Proceedings of the 5th WSEAS Int. Conf. on APPLIED INFORMATICS and COMMUNICATIONS, Malta, September 15-17, 2005 (pp110-119)

EST-Grid: An Efficient Scalable Peer-to-Peer Infrastructure for Web Service Discovery

S. SIOUTAS
Computer Engineering and Informatics department
University of Patras
Building B, University Campus, 26500, Rion, Patras
GREECE

L. DROSSOS
Technological Institute of Messolongi,
Department of Applied Informatics in Administration and Economics
Technological Institute Campus, 30200, Messolongi
GREECE

D.TSOLIS
Computer Engineering and Informatics department
University of Patras
Building B, University Campus, 26500, Rion, Patras
GREECE

T. S. PAPATHEODOROU
Computer Engineering and Informatics department
University of Patras
Building B, University Campus, 26500, Rion, Patras
GREECE

Abstract - Web services are becoming an important enabler of the Semantic Web. In this paper, we present a new P2P infrastructure for Web Services discovery. Peers that store Web Services information, such as data item descriptions, are efficiently located using a scalable and robust data indexing structure for Peer-to-Peer data networks, EST-GRID (**E**xponential **S**earch **T**ree). EST-GRID provides support for processing Exact match Queries of the form "given a key, map the key onto a node". EST-GRID adapts efficiently update queries as nodes join and leave the system, and can answer queries even if the system is continuously changing. Results from theoretical analysis show that the communication cost of the query and update operations scaling both in $O(\sqrt{\log n})$ time where n the number of nodes.

Key – Words: -P2P Networks, Indexing, Web Services, Data Structures, Grid Infrastructures

1. Introduction

Recently, P2P architectures that are based on Distributed Hash Tables (DHTs) have been proposed and have since become very popular, influencing research in Peer-to-Peer (P2P) systems significantly. DHT – based systems provide efficient processing of the routing/location operations that, given a query for a document id, they locate (route the query to) the peer node that stores this document. Thus, they provide support for exact-match queries. DHT-based systems are referred as structured P2P systems because in general they rely on lookups of a distributed hash table, which creates a structure in the system emerging by the way that peers define their neighbors. Related P2P systems like Gnutella [1], MojoNation [2], etc, do not create such a structure,

since neighbors of peers are defined in rather ad hoc ways.

There are several P2P DHTs architectures like Chord [3], CAN [4], Pastry [5], Tapestry [6], etc. From these, CAN and Chord are the most commonly used supporting more elaborate queries.

There are also other than DHTs structured P2P systems, which build distributed, scalable indexing structures to route search requests, such as P-Grid. P-Grid ([7]) is a scalable access structure based on a virtual distributed search tree. It uses randomized techniques to create and maintain the structure in order to provide complete decentralization.

In this work we present a new efficient grid structure for Peer-to-Peer (P2P) Data Networks, named EST-GRID. EST-GRID provides support for processing Exact match Queries of the form "given a key, map the key onto a node". EST-GRID uses a virtual Exponential Search Tree to guide key based searches. Data location can be easily implemented on top of EST by associating a key with each data item, and storing the key/data item pair at the node to which the key maps. We suppose that each node stores an ordered set of keys and the mapping algorithm runs in such way that locally ordered key_sets are also disjoint each other. EST-Grid adapts efficiently update queries as nodes join and leave the system, and can answer queries even if the system is continuously changing. Results from theoretical analysis show that the communication cost of the query and update operations scaling double-logarithmically with the number of EST-GRID nodes. Furthermore, our system is also robust on failures.

The rest of this paper is structured as follows. Section 2 remind us the fundamentals of hierarchical protocols giving examples, section 3 presents the EST-GRID, our new efficient and scalable P2P lookup system. In this section we also describe and resolve the communication cost of search and join/leave operations. Section 4 presents the results from theoretical analysis. Finally, we outline items for future work and summarize our contributions in section 5.

2. Preliminaries

This section reminds us the hierarchical and tree based algorithms that are useful in peer-to-peer contexts.

