Domain Name System for PeerHosting

Arnav Aggarwal, B.Tech

A dissertation submitted to the University of Dublin,
in partial fulfilment of the requirements for degree of
Master of Science in Computer Science

2010

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Arnav Aggarwal, B.Tech
September 13, 2010

Permission to lend and/or copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Arnav Aggarwal, B.Tech
September 13, 2010

Abstract

With exponential increase in web content on the world wide web, it is becoming increasing expensive and challenging for the Web Hosting Providers to ensure strict QOS. Peerhosting platform is an open, self balancing cluster of Web Hosting Provider web servers which collaborate to host content collectively while reducing the cost of managing and hosting web content. It interleaves content distribution management and client redirection services to ensure content availability. The current implementation of redirection service is centralised hence it is a single point of failure and a potential bottleneck.

To ensure efficient content availability there is a need to design a service which integrates well with the Peerhosting platform to resolve the current content location while observing current state of individual HPs. Peerhosting dns solves this problem by using a pastry routing protocol to improve scalability and enable load distribution among multiple nodes. The result is a fully operational name service which offers average lookup latency comparable to that of legacy dns.

Acknowledgements

My sincere thanks go out to all those who helped and supported me during the development of this project, not least my excellent supervisor, Dr. Stephen Barrett helped me throughout the project duration. The success of this project relied on both his advice and calming influence throughout. A special thanks to Paul Laird for all his help and insight into Peerhosting requirements and other things. Finally, thanks to all my NDS classmates and Dr. Siobhan Clarke for guiding us throughout our course. I feel blessed to be part of NDS, this year has been the best year of my life!

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

In recent years, we have seen dramatic increase in popularity of web based applications such as Facebook, Youtube and Google applications. A large number of businesses are solely Internet based and many others depend on Internet for a large number of reasons such as internal and external connectivity. As a result web Hosting Delivery is becoming mission critical for the success of the businesses. These factors coupled with the shift towards using web applications has increased the complexity of existing networks making it difficult for the web hosting providers to ensure content delivery with consistent QOS across different geographic regions.

[2]The hosting providers are faced with two key issues. First, flash crowd problem and the distributed denial of service attacks often cause the networks to be overloaded. Second, the growing scale of internet is making it increasingly infeasible for the hosting providers to match in-house usage to the available capacity. Both these challenges need to be addressed in both the core Peerhosting platform as well its name resolution service.

1.2 PeerHosting Service

Web Hosting provision has become very complex and increasingly costly with the increase in the complexity of systems and the need to account for sudden and huge bandwidth requirements. To solve this problem , [2] the Peerhosting platform aims to enable a worldwide, open and self-balancing cluster of Website Hosting Provider web servers, provided by different organisations, which can safely collaborate to deliver web content more efficiently. It delivers a broker service which enables trade in hosting capacity among participating HPs and orchestrates delivery of web content using peer to peer model. The Peerhosting system needs to integrate flawlessly with the existing client technologies and should be transparent to the its clients.

1.3 Research Objective

Currently the Peerhosting model is dependent on the use of a centralised database for name resolution service. This implementation of resolution service has significant drawbacks. First, it offers limited scalability and flexibility. Second, it does not support fast propagation updates from HPs. Third, it is a central point of failure and a potential bottleneck. These factors coupled with high latency, load imbalance and centralised mechanism has significant performance and security implications on the success of the Peerhosting platform. It is therefore essential that the target service be able able to distribute load among multiple hosts while enabling fast propagation updates from HPs. The project aims to contemplate and construct a domain name service

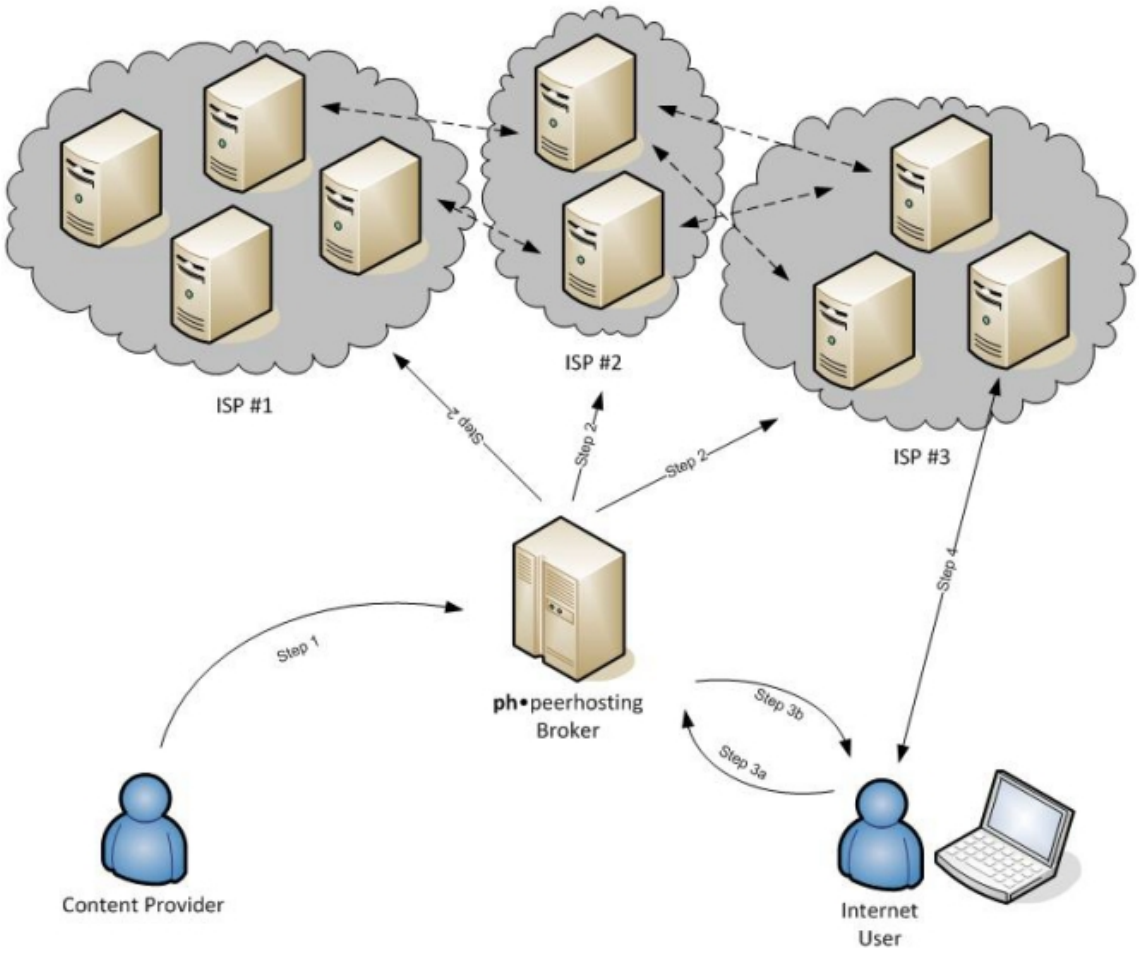

Figure 1.1: Peerhosting Architecture[2]

which integrates with [2] peerhosting platform and satisfies above stated requirements while offering name resolution service with delay times comparable to legacy DNS.

1.4 Detailed Hypotheses

To achieve the goals of this dissertation, a domain name resolution service for the Peerhosting platform will be investigated into, leading to the its implementation and evaluation. The known metrics for evaluation are query resolution time, replication efficiency, caching feasibility, load balancing and dynamic update propagation. The service would offer a name resolution service designed specifically to cater to the needs of the Peerhosting platform while providing some control over resource distribution to content providers. The service will be based on a completely decentralised, distributed architecture which would offer high scalability, low latency, natural resistance to flooding attacks, fast update propagation and improved load balancing. We discuss some of the peer to peer based name resolution services which satisfy some of the requirements of the peerhosting platform i.e.[21] CoDoNS and [6] DDNS. These systems will be studied and evaluated to design a service specific to the requirements of peerhosting platform. Our initial hypotheses is that peer to peer routing protocols based on structured distributed hash table (DHTs) might be a reasonable approach for constructing a domain name service. Therefore we will study some approaches before initiating the design of a name resolution service.

1.5 Outline

Chapter Two discusses the background of Peerhosting, legacy DNS service, various DHT protocols and several peer based DNS services which might be relevant to this project. The above mentioned technologies have been studied and reviewed and they act as the foundation for designing an appropriate name resolution service in the design chapter.

Chapter Three delves into the design of a domain name service for Peerhosting platform, this chapter covers a detailed explanation of Peerhosting, name service requirements along with a system design which satisfies above described requirements.

Chapter Four lists the steps taken to implement the Peerhosting domain name service, concentrating on numerous system components and libraries used.

Chapter Five evaluates Peerhosting DNS. In this chapter a test suite is designed and implemented which is used to evaluate the performance of the Peerhosting DNS implementation. Comparisons are drawn with legacy DNS system and approximations done for Peerhosting DNS system comprising several thousand nodes.

Chapter Six concludes the report and suggests future work for a better implementation of the Peerhosting platform.

Chapter 2

State of the Art

This chapter describes the current state of the art in the areas of domain name system (DNS), Distributed hash table (DHT) and peer based domain name systems. In particular, various applications and implementations of the distributed hash tables will be discussed in view of their suitability and relevance to the cause of designing the domain name resolution in [2] Peerhosting model. The chapter also documents metrics that can be used in the evaluation of the decentralised distributed name services.

2.1 Legacy DNS

2.1.1 Overview

[1] DNS is a hierarchical, managed distributed database coordinated by the DNS protocol which associates domain names with resource records (RRs) containing name data. DNS is the standard mechanism to advertise and locate content on the Internet. A resource record is a data set which stores information DNS consists of following components:

1. The domain name-space and resource records (RRs).
2. Name Servers containing information about domain's tree structure.
3. Resolvers which retrieve information from name servers in response to queries.

A domain name can be associated with multiple RRs either of same or different type. And a set of resource records for a same domain is called a RRset. The most common type of resource record is address type.

2.1.2 Structure and delegation

The domain name system is organised in a tree structure similar to the Unix file system where each node represents unique name. Each node in the tree is assigned a label with the exception of root node which is null. The domain names in the tree structure can be derived by the sequence of labels on the path from the current node towards the root node, separated by dots. The Domain Name System is divided into domains and the responsibility for maintaining a domain is delegated to independent organisations which can further delegate the responsibility of sub-domains to other organisations. The domains at the top level, second level and lower levels are broken into manageable units called zones. A domain can contain data delegated to other name servers whereas zones never contain delegated data.

[1] DNS defines two types of name servers, primary masters and secondary masters. The primary master reads the zone data from the file on the host while the secondary master (also called slave nowadays) reads data from the primary master. Both the primary master and the secondary master are authoritative for the

zone though the content owners use primary master server for maintenance. The slave servers refresh their information after fixed interval from the primary master server.

2.1.3 Name resolution

The root name servers maintain information about authoritative name servers for the top level domains.The top level domains have the knowledge of the authoritative name server of their sub domains and so on. The resolution starts from the root name servers and therefore they experience extremely high loads.

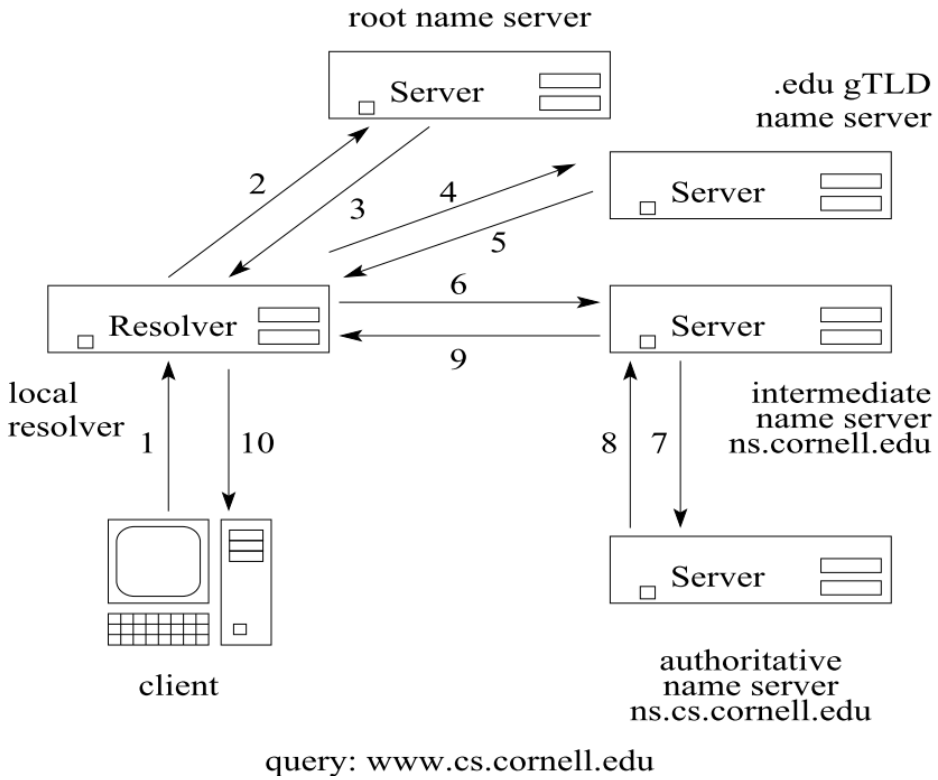

Figure 2.1: The DNS Resolver in Legacy DNS. Iterative resolver process shown by steps 2,3,4,5. Recursive resolver process represented by steps 6,7,8,9

Resolver programs are required to resolve a domain name to the resource record to find the address of the target machine hosting the desired content. Resolvers programs can be located on the local clients and on the ISPs. Resolvers resolve the received query by either recursive resolution or by iterative resolution approach.

Recursion

In recursive resolution approach, a resolver sends a recursive query to the name server which is then required to reply with the complete information or with an error stating that the domain can not be resolved. The task of resolving queries is performed by name server. The resolver sends a query to a name server which is then expected to return the requested information. As with recursive algorithms, the name server send queries to the various remote name servers and the follows referrals until it receives the answer from the final name server. The name server then returns the requested data to the resolver. Figure 2.1 shows a resolver receiving a dns query from the client. The resolver then contacts various name servers with an objective to resolve the query.

Iteration

In iterative approach, the task of resolving the query is performed by the resolver itself. The resolver sends a query to the name server but the name server is not obligated to return the final result. Name server returns the best possible answer also called a referral. The return data includes the list of possible name servers. The

Resolver is free to select any name server from the list. Figure 2.1 shows a resolver sending a query to the Server and the server returning the best response to the query without connecting to other name servers.

2.1.4 Caching

A name server stores the data it retrieves while it processes recursive queries. Each time it connects to other name server, it discovers new information which it stores for future reference. This information can be complete or incomplete, for example it can be resolved query or information about a more appropriate node.The mechanism in which intermediate name servers store the relevant information received from other name servers for some limited time is called caching. Name servers even implement negative caching, which is, they store negative responses received from the queried name server. For example if an authoritative name server responds to a query with an answer saying that domain name does not exist. The local name server will cache that information for a specified time and respond to queries without contacting any other name servers.

Name Servers cache data for a pre-defined time which is equal to the time to live (TTL). TTL is measured in mill seconds and it is the time for which any name server is allowed to cache the data. This mechanism is responsible for improving the response times in legacy DNS.

2.1.5 DNSSEC

With the growing concern about DNS security,[9] DNSSEC was conceptualised in the late 1990's to introduce authentication and integrity mechanism into the DNS. The DNS security extensions (DNSSEC) provides security infrastructure using hierarchy of cryptographic signatures, attached to each resource record. It achieves this by verification of resource records obtained by sources other than the authoritative name server using the public private key pair.

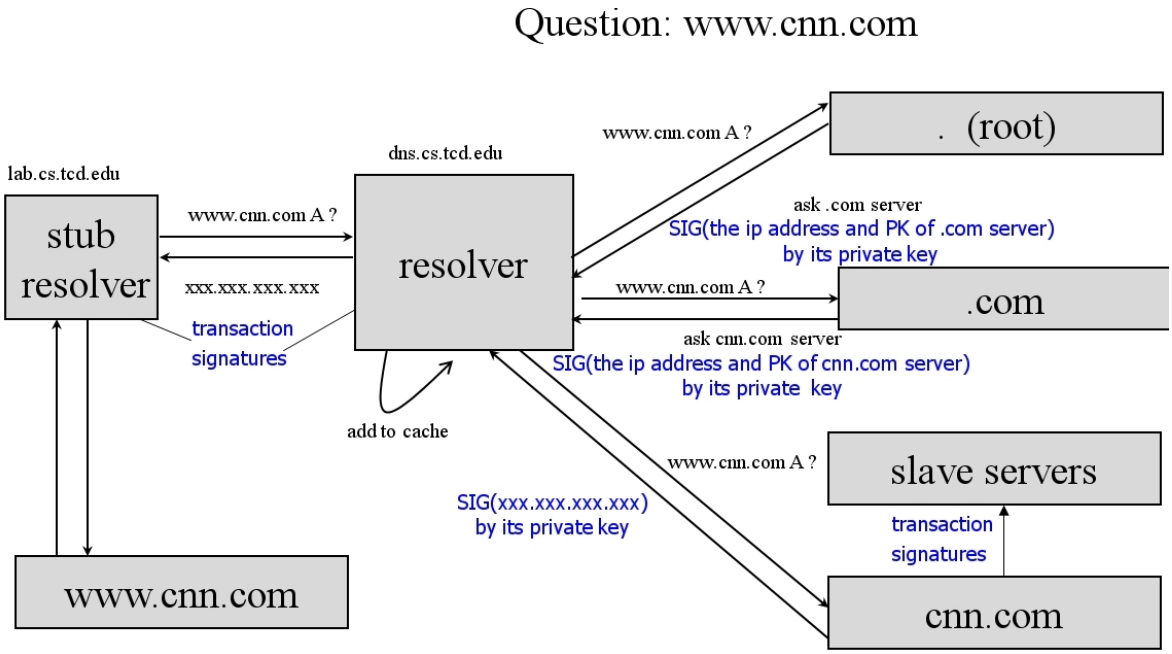

Figure 2.2: Secure DNS query and response

DNSSEC specification introduces four new types of resource records,[10] which are following:

1. Resource record signature(RRSIG)
2. DNS public key(DNSKEY)
3. Delegation signer(DS)
4. Next secure(NSEC)

Every name-space operator contains resource records signed with the private key stored in RRSIG, the private key need not be online as the records are signed at the time of issue. The public key generated is maintained by a zone that has the capability to sign its records, also called ”islands of security” [1]. The clients can verify the authenticity of the resource record by using the public key (DNSKEY) retrieved from the ”islands of security” name servers to verify the signature the private key (RRSIG).

