{G}_{F_0} = (V, \emptyset, \emptyset)$. $M_0 = (\mathbf{G}_{F_0}, \mathbf{G}_T, \emptyset, \emptyset)$ is an arc-path mapping. For any arc-path mapping, $M_n = (\mathbf{G}_{F_n}, \mathbf{G}_T, P_n, \mathbf{f}_n)$ where $\mathbf{G}_{F_n} = (V, A_{F_n}, \mathbf{j}_{F_n})$ is a graph and $\mathbf{G}_T = (V, A_T, \mathbf{j}_T)$ is a connected graph, for any $v_i, v_j \in V$ it is possible to create a new arc-path mapping $M_{n+1} = (\mathbf{G}_{F_{n+1}}, \mathbf{G}_T, P_{n+1}, \mathbf{f}_{n+1})$ that incorporates a new arc a_{n+1} , where $terminus(a_{n+1}) = v_j$ and $origin(a_{n+1}) = v_i$, into M_n . Let $A_{F_{n+1}} = A_{F_n} \cup \{a_{n+1}\}$. Let $\mathbf{j}_{F_{n+1}} = \mathbf{j}_{F_n} \cup (a_{n+1}, (v_i, v_j))$. Let $P_{n+1} = P_n \cup \{p_{n+1}\}$ where p_{n+1} is a path between v_i and v_j and let $\mathbf{f}_{n+1} = \mathbf{f}_n \cup (a_{n+1}, p_{n+1})$. The existence of such a path is guaranteed because of the connectedness of \mathbf{G}_T . Let

$M_{n+1} = (\mathbf{G}_{F_{n+1}}, \mathbf{G}_T, P_{n+1}, \mathbf{f}_{n+1})$ where $\mathbf{G}_{F_{n+1}} = (V, A_{F_{n+1}}, \mathbf{j}_{F_{n+1}})$. This process of incorporating a new arc into an arc-path mapping is repeated N times, where N is bounded by $|V|^2$, to generate from M_0 an arc-path mapping $M_N = (\mathbf{G}_{F_N}, \mathbf{G}_T, P_N, \mathbf{f}_N)$ where for every pair of vertices $v_i, v_j \in V$, $(v_i, v_j) \in A_F$ if and only if $F_C(v_i, v_j) = 1$. δ

3. Virtual Interconnection Networks

A *virtual interconnection network*, VI_N , is a triplet (M_P, I_N, h_P) where $M_P = (\mathbf{G}_F, \mathbf{G}_T, P, \mathbf{f})$ is an arc-path mapping in which $\mathbf{G}_F = (V, A_F, \mathbf{j}_F)$, $I_N = (\mathbf{G}_N, F_C, F_R)$ is an interconnection network in which $\mathbf{G}_N = (V_N, A_N, \mathbf{j}_N)$ is a graph, and $h_P: V \rightarrow V_N$ is a mapping between V and V_N . For all $v_i, v_j \in V$, $v_i \neq v_j$, there exists an arc $a \in A_F$ such that $\mathbf{j}_F(a) = (v_i, v_j)$ if and only if $F_C(h_P(v_i), h_P(v_j)) = 1$ or $h_P(v_i) = h_P(v_j)$.

Theorem 2. For any connected graph \mathbf{G}_T and any interconnection network I_N there exists at least one virtual interconnection network.

Proof. Let $N = |V_N|$, $M = |\mathbf{G}_T|$. Let g be an arbitrary function that maps $\{1, 2, \dots, M\}$ into $\{1, 2, \dots, N\}$. Define a mapping function $h_P = \{(v_i, v_{g(i)}) \mid v_i \in V\}$. Define a new connectivity function $F_{C_I} = \{((v_i, v_j), F_C(h_P(v_i), h_P(v_j))) \mid (v_i, v_j) \in V \times V\}$.

An example of a virtual interconnection network is shown in Figure 2. To simplify the figure, the example uses directed graphs.