2.1 Hierarchical protocols

Hierarchical protocols is nothing new, but provides an interesting approach to the balance between scalability and performance. The most well known service in use today that uses a hierarchical protocol is DNS. The purpose of DNS is to translate a human friendly domain name, such as www.ietf.org, to its corresponding IP address (in this case 4.17.168.6). The DNS architecture consists of the following:

- Root name servers
- Other name servers
- Clients

The other name servers can also be classified as *authorative* name servers for some domains. The early Internet forced all hosts to maintain a copy of a file named hosts.txt, which contained all necessary translations. As the network grew the size and frequent changes of the file became unfeasible. The introduction of DNS remedied this problem and has worked successfully since then.

2.2 An example of a DNS lookup

Assume a host is located in the domain sourceforge.net. The following scenario shows what a DNS lookup could look like in practice.

1. If a user on the aforementioned host, in the sourceforge.net domain, directs his web browser to http://www.ietf.org the web browser issues a DNS lookup for the name www.ietf.org.
2. The request is sent to the *local name server* of the sourceforge.net domain.
3. The name server at sourceforge.net is not able to answer the question directly, but it knows the addresses of the *root name servers* and contacts one of them.
4. There are 12 root name servers (9 in the US, 1 in the UK, 1 in Sweden and 1 in Japan). The root name server knows the address of a name server for the org domain. This address is sent in response to the question from the local name server at sourceforge.net.
5. The name server at sourceforge.net asks the name server of the org domain, but it does not have the answer either, but the name server of the org domain knows the name and address of the *authorative name server* for the ietf.org domain.
6. The name server at sourceforge.net contacts the name server at ietf.org and once again asks for the address of www.ietf.org. This time an answer is found and the IP address 4.17.168.6 is returned.
7. The web browser can continue its work by opening a connection to the correct host.

Note that a question sent to a name server can be either *recursive* or *iterative*. A recursive question causes the name server to continue asking other name servers until it receives an answer, which could be that the name does not exist. An iterative query returns an answer to the host asking the question immediately. If a definite answer cannot be given, suggestions on which servers to ask instead are given.

2.3 Caching in DNS

Caching plays an important part in DNS. In the example above the local name server will cache the addresses obtained for the name server of the org domain and the ietf.org domain as well as the final answer, the address of www.ietf.org. This causes subsequent translations of www.ietf.org to be answered directly by the local name server, and translations of other hosts in the domain ietf.org can bypass the root name server and the org server. The translation of an address such as www.gnu.org bypasses the root name server and asks the name server for the org domain directly.

2.4 Redundancy and fault tolerance in DNS

To make DNS fault tolerant any name server can hold a set of entries as the answer to a single question. A name server can answer a question such as ``What is the address of www.gnu.org'' with something like Table 1, which provides the names of name servers for the gnu.org domain. The results were obtained using the ***dig*** utility available on most Unix systems. In reality the response is much more compact.

;; ANSWER SECTION:				
gnu.org.	86385	IN	NS	**nic.cent.net.**
gnu.org.	86385	IN	NS	**ns1.gnu.org.**
gnu.org.	86385	IN	NS	**ns2.gnu.org.**
gnu.org.	86385	IN	NS	**ns2.cent.net.**
gnu.org.	86385	IN	NS	ns3.gnu.org.
;; ADDITIONAL SECTION:				
nic.cent.net.	79574	IN	A	**140.186.1.4**
ns1.gnu.org.	86373	IN	A	**199.232.76.162**
ns2.gnu.org.	86385	IN	A	**195.68.21.199**
ns2.cent.net.	79574	IN	A	**140.186.1.14**

Table 1: Sample response from a DNS query

The example shows that the gnu.org domain appears to have five name servers (NS), of which four of their addresses are known to us. The question of the address of www.gnu.org can be sent to anyone of the four servers. This means that we can receive an answer to our question as long as at least one of the name servers is reachable.