Figure 2.2 shows a dns query for resolving ”www.cnn.com” directed to a stub resolver located at ”lab.cs.tcd.edu”. The stub resolver forwards a recursive query to a resolver at ”dos.cs.tcd.edu”. The resolver resolves the resource record recursively by following the hierarchy and then verifies the resource record by fetching RRKEY from the designated island of security and using it with SIG resource record.

2.1.6 Legacy DNS challenges

The legacy DNS design focus primarily on system robustness against physical failures and neglects the impact of operational errors such as misconfiguration and bottlenecks. Venugopalan et. al [21] conducted a large scale survey to analyse and quantify vulnerabilities in the legacy DNS system. Some of the challenges faced by the current DNS implementation are mentioned below:

1. Implementation Errors
   One of the earliest and the commonest errors in DNS have been caused by incorrect implementation [16, 13]. While many have been highlighted, a large number of systems are still plagued by implementation errors.
2. Bottlenecks
   (a) Delegation bottleneck problem
   The DNS is negatively impacted by the Bottlenecks encountered at any point in the delegation chain. Essentially, the DNS servers are sensitive to delegation attacks caused by malicious agents. A delegation bottleneck is the minimum number of servers in the chain which can be targeted to disable name resolution of a domain. venugopalan et al. [21] show that 90 percent of DNS domains are served by three or fewer name servers and 78.63 percent of DNS are served by only two name servers, which is the minimum recommended. With such small number of servers, domains can be disabled even with a relatively small DOS attack.
   (b) Physical bottleneck problem
   The minimum number of routers and gateways between name servers and clients that need to be attacked to compromise domain is called the physical bottleneck. Venugopalan et. al[21] measured physical bottlenecks by performing trace routes to 10,000 name servers and found out that about 33 percent domains are bottlenecked at single router or gateway.
   (c) DNSSEC Implementation
   The DNSSEC is still to be widely adopted and this may create complex security vulnerabilities.

3. Delegation attacks
   Delegation attacks have been the reason for one of the biggest successful attack on the internet. [26]In 2001 Microsoft's DNS servers were successfully attacked primary reason being the presence of DNS servers in the same sub network.

4. Latency
   Latency is a critical metric for the measuring performance to web services. Jung et al.[11] reported that 10 percent of name resolution queries take more than 2 seconds, Wills and Shang[29] scrutinised NLANR logs and concluded that DNS look up queries add over a second to over 20 percent of name resolution queries. [18] Among others, one of the primary reason for high latency is the heavy tailed Zipf like query distributions in DNS which results in low cache hit rates. Service providers such as Digital island and Akamai perform network optimisations such as automatic load balancing and near server selection by setting the TTL in resource records of order of few seconds (example 30 seconds). This mechanism eliminates the advantages of caching in a legacy DNS system and impacts the latency negatively.

5. Load imbalance
   Originally DNS root servers comprised of only 13 servers, before it was successfully [15]attacked and brought down in November 2002. Thereafter root servers are maintained by more than 60 name servers to prevent any such attack in the future. While this has solved the problem at the root level, this problem still persists at the TLD's and lower level name servers and makes the DNS susceptible to such future attacks.

6. Update propagation
   While caching solves some of the performance related issues relating to DNS, it poses some serious drawbacks as well. [11]The selection of TTL is tricky, as a lower TTL results in reducing the effect of caching and reduces lookup performance while setting a large TTL restricts service relocation.

All the above mentioned problems coupled with the lack of support for fast dynamic updates, encourages me to proceed further and investigate the suitability of some of the peer based solutions for peerhosting project.

2.2 Distributed Hash Table based Technologies

2.2.1 Overview

Distributed hash tables have been used as a fundamental building block in many peer-to-peer systems[12][30]. Distributed hash Table (DHT) is a giant hash table that route a lookup query and map keys (derived from query) to their corresponding values, over a decentralised distributed system. Responsibility for maintaining the mapping information is distributed among the peers, with the aim of minimizing disruption in a volatile network. DHTs find many applications such as peer to peer file sharing networks (P2P), distributed file systems (DFS), domain name systems (DNS), instant messaging.

[3] DHTs commonly consist of the following components:

- Consistent hashing over a one dimensional space which is often used as a identify nodes in a network.
- Indexing topology to allow navigation in the peer network.

2.2.2 Implementations and protocols

Some of the DHT based protocols and implementations are the following:-

- Plaxton [19]
- Chord[25]
- Pastry [24]
- Tapestry [30]
- Kademlia[14]
- CAN(Content Addressable Network)[22]

Generally DHT systems work in the following manner. Each node is assigned a random unique identifier in one dimensional circular identifier space. Objects are assigned a unique random identifier in one dimensional space, which is stored at the node with has closest identifier as its own. Peers in the ring network are connected to each other by their neighbours. For searching objects, a node generates a query message, which traverses from node to node, each comparing the object identifier with its node identifier.[24][30] Many DHT algorithms such as Pastry and Tapestry offer lookup performance of $O(\log_b N)$, where N is the number of nodes in the DHT and b is the base.

[30] Tapestry,[24] Pastry,[25] Chord and [22] CAN represent a second generation peer-to-peer routing and location systems which have been inspired by the systems like Freenet and Gnutella. The above systems are better than other systems such as Plaxton as they guarantee a definite answer to a query in a limited number of network hops and offer similar scalability as that of Freenet with properties of self-organization.

[24] Pastry and [30] Tapestry resemble the work done by Plaxton et al. [19] in the historical hierarchy routing [27] mechanism. The approach of routing is based on address prefixes (regarded as a generalization of hypercube routing), which is common to all these systems, however Plaxton is not self organising.

[20] Pastry and Tapestry are different in their approach and are aimed at achieving network locality and replication. Unlike Pastry and Tapestry, Chord does not make an explicit effort to achieve good network locality as it routes messages to nodes based on numerical difference with the destination address.[22]In CAN, messages are routed in a d-dimensional space, where the size of routing table is O(d). The maximum cost of reaching a node in CAN network is $O(dN^{1/d})$ and the routing table does not grow with the size of the network (routing hops grows faster than logN) as in case of Pastry.

2.2.3 Characteristics

DHTs emphasise on the following characteristics:

- Decentralization
  The system is constituted by independent nodes without there being no or very little central coordination.
- Scalability
  The system should work efficiently even with thousands or millions of nodes in it.
- Fault tolerance
  The system is reliable and works as expected in cases of new nodes, failing nodes, lost nodes in the network. The system should be able to function without interruption until a large subset of nodes fail simultaneously.

A possible solution for achieving these goals is for each node to coordinate with only few other nodes in the system - most often a node is connected to O(log N) nodes where N is the total number of participating nodes. This is to decouple the node from the system by limiting its exposure to the possible changes in the composition of the system . Some models of DHT seek to be free of malicious participants [28] and often allow participants to remain anonymous, although this is less common. Finally, traditional DHT based distributed systems address issues such as load balancing, data integrity and performance to ensure that operations such as routing and data storage/ retrieval work flawlessly.

2.2.4 Performance metrics

[17]The performance of DHT's are often measured in view of the following metrics:-

- Cost of joining/leaving
  The system should should be flexible and robust such that there is minimal disruption to its working when existing nodes leave or new nodes join the network. The data should be well organised and distributed to ensure content availability.

- Cost of searching
  Searching cost is one of most important feature for evaluating a DHT in view of its use in a domain name system for Peerhosting. The cost of searching should be minimal in terms of number of steps taken for message passing between a pair of nodes.

- Congestion
  Focus must be on minimizing congestion on any given server/node. The load must be distributed rationally among all the nodes to ensure fair sharing of resources. This is also important in improving availability of content.

- Fault tolerance
  The system should be robust against failures such as node failures, application failure, network congestion, high latency, unreachable nodes, etc. The system must behave consistently upto a standardised limit in the following two conditions. First, when a large subset of nodes fail simultaneously. Second is based on Byzantine model in which large number of nodes may behave inconsistently.

The following characteristics are more specific to our problem of constructing a dns solution for Peerhosting platform.

- Load balancing
  This characteristic is crucial to the design of the DNS. However with the use of peer based DNS, this characteristic is satisfied to the required level.

- Data integrity
  The system must support secure delegation to maintain integrity of DNS records. There should be mechanism which distinguishes between a malicious node and untrustworthy one.

- Propagation of updates
  Propagation of updated DNS records quickly to the peers over the network is an necessity. The clients should only be directed to peers who are in possession of the desired content.

- Dynamic caching
  Caching is an integral mechanism which can be very efficient in reducing the number of hops traversed to answer a query. Therefore an important goal of the system is to implement some kind of caching mechanism which caches data based on network load and the popularity of objects. The performance trade-off of caching should be minimal.

2.2.5 Pastry

We considered Pastry routing protocol as a suitable candidate for designing a dns solution for Peerhosting. This protocol is fully described in [24]. We look more closely at the Pastry routing protocol as it fulfils some of the above stated requirements for implementing a dns solution for Peerhosting platform.

Overview

Pastry is a generic,decentralized routing and object location solution for creating a very large, self configuring network. It works on the principle of prefix-routing and allows creation of structured overlay networks which are completely decentralised, fault tolerant, scalable. [24]It is optimistic in controlling concurrent node arrivals and departures. Arrival and departures affect only a small number of existing nodes in the system, therefore contention is rare. Some of the noted applications of Pastry are SCRIBE[4] and PAST[8].

Design and routing

In pastry peer to peer overlay network, each node is assigned a 128-bit unique node identifier (node ID) using SHA-1 algorithm. NodeId can be used to locate a node's position in a circular nodeId space of range 0-2^{128}-1. NodeId is assigned to each joining node, and nodeId's are generated such that they are uniformly distributed by taking a cryptographic hash of any unique characteristic associated with the node, such as its ip-address. Objects in the a pastry network are assigned a unique 160 bit identifier (objectId) by hashing the object with using a hash algorithm. Objects are assigned to those nodes in the pastry ring whose nodeId is closest to the obectId.

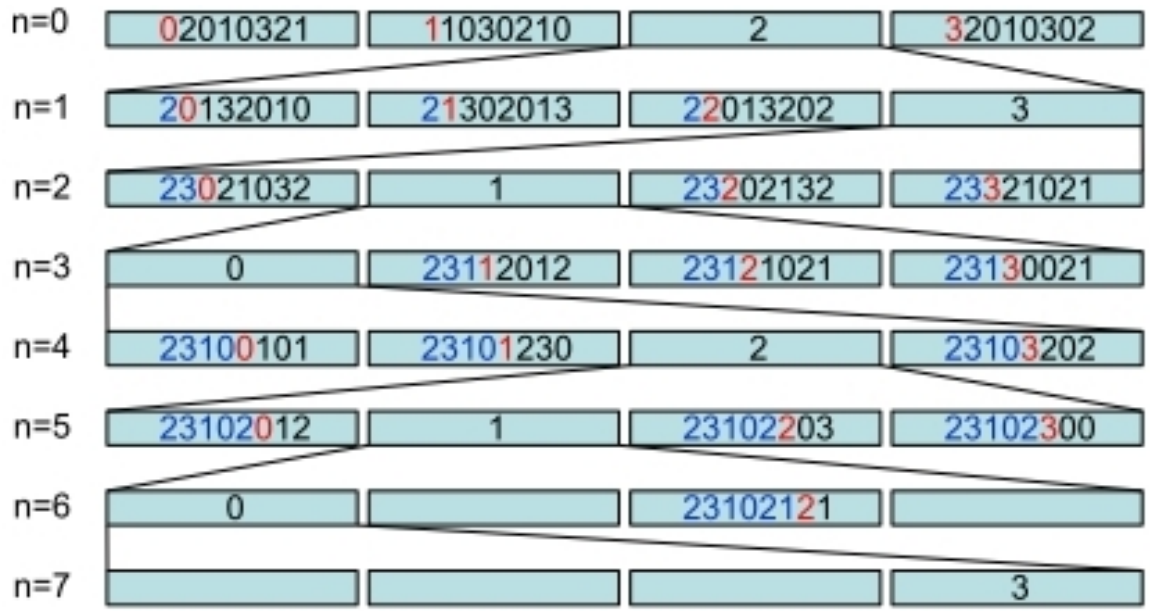

Figure 2.3: Pastry routing table. Here b = 2, nodeId = 23102103. It can be seen that first row contains nodeId having the same prefix i.e. "2" . The second value in nodeId is 3, row(n=2)shows possible values for prefix 23

Each node in Pastry ring maintains the following:

1. Routing table
   The routing table R consists of $\log_{2^b}$N rows and 2^b-1 columns. Each routing table entry at row "n" contains the IP address of only some of the nodes which share first n digits as prefix with condition that $(n+1)^{th}$ digit has one of the 2^b-1 possible values other than $(n+1)^{th}$ digit. The maximum number of rows in the routing table is 128/b but in practice only (approx)$\log_{2^b}$N rows are populated. The value of b establishes a trade-off between size of routing table and maximum number of hops required to route

between a pair of hops. With a value of b equalling four, the average number of hops is approximately equal to five.

2. Neighbourhood set
   The Neighbourhood set (M) contains nodeIds and IP addresses of $|M|$ closest nodes to current node in respect to proximity metric. This information is used to improve routing locality properties and is not used for routing normally.

3. Leaf set
   The Leaf set L is the set of nodes comprising of $|L|/2$ nodes with numerically lower NodeIds and $|L|/2$ nodes with numerically higher nodeIds relative to the nodeId of the current node. The Leaf set information is used for message routing and it contains the set of nodes of size L, which are numerically closest to current nodeId, with half nodes having smaller nodesId's and half with greater nodeIds than the current node.

Operation

Any node in the pastry ring can initiate a message towards a target node using a key. With a message and a key as a input , the node sends the message to other pastry nodes on the network, which then delivers the message to the node whose nodeId is numerically closest to key, among all the live Pastry nodes. This process continues until the message is delivered to the numerically closest node. The maximum number of routing steps possible before the message reaches the target node is O ($\log_{2^b}$ N), where b is a constant and N is the number of nodes in the pastry nodes.

Pastry advantages

Pastry aims to minimize the message travel distance considering network locality, using any given scalar proximity metric, for example the number of IP routing hops. Each Pastry regularly transmits keep-alive messages to inform the neighbouring nodes of their status. This mechanism notifies application on live nodes of node failures, new nodes and recovered nodes in the nodeId space. Since node ID's are randomly assigned, the probability of adjacent nodeId's being allocated in a wide geographic area is extremely high. This property is useful and ensures that the message reaches nodes with numerically closest nodeId in accordance with the proximity metric which is often completely different from the physical metrics. This property is also advantageous as messages are spread across diverse geographic region which enables better load distribution and improves system robustness against failure of part of network.

2.3 DDNS

2.3.1 Overview

[6]DDNS is a peer to peer domain name system which stores and retrieves resource records using DHash (Chord based distributed hash table).The system inherits Chords fault tolerance and load balancing properties and reduces administrative problems with the current DNS.

2.3.2 Operation

Each node in the system selects a n-bit identifier randomly generated using the hash of some fixed information such as IP address. The identifiers (nodes) are arranged in a circular ring called the name space. The objects are assigned lookup keys which is generated by taking the hash of the object with some random salt. Each node in a chord ring is responsible for storing objects with keys closer to its nodeId i.e in-between the the current node and its previous node. Figure 2.4 shows a chord ring showing responsibility of each node in the ring. [25]The

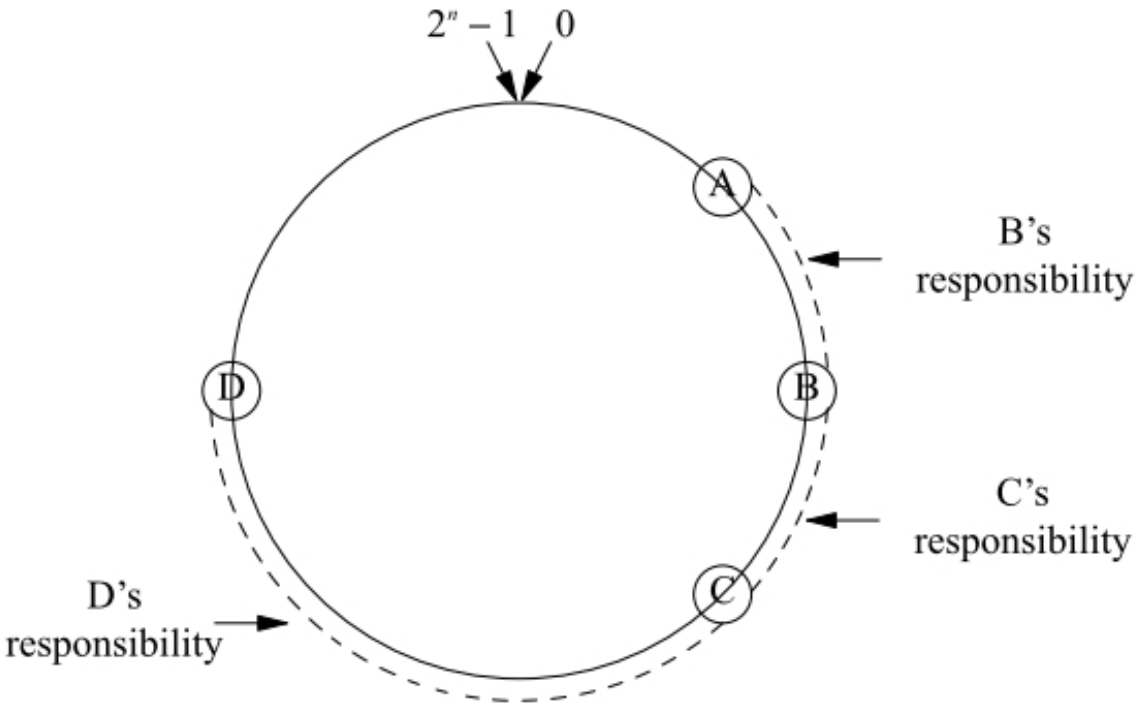

Figure 2.4: [23] Chord ring showing the responsibility of keys

routing is performed by using a finger table algorithm. The algorithm allows the nodes to memorise locations of other nodes in the peer network which reduces the number of traversals required to resolve a query. In a chord ring the maximum cost of reaching the desired destination node responsible for the queried key is log(n) where n is the size of the chord network.