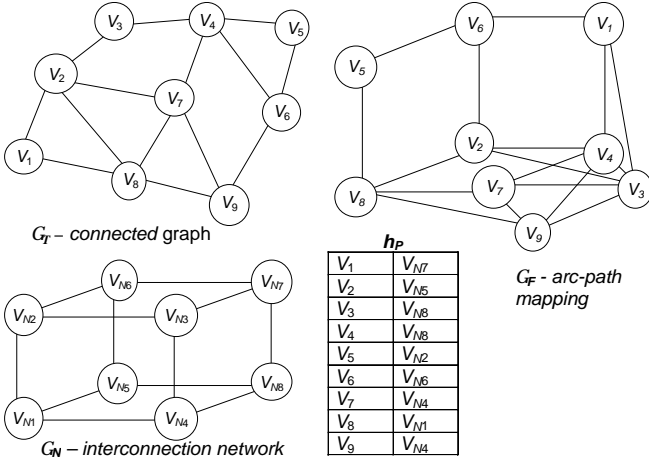


Figure 2. A virtual interconnection network

The number of vertices in the graph of the interconnection network does not need to match the number of vertices in the existing graph. As shown in Figure 2, the number of vertices in the existing graph is 9 and the number of vertices in the interconnection network is 8. Also the mapping function h_P does not need to be a surjection or injection, i.e., two vertices in the arc-path mapping can correspond to the same vertex in the interconnection network and there may be vertices in the interconnection network that do not have corresponding vertices in the arc-path mapping.

Theorem 2 proves that there is a virtual interconnection network for every connected graph and interconnection network. The mapping between the vertices of the network and the interconnection network, h_P , is defined as a function. However, finding an effective form of h_P is not trivial. The connectivity and routing functions associated with an interconnection network require a restrictive vertex identification scheme to route without relying on knowledge of all vertices (e.g., Cayley graphs). Interconnection networks use an invertible mapping between a number i , $i \in Z_N$ where $Z_N = \{0, 1, \dots, N-1\}$

and the vertex identification scheme of an interconnection network with N vertices. Let $F_K^{-1}: Z_{|\mathbf{G}_N|} \rightarrow V_N$ be such an identification function. Using this property, the developers of such networks usually number the vertices (e.g., the processors) sequentially from 0 ... $N-1$. However, in the domains in which we are interested, the identification scheme of the vertices cannot be selected or changed. The identification scheme used in such domains may be difficult to map invertibly to the identification scheme of an interconnection network. An overview of the schemes to find such functions is given in [3]. All such schemes require the vertices to be known in advance, an unreasonable restriction for dynamic networks.

The approach we take, similar to a hash table, permits multiple vertices to map to the same sequence number, and uses a special method to resolve any ambiguity after the vertices have been transformed to a number between 0 and $|\mathbf{G}_N| - 1$ which corresponds to a vertex on the virtual interconnection network.

A *hash-based virtual interconnection network* is a virtual interconnection network with $h_P = F_K^{-1} \circ h_V$ where $h_V: V \rightarrow Z_{|\mathbf{G}_N|}$ is a hash function that maps a vertex in the existing network to a number less than $|\mathbf{G}_N|$ and $F_K^{-1}: Z_{|\mathbf{G}_N|} \rightarrow V_N$ is the identification function of the I_N . Since h_V is usually not invertible, h_P , in a hash based virtual interconnection network is not invertible. To form a hash-based virtual interconnection network one needs a connected graph and an interconnection network. Next h_V is selected from which h_P is derived. Applying Theorem 2 with this h_P yields a hash-based virtual interconnection

network. The following definitions are useful for the discussion of virtual interconnection networks:

v_i is a *neighbor* of v_j if $F_C(h_P(v_i), h_P(v_j)) = 1$. In directed graphs the neighborhood concept is not necessarily symmetric.

v_i and v_j are *siblings* if $h_P(v_i) = h_P(v_j)$. The sibling concept is an equivalence relation, i.e., symmetric, reflexive, and transitive. It is also easy to prove that if v_i is a neighbor of v_j , and v_j and v_k are siblings, then v_i is a neighbor of v_k .

v_E is referred to as an *empty vertex* if for any $v_E \in V_N$ there exists no $v \in V$ such that $h_P(v) = v_E$.

