BDT-Grid: An Efficient Scalable Peer-to-Peer Lookup System for Web Service Discovery

Lampros Drosos, Spyros Sioutas, Evangelos Sakkopoulos, Spiros Sirmakessis

Research Academic Computer Technology Institute
Internet and Multimedia Technologies Research Unit
N. Kazantzaki Str. 26500 Rion, Patras, Greece

Technological Educational Institution of Messolongi
Department of Applied Informatics in Administration and Economics,
GR-30200, Messolongi, Hellas

E-mail: {drosos, sioutas}@teipat.gr {sakkopul, syrma}@cti.gr

ABSTRACT

Web Services constitute an essential factor for the realization of the envisioned Semantic Web. An important direction, apart from the optimization of the description mechanism, is the availability of WS information in an accessible way for machine processing. In this paper, we propose a novel decentralized approach for Web Service discovery based on a new P2P based approach. This novel approach enables efficient 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, BDT-GRID (**B**alanced **D**istributed **T**ree). BDT-GRID provides support for processing (a) Exact match Queries of the form "given a key, map the key onto a node" and (b) Range Queries of the form "map the nodes whose keys belong to a given range". BDT-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 our theoretical analysis point out that the communication cost of the query and update operations scaling in rooted and double-logarithmic time complexity on the number of nodes respectively. Furthermore, our system is also robust on failures.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**], Information Search and Retrieval - *Selection process*

H.3.5 [**Information Storage and Retrieval**], Online Information Services

H.3.4 [**Information Storage and Retrieval**], Systems and Software - *Information networks*

General Terms

Algorithms, Performance, Design, Experimentation.

Keywords

P2P Web Service, Quality of Service, DHT

1. INTRODUCTION

Web Services have evolved to become the framework for next generation business integration. Although discovery of services that match a conceptual and technical description is promoted by the use of WS registries, truly dynamic discovery has not been enabled yet.

Although Web Services technology is categorized among the newest members of the web engineering area, the widespread adoption of XML standards that support functionalities in terms of description (e.g. WSDL), communication (e.g. SOAP [21]), and discovery (e.g. UDDI) has stimulated to action the industry and academia in order to address the WS research issues.

Web Services, which are designed to enable applications to communicate directly and exchange data, regardless of language, platform and location, are currently emerging as a dominant paradigm [14] for constructing and composing distributed business applications and enabling enterprise-wide interoperability. In the years to come, Web Services are expected to become more widely adopted and thus, there will be an increasing demand for automated service discovery.

At present, catalogues based on the Universal Description, Discovery and Integration standard (UDDI) [19] constitute the prevalent technological environment for WS discovery. UDDI adopts a centralized architecture consisting of multiple repositories that synchronize periodically. However, this approach has already been proved to be inefficient, and as the number of Web Services grows and become more dynamic, such a centralized approach quickly becomes impractical. Additionally, since a query may return more than one results that meet the functional requirements but provide different quality of service attributes, the selection procedures become even more complicated in order to satisfy the QoWS requirements [23].

Current approaches to Web Service discovery can generally be classified as centralized or decentralized. Comprehensive reviews have been provided by [12][16]. The centralized approach includes UDDI registries [19][4], whereas the decentralized, distributed approaches are based on Peer- to-Peer infrastructures. In P2P discovery of Web Services network nodes are considered as peers that share information and are able to query other nodes.

Decentralized systems of WS discovery may have either structured or unstructured architecture [20][5]. Structured P2P architectures use hashing and in specific the most of them are based on Distributed Hash Tables (DHTs). These systems provide efficient processing of location operations so that, given a query with an object key, they locate (route the query to) the peer node that stores the object.

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) Web Service Discovery Overlay - Data Networks, named BDT-GRID. BDT-GRID provides support for processing

(a) Exact match Queries of the form “given a key, map the key onto a node” and

(b) Range Queries of the form “map the nodes whose keys belong to a given range”.

BDT-GRID uses a virtual Exponential Search Tree to guide key based searches. Data location can be easily implemented on top of BDT 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. BDT-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 BDT-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 BDT-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 and section 5 discusses experimental results. Finally, we outline items for future work and summarize our contributions in section 6.

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.

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.