3. The EST-GRID architecture

The EST-GRID provides an Exponential Search Tree-like structure where key based searching can be performed. In terms of bandwidth usage searching scales very well since no broadcasting or other bandwidth consuming activities takes place during searches. Since all searches are key based there are two possibilities:

- Let each host implement the same translation algorithm, that translates a sequence of keywords to a binary key.
- Let another service provide the binary key. This service accepts keyword based queries and can respond with the corresponding key.

The second approach is more precise. It is also possible to use a more centralized implementation for such a service. From now on we assume that the key is available. The paper describes an algorithm for the first case. We also suppose that the set of keys on each host retain a **global order**. Details are described on next paragraph.

3.1 Preliminary Structures

3.1.1 The Structure of B-Trees [14]

Unlike a binary-tree, each node of a b-tree may have a variable number of keys and children. The keys are stored in non-decreasing order. Each key has an associated child that is the root of a subtree containing all nodes with keys less than or equal to the key but greater than the preceeding key. A node also has an additional rightmost child that is the root for a subtree containing all keys greater than any keys in the node.

A b-tree has a minimum number of allowable children for each node known as the minimization factor. If t is this minimization factor, every node must have at least t - 1 keys. Under certain circumstances, the root node is allowed to violate this property by having fewer than t - 1 keys. Every node may have at most 2t - 1 keys or, equivalently, 2t children.

Since each node tends to have a large branching factor (a large number of children), it is typically necessary to traverse relatively few nodes before locating the desired key.

3.1.1 Fusion Trees [12]

The Fusion tree is a static data structure which permits O(logN/loglogN) amortised time queries in linear space. This structure is used to implement a B-tree [14] where only the upper levels in the tree contain B-tree nodes, all having the same degree (within a constant factor). At the lower levels, weight balanced trees are used. The amortised cost for searches and updates is O(logN/logd + logd) for any d = O(w1/6). The first term corresponds to the number of B-tree levels and the second to the height of the weighted-balanced trees.

The Fusion tree has the following properties:

For any d, d = O(w1/6), a static data structure containing d keys can be constructed in O(d4) time and space, such that it supports neighbour queries in O(1) worst-case time.

The main advantage of the fusion technique is that we can decide in time O(1) in which subtree to continue the searching by compressing the k-keys of every B-tree node using w - bit words.

3.1.2 Exponential Search Trees [8]

The Exponential Search tree answers queries in one-dimensional space. It is a multi-way tree where the degrees of the nodes decrease exponentially down the tree. Auxiliary information is stored in each node in order to support efficient searching queries. The Exponential Search tree has the following properties:

- Its root has degree $\Theta(N^{1/5})$.
- The keys of the root are stored in a local data structure. During a search, the local data structure is used to determine in which subtree the search is to be continued.
- The subtrees are exponential search trees of size $\Theta(N^{4/5})$.

The local data structure at each node of the tree is a combination of van Emde Boas trees [11] and perfect hashing [13] thus achieving O(logwloglogN) worst case cost for a search.

Anderson, by using an exponential search tree in the place of B-trees [14] in the Fusion tree structure [12], avoids the need for weight-balanced trees at the bottom while at the same time improves the complexity for large word sizes. This structure further improves exact match searching by achieving $O(\sqrt{\log N})$ time using O(N) storage.

3.2 EST-Grid Infrastructure

The EST-Grid is an exponential tree T where the degree of the nodes at level i is defined to be $d(i)=t(i)$ and $t(i)$ indicates the number of nodes present at level i. This is required to hold for $i \geq 1$, while $d(0)=2$ and $t(0)=1$. It is easy to see that we also have $t(i)=t(i-1)d(i-1)$, so putting together the various components, we can solve the recurrence and obtain for $i \geq 1$: $d(i)=2^{2^{i-1}}, t(i)=2^{2^{i-1}}$. One of the merits of this tree is that its height is $O(\log\log n)$, where n is the number of elements stored in it.