Given a virtual interconnection network, $VI_N = (((V, A_F, \mathbf{j}_F), \mathbf{G}_T, P, \mathbf{f}), ((V_N, A_N, \mathbf{j}_N), F_C, F_R), h_P)$ a message T from $v_1 \in V$ to $v_2 \in V$ can be sent by utilizing F_R . Such a task can be done in two phases: In the first phase a route is calculated using the interconnection and in the second the route is translated to a route in the existing network through which the message is sent. An alternative approach would be a combination of these two phases where the actual message is sent during the route discovery. Both approaches would use the F_R of the interconnection network to create a route in the virtual network and define the actual path using the arc-path mapping. There is a vast literature on high-performance fault-tolerant routing in different types of interconnection networks. See [16; 17] for recent surveys on the topic and [18] for an introductory text. Due to its simplicity, generality, and performance in the experiments we tested, but definitely not for its performance when a large number of faults are present, a protocol for the second approach is

```

Send( $v_1, v_2, T$ ) // Send the message  $T$  from  $n_1$  to  $n_2$ 
//  $L$  is a set containing the vertices already traversed
 $L = \emptyset$ 
Route( $v_1, v_2, T, L$ )

Route( $v_1, v_2, T, L$ )
 $v_{N_1} = h_P(v_1)$ 

// do not attempt routes longer than a maximum length
If  $|L| > \text{max\_route\_length}$ 
Return with failure

Add  $v_{N_1}$  to  $L$  // Add this vertex to  $L$  to avoid cycles

 $v_{N_2} = h_P(v_2)$ 

If  $v_2$  is a neighbor or sibling
Send  $T$  to  $v_2$  in the actual network
Return with success

// make a list of the distance from each neighbor
// to the destination
For each virtual neighbor  $v_i$  of  $v_1$ 
 $v_{N_i} = h_P(v_i)$ 
If  $v_{N_i} \in L$  then skip to next  $v_i$  // avoid cycles
 $P = F_R(v_{N_i}, v_{N_2})$ 
//  $|P| \approx$  the diameter of interconnection network
Store  $\{v_i, |P|\}$  in a list sorted by ascending  $|P|$ 

// start routing attempts with the neighbors closest
// to the destination:
For each  $\{v_i, p\}$  in the sorted list
result = Route( $v_i, v_2, L, T$ )
If result indicates a successful routing
Return with success

// if none of the neighbors can reach  $v_2$ , return failure
Return with failure

```

Figure 3. Message sending in a VIN

presented in Figure 3. In case a very large number of faults are present, the more complex routing functions presented in [16; 17; 18] should be used.

A major problem of using F_R to create a route in the existing network is that someone can never guarantee that there will be v_j such that $h_P(v_j) = v_{N_j}$. Since optimal hashing functions behave as pseudorandom number generators someone cannot even guarantee

the connectivity of the network. However it is possible to make probabilistic guarantees.

To analyze the behavior of this method, we will consider an approximate model of the virtual interconnection network. In this model, we assume that each vertex, $v \in V$ is mapped to a completely random vertex through h_p so that each of the $\binom{M}{N}$ possible configurations of N occupied interconnection network vertices and $M - N$ empty interconnection network vertices is equally probable where $M = |V|$ and $N = |\mathbf{G}_N| = |V_N|$. This assumption can be satisfied in practice by selecting a good hashing function.

In this model, the number of vertices that are assigned to a vertex in the interconnection network follows a binomial distribution.

Lemma 1. In the virtual interconnection network model with random hashing, the probability that a vertex in the graph of the interconnection network does not have a vertex from the existing graph associated with it is less than $e^{-M/N}$.

Proof. See Appendix 1.

When $M = N > 10$, $0.35 < prob_0 < e^{-1} < 0.37$. For most practical purposes, $prob_0$ can be conservatively assumed to be 0.37.

Lemma 2. In an interconnection network, it is expected that every vertex in the interconnection network will have a vertex associated with it if $M \geq N \log_e(2N)$.

Proof. See Appendix 2.

To provide a probabilistic guarantee, it is possible to reduce the size of the interconnection network (i.e., decrease N to increase M/N ratio). Doing so would increase the probability that at least one vertex will be mapped to every vertex in the virtual interconnection network. However most interconnection networks are fault tolerant by their nature and can withstand the failure of a large number of random vertices while still guaranteeing connectedness with very high probability. For example, a ten thousand vertex, diameter two interconnection network, such as the one in [4], that experiences the random failure of 40% of its vertices will become disconnected with a probability of less than 10^{-40} . Using this property of interconnection networks M can be selected to populate network vertices. For example, one could select $M = 2N$. When $M = 2N$, one can expect 14% ($e^{-2} < 0.14$) of the vertices of the interconnection network to be empty at the start and can afford the failure of more than 50% of the vertices populating the vertices in the interconnection network because with 50% failure $M=N$ and only 37% of the vertices should be empty, thus expect the interconnection network to be connected with a probability higher than $1 - 10^{-40}$.

4. Network-Embedded Databases

As described in the previous section, our solution imposes an interconnection network that exists on top of an underlying network infrastructure. The connection between the nodes of the network is achieved by using the underlying network. The physical connectivity requirement between the nodes is replaced with point-to-point communication requirement between the nodes, i.e., as long as a node

can send a message to another node either directly through a physical connection or indirectly through a set of physical connections passing from multiple nodes, the nodes are considered connected.

Since the underlying network handles the connectivity and message passing, the major advantage of such a virtual network is in the structure of the network. The structure of an interconnection network allows the individual nodes to store connectivity information about only a small subset of the nodes in the network. However, the combined actions of these nodes guarantee that the global state of the network is maintained reliably.