2.3.3 Advantages

The DDNS offers the following benefits over existing DNS system.

1. DNS is a administered hierarchical system, hence requires certain level of expertise to administer. Whereas DDNS is a peer based DNS and separates service from its authority. The content owner and its Data can simply be entered into the chord storage ring without worrying about the ISPs to be on-line all the time.

2. It offers a better approach to load balancing as the concept of the hierarchical managed DNS is eliminated and the dns information is replicated on multiple nodes selected randomly.

3. It offers better resistance to denial of service attacks as the data is replicated on nodes connected to the chord ring. Taking out a few nodes from the system will not affect availability of data. Also to cater to increased demand, resource record sets can be replicated to more nodes.

4. It is based on DHash which offers improved robustness and load balancing by allocating keys to nodes evenly through consistent hashing.

5. The data is cached by nodes along the lookup path which increases the probability of spreading data on each node by only log m transmissions where m is the size of chord ring.

6. Dhash moves the data automatically so that the data is replicated across a fixed number of nodes in a pseudo-random fashion. This makes system resistant to denial of service attacks and improves fault tolerance.

2.3.4 Challenges

DDNS system was designed by Russ Cox et. al [6], they demonstrated that DDNS system is faced with some challenges, which is significant in context of the functionality required in the peerhosting model. The challenges have been discussed below.

1. The time taken to resolve a dns query in DDNS is higher than the conventional DNS. This is because in a chord ring , a message requires O($\log_b$ n) remote procedural calls per lookup where b= 2. This equals to 20 RPCs in a network consisting of million nodes. The latency in traversing 10-20 RPC is unrealistic for a domain name system.

2. The client needs to be updated each time a new functionality is to be offered by the system which is infeasible and problematic.

3. The system depends on coupling of administrative hierarchy with the service structure. Clients are a part of the chord ring. Thus there is no incentive for clients to serve DNS for other people servers.

2.4 CoDoNS

2.4.1 Overview

[21]CoDoNS is a peer based cooperative domain name system which provides high performance using Beehive replication layer to reduce the lookup latency and increase query throughput. CoDoNS is based on Pastry routing protocol which works on the principle of prefix matching for searching objects on the CoDoNS network. The beehive replication layer has been explained in detail in the following paragraphs.

2.4.2 Beehive

[20]Beehive is a high performance peer to peer replication layer. It uses structured replication and enables O(1) search latencies on the overlay, meaning that the maximum cost of finding a solution can be one hop (for a fraction of queries). Beehive considers replication and caching as a fundamental problem. The caching optimization is expressed taking into account the trade-off between the cost of replication and the resulting system performance. This analytical formulation enables Beehive to predict expected behaviour of the system.

Beehive improves performance by replicating objects dynamically on the overlay based on factors such as content demand, demand patterns etc. It performs mathematical computation to calculate the cost of replication and compares it with the actual benefit of replication in reducing the lookup time.

Operation

Beehive works on the principle of replicating objects at some established level in the DHT circular space. Level of replication signifies the number of nodes on which the object is replicated. For example, objects whose level is 0, has to be distributed at all nodes across the system, similarly objects whose level is ”i” are replicated on all nodes having at least ”i” matching prefixes. Beehive uses various replication strategies through which it decides a replication level for each object. Figure 2.5 shows the effect of replication in the system.

Beehive employs several inexpensive algorithms through which it computes the replication level for each node dynamically, the key parameter used by such algorithms being the popularity of the object. It does so with the aim of providing a constant cost of finding an object in the system while reducing administrative control to ensure availability.

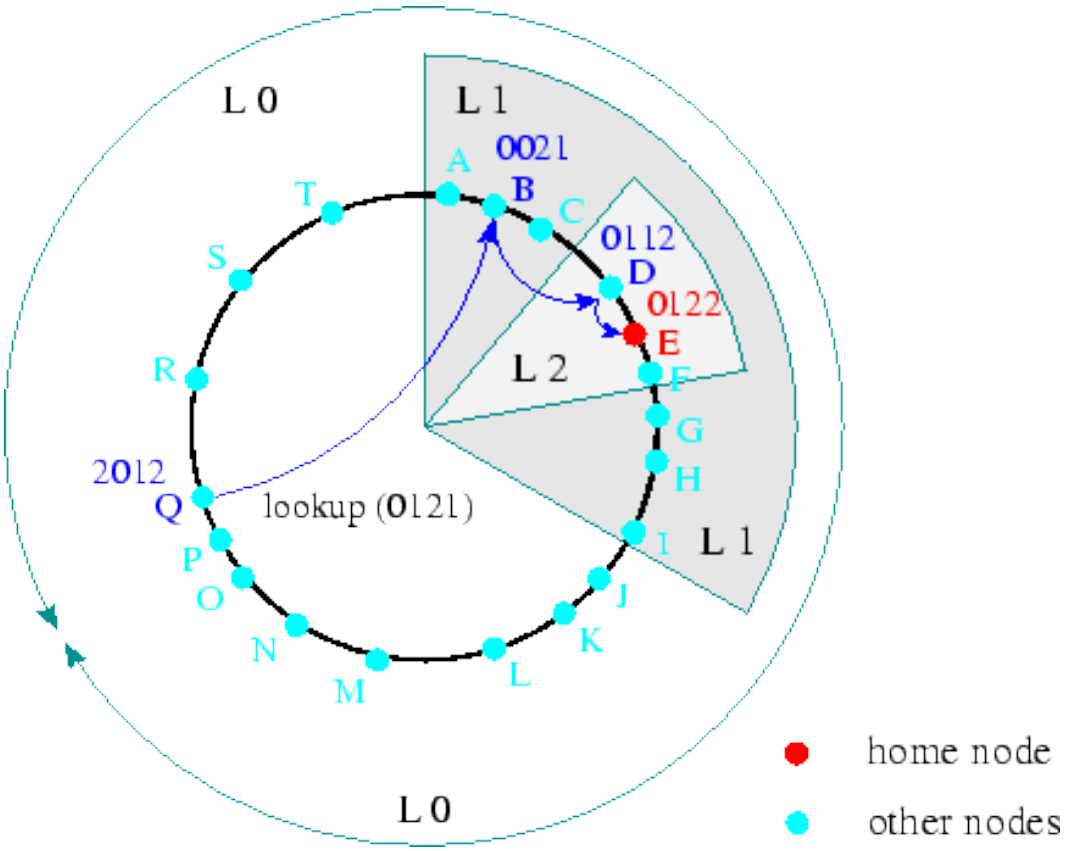

Figure 2.5: Replication level in Beehive system.[20] "A query for finding object with objectId "0121" takes three steps from node Q to node E (Home Node)." By replicating the object at level two (D and F), the latency is reduced.

Beehive uses the following models for computing the level of replication:-

1. Monitoring protocol
   Monitoring protocol estimates properties of query distribution and popularity of objects by collecting, monitoring and aggregating the locally available data. Monitoring is performed by individual nodes by measuring the object access frequency and then aggregating them with other nodes statistics after regular intervals. Each node gather aggregate data from other nodes which are one level higher in routing table, which results in trickling of aggregates to the lowest level node, ultimately reaching the home node. The home node computes the total aggregate and spreads this information to all the replicas in the network.

2. Analytical model
   Each Beehive node invokes the analytical model after every "analysis interval" to find out the level of replication for its objects. Analytical model consists of closed form optimal solutions through which a node finds the appropriate level of replication for each object it stores. The results of the monitoring protocol i.e. the estimates are fed as a input to the analytical model. The analytical model captures space time trade-off while incurring a minimum bandwidth and storage overhead cost by replicating objects across several nodes.

3. Replication protocol
   Replication protocol replicates objects to a level specified by the analytical model. Each node is responsible for deciding the local replication factor according to which it replicates the object on the adjacent nodes (nodes which are one hop away).

Characteristics

Beehive adds to the resilience and scalability of structured overlays by providing the following critical characteristics necessary to support performance demanding applications/infrastructures.

1. Adaptability
   Beehive quickly recognizes and respond to large-scale increase in popularity of objects occurring due to flash crowds and denial of service attacks. Beehive detects change in popularity of objects and adjust the extent of replication accordingly.

2. Quick updates
   Updates in Beehive percolate using a mechanism called Proactive replication. Beehive nodes notify other nodes of updates by distributing copies of object. Unlike legacy DNS mechanism, it does not use TTL parameter for caching and ensures consistency of modifiable objects using proactive propagation of updates.

3. High Performance
   Beehive guarantees content availability and consistent lookup performance while requiring minimum effort, memory, network bandwidth and load.

2.4.3 Operation

CoDoNS consists of distributed nodes that can self organise to form a network. It associates the domain to a node based on the similarity of objectId to the globally unique node identifier, such a node is called the home node. In a CoDoNS system, the home node is responsible for storing, maintaining and replicating the resource record on the network. In case of failure of home node, the next closest node takes up the responsibility and becomes the home node of the domain. CoDoNS prevents data loss by replicating the resource records on nodes (servers) adjacent to the home node in the CoDoNS network.

CoDoNS system uses Pastry routing protocol for routing messages. A brief explanation of it is as follows. A client sends a query(in Legacy DNS format) to a local CoDoNS server. The server looks up for the cache table and responds if finds any entry, else it directs the request on the internal CoDoNS network. The request terminates at either the home node or the intermediate node having cached copy. The node answers the request by sending the data to the nearest CoDoNS server which then replies to the client. In case the server is not able to find the resource record in the system, the home node fetches the record from the legacy DNS and sends the result back to the requesting server.

2.4.4 Characteristics

CoDoNS relies on cooperating peer caches which change dynamically according to the popularity of an object. It offers following unique characteristics:

1. DOS resilience
   CoDoNS is highly scalable as it responds favourably to denial of service attacks by dynamically changing the replication level of an object to distribute the load among multiple peers.

2. High performance
   Using the analytical model of Beehive, CoDoNS servers can answer queries even in a single hop. The performance of such a service is way better than the legacy DNS and DDNS as the number of hops required to answer a query is greatly reduced.

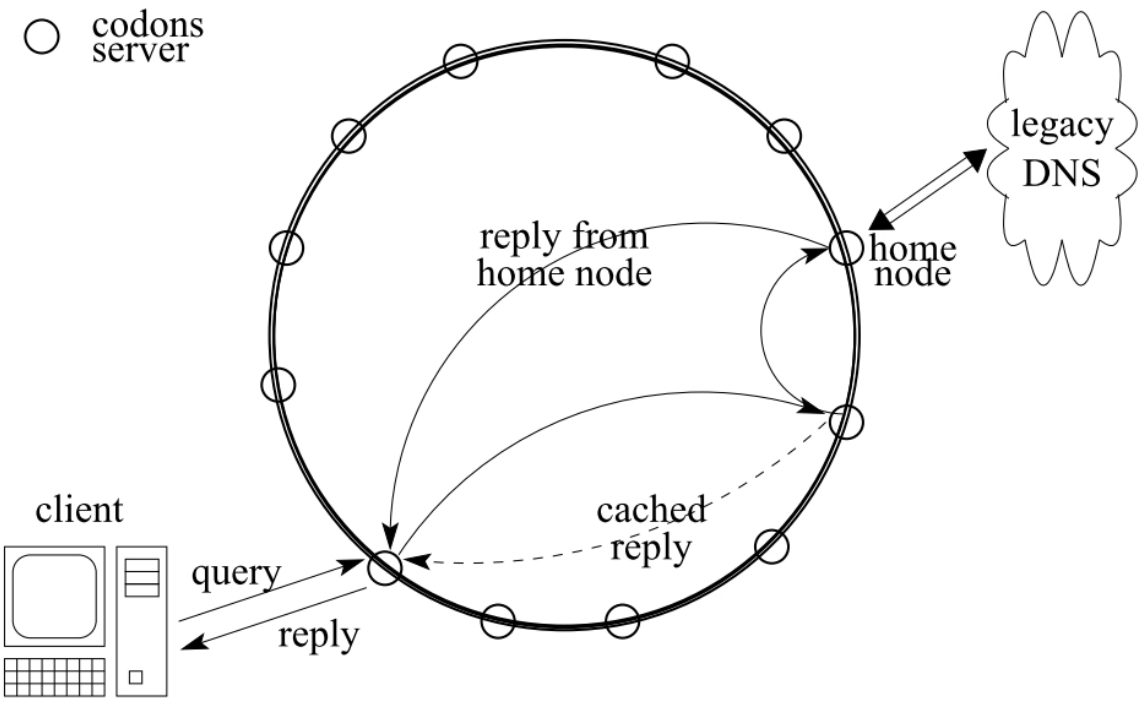

Figure 2.6: CoDoNS System: Clients sending request to local CoDoNS server. The server obtains RR from the home node/ intermediate node or from legacy DNS. The home node sends back reply through the server back to the client.

3. Lower latency
   CoDoNS has lower latencies than the conventional DDNS (20 RPCs in a network of million nodes)and legacy DNS which typically needs 2 RPCs. Pastry takes $O(\log_b n)$ remote procedural calls per lookup where b= 16. This equals to only 5 RPCs in a network consisting of million nodes.

4. Quick updates
   Updates in the CoDoNS are spread in relatively short time, nodes can be updated at any given time. In traditional DNS updates become relevant after the TTL of the data has expired. Thus, CoDoNS enables dynamic relocation of services, particularly in response to emergencies.

2.4.5 Challenges

Some of the challenges and ad faced by CoDoNS systems are:-

1. The peer based systems require that the results of all potential queries be anticipated and stored in advance which is not always possible. In the conventional DNS, some systems such as Akamai use custom DNS responses for better load balancing and to route the clients to nearby servers reducing latency.

2. [23] The CoDoNS system has unrealistic memory requirements on each node.

2.4.6 Relevance

The system is relevant in solving the DNS problem in context of peerhosting model due to the following factors:

1. No change is required on the resolver or the client side. The system integrates well with the current DNS clients and no change in the client-side resolver libraries is required.

2. Domain names can be explicitly added and securely managed by the owners. It can also acquire mappings from the legacy DNS.

3. It achieves high lookup performance using Beehive, an analytically driven proactive caching layer. [20]. Beehive anticipates demand and replicates DNS data automatically throughout the network.

4. The peer based architecture of the CoDoNS has decoupled the name-space management from the service hierarchy in the network.

5. Domain name records in the system are self validating and tamper proof. All the clients(nodes) on the network can answer the query authoritatively without the need to refer to the authoritative server.

6. It counters the possibility of delay by allowing direct caching. CoDoNS allows name owners to directly insert, modify, update and delete their records in their own administrative domain.

111

Chapter 3

System requirements and architecture

In this chapter we will discuss the underlined principle, key requirements and a system design for Peerhosting dns. The system is designed to create a robust peer based dns solution for Peerhosting platform.

3.1 Peerhosting

3.1.1 Overview

Current web hosting environments require Web Hosting Providers (WHPs) to maintain fixed resources such bandwidth much greater than required to account for sudden unexpected demand peaks. This is absolutely essential to adhere to the contracted QOS. This model is expensive and wasteful as WHPs are required to make provision for additional resources even when there is no need. It should be noted that the term peer has been used interchangeably with node as both of them refer to participant nodes in the dns network. Figure 3.1 shows a technical diagram of Peerhosting platform customised to interface with its dns network.

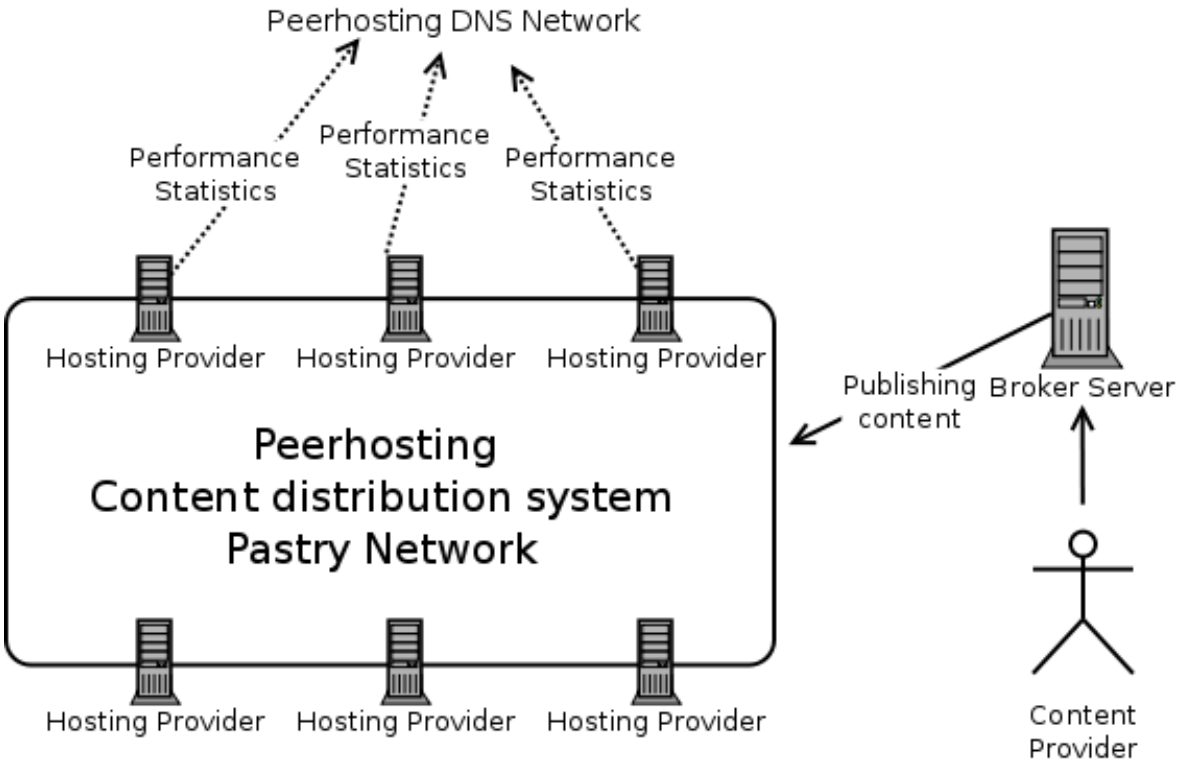

Figure 3.1: Peerhosting Architecture

Peerhosting provides a hosting infrastructure which aims to improve QOS offered by HPs while reducing cost to their clients. Peerhosting is an open, global and self balancing cluster of WHP web servers belonging to different organisations. WHPs safely collaborate to deliver the content through content sharing and distribution.