3.2.1 Peers in EST-Grid Infrastructure

We distinguish between leaf_peers and node_peers: If peer i, henceforth denoted p_i, is a **key_host_peer** (leaf) of the EST-Grid network it maintains the following:

- A number of ordered k-bit binary keys ki = $b_1...b_k$, where k is less than or equal to n_1, for some bounded constant n_1 which is the same for all p_i. This ordered set of keys denotes key space that the peer is responsible for. Let K the number of k-bit binary keys and n the number of key_host_peers. While we can initially distribute the keys in that way such as each host peer (leaf) stores a load of $\Theta(K/n)$ keys it is not at all obvious how to bound the load of the host peers, during update operations. In [9], an idea of general scientific interest was presented: modeling the insertions/deletions as a combinatorial game of bins and balls, the size of each host peer is expected w.h.p. $\Theta(\ln n)$, for keys that are drawn from an ***unknown*** distribution.
- The key sets $S_j=\{k_i \mid 1 \leq i \leq \Theta(K/n)\}, 1 \leq j \leq n$ retain a global order. That means, $\forall S_j, S_q, 1 \leq j \leq n, 1 \leq q \leq n, j \neq q$, if $\min\{S_j\} < \min\{S_q\}$ then $\max\{S_j) < \min\{S_q\}$. Thereupon, we are sorting the key_sets above providing a leaf oriented data structure as you can see in figure 1.

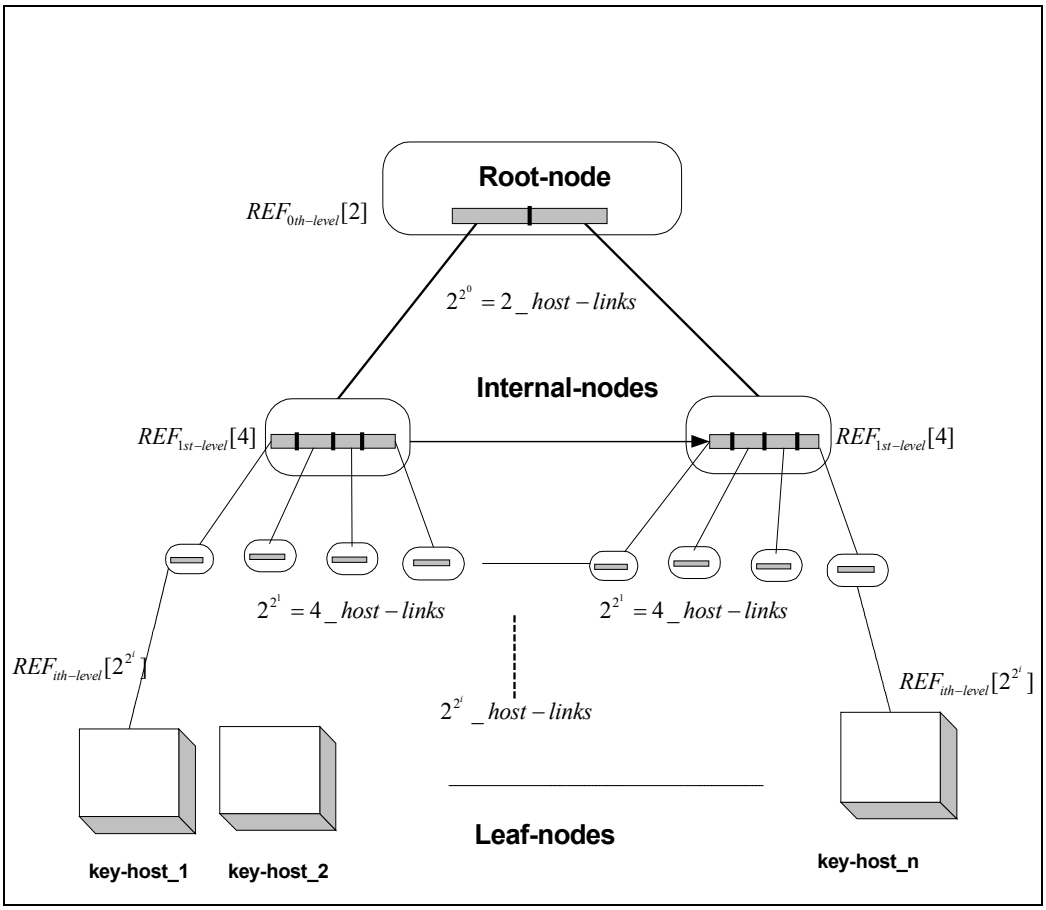

Fig. 1: The EST-Grid Infrastructure

If p_i, is a **node_peer** (root or internal node) of the EST-Grid network is associated with the following:

- A local table of sample elements REF_{P_l}, one for each of its subtrees. The **REF** table is called the *reference table* of the peer and the expression $REF_{P_l}[r]$ denotes the set of addresses at index r in the table. Each REF table is organized as the innovative linear space indexing scheme presented in [8] by Anderson which achieves an $O(\sqrt{\log n})$ worst-case time bound for dynamic updating and searching operations, where n the number of stored elements. We will use this solution as the base searching routine on the local table of each network node.

For each node p_i we explicitly maintain parent, child, and sibling pointers. Pointers to sibling nodes will be alternatively referred to as ***level links***. The required pointer information can be easily incorporated in the construction of the EST-Grid search tree.

3.3 Lookup Complexity

Theorem 1: Suppose a *EST-grid* network Then, Exact Match operations require $O(\sqrt{\log n})$ hops where n denotes the current number of peers.

Proof: Assume that a key_host_peer p performs a search for key k. We first check whether k is to the left or right of p, say k is to the right of p. Then we walk towards the root, say we reached node u. We check whether k is a descendant of u or u's right neighbor on the same level by searching the REF table of u or u's right neighbor respectively. If not, then we proceed to u's father. Otherwise we turn around and search for k in the ordinary way.

Suppose that we turn around at node w of height h. Let v be that son of w that is on the path to the peer p. Then all descendants of v's right neighbor lie between the peer p and the key k. The subtree T_w is an EST-tree for $n' \leq n$ elements, and it's height is $h=\Theta(\log\log n')$.

So, we have to visit the appropriate search path $w, w_1, w_2, \ldots . w_r$ from internal node w to leaf node w_r .In each node of this path we have to search for

the key k using the REF_{w_i} indices, 1≤i≤r and r=O(loglogn), consuming $O(\sqrt{\log d(w_i)})$ worst-case time, where d(w_i) the degree of node w_i. This can be expressed by the following sum: $\sum_{i=1}^{r=O(\log\log d)} \sqrt{\log d(w_i)}$

Let L_1, L_r the levels of W_1 and W_r respectively. So, $d(w_1)=2^{2^{L_1}}$ and $d(w_r)=2^{2^{Lr}}$

But, L_r=O(loglogn). Now, the previous sum can be expressed as follows:

$$\sqrt{\frac{2^{L_1}}{L_1}}+\sqrt{\frac{2^{L_1+1}}{L_1+1}}+.....+\sqrt{\log n}=O(\sqrt{\log n})$$

To perform the search a connection to a peer p in the EST-Grid is established and the call bdtgrid_search(p, k) is performed. The function bdtgrid_search is shown in figure 2.

```
Node p ESTtgrid_search (p, k)
{
int j;
bool move_right=false;

if (p=host_key && p is responsible for this k)
return p;
if (someone else is responsible)
{
 Check whether k is to the left or right of p; \\ say k is to the right of p\\
p_next=father(p)

While (k > REF p_next [right _ most ] & & move _ right = false )
{
p′=right_sibling of p_next;

if (k <= REF p'[right _ most ])
{
p_next= p′;
move_right=true;
}
else
p_next=father (p_next);

host = send_search(p_next, k);
}
While (p_next is not a key_host)
{
j=search (k, p_next);
\\ Where search (key, node) denotes the procedure [8] which returns an integer position j indicating
the appropriate descendant we must continue the further searching\\
p_next =&REF p-next [j];
host = send_search(p_next, k);
}
p=p_next;
return p;
}
```