Virtual interconnection networks allow users to access network resources and store information in the network in a non-machine-specific manner. Such an approach provides significant advantages in terms of availability of data. Much in the way that vertices are mapped through hashing functions h_p to the vertices in an interconnection network, data can be mapped to the vertices of an interconnection network through the multiple hash functions.

A *network-embedded database* (NED) is a triplet, (VI_N, DB, h_K) , where $VI_N = (((V, A_F, \mathbf{j}_F), \mathbf{G}_T, P, \mathbf{f}), ((V_N, A_N, \mathbf{j}_N), F_C, F_R), h_P)$ is a virtual interconnection network and DB is a database and $h_K: K \rightarrow P_s(V_N)$ is a function that maps a key to a set of vertices in the interconnection network. In a network-embedded database, a vertex $v \in V$ stores a copy of the key-data item pair, (k, d) , if $h_P(v) \in h_K(k)$. In a network-embedded database $h_K = \{(k, \{(F_K^{-1} \circ h_{k_1})(k)\} \cup \dots \cup \{(F_K^{-1} \circ h_{k_n})(k)\}) \mid k \in K\}$ where $h_{k_i}: K \rightarrow Z_{G_N}$, is a set

of hash functions and F_K^{-1} is the identification function of the I_N .

To add a new $(key, data)$ pair to a *NED* through v_1 one can send $(key, data)$ pairs to all the vertices v_i such that $h_P(v_i) \in h_K(key)$ and store the $(key, data)$ pair on each. This can be easily done by modifying the Route function presented in Figure 3, such that it accepts vertices in the interconnection network as arguments and the destination vertex can be found using $h_K(key)$. However, similar to any distributed system, the integrity of the information must be protected through a concurrency protocol. While the ability to store multiple copies of the data in multiple locations is similar to many distributed systems, the independence of each storage location from the other points of storage leads to a balanced load in the network.

In a virtual interconnection network classical replication could be augmented with an information dispersal [10] or secret sharing [9; 2] scheme. An information dispersal scheme or secret sharing scheme divides the information into a certain number of chunks, called *shares*, of which only a subset, m , of them is required to recover the information. Such an approach can increase reliability and security of the information while reducing the amount of replication with minimal overhead. The overhead of multiple messages to recover a single data item is counterbalanced by lower storage requirements and better security for the same availability. There can be at least two different approaches to distributing shares in a network-embedded database. In the first approach each sibling can be given a different share and in the second approach can be given the same

share. Because the first approach clusters the information in the same sets of siblings, the security of this approach is less than the second.

Since classical replication is an information dispersal scheme where $m = 1$, we will analyze the properties of a network-embedded database that use the IDA [10] protocol that requires the collection of m shares to recover a data item.

Let $M = |V|$, the number of vertices in the actual network. Let N be the number of vertices in the interconnection network. Let us assume that M_s vertices are unavailable due to attacks or node failures. We assume that the underlying network is still connected and can be used to route messages between any two nodes after M_s node failures and additional link failures. Let us assume that the hash functions h_{k1}, \dots, h_{kn} are random but dependent so that $(\{F_K^{-1} \circ h_{k1}\}(k)) \cap \dots \cap \{(F_K^{-1} \circ h_{kn}\}(k)) = \emptyset$, i.e., the hash functions always map to random but distinct vertices in the interconnection network.

Lemma 3. The expected amount of replication in a network-embedded database is $(nM)/(Nm)$ where n is the number of hash functions.

Proof. See Appendix 3.

Lemma 4. The probability of losing a single data item in a network-embedded database that creates a new share for each sibling and that contains w data items when M_s nodes are unavailable is less than

$$w \sum_{i=0 \dots m-1} \text{binom}(i; M - M_s, n/N).$$

Proof: See Appendix 4.

With suitable parameters, this probability can be very low. For example, if the database contains $w = 10^{12}$

pieces of information distributed among $M = 2 \times 10^9$ computers through with $n = 40$ hash functions in a diameter $D = 3$ virtual interconnection network with a degree of $\delta = 1000$ nodes, then the probability of intruders destroying a coherent piece of information is less than 10^{-13} if they gain access to or destroy $M_s = 10^9$ nodes (half of the network nodes) where the minimum number of shares required to recover a piece of information, m , is 8. The expected amount of storage overhead due to replication in this database is a factor of 10, which drops to 5 when 10^9 nodes are destroyed.

Lemma 5. The probability of losing a single data item in a network-embedded database that replicates the shares of the siblings and that contains w data items when M_s nodes are unavailable is less than

$$w \sum_{i=0 \dots m-1} \text{binom}(i; n, 1 - e^{-(M - M_s)/N}).$$

Proof. See Appendix 5.

For the same example given above, this probability can be calculated to be less than 10^{-11} .