The request is sent to the local name server of the sourceforge.net domain.

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.

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.

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.

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.

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 BDT-GRID architecture

The BDT-GRID provides a Balanced Distributed Tree-like structure where key based searching can be performed in order to discover and select a requested Web Service key or URI. 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 BDT-Grid network

The BDT-Grid is a balanced distribution 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 Peers in BDT-Grid

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

- A number of ordered k-bit binary keys ki = b1...bk, where k is less than or equal to n1, for some bounded constant n1 which is the same for all pi. 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.

If pi, is a node_peer (root or internal node) of the BDT-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 and Thorup which achieves an optimal $O(\sqrt{\log n / \log \log n})$ 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 pi 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 BDT-Grid search tree.

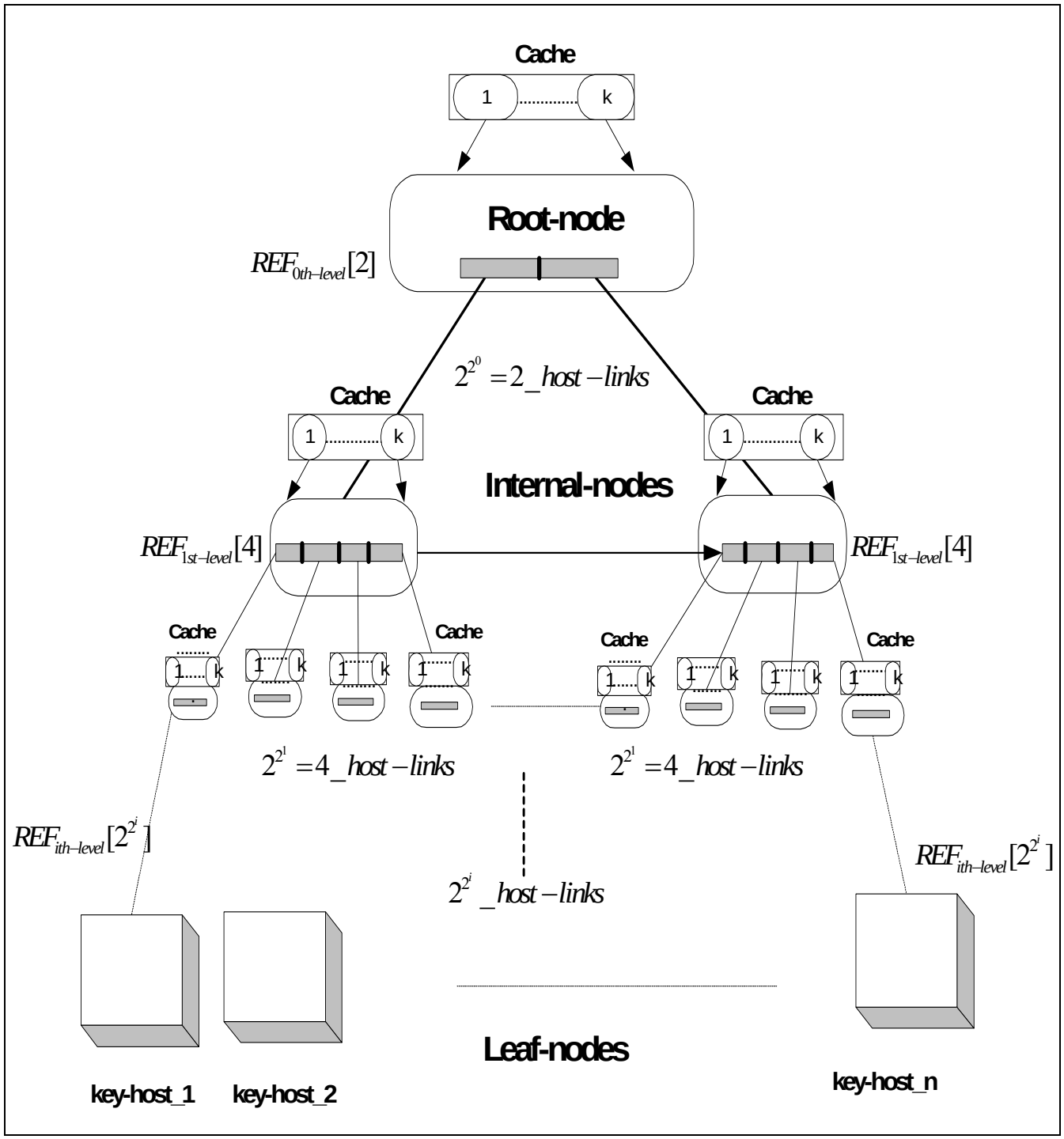

Figure 1:The BDT-Grid peer-to-peer system

3.3 Lookup Complexity

Theorem 1: Suppose a *BDT-grid* network Then, Exact Match operations require $O(\sqrt{\log n / \log\log n})$ 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 BDT-tree for $n' \le n$ elements, and it's height is $h=\Theta(\text{loglog } n')$.

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, $1 \le i \le r$ and $r=O(\text{loglogn})$, consuming $O(\sqrt{\log d(w_i) / \log\log d(w_i)})$ worst-case time, where $d(w_i)$ the degree of node w_i. This can be expressed by the following sum:

$$\sum_{i=1}^{r=O(\log\log d)} \sqrt{\frac{\log d(w_i)}{\log\log d(w_i)}}$$

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, $L_r=O(\text{loglogn})$. Now, the previous sum can be expressed as follows:

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

Remark 1:. Exploiting the order between the key_sets on the leaves it is obvious that Range Queries of the form $[k_\ell, k_r]$ require $O(\sqrt{\log n / \log\log n} + |A|)$ hops, where A the answer set. In such a query, the hosts whose keys belong to the range $[k_\ell, k_r]$ can be found by first searching the BDT-Grid structure for k_ℓ and then perform an in-order traversal in the tree from k_ℓ to k_r.