Figure 2: Pseudo-code for EST-Grid searches

3.4 Key_Host_Peers Join and Leave the System

What will happen when a key_host_peer overflows (or underflows)?? In the first case we have to nearby insert a new host_peer. In the second case

we have to mark as deleted the key_host_peer by moving first the few remaining keys to the left or right neighbors. Obviously after a significant number of join/leave operations a global rebuilding process is required for cleaning the redundant nodes and rebalancing the EST structure.

```
Procedure INSERT_host_peer (p)
{
Insert a new leaf node p;
counter=counter+1;
p_next=father(p);

While (p_next !=root)
{
update REF_p_next ; //add one more link according to algorithm
                    presented in [8] //
p_next=father(p_next);
}
if counter = Θ(n) then Rebuild(T);

}
```

Figure 3: Pseudo-code for INSERT host_peers

```
Procedure DELETE_host_peer p
{
search for p; // according to ESTgrid_search routine //
mark p;
counter=counter+1;
if counter = Θ(n) then Rebuild(T);
}
```

Figure 4: Pseudo-code for DELETE host_peers

```
Procedure Rebuild(T)
{
Build a new EST_Grid structure;
Counter=0;
}
```

Figure 5: Pseudo-code for Rebuilding operation

Theorem 2: Suppose a *EST-grid* network Then, *join* and *leave* operations require $O(\sqrt{\log n})$ amortized number of hops where n denotes the current number of peers.

Proof: A join (insert) operation affects the path from the new leaf node to the root of the EST–GRID. In each path-node w_i ($1 \le i \le c \log\log n$ and c is a constant) we have to update the REF_{w_i} index. This process requires $O(\sqrt{\log d(w_i)})$ time, where $d(w_i)$ the degree of the node w_i. This can be expressed by the following sum: $\sum_{i=1}^{r=O(\log\log d)} \sqrt{\log d(w_i)} = \sqrt{\log n}$.

The leave (delete) operation requires $O(\sqrt{\log n})$ hops for detecting the node and O(1) time to mark as deleted that node.

After $\Theta(n)$ update operations we have to rebuild the Balanced Distributed backbone. By spreading the $\Theta(n)$ rebuilding cost to the next $\Theta(n)$ updates, the theorem's amortized bound follows.

4. Evaluation and Outline of Contributions

As you can see in Table 2 below, our contribution provides for exact-match queries, improved search costs from O(logn) in DHTs to $O(\sqrt{\log n})$ in EST-GRID and adequate and simple solution to the range query problem. Update Queries such as WS registration and de-registration requests are not performed as frequently as a user login and logout in a typical P2P data delivery network. Web Services are software developed to support business structures and procedures which are expected to stay available in the WS discovery registries more than a P2P user session time span. EST -GRID scales very well in the amortized case and it is better than Chord in the expected business oriented weak – sparse updates. EST -GRID does not scale well in worst-case due to a likelihood reconstruction overhead, which is not typically met in WS registry/catalogue implementation cases, though. Additionally, a fault tolerance schema is available to support with fidelity an elementary web services business solution.

P2P Network Architectures	Lookup Messages	Update Messages	Data Overhead-Routing information
CHORD	O(logn)	$O(\log^2 n)$ *with high probability*	O(logn) nodes
BDT-GRID	$O(\sqrt{\log n})$	$O(\sqrt{\log n})$ *Amortized* $\Theta(n)$ *worst-case*	Exponentially increasing