Both lemmas assume that the network is still connected after the failure of M_s nodes and additional link failures. However if the underlying network is hierarchical or mostly hierarchical like the Internet, then it is possible to bring the network and thus the database down with much less effort. What these numbers demonstrate is that the structure we propose is fault tolerant and robust, and as long as the underlying network can withstand the attacks and is not the failure mode, an embedded database will prevail and will not be the failure mode.

5. Growing the network

In the models described in sections 3 and 4, the location of the nodes and the data were mapped to vertices of a graph of an interconnection network using hash functions. When new nodes are added to the network, the number of siblings at a given vertex increases. However, increasing the number of siblings increases the amount of replication and the connectivity that must be maintained by the neighbors. When the number of nodes in the network reaches a critical value, the interconnection network must be resized to reduce the amount of replication and the number of neighbors that a node must maintain connectivity. In an efficient network, resizing the network should not cause any message exchange in the network (either to create new connections or to move data items stored in some nodes to other nodes). We achieve this by modifying the definitions of the hash functions and by using a generalized de Bruijn graph. Most interconnection networks including the original de Bruijn graph, its extensions and generalizations, do not provide this property. Network expansion methods investigated in the literature propose methods that can be used to add new nodes without requiring major updates in the existing connections. However our requirements are more strict: such expansions should also not cause any movement of the data items stored in the database to other nodes. Even if a small percentage of the data items in a large database needs to be moved in the VIN, the database may need to be locked for very long periods.

De Bruijn graphs and their generalizations have been extensively studied in the literature. See [20] and [21]

for surveys. Here we define a new type of generalization. We define a generalized de Bruijn graph, $B(d, k, t)$, with three parameters. For given integers $d \geq 2$ and $k \geq t \geq 1$, we represent the vertices of the $B(d, k, t)$ by strings of length k . The vertex $(a_1, \dots, a_{k-t}, a_{k-t}, \dots, a_k)$ is adjacent to the vertices $(a_{t+1}, \dots, a_k, a_{k+1}, \dots, a_{k+t})$ where $a_i \in Z_d$ is called a digit. Such a directed graph has d^k vertices with diameter $\lceil k/t \rceil$ and in- and out-degree d^t .

This new generalization and defining the hash functions as follows allows us to scale the network gracefully. $h_p = F_K^{-1} \circ h_{V_1} \circ h_{V_2}$ where $h_{V_2}: V \rightarrow Z_M$ and $h_{V_1}: \{(a, a \bmod d^k) \mid a \in Z_M\}$ where M is a number larger than the maximum number of nodes that can be added to the network (e.g., a 128-bit value), i.e., the vertex id is first hashed to a large value and then reduced to the order of the graph. Similarly h_{k_i} is also defined using two functions where the key is first hashed to a large value and then reduced to the order of the graph using a modulo operation. These definitions allow us to change the parameters of the de Bruijn network without any message exchange in the network.

Method 1: $B(d, k, t) \rightarrow B(d, k + 1, t)$. This is the most important expansion for a large network, which reduces the number of siblings and neighbors, thus decreases the amount of replication and the number of neighbors that each node must maintain connectivity. Through this expansion, another digit is added to the end of the string representation of each vertex. This reduces the expected number of siblings at each node by a factor of d . Thus the expected number of neighbors is reduced by a factor of d . By

lemma 3, the amount of replication is also reduced by a factor of d . The diameter either remains the same or increases by one. The neighbors of a node after the expansion is a subset of the neighbors it had before the increase, i.e., after such an expansion the nodes just “forget” some of their old neighbors and there is no need to establish connectivity with new neighbors. Similarly the data items stored in a node after the expansion is a subset of the data items it had stored before the increase, i.e., after such an expansion the nodes just “forget” some of the data items they used to store and there is no need to move the data items to other nodes.

Method 2: $B(d, k, t) \rightarrow B(d, k + 1, t + 1)$. Another digit is added to the string representation of the vertices. To guarantee no data item movement, this expansion requires $t \geq k/2$. The new digit, pulled from the unused portion of h_{v_1} , is added between the $\lfloor k/2 \rfloor^{\text{th}}$ and $(\lfloor k/2 \rfloor + 1)^{\text{th}}$ digits. The diameter of the network is kept the same. The expected number of siblings at each node is reduced by a factor of d . However the expected number of neighbors is kept the same. The amount of replication is reduced by a factor of d . This is a very useful change when the network is still small. This update does not require any data item movement.