Peerhosting comprise of a centralised market model (Broker service) which enables real time trade amongst partner HPs. Broker service allows HPs to purchase credits enabling them to transfer their content to other HPs to cater to increased demand. The Broker is also responsible for redirecting the clients to new content location (HPs). This service is completely transparent to the clients and hence there is no need to change client resolvers.

Peerhosting integrates the hosting capacity of individual web hosting providers through the use of:

- Content distribution system
- Client Redirection
- Execution of monitoring services

3.1.2 Stakeholders

The Peerhosting comprise of following four stakeholders:

1. Hosting Providers
   Hosting providers are members of Peerhosting infrastructure providing active nodes which host or deliver content upon the request from the broker.
2. Broker
   It is the key component of Peerhosting infrastructure. It enables trade of hosting capacity between partner HPs and manages peer based delivery of web content to clients by redirecting them to the appropriate HP.
3. Content Providers
   Customers of any HP and a members of the market who want to share their content (such as websites, audio, video, virtual machines etc.) on the world wide web.
4. Internet Users
   Clients who access/consume the content on the world wide web through the use of standard browsers.

3.1.3 Operation

The Peerhosting platform operates as follows:

1. Initially, the content providers register with hosting providers to host their content on the world wide web. Then content providers register with the Peerhosting infrastructure by purchasing capacity from the Peerhosting broker. This capacity is aggregated from the underlying peer network of the Peerhosting infrastructure and used to host content of the content provider. To host their content, the content providers can either select HP's themselves or the selection can be dynamic based on HP nodes offering required QOS at the lowest possible expense to the content provider.
2. The content is replicated across the peer network(HPs) to prevent data loss due to node failure. In most cases virtual machines containing the content and runtime environment are transferred over the network which gets installed automatically on the peer systems(HPs). The HPs hosting this content register with the broker to activate the monitoring system for accounting resource usage.

3. Broker is responsible for providing domain name resolution services. On receiving request for content powered by Peerhosting, the broker directs the clients to the appropriate HP which is hosting the content. Broker selects the HPs based on technical and financial factors. This decision is made in real time.

4. The client receives the resolved IP address for the dns query and accesses the content from the resolved node(HP).

5. The resource utilisation at the HP on Peerhosting infrastructure that served the request is registered periodically for billing the content provider.

3.1.4 Challenges

There are a few unsolved problems in the Peerhosting infrastructure though we are primarily concerned with the domain name resolution service offered by the Broker. Therefore we will focus only on the challenges in creating a name service for peerhosting. Some of the apparent drawbacks of the current implementation of domain name service provided by Broker in Peerhosting model are:

1. It uses a centralised database server which maps domain names to IP address. Such an implementation is prone to denial of service attacks as it creates becomes central point of failure.

2. A central database is not scalable and would not be able effectively resolve queries for many thousand domains simultaneously.

3. It would require huge resources to perform computations before selecting the most appropriate node at a given time for each domain.

4. It co-locates the resources and its management by domain owners which puts additional load on the system.

It is worth nothing that the legacy dns system is unsuitable for Peerhosting model due to the following reasons:-

1. The content location is varying dynamically and hence it would not be able to support fast update propagation

2. Requires dedicated resources and its management by domain owners.

3. Extensive dependence on caching which is unsuitable as computations are performed in real time.

3.2 DNS Requirements

The system under consideration is domain name resolution service for Peerhosting platform. The key attributes of the desired solution are that the system should be scalable, fault tolerant and offer optimal query resolution time while providing dynamically managed, distributed and decentralised domain service to the Peerhosting platform. On a dns request, the system should be capable of making decisions in real time to select the most appropriate HP to redirect the client. This selection is performed using performance data fed by dns peer nodes in Peerhosting platform. A key point to be observed is that the Peerhosting dns network is different from the Peerhosting content network consisting of HP's. The dns peer network takes the responsibility of resolving domain names from the broker and make a decision on the HP to be served to a client request based on the statistical performance data from all HP's in possession of content for a unique content provider.

3.2.1 Primary Goals

The primary goals of the system are as follows:-

- It should be based on the peer to peer based technology for efficient load balancing to improve robustness against dos attacks. Each hosting provider (HP) contributes a node to the dns peer network to ensure a continuous resource pool of sufficient bandwidth, CPU and storage. Similar to the Peerhosting model, the resource contribution by individual HP's will be logged and billed to the content provider.
- It should provide domain name resolution service to users without having the need to modify the existing client resolvers. The system should be transparent to the user.
- Only designated nodes (HPs) and content providers should be given authoritative access to their resource records in the system.
- The system should prevent insertion by unauthorised nodes and prevent duplication of resource records for a unique domain by blocking new inserts for an already inserted domain.
- The Hosting provider servers should be able to insert their individual performance statistics regularly for the domains they serve to the nodes which are responsible for storing corresponding dns resource records through a designated gateway node.
- Each dns peer node is independently capable of performing calculations using the received data to return the most appropriate result (IP address of most appropriate HP to serve the content). It reduces centralised computation by allowing embarrassingly parallel computations to be performed on individual nodes. A resource record set (RRset) is a list of records containing different IP addresses for a particular domain. The nodes receive a domain name as a request and it replies by selecting the most appropriate IP address among the RRset after performing relevant computations.
- Only designated nodes are capable of accepting dns queries from clients.
- Only designated nodes are capable of accepting statistical information from Peerhosting HPs or from the Broker to prevent unauthorised access
- The system provides load balancing mechanisms in two places. First by distribution of dns resolution responsibility on multiple nodes through replication of objects on the peer network. Second, resolving the domain to different IP-addresses based on the performance statistics returned by the domain servers.
- Evaluation of the system to analyse its feasibility in view of low query latency requirements.

3.2.2 Secondary Goals

The secondary goals of the system are as follows:-

- Replication and proactive caching mechanisms could be implemented to reduce the number of steps to resolve a query while reducing query latency.
- Ensure a constant replication level for a resource record. For example, in the case of node failure responsibility of failed node should be transferred to other available nodes dynamically.
- The system should be fault tolerant and should function normally providing consistent uninterrupted service to users.
- The system attains a maximum latency target of 1 second for any dns query resolution. Ideal latency requirements are of the scale of 100-500ms.
- The system must provide high availability and must be robust against high churn.

3.2.3 Stretch Goal

- The system decouples ownership authority from the responsibility of resolving dns requests by placing in additional security mechanisms such as public key cryptography mechanisms to sign RRsets.
- The system implements dns protocol and interfaces with the legacy dns resolvers to receive dns resolution request.
- An efficient scheduling algorithm which selects hosting servers(IP-address) as an answer to the dns request.
- Monitoring Engine: Monitoring Engine is an essential component of the system as it provides a mechanism which calculates and aggregates the resources offered by all participants (dns peers) to host a domain name for a content provider.
- To establish trust among content providers about system integrity, security mechanisms needs to be devised. One such mechanism is the independence given to the content providers to select trusted peers to resolve resource records for their domain.

3.3 Proposed System Design

In this section we will discuss our application design in view of stated requirements. We use the underlying architecture of the past [8] in addition to additional components such as replication strategies, caching policies, performance monitoring system. We have created a performance engine on each peer through which a peer performs calculation on the statistical data it receives from the HP servers serving the content to select the most appropriate IP address. Also we have introduced a host of new features which are discussed below, in view of the specific requirements of the Peerhosting network.

We will discuss the proposed dns design with detailed explanations of various system components.

3.3.1 System Overview

Peerhosting dns comprises of geographically distributed nodes with self organisation capabilities to create a peer to peer network. We assume that each participating entity (HP) on the Peerhosting network will dedicate a server with pre-agreed resources to the Peerhosting dns network to create a very large pool of resources for storing dns resource records (RRset) and resolving the stored domains. Content providers are automatically registered in the Peerhosting dns system upon their registration with the Broker. The proposed dns system provides query resolution services to clients using dns protocol and thus existing resolvers do not require any change.

Peerhosting dns separates query resolution from name space management. Unlike legacy dns system, Peerhosting dns is based on a flat structure where each domain name is a separate entity which is stored on distinct peer nodes. The dns resource records in the system are inserted by the broker in agreement with the content provider. To host their dns resource records, the content provider may select the peers manually or delegate this responsibility to the broker. In both cases, a resource record set (RRset) is prepared by the Broker which consists of IP address of all HPs who are responsible for hosting content for a domain name. A hash of RRSset domain is taken using the hash algorithm, which forms the objectId. Using objectId, RRset is inserted in the Peerhosting dns and its responsibility delegated to a node (Home Node). The home node is the node whose nodeId is closest to the ObjectId in the system. The home node is responsible for managing replication and ensures constant replication factor for the domain. Figure 3.3 shows a client directing a dns request to the Broker. The Broker performs two key functions of inserting the dns records and forwarding the query

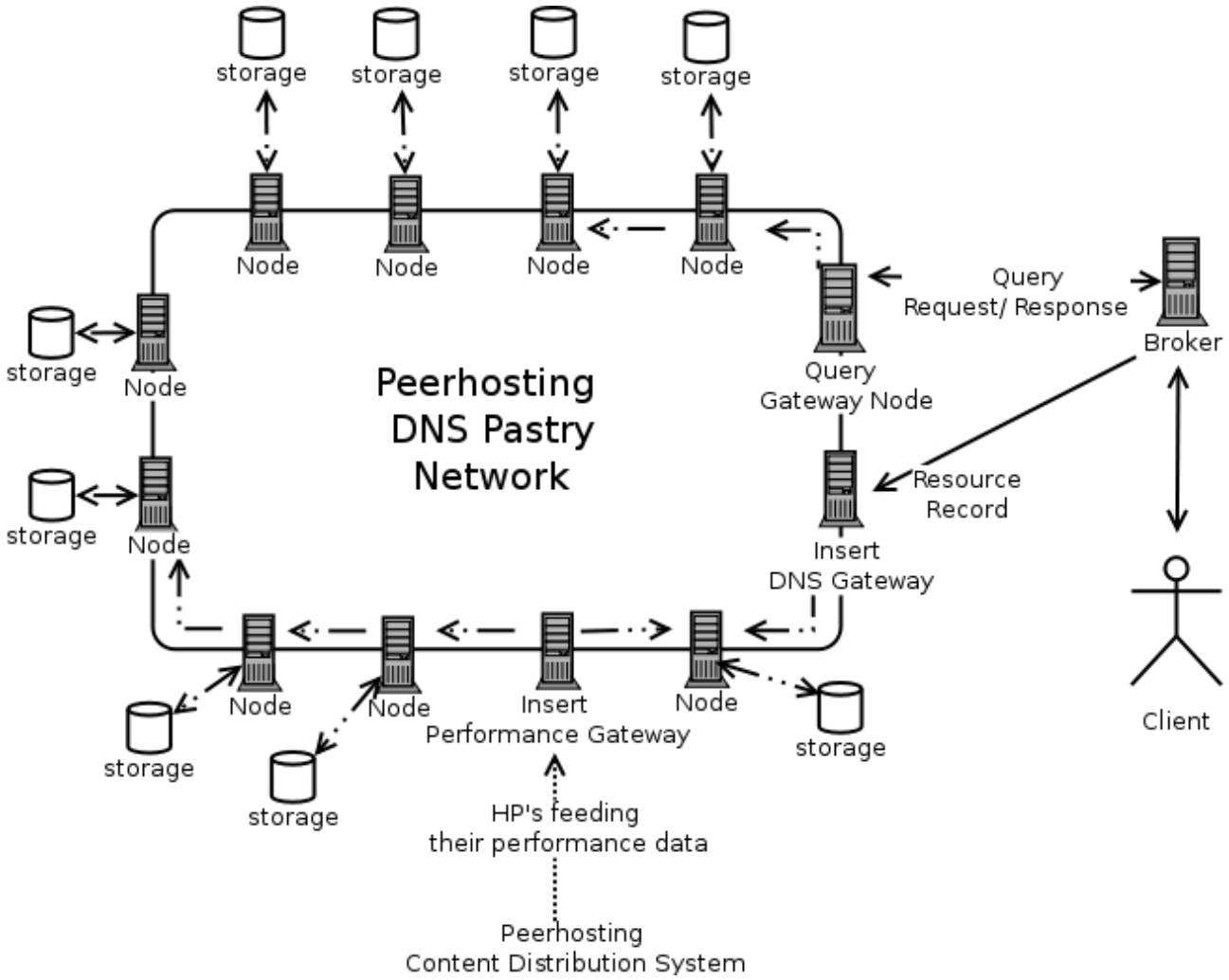

Figure 3.2: Peerhosting DNS Architecture

to the Peerhosting dns system. An Insert Performance Gateway is also shown to be receiving performance information from HPs in Peerhosting platform.

3.3.2 System Architecture and components

This section describes the system design. It also explains various components and mechanisms in the Peerhosting dns.
The system consists of the following main components:

1. Insert Gateway Node
   Insert Gateway Node is a node on the Peerhosting DNS network which is authorised to interact with the Broker to insert resource records in the system.

2. Insert Performance Node
   Insert Performance Gateway Node is a node on the Peerhosting DNS network which is authorised to interact with the HPs to insert performance information for a resource record (domain) in the domains home node.

3. Query Gateway Node
   Query Gateway Node is a node on the Peerhosting DNS network which is authorised to interact with clients to resolve domains hosted with Peerhosting platform.

Insert Gateway Node

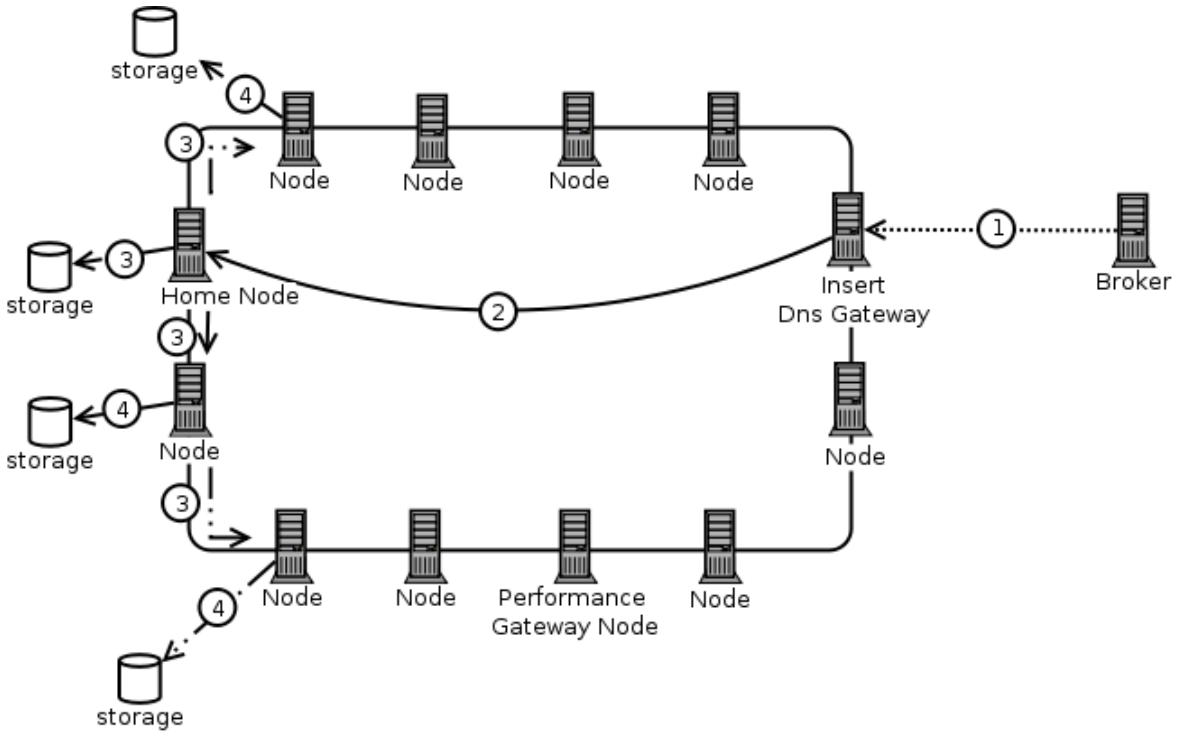

Figure 3.3: Insert dns resource record with replication factor of four

Insert Gateway node on the Peerhosting dns system interfaces with the Broker to insert resource records. Insert Gateway node receives resource records from the Broker and computes a 160 bit hash for each entry (resource record). The computed hash of an object is referred to as the objectId and it is used to locate the appropriate home node of the resource record. A domain object is delegated to a live node whose nodeId is closest to the ObjectId, this node is referred as Home Node. When the Broker inserts the domain object with nodeId as the key, the object is routed to the Home Node. The home node either accepts or denies responsibility for a domain, for either operation it sends a acknowledgement message to the Broker. Once Home Node accepts responsibility of a domain, it is the responsibility of the home node to resolve the domain and manage domain's object replication. An insert operation is performed in the following sequence:

- Step 1
  Broker initiates an insert operation.
- Step 2
  Gateway node computes ObjectId(key) for the domain using a hash algorithm . It then prepares a insert message before transmitting it on the dns system.
- Step 3
  The insert message is delivered to the home node. The home node stores the domain object in local store and becomes the authoritative node for the domain.
- Step 4
  Home node reads the domain's replication factor(k) and initiates an insert operation on k-1 nodes. These nodes are selected from the leaf set of the home node. All the selected nodes either accept or deny the domain object responsibility and conform their action to the home node.
- Step 5
  Home node returns a message to the Insert Gateway confirming the insert operation with a log of successful and failed inserts. The log contains a list of NodeIds on which the object is replicated.