Table 2. Performance Comparison with the best Known Architecture

5. Simulation and Experimental Results

In this section we evaluate the EST protocol by simulation. The simulator generates initially K keys drawn by an unknown distribution. After the initialization procedure the simulator orders the keys and chooses as bucket representatives the 1st key, the lnnst key, the 2lnnst key. …and so on. Obviously it creates N buckets or N Leaf_nodes where N=K/lnn. By modeling the insertions/deletions as the combinatorial game of bins and balls presented in [9], the size of each bucket (host peer) is expected w.h.p. $\Theta(\ln n)$. Finally the simulator uses the lookup algorithm in Figure 2. We compare the performance of EST simulator with the best-known CHORD simulator presented in [10]. More specifically we evaluate the Load balance and the search path length of these two architectures. In order to understand in practice the load balancing and routing performance of these two protocols, we simulated a network with $N=2^k$ nodes, storing $K=100x2^k$ keys in all. We varied parameter k from 3 to 14 and conducted a separate experiment of each value. Each node in an experiment picked a random set of keys to query from the system, and we measured the path length required to resolve each query. For the experiments we considered synthetic data sets. Their generation was based on several distributions like Uniform, Regular, Weibull, Beta and Normal. For anyone of these distributions we evaluated the length path for lookup queries and the maximum load of each leaf node respectively. Then we computed the mean values of the operations above for all the experiment shots. The figures below depict the mean load and path length respectively.

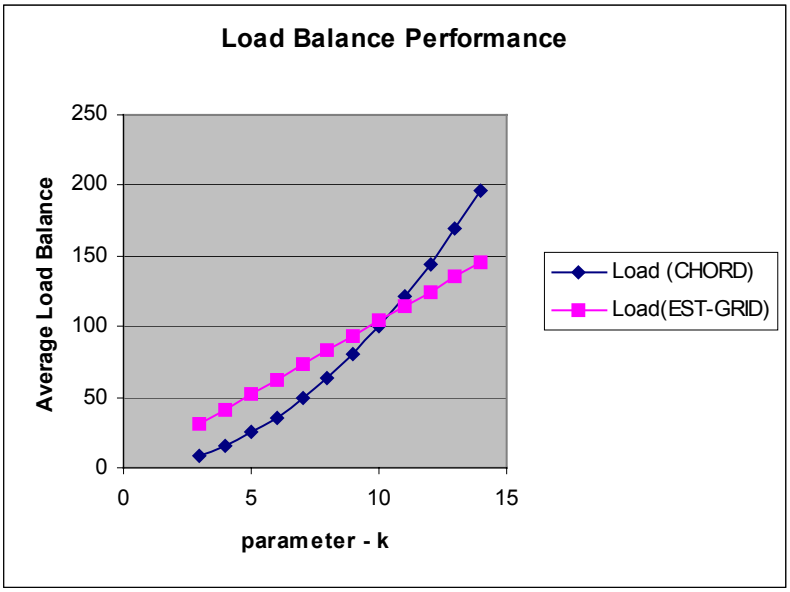

Table 3. Load Balance Performance Comparison with the best Known Architecture

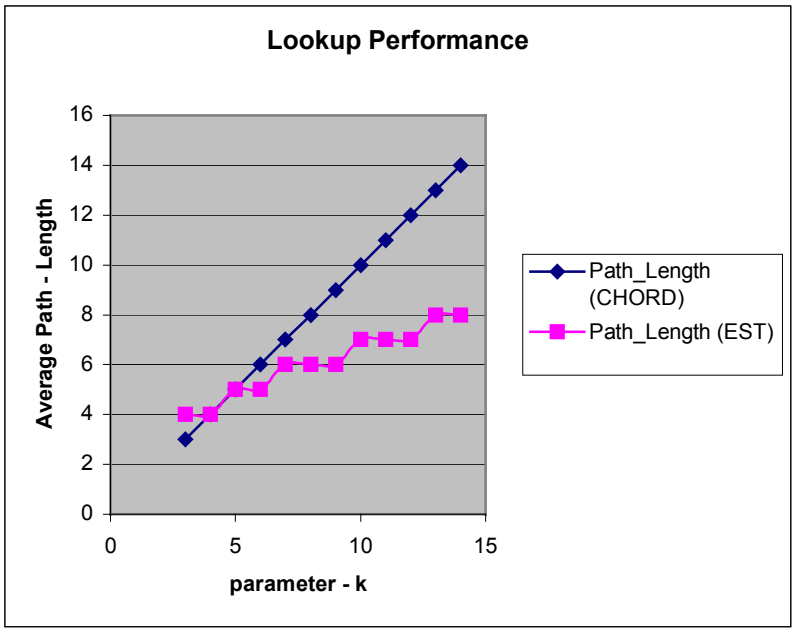

Table 4. Lookup Performance Comparison with the best Known Architecture

From the experimental evaluation derives that the mean value of bucket load is approximately $15 * \ln n$ in EST protocol instead of $k * \log_2 n$ in CHORD protocol. Obviously, for $k > 15$ the EST protocol has better load balancing performance. Considering now the lookup performance depicted in Table 4 the Path-Length in EST protocol is almost constant (between 4 and 8 hops) instead of CHORD where the path-length is increased logarithmically.