Method 3: $B(d, k, t) \rightarrow B(2d, k, t)$. The number of bits of the digit alphabet is increased by one. The number of siblings is decreased by a factor of 2^k . The number of connections that a node maintains and the amount of replication is decreased by a factor of $2^{k/t}$. F_k^{-1} is changed to pull the new bit of each digit from

the unused portion of h_{v_1} . This allows expansion without any data movement.

Other types of updates that do not require data movement are also possible (e.g., $B(d, k, t) \rightarrow B(d, 2k, 2t)$; no restriction between k and t). However such updates are less useful due to the very low granularity of the updates.

6. Results and discussions

The research presented in this paper has also been analyzed using a software simulation of the protocols using a generalized de Bruijn graph. We only simulated a binary de Bruijn graph ($d = 2$) because it provides better granularity.

Figures 4 and 5 are both taken from simulations that were started with $k = 1$ and $t = 1$, and both k and t were incremented by 1 each time the ratio of the number of nodes to the number of vertices (r) in the VIN reached a set threshold as new nodes were added. In Figure 4, the three plots selected as representatives of the tests we performed show the average route length for three different values of r . Figure 5 represents a single case where r is 3. The network contains sequentially generated node names to also take the performance of the hash functions into account. In all cases, t was not incremented any further after it reached 10, and only k kept on increasing after that point. Average route lengths in both figures were computed by statistical sampling. We limited t to 10, because at $t = 10$ the in- and out-degree of a vertex is 1024, which means each node will have to maintain connectivity with approximately $t \cdot r$ neighboring nodes. This represents an amount of storage and communications overhead

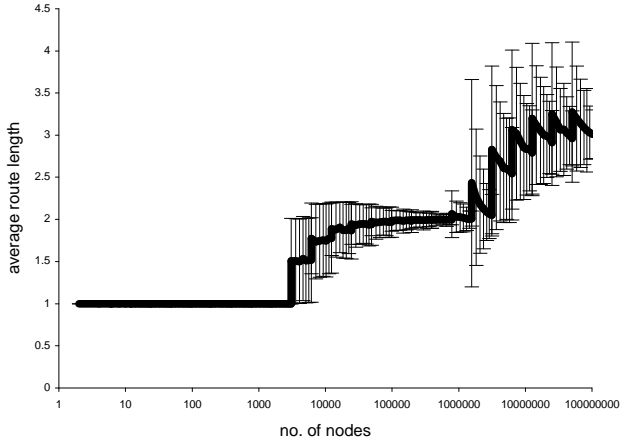


Figure 5: Average route length and standard deviation

that is assumed to be acceptable for a typical node on a network like the Internet.

As shown in Figure 4, when r is high, the average route length is approximately equal to the diameter of the interconnection network, which 2 for up to 2^{22} nodes. For low values of r , due to the large number of empty vertices, the average route length can be an order of magnitude more than the diameter of the interconnection network. However, using a high r will result in a larger number of neighbors (e.g., an average of 3000 vs. 200 neighbors for the 4 vs. 0.25 ratios).

In Figure 5, the plot shown is that of a nodes-to-vertices ratio of 3 as the threshold for incrementing k . The error bars represent a range of ± 1 standard deviation for each data point. The standard deviation declines up to the point where the average route length exceeds 2. This is due to the fact that, since the routing is done to a node that is known to exist (i.e., to a vertex that is not empty) and since the only hop between the origin and the destination is a node that is a direct neighbor of the originating node (and therefore its existence can be checked by the originating node), there is no possibility for

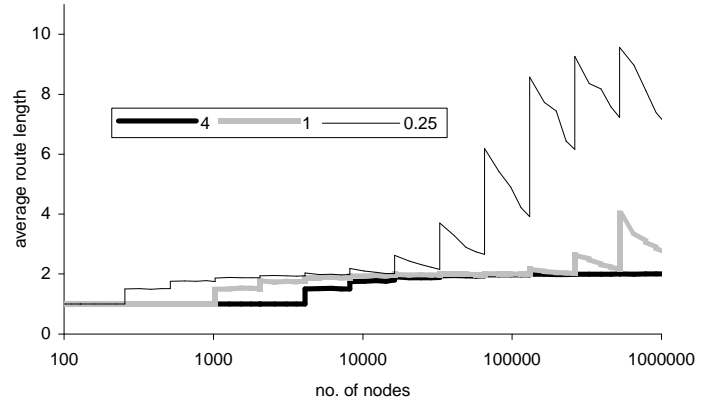


Figure 4: Average route length in VIN

inefficient or unsuccessful routing. Starting with the third hop, however, these uncertainties are introduced into the routing behavior and the standard deviation is maintained at a moderate amount.