Insert Performance Gateway

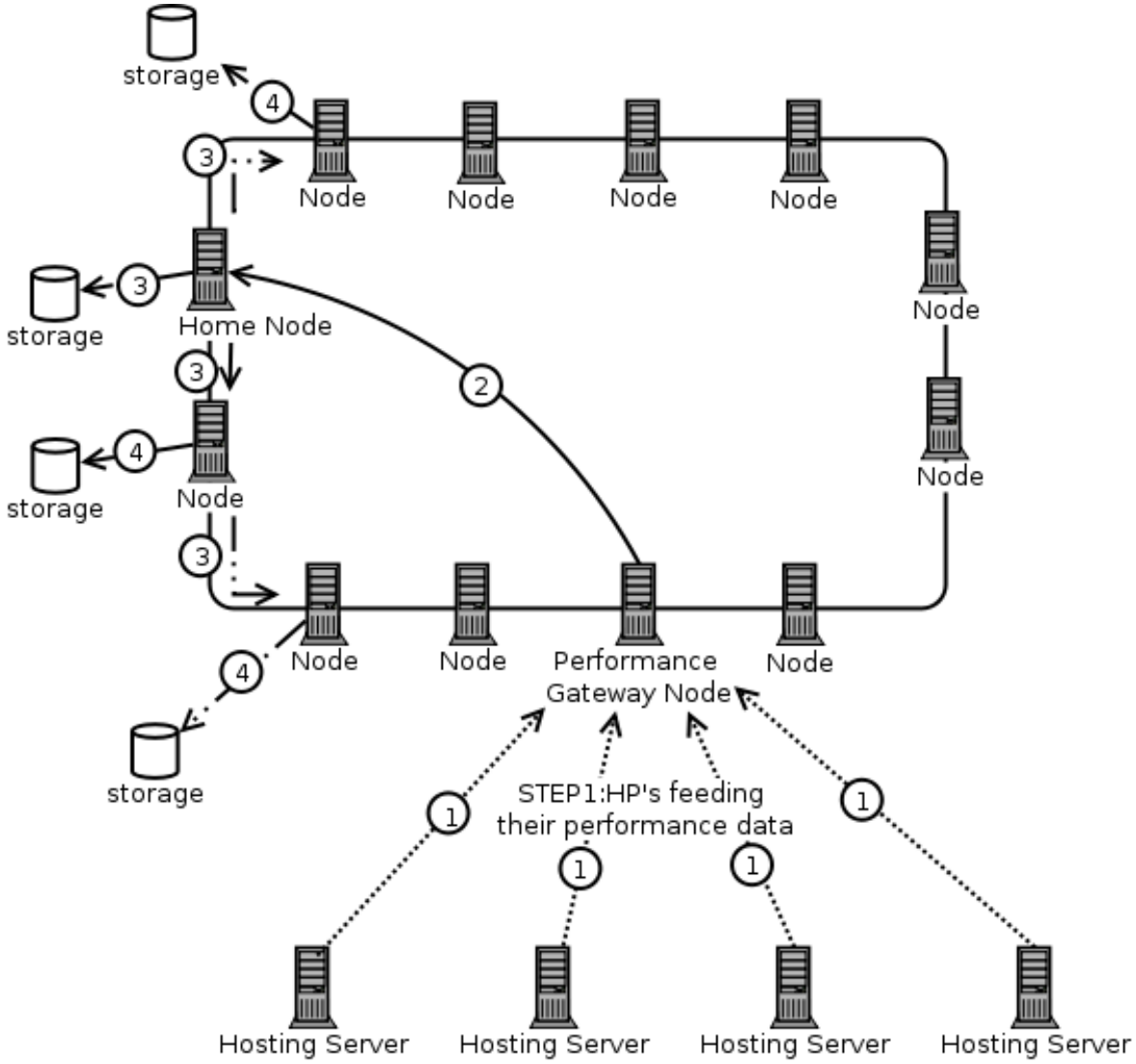

Figure 3.4: Insert performance statistics from HPs in Peerhosting DNS

Insert Performance gateway interfaces between the Broker and Hosting provider servers(HP servers) on the Peerhosting content system. It receives and inserts the performance statistical information from the HP servers in the dns peer network. It also ensures that the performance information reaches only the authorized home node by sending this data directly to the home node. If it does not receive confirmation message from any of the node, it re-tries to insert the data after fixed time intervals before it reports the problem to the Broker. An insert operation is performed in the following sequence:

- Step 1
  HP Servers initiate an insert operation by sending the performance data to the Performance Gateway node.
- Step 2
  Gateway node computes ObjectId for the domain and locates the home node of the domain.
- Step 3
  The Gateway node sends an insert message to the home node.

- Step 4
  The home node locates the resource record for the given domain to retrieve the list of replica nodes. It then sends an insert message containing the performance data directly to the replica nodes.

- Step 5
  All the replica nodes send the confirmation message back to the Home node which then conforms it to the Performance Gateway node.

Query Gateway

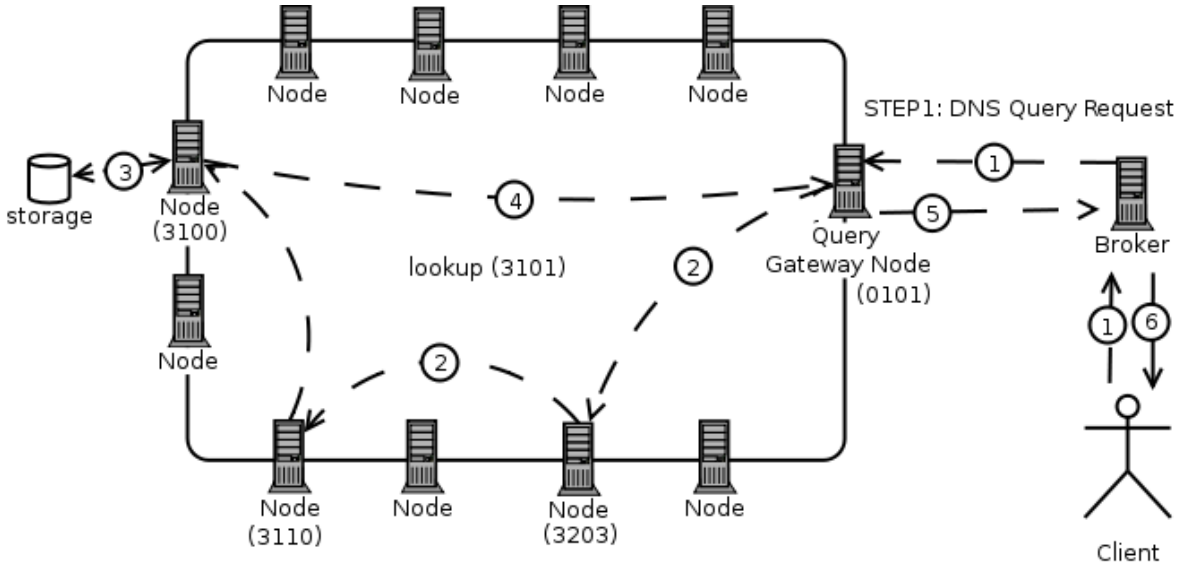

Figure 3.5: DNS query resolution operation

Query gateway is one of the most critical components in the system. Multiple query gateway nodes are created in diverse geographic locations to improve system robustness against failure of a single gateway due to bandwidth and throughput constraints. This is to distribute the query resolution load while reducing lookup times. To further improve the QOS, a Broker may multicast a query to multiple Query Gateways and sends a reply to the client with the result from the Gateway which responds the earliest. For each query it calculates the hash of the domain (ObjectId) and initiates a lookup message, using the ObjectId as the destination. The destination node upon receiving the query message, sends the resolved IP address to the Query Gateway which then it to the Broker.

The dns query resolution involves the following steps:

- Step 1
  Clients initiate a dns query directed towards the Peerhosting Broker. The Peerhosting Broker redirects the query to single of multiple Query Gateway nodes.

- Step 2
  A Query Gateway node then computes the objectId and generates a query for that objectId. The query traverses multiple hops before it reaches its destination. In the given example, it takes three hops before the query(objectId 3101) reaches its destination(3100) as the nodeId of the source node was completely different from the objectId.

- Step 3
  Query is received by the destination node, which has the resource record for the queried domain. The destination node resolves the query by accessing the persistent storage and retrieves the appropriate IP for the queried domain.

- Step 4
  Target node then sends the resolved query back to the Query Gateway node.
- Step 5
  Query Gateway node returns the resolved IP address to the Broker.

3.3.3 Introduction to Past

[8]Past is a large, scalable, self organising, persistent storage application built on top of pastry. It is essentially a peer network connected to the internet where each peer contributes its memory, bandwidth and persistent storage to create a very large common pool of resources.

Usage of Pastry

Past uses pastry protocol for routing messages between a pair of nodes. Therefore we provide a brief overview of pastry protocol and its use in Peerhosting DNS. Pastry is a highly scalable, fault resilient and self organising second generation peer to peer routing algorithm. It is a multi-hop routing protocol in which the number of hops to find an object is O(log(N)) in the worst case. Pastry network ensures network locality and minimises distance traversed by a message between any two peers. Upon node initialisation, each participating node is allocated a unique 128 bit identifier (nodeId). The nodeId is assigned randomly which ensures uniform node probability in the network space. A heuristic ensures that among all the live nodes in a node's routing table, the message is routed to a node whose objectId is closest to the originating node in respect to the proximity metric. Each node selects the next hop by selecting the node with closest matching prefix (longer prefix match by one digit) to the objectId from all the entries in the routing table which stores the information about the immediate neighbouring nodes in the nodeId space. This mechanism is unable to produce globally shortest routes but simulations show that the average distance travelled by a message is only 50 percent greater than the actual distance between source and destination[24].

Node addition

Here we briefly talk about the procedure using which nodes join the peer dns network and what happens when new nodes join.

When a new node joins a network, it connects to any bootstrap node which allocates this node a unique nodeId. To initialise its routing table, leaf set and neighbourhood set, the node contacts a nearby node A and asks it to forward a special message to the destination node X. This message is routed to any node Z whose nodeId is numerically closest to X^2. X fetches the neighbourhood set from A, leaf set from Z and constructs the routing table by obtaining the ith row of routing table along route from A to Z.

The other affect of a new node joining the network is that its nodeId may be closer to an ObjectId than the existing nodeIds on which an object is already stored by the home node. This creates an imbalance in the network and has to be corrected to ensure successful dns resolutions while maintaing a constant replication factor. To correct this imbalance the home node redistributes the object responsibility by revoking the object from old node and storing it on the new node. This is how home node ensures that a object is replicated on only the k closest nodes with nodeIds closest to the objectId.

Node failure

Here we will briefly talk about the procedure by which other nodes in the peer dns network are informed of possible node failures.

Usually, all the nodes in Pastry dns network are made aware of their immediate neighbouring nodes by periodic exchange of keep-alive messages (beacons). These beacons are sent to all nodes in the node leaf set. Often a set of neighbouring peers share the same leaflet nodes thus when a node disappears, there is no keep alive beacon from that node and this node is then removed from the leaf sets of all nodes. Also the nodes periodically share their leaflet information and this expedites the whole process of removing dead nodes.

3.3.4 Object Identifier Generation

In the current system, we use a hash table (SHA-1) to generate a 160 bit unique objectId (key) for any domain object before inserting the object in the dns peer network. Node selection can either be manual or automatic based on the content providers requirements. In case of automatic assignment objectId is used as a key in selecting a home node(HN) to delegate responsibility of hosting and resolving a domain name in the peer network. But such approach does not provide transparency in the system about the delegation of home node as objectIds are computed as a hash of the domain name. The objective is that the content providers are given the option to select the authoritative Home Node responsible for resolving their domain name based on parameters such as trusted HP's, their QOS history, available bandwidth resources, their geographic location and reputation. .

In the Peerhosting dns system it is also crucial to instil a sense of trust, security and confidence in the Peerhosting platform. This can be achieved if content providers are allowed to select a home node trusted by them. It will assure content providers that there content is being managed by a known node trusted by them. For this purpose we propose to develop an system through which objectIds can be customised in a way which allows manual selection of home node. This can be achieved by generating objectId with value extremely close to the desired nodeId so that the domain name's responsibility is automatically transferred to the desired peer. However, if the content provider requires that the dns RRset be replicated across the network on multiple peers for better services, then the home node will decide the replica peers locally. Thus content providers will only have limited control in distributing objects on the peer network ensuring fair sharing of resource while preventing excessive load on well known peers. Selection of Home Node peer by content provider and automatic selection of other peers for replica distribution will ensure fair utilisation of resources on all peers while providing fair gains to all the participants.

3.3.5 Storage Load Balancing and Replication

In any content storage system, there is a need to ensure fair storage utilisation and global even distribution of objects. Therefore we have used the Past mechanism for ensuring better load distribution and maintaining a constant replication factor in the peer system. The replication factor k can be defined either by the Broker in agreement with the content provider or it can be dynamically selected based on the demand of the object. For dynamic selection of the replication factor, the possibility of using Beehive replication layer can be looked into. In case of manual selection, the replication factor is specified during the insert operation. When an object is inserted in the dns system with object replication factor as k, the administrative dns node in the peer system replicates the dns object among k-1 leaf set nodes with closest nodeIds for the objectId. It is also important that the desired replication factor is achieved at all times by the administrative dns node even if the number of closest nodes in the leaf set is less than k. In case a selected leaf set node is unable to store a replica as directed by a home node. The home node selects the next closest node (k+1) from the leaf set using the same algorithm as used earlier. This mechanism is called replica diversion.

Conditions for replica diversion

In this section we will discuss the conditions for initiating and controlling replica diversion which are following:

1. When an insert operation is performed with a replication factor k, the insert messages reaches the node with the closest nodeId to the objectId in terms of scalar proximity metric. This node becomes the authoritative node A and stores the object. If the authoritative node A is able to find k-1 available nodes for storing the object from its leaf set then there is no need for initiating replica diversion. All the contacted leaf set nodes store the object and return a conformation message to the authoritative node.

2. In case one of the authoritative leaf set node B is unable to store the object. In that case B selects a node C from its leaf set with nodeId closest to the objectId and stores the object on that node while creating a pointer entry for redirecting any query message to C. B also creates a pointer entry on another node D from its leaf set having k+1th closest nodeId with the objectId. D ensures availability of content on C even in case of failure of B.

Replica diversion is an important characteristic as it provides a mechanism for local storage space balancing and enables graceful degradation of the system as it reaches maximum storage limit. However this mechanism has following disadvantages:

1. Overhead in an insert operation for each replica diversion is two RPC's. And an additional entry has to be created at both D and B both pointing to C.

2. Query overhead increases by one hop only when a query request is received by B. If a query request is received by any other node holding the object, the query latency is normal.

3.3.6 Storage Mechanism

Each node consists of two local object stores, one of which is persistent and the other non persistent (stores object in local program memory). The storage capacity of the Persistent store is set to 20 mega bytes even though it can be input as an argument as well. Persistent object store is called resource record store while the non persistent object store is called performance store. These stores are accessed by access objects whose prime function is to manage and implement synchronisation. We implement the performance store as a non persistent memory as nodes act on data for each insert to conclude a new serving HP as the answer and thus there is no need to maintain old data for a very long time.

Performance object store

As the name suggests, the performance store is used exclusively for storing the statistical performance information about the serving HPs. Data type of a performance store object consists of a domain name, corresponding system IP(IP of serving HP), system load at the serving HP, data instance (used for version control of performance data). Once a node receives fresh performance information from a serving HP (system IP)for a particular domain, the node flushes previously held data. For example, if a node receives performance data for "www.google.com" domain from a serving HP at IP address (12.1.23.4) with performance instance of 14, the node will remove all previous performance data for "www.google.com" domain for IP 12.1.23.4. This is done to prevent the memory store from growing too big and to reduce object store lookup time.

Resource record object store

Resource record store is used for storing the complete resource record sets. Data type of resource record object consists of a domain name, an answer field and two lists, one containing IP addresses and the other null. The first list is optional and stores a list of IP addresses which serve this domain permanently. Other list stores IP addresses (serving HPs) which are added dynamically based on the output of the Performance Engine.

3.3.7 Caching

Caching is an important tool in reducing the client fetch latencies (proportional to number of hops traversed in underlying pastry network) and in improving the query throughput of the system by load distribution on the nodes along the query path. However, we have a system in which the IP address of HPs (resolved dns queries) returned by the nodes change based on intermediate / metadata (performance data) after fixed time intervals. Therefore in such a system caching is not a viable option unless the refresh rate of performance data is several hours. In the Peerhosting content hosting system the expected churn is high, new nodes join frequently and the older ones discarded. And thus it is expected that the home node receives fresh performance data along with new node information once or twice an hour or may be more frequently. Therefore as of now we have disabled caching in our system.

3.3.8 Performance Engine

In the legacy dns system the resolution of domain names is done to a single or multiple dedicated static systems responsible for hosting content of a particular content provider. In such mechanism the responsibility of hosting the content is delegated or controlled directly by the content provider itself. Thus the content providers remain static or rarely change. Whereas Peerhosting content distribution system is based on a peer network which works by aggregating, allocating and delegating the resources contributed by individual nodes (HPs). In this system, individual peers (HPs) can pull out of the Peerhosting platform if they run out of available resources. This coupled with genuine node failures could lead to a extremely high churn in the system. Due to the high churn the HPs responsible for hosting the content for a domain may change very rapidly in time. The content is re-located on new nodes based on content providers HP preference, system load and available resources at a peer node, content demand, geographical location of HP's and other network conditions.

It shall be observed that in Peerhosting the content is hosted by multiple HP's as a single peer may not be able to cater to demand of all the clients. Therefore, selection of the appropriate peer(HP) among the multiple HPs by the dns client becomes a critical task to achieve even load distribution, fair resource sharing and improved availability of content. To create a stable system and improve the content availability we create a Performance Engine (PE) on each node in Peerhosting DNS. The Performance Engine analyses and calculates the load factor on each node responsible for distribution of the content for a particular domain before resolving the dns query to an appropriate HP. All the HP's serving content for a domain must broadcast their current system information such as system load, available memory, available storage, bandwidth consumption to the 'Performance Gateway' node in the dns peer system. The 'Performance Gateway' node then locates the home node for a domain and transfers the performance information to it. One apparent disadvantage of this system is that the HP have to broadcast their system performance to unknown hosts which is a potential security threat. One possible way of countering this disadvantage is by encrypting the content at the the originating node and decrypted at the Home node of the domain in the dns peer system. There is still a problem of applying a private key pair mechanism in this setup.