6. Conclusion

A Web Service discovery structure over a P2P network needs to determine the node that stores the web service item or the nodes that store a set of WS items, which satisfy a range criterion. In this paper we introduced and analyzed EST-GRID, a protocol that solves this challenging problem in decentralized manner. Current work includes the implementation and experimental evaluation of EST -GRID for large scale WS discovery when the insertion/deletion of WS items draw unknown distributions. Furthermore includes a detailed study of embedding fault-tolerance techniques into EST-GRID system.

Acknowledgements

The authors would like to thank the Operational Program for Educational and Vocational Training II (EPEAEK II) and particularly the Program PYTHAGORAS, for funding the above work.

References

[1] Andersen D.. *Resilient overlay networks.* Master's Thesis, Department of EECS, MIT, May 2001,

[2] Bakker A., Amade E., Ballintijn G., Kuz I., Verkaik P., Van Der Wijk I.. Van Steen M., and Tanenbaum A.. *The Globe Distribution Network. In Proc. 2000 USENIX Annual Conf. (FREENIX track)* (San Diego, CA, June 2000), pp. 141 – 152.

[3] Chen Y., Edler J., Goldberg A., Gottlieb A., Sobti S., and Yianilos P.. *A prototype implementation of archival intermemory. In Proceedings of the 4th ACM Conference on Digital libraries* (Berkeley, CA, Aug. 1999), pp. 28-37.

[4] Clarke I. *A distributed decentralized information storage and retrieval system.* Master's Thesis, University of Edinburgh, 1999.

[5] Clarke I., Sandberg O., Willey B., and Hong T.W. *Freenet: A distributed anonymous information storage and retrieval system. In Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability*

(Berkeley, California, June 2000). http://freenet.sourceforge.net.

[6] Dabek F., Brunskill E., Kaashoek M.F., Karger D., Morris R., Stoica I., and Balakrishnan H.. *Building P2P systems with Chord, a distributed location service. In Proccedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)* (Elmau/Oberbayern, Germany, May 2001), pp. 71-76.

[7] Dabek F., Brunskill E., Kaashoek M.F., Karger D., Morris R., and Stoica I.. *Wide-area cooperative storage with CFS. In Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)* (To appear; Banff, Canada, Oct. 2001).

[8] Anderson, "Faster deterministic sorting and Searching in linear space", TR- LU-Cs-TR:95-160, Department of Computer Science, Lund University, 1995.

[9] Kaporis, Ch. Makris, S. Sioutas, A. Tsakalidis, K. Tsichlas, Ch. Zaroliagis, "*Improved Bounds for Finger Search on a RAM*", LNCS 2832, pp 325-336, 11th Annual European Symposium on Algorithms (ESA 2003) – Budapest, 15-20 September, 2003.

[10] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, "Chord: A Scalable Peer – to – Peer Lookup Service for Internet Applications", ACM-SIGCOMM, 2001.

[11] van Emde Boas, "Preserving Order in a forest in less than logarithmic time and linear space", IPL 6(3), 80-82, 1977.

[12] M. L. Fredman and D. E. Willard, *"Surpassing the information theoretic bound with fusion trees"*, J. Computer Systems Science 47, pp: 424-436, 1994.

[13] Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer Auf Der Heide, Hans Rohnert, and Robert E. Tarjan, *Dynamic Perfect Hashing: Upper and Lower Bounds.*, SIAM J. Comput. Volume 23, Number 4 pp. 738-761

[14] Comer, D., *"The ubiquitous B-tree"*, ACM Computing Surveys, 11(2) 1979.