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

```
Node p *bdtgrid_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 BDT-Grid searches*

3.4 Fault Tolerance

What will happen when few internal nodes have been shutdown?? In data caching applications, it is useful to solve the more general problem of finding a nearby node that actually has a copy of the desired data. In our structure, thus, we equip each node with k redundant nodes each of them stores replicated copies of a data item, where k>1 a small positive constant. We also suppose that each node is k-robust, that means the simultaneous shutdown of all these nodes is impossible, thus at least one is active on the network.

3.5 Key_Host_Peers Join and Leave the System

In the case of key_host_peer overflow we have to nearby insert a new host_peer. In the second case of underflow 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 BDT 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 bdtgrid_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 BDT_Grid structure;
Counter=0;
}
```

Figure 5: *Pseudo-code for Rebuilding operation*

Theorem 2: Suppose a *BDT-grid* network Then, *join* and *leave* operations require $O(\sqrt{\log n / \log\log n})$ 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 BDT –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 $O(\sqrt{\log d(w_i) / \log\log d(w_i)})$ time, where $d(w_i)$ the degree of the node w_i. This can be expressed by the following sum:

$$\sum_{i=1}^{r=O(\log\log d)} \sqrt{\frac{\log d(w_i)}{\log\log d(w_i)}} = \sqrt{\frac{\log n}{\log\log n}}.$$

The leave (delete) operation requires $O(\sqrt{\log n / \log\log n})$ 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

For comparison purposed, an elementary operation's evaluation is presented (table 2).

P2P Network Architectures	*Lookup Messages*	*Update Messages*	*Data Overhead-Routing information*
CHORD	*O(logn)*	*O(log2n) with high probability*	*O(logn) nodes*
BDT-GRID	$O(\sqrt{\log n / \log\log n})$	$O(\sqrt{\log n / \log\log n})$ *Amortized* $\Theta(n)$ *worst-case*	*Exponentially increasing*

Table 2. Performance Comparison with the best Known Architecture

Our contribution provides for exact-match queries, improved search costs from O(logn) in DHTs to $O(\sqrt{\log n / \log\log n})$ in BDT-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.

BDT-GRID scales very well in the amortized case and it is better than Chord in the expected business oriented weak – sparse updates. BDT-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.