The performance data is received by the home node which broadcasts this data to all the replica nodes responsible for resolving dns queries for this domain. The raw data is broadcast with the objective of allowing the nodes to perform embarrassingly parallel computations while reducing system load on a single HP. It should be noted that each node is independent and performs computations to reach to independent conclusions. When a node receives new performance data it activates the PE with performance data as inputs. The PE then performs analysis to select the most appropriate node(IP address) to be resolved for a dns query for a fixed time interval(time interval is greater than the frequency of receiving updates from HPs serving content). At the moment, the performance engine performs selection based only on the system load parameter though there is scope for adding in more parameters.

There can be an argument that performing calculation before answering each query may not be a feasible option as it may impact the overall system performance by increasing the query delay significantly. To counter this, we have designed the peer nodes such that each node initiates the Performance Engine only when it receives a fresh update of performance information and not before answering each query. The execution of Performance Engine is a non blocking operation so the system is available and resolving queries even when it is running.

Chapter 4

Implementation

In this chapter, we will discuss the libraries and the models used for development of the Peerhosting dns. User scenarios will be explained in this chapter for better understanding of the system.

4.1 Language and environment

The ideal target operating system for supporting our implementation of Peerhosting DNS is Debian based Linux even though its implementation is platform independent as it executes on Java Platform (JRE6 SDK). All the programs have also been tested on JRE 1.7.0-icedtea verison which is an open source implementation of Java. Peerhosting DNS has been tested on Ubuntu Linux OS, windows XP and Fedeora 8 machines. It should run well on most other operating systems which have a rightly configured Java environment. Insert Performance clients which monitor and report average CPU load may not work correctly on some Windows based systems as the library used does not ensure accurate system performance on all Windows platforms. Netbeans IDE version 6.5 was used during implementation and testing. As the development involved a distributed environment, initially the system was tested on a single machine with all nodes being created on a single JVM. Later the system was tested on a local area network and Planetlabs environment.

4.2 System Implementation

System implementation is based on design requirements presented in the previous chapter. Focus of implementation is to get a feature complete working prototype of the proposed system and thus the proposed design is simply taken as a reference point.

The System is built using the iterative software approach with greater focus on scalability, fault tolerance and performance. The following features have been implemented in the first version of the software:

- Pastry network with ability to deliver messages based prefix routing.
- Insert dns resource records mechanism.
- Insert average system load from hosting providers through a Gateway to home node.
- Performance Engine to select the most appropriate HP to serve to a dns request.
- Query resolution mechanism.
- Mechanism to monitor and transfer the HP system load to the Input Performance Gateway.
- Mechanism to generate dns queries.

Peerhosting dns servers are layered on top of [7]Freepastry library version 2.1 which is the most stable version of freepastry. Freepastry libraries and key API's have been discussed in section 4.2.3 extensively. Each Peerhosting dns peer (daemon) implements dns resolver which is event driven and non blocking.

4.2.1 Continuations

Throughout the system, continuations have been used to enable the possibility of non-blocking operations. A continuation is similar to a call-back and a listener in Java and it is used to handle network latency and other types of Input Output that may block or take a significant amount of time. It is specifically useful in insert and lookup queries as the program will not be blocked, allowing it to initiate multiple requests simultaneously. Similar functionality can be achieved using a lot of threads with synchronisation and thread locking mechanisms. But this would increase the system complexity and may result in possible system degradation. The following code illustrates a continuation command.

```
public interface Continuation
{
  /**
   * This method gets called when the requested result gets available.
   *
   * @param result= the result object received for the request made.
   */
  public void receiveResult(Object result)
  {
        // code to execute using the result
  };
  /**
   * This method gets called when an exception occurs
   * Due to the request made.
   *
   * @param result=  the exception which occurs.
   */
  public void receiveException(Exception result)
  {
        // code for handling the exception
  };
}
```

4.2.2 Environment

Freepastry provides a class called 'Environment' which initialises a new node after reading pastry routing protocol parameters such as leaf set size, maintenance frequencies, number of handles stored per routing table, maximum size of a message from a file named 'freepastry.params'. These parameters can be stated explicitly by either altering the freepastry.params file or by creating a new '.params' file. A new node is required to create an environment to initialise its routing table. Some of the key features of Environment class which have been used in the system are:

- Logging
  Environment provides a simple standardised logging mechanism through which system logs are generated for debugging and monitoring purposes. We have used this class for generating logs on individual nodes used for debugging purposes.

- Parameters
  Parameters are necessary to initialise the program. Parameters can either be provided as arguments or be entered in a params file as default value. To initialise the node we read the local IP, local bindport, bootstrap IP , bootstrap port parameters as arguments and read routing protocol information from the params file in the default package.

- SelectorManager
  Selector manager is a single non daemon thread which manages timer events and network input output. Selector manager is used for managing multiple threads for improved thread synchronisation. It also reduces complexity of implementing synchronisation manually.

- TimeSource
  TimeSource is an alternate library for measuring time on each node. It functions similar to "System.currentTimeMillis()" and provides a virtual clock in Free Pastrys event simulator. Another benefit of using this library is that the system is not dependent on correct configuration of NTP servers. Thus this library has been used for testing system performance by measuring time difference between operation request and operation response.

4.2.3 Usage of Pastry

The system has been developed largely using the Pastry Api's with major modifications in its code base for implementing the extra functionality to fulfil the design requirements. It should be mentioned that even though the proposed design suggests the use of UDP as the underlying network layer for communication, the current network layer implementation is based on use of TCP/IP sockets as provided by the Freepastry library. UDP messages are only used by nodes to exchange keep alive messages to maintain their leaf sets . The future versions would support dns protocol which uses UDP as its network layer. This change will make the system more robust while improving query throughput. We now discuss some of the APIs that have been used for implementing key features:

- nodeId= pastryInit(credentials, application)
  This is used to create a new environment through which a node can either join an existing ring or start its own ring. If no credentials are entered the node starts a new ring else we need to provide the bootstrap IP address for the current node to join an existing ring. This API also generates and allocates a unique nodeId to the new node using a random nodeId generator algorithm. The node object is passed as an argument to the application which specifies the node behaviour on various events.

- route (msg, key) This is used to route the given message to the node whose nodeId is closest to the key. Key is represented by the objectId which is generated a hash of the domain name.

- deliver(msg, key) This message is called when the message is received by the destination node (nodeId closest to the ObjectId). Several types of messages can be received by a node and each type of message requires distinct operations to be performed. A brief explanation of each message is provided below:

  - InsertMessage
    When a node receives this message, it initiates a store operation to store the dns resource record in the persistent store.

  - InsertPerformanceMessage
    When a node receives this message, it initiates the Performance Engine to calculate the most appropriate node for the given domain. It then associates the output from the Performance Engine with the dns resource record available in the persistent storage.

  - LookupMessage
    On receiving a Lookup message, the node looks for the objectId (key) in its persistent storage, retrieves the corresponding object and then responds with the object.
  - LookupHandlesMessage
    On receiving a lookuphandlesMesssage, a node looks up in the local storage for the corresponding object and then locates all other nodes responsible for storing this object. It then prepares a list of all the nodeIds before replying to the source with this list.

- forward(msg, key, nextId)
  This method is called by a intermediate pastry node before it passes the message to the next node towards the destination. This method can be used to trace the object path or to change the contents of message. This could also be used to count the number of hops before a query reaches the destination node.

- newLeafs(leafSet)
  This method is called when a change in leaf set of a node is observed. Through this method the application(Home node) is informed of any node failures in the leaf set. This is used to ensure a constant replication factor of an object. For example, if a node on which object had been replicated goes down, a new replica node can be selected for the object.

- objectId= Insert(object name, owner credentials, replication factor ,object)
  This method generates a 160 bit unique objectIdId for the object name by parsing its name, owner credentials and random salt through the SHA-1 hashing algorithm. Before the file is inserted, the objectId is checked for collisions to prevent multiple insertions with the same objectId. This objectId is then used for inserting the object on multiple peers depending on the replication factor of the object.

- file= Lookup(object name)
  This method computes the SHA-1 hash of the object name to construct an objectId for the domain name. Using the objectId a lookup operation is performed to retrieve the resolved IP address of the domain name from any replica node(out of possible k replicas).

4.2.4 Resource record selection

Resource records are selected on the basis of performance data inserted by the HP servers in the dynamically. The resource records are selected on each performance data insertion only, thereby ensuring fast query resolving times. Currently the performance engine implements a simplistic algorithm which compares the system load of all the serving HPs for a unique domain and finds the HP with the lowest system load. Whenever a target node receives a performance insert operation, it runs the performance engine which outputs the IP address of the most appropriate HP serving that domain. This value is then inserted in the answer field of the resource record set of the corresponding domain name.

4.3 System Components

In Peerhosting dns each peer is assigned a distinct role with different responsibilities. Every Peerhosting dns peer creates an environment and implements a stand-alone daemon running on each node which is accessible via TCP/IP requests from the master daemon. The daemon is event driven and is implemented as a non blocking master process with multiple slave processes. Slave processes may either be blocking or non-blocking depending on the type of the request received. For example, in case of an insert request by the remote client or an insert gateway, the slave process is blocking whereas for query requests from remote clients or peers the slave process is non-blocking.

Creating a new node requires creating a local environment. A node can either create a new pastry ring or join an existing pastry ring by connecting with the bootstrap node using the Pastry socket class to synchronise its PastryNodeFactory and a NodeIdFactory for generation of nodeIds and objectIds respectively. This is how a pastry node initialises its routing table, nodeId and initial leaf set values. The following code illustrates node initialisation.

```
public CreateNode(int bindport, InetSocketAddress bootaddress,
Environment env) throws Exception
{
    // Generates Random NodeId
    NodeIdFactory Factory = new RandomNodeIdFactory(env);

    // constructs a PastryNodeFactory using rice.pastry.socket
    PastryNodeFactory factory
    = new SocketPastryNodeFactory(Factory, bindport, env);

    // Creates a new node
    PastryNode node = factory.newNode();

    // Passes the newly created node to the application
    Application app= new Application(node);

    // Boots the node into the ring
    node.boot(bootaddress);
  }
```

Now, we will discuss some of the system components implemented for testing the system performance.

4.3.1 Bootstrap Node

As the name suggests a Bootstrap node's responsibility on the peer network is to help other nodes to launch into the dns peer network. This node also contributes its resources such as memory, storage, CPU to the effective running of the peer dns system.

To initiate a standard bootstrap node, the following arguments need to be passed to the program "Bootstrap Node" with local bind port matching with bootstrap port address and bootstrap IP address matching with bind IP address.

argument[0]= local bind Port (Local Port of the application)
argument[1]= Bootstrap IP-Address
argument[2]= Bootstrap Port Address
argument[3]= Bind IP-Address (Local IP-Address of the local Machine)

4.3.2 Standard Node

A standard node is simply a contributory node on the network with no specific responsibility other than offering its resources such as memory, storage and CPU to the effective running of the peer dns system. Such a node can be launched in the network only by establishing a connection with Bootstrap node. Arguments for initiating such a node are similar to the Bootstrap node, only difference being that the Bootstrap IP and bootstrap port address are different from local bind-port and local bind IP-address.

argument[0]= local bind Port (Local Port for the application)
argument[1]= Bootstrap IP-Address
argument[2]= Bootstrap Port Address
argument[3]= Bind IP-Address (IP-Address of the local Machine)

4.3.3 Insert Gateway

Insert Gateway peer is implemented as a non-blocking ServerSocket thread listening on a pre-defined port on the standard node. The ServerSocket thread is listening for insert requests from the Peerhosting Broker. The ServerSocket opens a non blocking new thread for each request which then passes an insert request to the most appropriate neighbouring node from the routing table (prefix matching) by opening a TCP/IP socket. The message hops from node to node until the destination is reached using prefix routing. The target node initiates a blocking call on the node's local storage to store the resource record. The following code explains the implementation of insert continuation command.

```
// Content object
final PastContent myContent = new MyPastContent(id,domainName,
        ipAddress,instance,performance);

// inserts an myContent object
     pastApp.insert(myContent, new Continuation<Boolean[], Exception>()
      {
        // the result is an Array of Boolean for each insert
        public void receiveResult(Boolean[] results)
        {
          int numSuccessfulStores = 0;
          for (int ctr = 0; ctr < results.length; ctr++)
          {
            if (results[ctr].booleanValue())
              numSuccessfulStores++;
          }
          long newtime=env.getTimeSource().currentTimeMillis();

          //measures time taken for an insert operation
          long timetaken=newtime-time;
        }

// Prints the result object received.
        public void receiveException(Exception result)
        {
          System.out.println("Error storing "+result);
        }
  });
```

To launch an Insert Gateway node on the dns peer network, the gateway node must connect with the bootstrap node using bootstrap IP and port address. Only one extra argument needs to be passed to "InsertGateway.jar" to launch this node into the peer dns network which is the local port address at which the server is listening for insert requests. The arguments which need to be passed are mentioned below:

argument[0]= local bind Port (Local Port of the application)
argument[1]= Bootstrap IP-Address
argument[2]= Bootstrap Port Address
argument[3]= Bind IP-Address (IP-Address of the local Machine)
argument[4]= ServerSocket Port (Local Port at which the server is listening for 'Insert Data')

4.3.4 Insert Performance Gateway

For every Insert Performance request this node fetches the nodeHandles of all the peers who are in possession of resource record object for the domain whose performance value is to be inserted. It then opens a direct socket connection with each peer and feeds the latest data directly to the target nodes. The cost of this operation is one step. The insert operation method of Insert Performance Gateway is similar to the Insert Gateway node, difference being the type of objects passed in the insert operation and the messages are transmitted directly to the target nodes. The list of arguments to launch an Insert Performance Node is as follows:

argument[0]= local bind Port (Local Port of the application)
argument[1]= Bootstrap IP-Address
argument[2]= Bootstrap Port Address
argument[3]= Bind IP-Address (IP-Address of the local Machine)
argument[4]= ServerSocket Port (Local Port at which the server is listening for 'Insert Performance Data')

4.3.5 Query Gateway

Query Gateway peer is implemented as a non-blocking ServerSocket thread listening on a pre-defined port on the standard node. The ServerSocket thread is listening for insert requests from the Peerhosting Broker. The ServerSocket opens a non blocking new thread for each request which then passes a lookup message to the next neighbouring node from the routing table by opening a TCP/IP socket. The message hops from node to node until the message reaches its destination. The target node accesses the resource record store to fetch the 'answer' value for the given domain name which is then returned directly to the source node (Query Gateway Node). The following code explains the implementation of lookup continuation command.

```
pastApp.lookup(lookupKey, new Continuation<PastContent, Exception>() {
  public void receiveResult(PastContent result)
  {
      MyPastContent content=(MyPastContent) result;
      //Resolved IP Address for the domain
    String ipaddress[]=content.getIpaddress();

  }

  public void receiveException(Exception result) {
    System.out.println("Error looking up "+lookupKey);
    result.printStackTrace();
  }
});
```

To launch a Query Gateway node on the dns peer network, the gateway node must connect with the bootstrap node using bootstrap IP and port address. An extra argument needs to be passed mentioning the port to create an external ServerSocket to launch this node into the peer dns network. The arguments which need to be passed are mentioned below.

argument[0]= local bind Port (Local Port of the application)
argument[1]= Bootstrap IP-Address
argument[2]= Bootstrap Port Address
argument[3]= Bind IP-Address (IP-Address of the local Machine)
argument[4]= ServerSocket Port (Local Port at which the server is listening for Queries from external clients)

4.3.6 Insert client

The insert client fetches resource records from a repository and creates a new thread for each entry. Each resource record contains a domain name, list of IP address, an empty list, empty answer value and a replication factor. Each thread then opens a TCP/IP socket with the Insert Gateway and passes the resource record to it. The client waits for an insert confirmation and records success or failure in the repository. The Insert Client can be initiated using the following arguments.

argument[0]= IP Address of Insert Gateway Node
argument[1]= Port Address of Insert Gateway Node

4.3.7 Insert Performance client

Insert performance client program monitors the average system load on the host it is running. This program also retrieves the system name, system architecture and various other system specific information using the java.lang.management.OperatingSystemMXBean library. This program initiates two types of threads. The first one initiates a program which consumes CPU cycles by calculating primes of large random numbers. This program is designed to generate a random CPU consumption pattern thus simulating a real HP on Peerhosting platform. The second thread monitors the system performance after fixed intervals and reports it the Insert Performance Gateway client using TCP/IP socket connection. The Insert Performance client can be initiated using the following arguments.

argument[0]= IP Address of Insert Performance Gateway Node
argument[1]= Port Address of Insert Performance Gateway Node
argument[2]= Domain Name which is being served by the serving HP
argument[3]= IP-Address of the serving HP
argument[4]= Frequency of inserting performance data (in ms)

4.3.8 Query client

The query client program simulates the Broker and produces queries for resolving the domain name specified in the arguments. The program sends the queries to the Query Gateway node by creating a socket connection on a new thread for each request. This program fetches the resolved IP for the domain from the Query Gateway node. This program also computes the time taken to resolve each query while calculating the average time for all queries performed. The Query client can be initiated using the following arguments.

argument[0]= IP Address of Insert Gateway Node
argument[1]= Port Address of Insert Gateway Node
argument[2]= Domain Name to be resolved
argument[3]= Query frequency (in ms)

Chapter 5

Evaluation

5.1 Introduction to evaluation

Evaluation is an integral part of this project, and was declared as a key goal in the design section. The frequent task of storing, computing and resolving the domain name with the most appropriate serving HP system is important. Also thus system should be robust against node failures and perform well on a much larger scale. The design of Peerhosting DNS minimises reliance on single nodes by distributing various responsibilities to multiple nodes located in diverse geographic locations.