The preliminary simulations indicated that the virtual interconnection network (VIN) and the network-embedded database (NED) presented here have excellent fault-tolerance properties if the underlying network has similar fault-tolerance. A NED that uses a generalized de Bruijn graph, $B(2, k, t)$ can accommodate 2^t nodes. For example a NED with a degree of approximately 1000 ($t = 10$) and a diameter of two ($k = 20$) can accommodate approximately one million nodes. A similar network with approximately 10^{15} vertices could have an average diameter of just five “virtual” hops when populated with approximately 10^{16} nodes.

A diameter-0 VIN, where every node is a sibling of every other node, provides a fully replicated route translation, which is equivalent to the old Internet model where every computer had the IP address of every other computer in the network. A NED based on such a VIN replicates every data item at every node.

A diameter-1 VIN, where every node is either a neighbor or a sibling of every other node, also provides a fully replicated route translation. A NED based on such an interconnection network partitions the database into Δ nodes. The performance of such a NED is roughly equivalent to that of the client-server architecture (CSA) with better robustness. The switch from a diameter-0 VIN to diameter-1 VIN can occur using the third method. Then the network should continue to grow through alternating applications of the first and second methods.

A diameter-2 VIN, where every node is at most two “virtual” hops away, has roughly the same performance as the name resolution strategy used in current Internet (we assume there is no caching in both models) with better robustness. The message complexity of a NED based on such a VIN is at least twice that of a CSA. The switch from a diameter-1 VIN to diameter-2 VIN can be done using the first method. Then the network should continue to grow through applications of the first method while keeping the number of neighbors a constant.

A diameter- t VIN, where $t > 2$, requires more messages than a CSA to route or to read a data item. However a diameter-3 VIN can accommodate more than 10^9 nodes if $\Delta=1000$. With suitable replication parameters, the number of nodes that must be attacked or offline to bring down the entire network or the database is more than 10^8 if the underlying network can withstand such an attack. On the other hand, a 1000-server CSA will be down if all 1000 servers were attacked. Also, even though such a NED requires more messages than a CSA, the load on the NED is evenly distributed. A diameter- t VIN ($t > 2$)

can continue to grow through applications of the first or third methods.

Thus, among the characteristics of a NED are graceful scalability, robustness, lack of bottlenecks and significant fault tolerance against natural faults and malicious attacks.

References

- [1] Akers, S. and B. Krishnamurthy. On group graphs and their fault tolerance. *IEEE Transactions on Computers*. 36(7): 885-888, 1987.
- [2] Blackley, G.R. Safeguarding Cryptographic Keys. In *Proceedings of the National Computer Conference*. 48: 313-7, 1979.
- [3] Czech, Z. J, G. Havas, B. S. Majewski. Perfect Hashing. *Theoretical Computer Science*. 182: 1-143, 1997.
- [4] Faber, V, W. J. Moore, and W. Y. C. Chen, Cycle Prefix Digraphs for Symmetric Interconnection Networks. *Networks*. 23: 641-9, 1993.
- [5] Fragopoulou, P. and S.G. Akl. Optimal communication primitives on the generalized hypercube network. *Journal of Parallel Distributed Computing*. 32:173-87, 1996.
- [6] Hsu, D. F. Introduction to a special issue of interconnection networks. *Networks*. 23: 211-3, 1993.
- [7] Lakshminaraha, S. J. Jwo and S.K. Dhall. Symmetry in interconnection networks based on Cayley graphs of permutation groups: a survey. *Parallel Computing*. 19: 361-407, 1993.
- [8] Sen, A. Supercube: An Optimally Fault Tolerant Network Architecture. *Acta Informatica*. 26: 741-8, 1989.

[9] Shamir A. How to share a secret. Communications of the ACM. 24: 612-3, 1979.

[10] Rabin, M.O. Efficient dispersal of information for security, load balancing, and fault tolerance. Journal of the ACM, 36: 335-48, 1993.

[11] Lomet, D. Replicated indexes for distributed data. In Proceedings of 4th International Conference on Parallel and Distributed Information Systems, 108-19, 1996 .

[12] Johnson, T., Colbrook, A. A Distributed, Replicated, Data-balanced Search Structure. International Journal of High Speed Computing, 6(4): 475-500, 1994.

[13] Pradhan, D.K., and S. M. Reddy. A fault-tolerant communication architecture for distributed systems. IEEE Transactions on Computers. 31: 863-70.

[14] Naor, M and R. M. Roth. Optimal file sharing in distributed networks. SIAM Journal on Computing, 24(1): 158-83. 1995.

[15] Kant, G and J. van Leeuwen. The file distribution problem for processor networks. In SWAT 90. 2nd Scandinavian Workshop on Algorithm Theory Proceedings, 48-59, 1990.