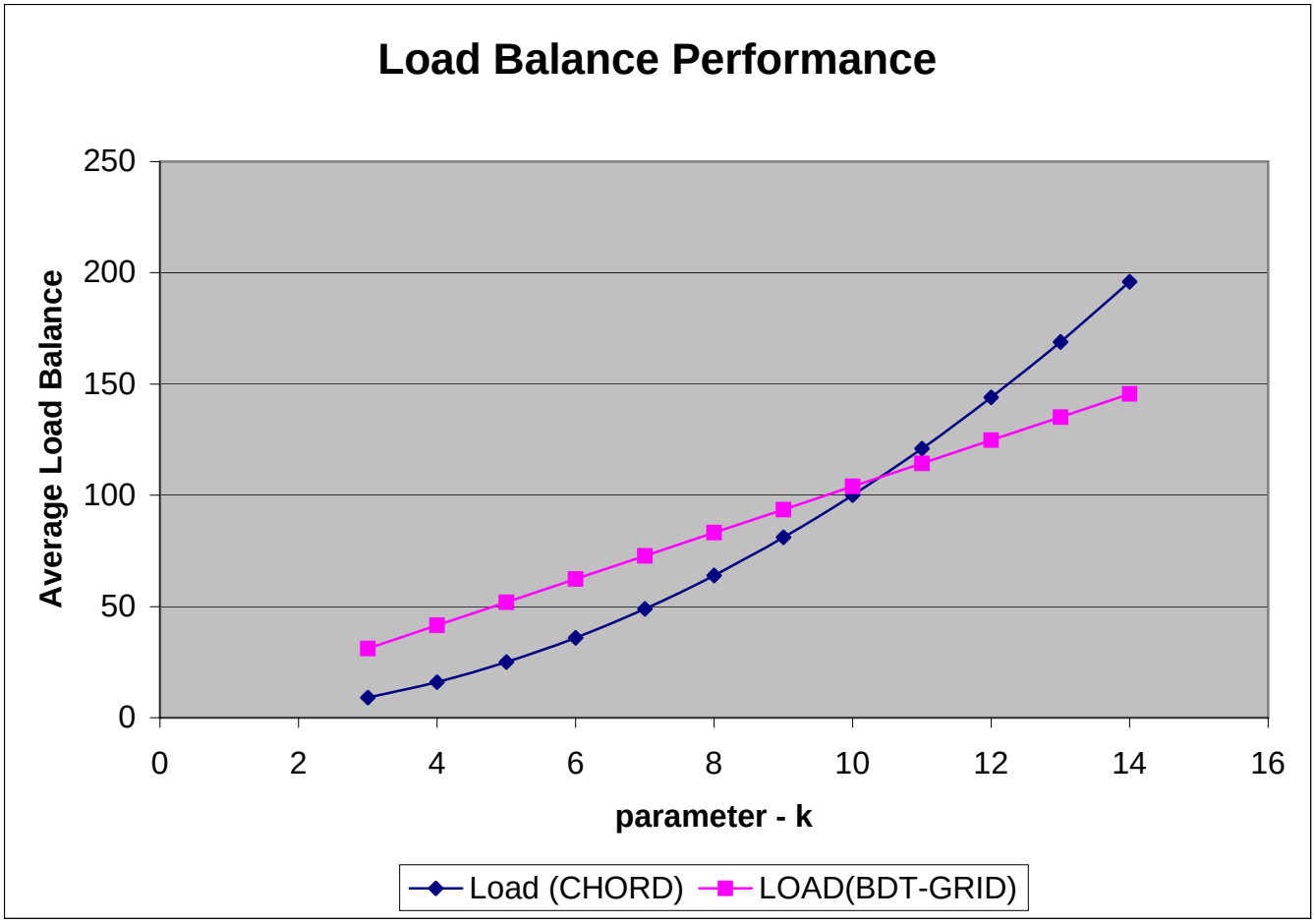

Table 3. Load Balance Performance Comparison with the best Known Architecture

5. Simulation and Experimental Results

In this section we evaluate the BDT 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 BDT 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.

From the experimental evaluation derives that the mean value of bucket load is approximately $15 * \ln n$ in BDT protocol instead of $k * \log_2 n$ in CHORD protocol. Obviously, for $k > 15$ the BDT protocol has better load balancing performance. Considering now the lookup performance, the Path-Length in BDT protocol is almost constant in comparison to CHORD where the path-length is increased logarithmically.