On a global network such as the Internet, the network conditions and resources available at different nodes are hugely varying with time. Thus it increases the performance uncertainty and makes the system susceptible to arbitrary failures due to failed nodes. The other possible effects could be system performance degradation due to high network latency due to network clogging thereby resulting in slow access to some nodes on the network.

This section describes the experimental methods that we use to determine the performance of Peerhosting DNS. We then investigate and evaluate the performance achieved while comparing it with desired performance.

5.2 Evaluation system configuration

For the purpose of experimentation of Peerhosting DNS system, all the tests were carried out in two environments with completely different network, hardware and software configurations. This is done so with an aim to establish feasibility of Peerhosting DNS network in different environments and network configurations. It must be observed that the tests were conducted in networks containing nodes ranging from 10-500. The two environments are:

1. Local area network (10-500 nodes)
2. PlanetLabs (10-30 nodes)

The configuration of both of these environments has been described as follows:

5.2.1 software configuration

Local area network

All the test/evaluation machines (nodes) in the LAN network were configured with an installation of Windows XP Professional service Pack 3 and were updated with all security patches. Various university authorised third party software was installed on each machine, though it was ensured that none of it was active while testing the

system.
To ensure that idle tasks were not running, various tasks such as Windows Indexing service and windows restore facility were disabled. However, these services do enable automatically from time to time, there impact on system performance is not significant as it is not likely to affect network connections in any way.

PlanetLab

All the test/evaluation machines on the PlanetLab network were configured with a fresh installation of Fedora 8 virtual machine. Since virtual machines are allocated on each node, the underlying physical resources and network bandwidth is shared among many users and processes. The virtual machine allocated does not run any application software other than the Peerhosting DNS software. Java Runtime version 1.7.0-icedtea (open source) was installed on all the test machines for execution of Peerhosting DNS application.

5.2.2 Hardware configuration

Local area network

The local area network nodes featured the following hardware components:

- Intel Core2Duo processor, clocked at 2.4 GHz
- 4GB DDR2 800MHz RAM
- 1 TByte

PlanetLab

The PlanetLab nodes featured the following components

- Intel Xeon 30x0 2.4Ghz (Mcore/Core2Duo)
- 4GB DDR2 RAM
- 1 TByte

5.2.3 Network Configuration

Local area network

It must also be observed that all the nodes in this environment were connected through the same local area network and nodes had access to Internet. However, the Internet connectivity was not used while testing the system. At the time of testing, the local area network traffic was observed to be ideal and churn was negligible.

PlanetLab

[5]PlanetLab is a global research network which supports development and testing of new network services such as peer to peer systems, network mapping services and distributed storage etc. PlanetLab consists of 1120 nodes located across 510 distinct geographical locations.

It must also be observed that all the nodes in this environment had Internet connectivity and were not connected by a local network. All the communication between these nodes occurred over the internet on TCP/IP connections. This is a relatively unstable network in comparison with local area network as each node in PlanetLab shares its physical and network resources among many users and processes. Nodes often go down for maintenance purposes and some nodes may not be reachable due to network clogging or other system failures. Thus churn is higher than ideal.

5.3 Design of evaluation system

5.3.1 Introduction to design of evaluation system

For testing and evaluating the performance of Peerhosting DNS, programs were customised to add capability to measure the performance metric such as the query delay. We perform tests with a view of measuring query latency as it is the single biggest determinant in establishing the feasibility of our design approach. Freepastry offers libraries which allow simulation of a large network containing thousands of nodes in single JVM. It is also possible to change network topologies using the simulator but it has two drawbacks. Firstly, the simulator does not account for message loss, varying bandwidth and varying latency. Secondly, simulator does not simulate other factors that a comprehensive simulator such as ns-2 simulates. Due to the above drawbacks and the inability of the simulator to recreate a volatile network with varying latencies, measuring query delay using a simulator would not yield accurate results.

5.3.2 Evaluation of Peerhosting DNS

To evaluate the Peerhosting DNS system, changes were made in Insert Gateway and Query Gateway program to reflect the delay in performing each request received from the corresponding client programs. The storage mechanism on the Query Gateway nodes was disabled thus preventing local storage and retrieval of requested DNS records. New functionality was added to allow Gateway programs to receive requests from clients on specific predefined ports. It should be observed that the delay is calculated as the time taken by the program to perform a complete operation as observed by the Gateway node. Insert DNS client and query client simulate Broker while insert performance client simulate serving HPs in the Peerhosting infrastructure. Clients are capable of generating query resolution requests, insert DNS record, insert performance statistic requests at varying rates (requests/second).

5.4 Implementation of evaluation system

The evaluation system comprises of basic test infrastructure and specialised infrastructure for both LAN and PlanetLab testing.

5.4.1 Basic Test Infrastructure

The basic test infrastructure remains same for both LAN and PlanetLab testing. To run the tests, the evaluation system executes 'Insert DNS client' to insert DNS resource records in the system. For each test, the number of domains inserted equals ($2\sqrt{N}$+3) where N is the total number of nodes in the network. The first 2*N domains inserted had a replication factor of 10 to maximise the DNS resource density on nodes to better account for time taken to locate resource records. The other three domain values were inserted with different replication factors of zero, five and ten, which were then queried to observe delay. For each test suite different domain names were queried to change the query pattern in the network.

5.4.2 Query Latency

Query delay is the most significant factor in establishing the feasibility of our design approach. Therefore we measure the query delay at the Query Gateway nodes by querying domain names in peer network containing 100-500 nodes.

LAN

To setup the evaluation infrastructure, all the jar files namely "QueryGateway.jar", "InsertGateway.jar, "Insert-GatewayPerformance.jar", "InsertDnsClient.jar", "InsertPerformanceClient.jar," and "QueryClient.jar" were

copied manually on each host. Fresh JRE was installed on all the machines and the required jar files were executed using appropriate arguments. In case of LAN, query delay test infrastructure consists of four pairs of query clients and Query Gateways out of which query delay performance is observed on two pairs while the other two pairs (unobserved) are used to generate network traffic. The two unobserved pairs query random domains to generate random traffic patterns in the system while the two observed pairs query domains as directed by the user. The clients for both observed and unobserved pairs generate queries at the rate of 100 queries per second. Tests are performed for network sizes of 100 nodes, 250 nodes and 500 nodes. It should be observed that only 10 physical machines were used for actual testing and multiple nodes were simulated on each machine to increase the network size. For each domain, delay was observed on two Query Gateways (pairs). The first 50 values from both the gateways were taken, summed and averaged to determine mean and median score.

PlanetLab

To setup the evaluation infrastructure on PlanetLab, 311 nodes with sufficient free bandwidth were selected and added to the slice "tcd4" provided by the principle investigator (PI). Then a private public key pair was generated and public key added to all the nodes in the slice. Private Key was used to authenticate the user and access the remote nodes on the slice. A script was written which installed openJDK on all the host machines. All the jar files namely "QueryGateway.jar", "InsertGateway.jar,"InsertGatewayPerformance.jar", "InsertDnsClient.jar", "InsertPerformanceClient.jar" and "QueryClient.jar" were copied manually one by one in the home folder of each node. Fresh JRE was installed on all the machines and the required jar files were executed using appropriate arguments.

In case of PlanetLab testing, query delay test infrastructure consists of three pairs of query clients and Query Gateways out of which query delay performance is observed using two pairs while the other pair (unobserved) is used to generate network traffic. The unobserved pair queries random domains to generate random traffic pattern in the system while the two observed pairs query domains as directed by the user. The clients for both observed and unobserved pairs generate queries at the rate of single query per second. Tests are performed for network size of 10 and 35 nodes. It should be observed that for PlanetLab testing different physical machines were used for each experiment. The values were then added and averaged to determine mean and median scores.

5.5 Evaluation results

In this section we discuss and elaborate our findings for various tests performed. It should be noted that the terms query delay, lookup latency and query latency have been used interchangeably and they refer to the time taken by a Query Gateway node to resolve a domain name.

5.5.1 LAN test with no replication

Introduction to this test

The objective of this test to establish and compare the query delay statistics observed by a Query Gateway while querying a domain name which has not been replicated at all. The delay is observed for a peer network consisting of 100, 250 and 500 nodes. The test is performed on nodes in a local area network and maximum steps required to resolve a query should not be greater than 3, 2 and 2 hops for networks containing 500, 250 and 100 nodes respectively. We have arrived at the above figures using the formula $log_{2^b}N$ where b equals 4 and N equals the number of nodes in the pastry network.

Results of this test

The results can be seen in the figure 5.1

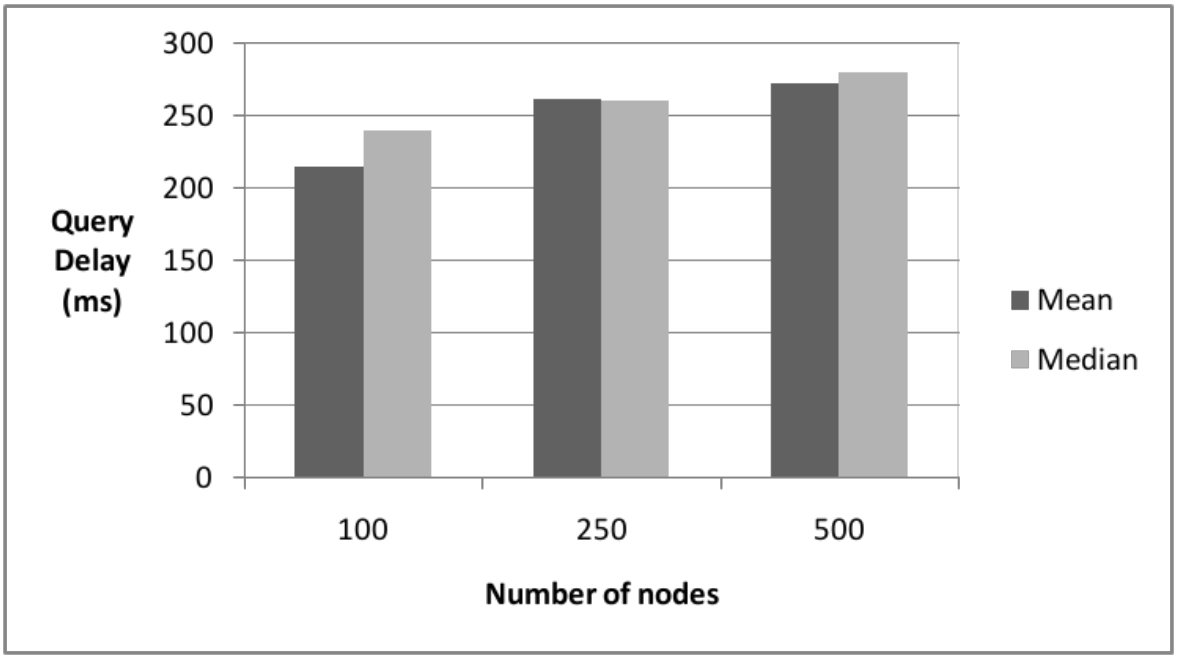

Figure 5.1: Query delay observed when replication factor (k) equals zero

Evaluation of Results

The results show degradation in average query delay while increasing network size. Even though, there is a greater query delay difference between network of 100 and 250 nodes than between network of 250 and 500 nodes, the actual variation is in range of 70ms which is not insignificant. The results also show that in a network of 500 nodes, 50 percent of the queries were answered in less than 280ms.

5.5.2 LAN test with five replicas

Introduction to this test

The objective of this test to establish and compare the query delay statistics observed by a Query Gateway while querying a domain name which has been replicated at five nodes including the home node. Due to replication the maximum steps required to resolve a query should be less than 3, 2 and 2 hops for networks containing 500, 250 and 100 nodes respectively. The test has been performed on a LAN therefore the network traffic was not bursty and the churn was negligible.

Results of this test

The results can be seen in the figure 5.2

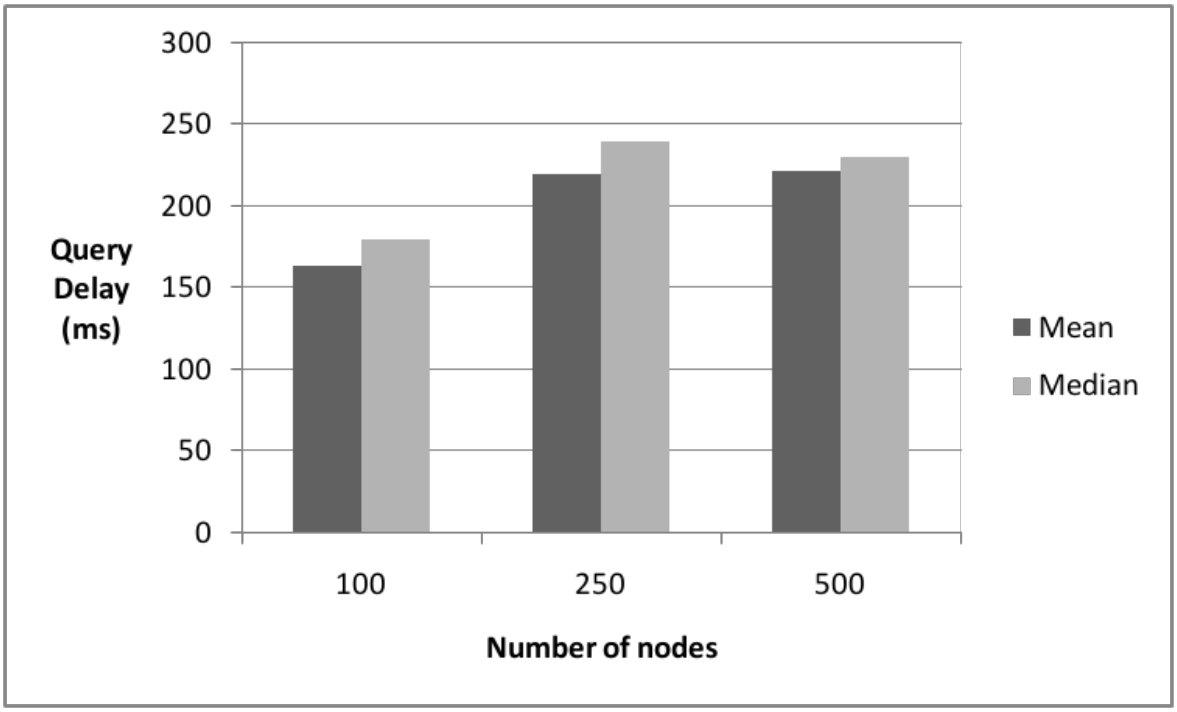

Figure 5.2: Query delay observed when replication factor (k) equals five

Evaluation of Results

The result show that replication of a DNS resource record has a significant impact on query delay times. The query delay observed for resolving replicated domains is less than delay observed for domains which were not replicated at all. This is true for network containing 100, 250 and 500 nodes. A significant difference in average delay is observed in case of 100 node network in comparison to 250 and 500 node network.In this test scenario The results for resolving a domain name with some replication follow a similar pattern to the domain resolution for no replication, though with improved query resolution time.

5.5.3 LAN test with ten replicas

Introduction to this test

This test is performed to observe the effect of excessively high replication on query resolution time. The domains with a replication factor of 10 are queried by Query clients and the delay is observed at Query Gateway. Due to the excessive replication the maximum steps required to resolve a query should be less than 3, 2 and 2 hops for networks containing 500, 250 and 100 nodes respectively. This test has been performed on a LAN therefore the network traffic was not bursty and the churn was negligible.

Results of this test

The results can be seen in the figure 5.3

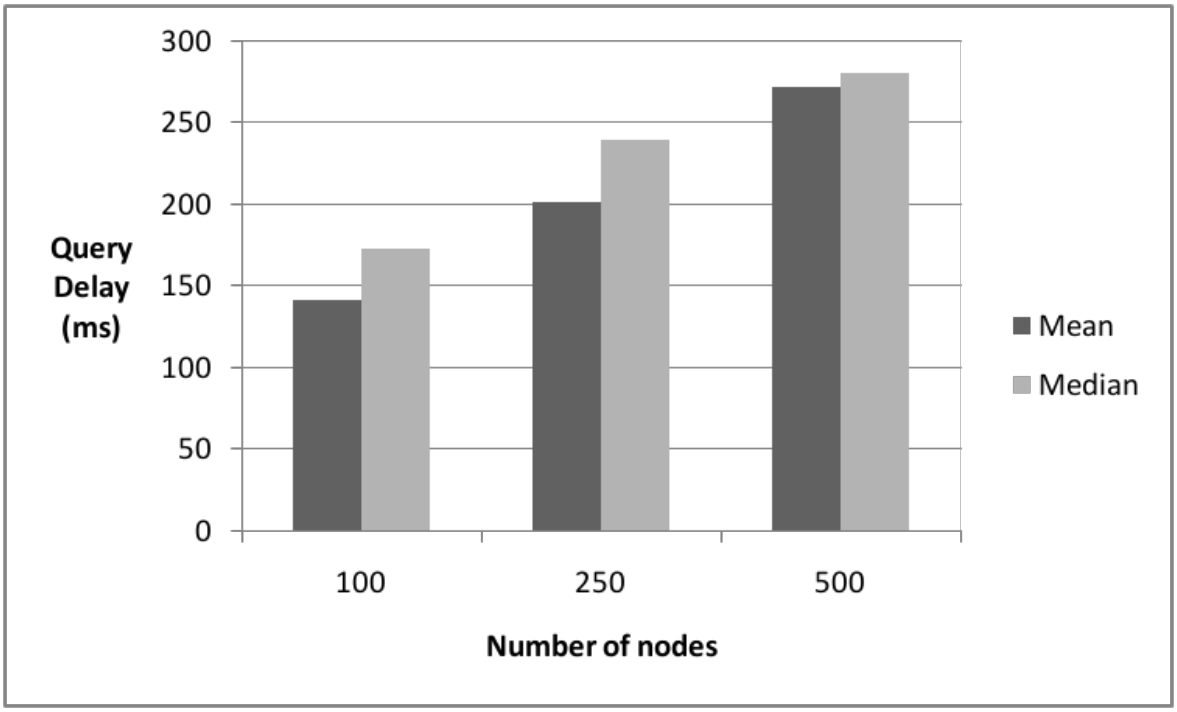

Figure 5.3: Query delay observed when replication factor (k) equals ten

Evaluation of Results

The results show that the effect of replication is more dominant in small peer to peer networks. The effect reduces rapidly as the network size increases. The delay observed for resolving a resource record with a very high replication factor of 10 is comparable to records which are replicated at only five nodes. It can also be seen that the median is greater than the average delay for all network sizes, which point towards an imbalance in query resolution times for the same domain. This shows that increasing the replication factor can result in highly varying query delay. This observation is not surprising due to the random routing paths in pastry network.