[16] Grammatikakis, M.D., Hsu, D.F.; Kraetzl, M. and Sibeyn, J.F. Packet routing in fixed-connection networks: a survey. Journal of Parallel and Distributed Computing, 54(2), 77-132, 1998.

[17] Agrawal, N.; Ravikumar, C.P. Adaptive routing techniques for high reliability in multiprocessor interconnection networks. IETE Technical Review, 12(3): 191-203. 1995.

[18] Duato, J., Yalamanchili S., and Ni L. Interconnection Networks: An Engineering

Approach. IEEE Computer Society, Los Alamitos, CA, 1997.

[19] Friedman, R. and S. Manor. Scalable Stability Detection Using Logical Hypercube. Proceedings of IEEE SRDS, 1999, 124-133.

[20] Ergincan, F. O. and R. W. Dawes. De Bruijn Bus Networks and Their Generalization, Technical Report from the Department of Computing and Information Science, Queen's University at Kingston, Ontario. QUC 91-310.

[21] Leighton, F. T. Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes. Morgan Kaufmann Publishers, 1992.

Appendix 1. Proof of Lemma 1

Proof. The probability that an interconnection vertex has k vertices map to it through h_p is

$$prob_k = binomial(k, M, 1/N) = \frac{\binom{M}{k} (N-1)^{M-k}}{N^M}.$$

Thus the probability that an interconnection vertex has no vertices map to it through h_p is

$$prob_0 = \frac{\binom{M}{0} (N-1)^M}{N^M} = \left(1 - \frac{1}{N}\right)^M \leq e^{-M/N}. \quad \delta$$

Appendix 2. Proof of Lemma 2

Proof. We can select M such that we expect (probabilistically of course) that each vertex of the interconnection network is full. This value can be calculated by incrementally filling the vertices of the interconnection network. The first vertex has a probability of 1 of mapping to an empty interconnection vertex. The second vertex has a probability of $(N-1)/N$. Thus, the second full vertex of

the interconnection network will require $N/(N-1)$ vertices. To fill the i^{th} vertex, will require $N/(N-i)$ tries. Thus to fill all N vertices, $N \sum_{i=1}^N \frac{1}{i}$ vertices must be mapped. This formula can be approximated using Euler's approximation for Harmonic sums as $N \log_e N + \frac{1}{2} - \frac{1}{12N} + \dots$ where $\gamma \cong 0.577$ is Euler's constant. For every value of $N > 5$, the approximation is bounded above by $N \log_e(2N)$ which implies $N \sum_{i=1}^N \frac{1}{i} \leq N \log_e(2N)$. Furthermore for every N , $N \sum_{i=1}^N \frac{1}{i} \leq \lceil N \log_e(2N) \rceil$ Thus if $M \geq N \log_e(2N)$, it's expected that all the vertices in the interconnection network will have at least one associated vertex. δ

Appendix 3. Proof of Lemma 3

The expected number of siblings in a network-embedded database is M/N . Since each hash function maps to a distinct vertex in the interconnection network, where each sibling receives a copy, the number of vertices in the existing network that receives a copy is nM/N . In IDA [10], the amount of replication is $(nM)/(Nm)$. δ

Appendix 4. Proof of Lemma 4

Proof: Each application of h_K generates a random sampling of n vertices in the interconnection network. The probability of H_p not mapping a vertex in the existing network to any of these follows a hypergeometric distribution, which is bounded by $1-n/N$. Since there are $M - M_s$ vertices available, the

probability of one data item being stored in k vertices in the existing network follows a binomial distribution $prob_k = binomial(k, M-M_s, n/N)$. Since any m shares can be used to recover the data item, the probability of not being able to find m shares follows a cumulative binomial distribution. Since there are w such independent trials, the final probability is $1 - \left(1 - \sum_{i=0 \dots m-1} binom(i; M - M_s, n/N)\right)^w$ which is less than $w \sum_{i=0 \dots m-1} binom(i; M - M_s, n/N)$. δ

Appendix 5. Proof of Lemma 5

Proof. When the shares of siblings are not independent but replicated, the probability of losing a data item is dependent on the probability of a vertex in the interconnection network being empty. This probability can be approximated as $e^{-(M-M_s)/N}$. The probability of losing one data item is approximately equal to and bound by n repeated independent trials (i.e., n hash functions) and not being able to find at least m vertices in the interconnection network. The probability of not being able to find at least m vertices in the interconnection network for w data items is:

$$1 - \left(1 - \sum_{i=0 \dots m-1} binom(i; n, 1 - e^{-(M-M_s)/N})\right)^w$$

which is always less than $w \sum_{i=0 \dots m-1} binom(i; n, 1 - e^{-(M-M_s)/N})$. δ