6. Conclusion

The combination of Peer-to-Peer architectures and Web Services technology has an added value result: functional integration problems are exceeded due to the universal information architecture provided by WS, and moreover, direct and fault tolerant access to computational resources is achieved thanks to

the network infrastructure architecture that P2P provide. The decentralization of WS discovery, which is expected to be one of the most favorable outcomes of this combination, will increase fault tolerance and search efficiency.

In the case of a Web Service discovery structure over a P2P network, a web service item (or a set of WS items) that satisfies the range criterion is stored in a node (or nodes respectively). In order to discover the WS that meets the search criteria, this nodes need to be determined. In this paper we introduced and analyzed the protocol BDT-GRID, which tackles this challenging problem in a decentralized manner.

Current work includes the implementation and experimental evaluation of BDT-GRID for large scale WS discovery when the insertion/deletion of WS items draw unknown distributions. Furthermore includes the application of BDT-GRID in other domains such as large-scale distributed computing platforms. Future steps also include research on semantic based P2P solutions [17][18] in order to support semantically enriched Web Services and respective ontologies [11][22]

7. REFERENCES

[1] A. Anderson and M. Thorup. Tight(er) Worst – Case Bounds on Dynamic Searching and Priority Queues, ACM STOC 2000.

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

[3] 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.

[4] C. Makris, Y. Panagis, E. Sakkopoulos, and A. Tsakalidis, "Efficient and adaptive discovery techniques of web services handling large data sets", in J. Systems and Software, Elsevier Science, to appear.

[5] Ch. Makris, E. Sakkopoulos, S. Sioutas, P. Triantafillou, A. Tsakalidis, B. Vassiliadis, "NIPPERS: Network of InterPolated PeERS for Web Service Discovery", in the proceedings of the 2005 IEEE International Conference on Information Technology: Coding & Computing (IEEE ITCC 2005), Track Next Generation Web and Grid Systems, Full Paper, Las Vegas, USA, pp. 193-198

[6] 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.

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

[8] 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.

[9] 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).

[10] 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.

[11] DAML Services (DAML-S / OWL-S). Semantic Web Services Architecture (SWSA) Committee. URL http://www.daml.org/services

[12] Garofalakis, J., Panagis, Y., Sakkopoulos, E., Tsakalidis, A., 2004. Web service discovery mechanisms: Looking for a needle in a haystack? In: International Workshop on Web Engineering, Hypermedia Development and Web Engineering Principles and Techniques: Put them in use, in conjunction with ACM Hypertext 2004. Extended version submitted. URL http://www.ht04.org/workshops/ WebEngineering/HT04WE Garofalakis.pdf

[13] 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.

[14] J. Borenstein, J. Fox, "Semantic Discovery of Web services: A step towards fulfillment of the vision", Web Services Journal, [on line] http://www.sys-con.com/webservices, 2004.

[15] 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.

[16] Moreau, L., Avila-Rosas, A., Miles, V. D. S., Liu, X., 2002. Agents for the grid: A comparison with web services (part ii: Service discovery). In: In Proceedings of Workshop on Challenges in Open Agent Systems. pp. 52-56.

[17] Nejdl, W. Wolpers, M. Siberski, W., Schmitz, C., Schlosser, M. Brunkhorst, I. Löser, A. (2004) Super-peer-based routing strategies for RDF-based peer-to-peer networks, *Journal of Web Semantics,* 177-186.

[18] Nejdl, W., Siberski, W. & Sintek, M. (2003) Design Issues and Challenges for RDF- and Schema-Based Peer-to-Peer Systems, *SIGMOD Record.*

[19] OASIS UDDI Specifications TC - Committee Specifications.

[20] Schmidt, C., Parashar, M., 2004. A peer-to-peer approach to web service discovery.World Wide Web 7 (2), 211-229.

[21] W3C, 2003. SOAP Version 1.2 W3C Recommendation Documents. URL http://www.w3.org/TR/SOAP

[22] Web Services Modeling Ontology URL http://www.wsmo.org/

[23] Yutu Liu, Anne H.H. Ngu and Liangzhao Zeng . QoS Computation and Policing in Dynamic Web Service Selection. In proceedings of the WWW2004, New York, USA. pp. 66-73