5.5.4 Delay range comparison

Here we elaborate and compare the results obtained from performing the above tests. Figure 5.4 shows a comparison table showing the query delay range observed in above tests.

Replication Level	Query Latency Range (ms) for 100 Nodes	Query Latency Range (ms) for 250 Nodes	Query Latency Range (ms) for 500 Nodes
0	120-301	100-421	211-490
5	60-291	110-361	100-381
10	41-310	80-321	170-411

Figure 5.4: Query delay comparison observed on local area network

Peer network containing 100 nodes

It can be seen that in a peer network containing hundred nodes, the minimum query latency for domains with no replication is twice the minimum latency observed for domains with replication level of five or ten even though their maximum delay is comparable. Excessive replication of object does not have any significant impact on query delay in a peer network of this size.

Peer network containing 250 nodes

In a peer network containing two hundred fifty nodes, the query delay range remains similar to a network containing 100 nodes albeit with an upward shift in minimum and maximum delay times. The minimum and maximum observed query latencies reduce marginally as we increase the replication factor of the queried domain. A huge overlap is observed in the delay range across various replication factors even though fifty percent queries have been resolved in less than 250 ms irrespective of the replication factor.

Peer network containing 500 nodes

The query delay observed in a 500 node network is significantly higher than a network containing 100 or 250 nodes for domains which are not replicated at all. This difference is less apparent in case of DNS records which have been replicated across multiple domains even though the maximum observed delay is higher than the network containing 100 or 250 nodes. Surprisingly, the minimum query latency in case of objects replicated across ten nodes is higher than objects which are replicated across less number of nodes. The query delay median for different domains with distinct replication factors of 0, 5 and 10 is consistent and fifty percent of queries get resolved in less than 300ms.

5.5.5 PlanetLab test with no replicas

Introduction to this test

This test was conducted with an objective of finding latency metrics in the PlanetLab environment consisting of ten and thirty five nodes. The query latency for a DNS record with replication factor of zero is observed at Query Gateway program. The maximum steps required to resolve a query should not be greater than two hops for both networks containing 10 and 35 nodes. We have arrived at the above figures using the formula $log_{2^b}N$ where b equals 4 and N equals size of network, while adding one hop for the target node to reply to the Query Gateway node.

Results of this test

The results can be seen in figure 5.5

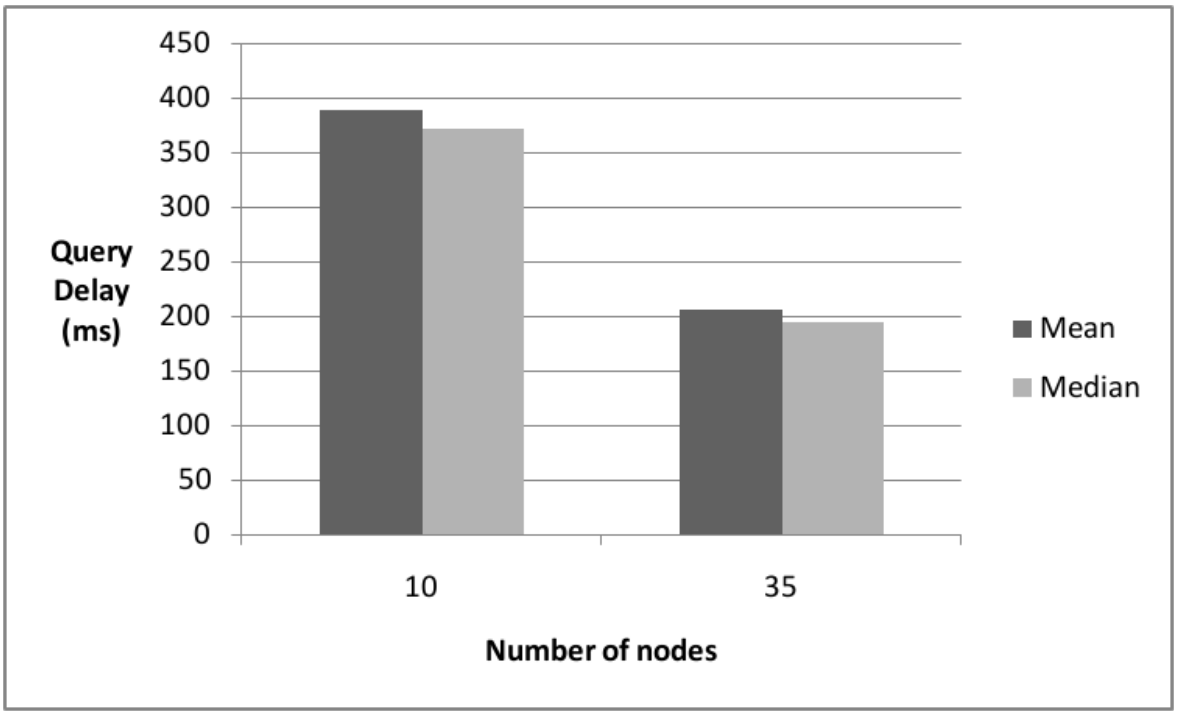

Figure 5.5: Query delay observed when replication factor (k) equals zero on PlanetLab nodes

Evaluation of Results

The results show that the fifty percent of the queries were resolved in less than 400ms and 200ms for networks consisting of ten nodes and thirty five nodes respectively. According to the calculations performed, queries in both the network should not take more than two steps in resolving the DNS request. The huge difference in the observed query latencies underscores the effect of external factors in query resolution times. It is again highlighted that these networks were created using different node sets.

5.5.6 PlanetLab test with five replicas

Introduction to this test

This test was conducted with an objective of finding latency metrics in the PlanetLab environment consisting of ten and thirty five nodes when the queried domain objects were replicated across five nodes including the home node. The query delay is observed at the query gateway node. The maximum of two steps are required to resolve a query in both networks containing 10 and 35 nodes. We have arrived at the above figures using the formula $log_{2^b}N$ where b equals 4 while adding one hop for the target node to reply to the Query Gateway node.

Results of this test

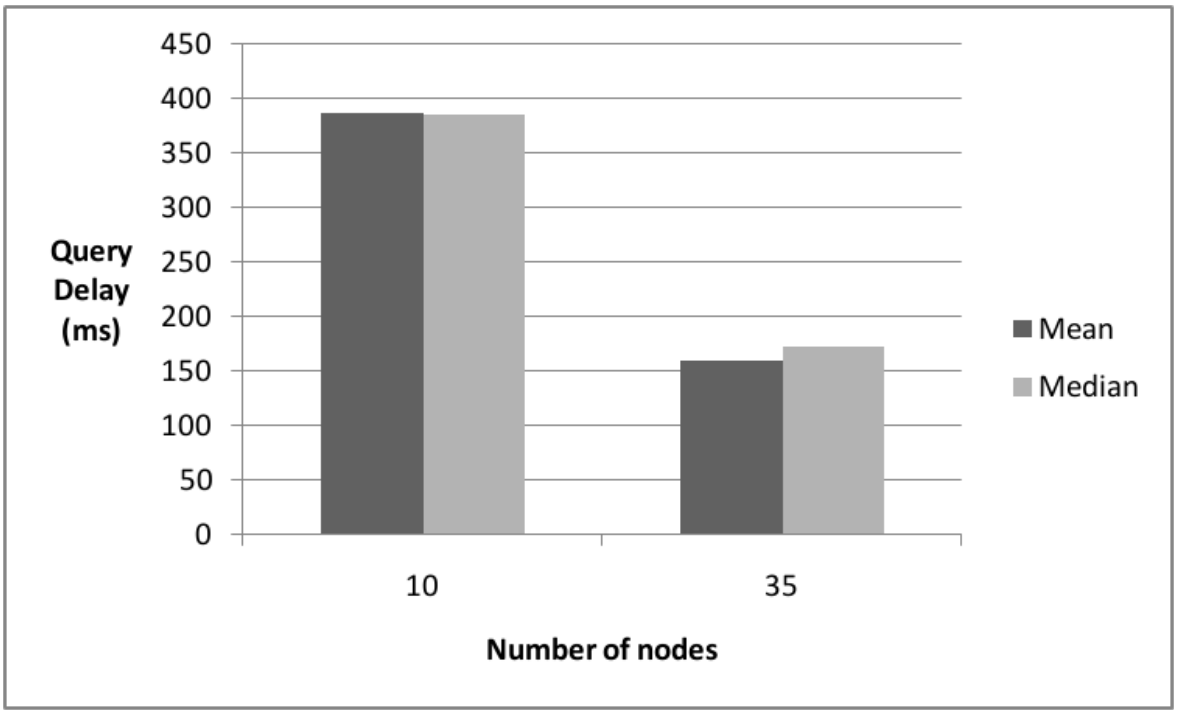

Figure 5.6: Query delay observed when replication factor(k) equals five on PlanetLab nodes

Evaluation of Results

The results show that the fifty percent of the queries were resolved in less than 400ms and 170ms for networks consisting of ten nodes and thirty five nodes respectively. According to the calculations performed, queries in both the network should not take more than two steps in resolving the DNS request. The huge difference in the observed query latencies further underscores the effect of external factors in query resolution times. It is also highlighted that these networks were created using different node sets. Also it is observed that in small peer networks (10-35 nodes), the effect of replication is negligible, and the only advantage of replication would be to increase resource availability albeit at the cost of more management data on the network.

5.5.7 Comparison of delay range

Here we elaborate and compare the results obtained from performing the above tests. Figure 5.7 shows a comparison table showing the query delay range in a PlanetLab setup for object replication factor of zero and five.

Replication Level	Query Latency Range (ms) for 10 Nodes	Query Latency Range (ms) for 35 Nodes
0	351-516	123-336
5	347-496	41-281

Figure 5.7: comparison

Peer network containing 10 nodes

In a peer network containing ten nodes, the query delay for searching replicated DNS records is similar to searching non replicated DNS records. The effect of replication is observed is negligible.

Peer network containing 35 nodes

In a peer network containing thirty five nodes, the query delay for searching replicated DNS records is significantly lower in comparison to searching non replicated DNS records.

5.6 Evaluation conclusion

The results show that the time taken to resolve a domain does not exceed a mark of 550ms in any of the tests performed. The maximum average time required to resolve a query in a stable peer to peer network consisting of 500 nodes in a local area network setting does not exceed 300ms irrespective of object replication factor.

LAN conservative estimate

Our tests showed average delay without object replication to be no more than 260ms. This figure allows us to believe that even if we take a conservative estimate of delay for each hop, it should not greater than 130ms as the minimum number of steps taken would not be less than two. Computing the number of steps in a very large pastry network using the formula $log_{2^b}N$ where b equals 4 and N equals 10,000, we find that the number of steps required to resolve a domain name is not greater than five. Based on the above calculation and the derived results we approximate the query delay in Peerhosting DNS system containing thousands of nodes. Even if we take a conservative estimate of the query delay time in a Peerhosting DNS network comprising of ten thousand nodes when objects are not replicated at all, the average query delay would not be greater than 130*5 which equals 650ms.

LAN actual estimate

We also take an estimate of the average lookup times in a large Peerhosting DNS network consisting of 10,000 nodes in accordance with the mean delay observed in a peer network of 500 nodes. Our tests showed average delay without object replication to be no more than 260ms. The maximum number of steps to resolve a query cannot be greater than three. Assuming that a query takes three steps to resolve, we come at a figure of 87ms for resolving a query. Now, we account the approximate average query resolution time for a Peerhosting DNS network consisting of ten thousand nodes. As calculated earlier, the maximum steps for resolving a domain in such network would not take more than 5 steps. If we now estimate the average lookup time for traversing five hops, we arrive at a figure of 433ms. Our approximation of average lookup time is very close to the [21] average latency offered by the current legacy DNS which is equal to 382ms.

PlanetLab performance

Our PlanetLab test results show that the average query delay observed is of range 200ms to 400ms irrespective of object replication levels for pastry network comprising of 10 - 35 nodes. The maximum steps for resolving a query in a network of ten to thirty five nodes is not greater than two and thus the average hop time can be calculated to be in range of 100ms - 200ms. For a network comprising of ten thousand nodes, a maximum of five steps would be need to resolve a domain name. Therefore the average query time can be approximated to be in range of 500ms to 1000ms. In addition to the approximated results, we believe that PlanetLab tests show a large variation due to varying external factors. Therefore we propose to perform further extensive tests using PlanetLab infrastructure to produce a better approximation of results which would serve as a foundation for system performance on larger networks.

Chapter 6

Conclusions

6.1 Introduction

In Chapter 5 we presented the evaluation of Peerhosting dns with respect to the query delay times. We also presented an approximation of the Peerhosting system performance for a very large peer network consisting of ten thousand nodes and then compared it with the average query resolution time in the legacy DNS. In this chapter we will draw conclusions from the system design and evaluation to explore the suitability of the solution. We will also discuss some possibilities which would enable to improve system performance and derive a better approximation of system performance.

6.2 Conclusions

We believe that pastry routing protocol has real potential of being used to create a peer to peer dns service especially in view of Peerhosting requirements. Evaluation of proposed solution shows that the computational cost at each node is reasonable and it does not impact the query delay significantly. The design has been successful in eliminating cost of selecting most appropriate answer for a domain request at a given node by separating query mechanism from execution of Performance Engine. The design also addresses the requirement of load distribution among multiple nodes by replicating objects and broadcasting performance data from serving HPs in Peerhosting content system to the designated replica nodes in peerhosting dns system.

A pastry based dns service is a likely possible solution for Peerhosting in view of the query times observed, as has been shown in Evaluation conclusion in Chapter 5. The results showed that the average query delay in a dns peer system containing five hundred nodes to less than 350ms. An approximation using the observed statistics was made to conclude that average query delay in a peer network consisting of 10,000 nodes would be in range of 433ms - 650ms in stable networks. We believe that delays of this range are acceptable for a dns solution and the query delay can be optimised using better system implementation and caching mechanisms for domains whose HPs are stable and change less frequently.

Overall we find that a peer to peer dns service based on pastry routing protocol is an acceptable solution in view of the average delay latency observed.

6.3 Future work

This section details areas of future work that are related to this dissertation.

6.3.1 Peerhosting integration

Peerhosting dns is designed to be an integral part of the Peerhosting project. In this thesis we designed and evaluated the feasibility of pastry based dns solution for Peerhosting. In our current implementation we have used client programs to simulate Peerhosting behaviour. And now that we have found the solution as a appropriate candidate we propose integration of our dns implementation with Peerhosting code base in the future.

6.3.2 PlanetLabs testing

The current system was not intended to run on Planetlab infrastructure. As a result, we were able to perform limited Planetlabs testing, therefore we propose to modify the Peerhosting dns code base to suit Planetlab test environment. We also propose to test the system thoroughly on PlanetLab infrastructure to obtain realistic results on larger networks.

6.3.3 DNS protocol

The current implementation does not conform to the dns protocol, and the resource records objects are implementation differently. Nodes communicate using TCP/IP socket connection instead of UDP messages as used in legacy DNS system. We intend to implement the DNS protocol in the next version of the Peerhosting dns.

6.3.4 Delegation authority

Control over delegation of home node for a domain name by its content provider is an integral requirement of Peerhosting project. We have proposed a rudimentary system to achieve this functionality. In the future, this aspect shall be researched further.

6.3.5 Security layer

The current implementation of Peerhosting dns lacks security features as proposed by DNSSEC. Freepastry enables generation of private public key pair for each object to provide some level of control to the object owner. But clearly this mechanism does not satisfy the needs of Peerhosting project. The control over a domain name is exercised regularly by several HPs hosting a domain to insert their performance statistics in the dns peer network. Therefore private key needs to be transferred regularly to the new nodes which may lead to authoritative access to dns record by unscrupulous HPs.

6.3.6 Caching layer

Initially, we intended to implement Beehive caching layer as an overlay on Peerhosting dns, but soon it we realised that it would be infeasible to implement any kind of caching mechanism until their is some stability in the systems serving a domain. This requires establishment of Broker infrastructure which would determine stable nodes eligible to use benefits of caching layer. This feature is dependent on the Peerhosting Infrastructure and will be investigated based on further development of Peerhosting platform.

6.3.7 Monitoring tools

Tools to manage and monitor the resources offered by individual nodes need to be created. As the Peerhosting dns works on resources contributed by independent HPs there is a need to account and monetise the resources offered by these nodes to enable fair sharing of resources while establishing a sense of trust among HPs in the system.

6.4 Lessons learned

This chapter concludes this dissertation. In this chapter we have presented the benefits and limitations of this approach and conclude that this approach is suitable to requirements of domain name service for Peerhosting system.

Bibliography

[1] P. Albitz and C. Liu. *DNS and Bind*. O'Reilly Media, 2001.

[2] Stephen Barrett. Peerhosting. A request for Comment for Peer Technology for Web Hosting Industry, May 2009.

[3] J. Byers, J. Considine, and M. Mitzenmacher. Simple load balancing for distributed hash tables. *Peer-to-Peer Systems II*, pages 80–87, 2003.

[4] M. Castro, P. Druschel, A.M. Kermarrec, and A.I.T. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications*, 20(8):1489–1499, 2002.

[5] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):12, 2003.

[6] R. Cox, A. Muthitacharoen, and R. Morris. Serving DNS using a peer-to-peer lookup service. *Peer-to-Peer Systems*, pages 155–165, 2002.

[7] P. Druschel, E. Engineer, R. Gil, Y.C. Hu, S. Iyer, A. Ladd, et al. FreePastry. *Software available at http://www. cs. rice. edu/CS/Systems/Pastry/FreePastry*.

[8] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proc. HotOS VIII*, pages 75–80. Citeseer, 2001.

[9] D. Eastlake. Secure domain name system (dns) dynamic update. *draft-ietf-dnssec-update2-00. txt, Transfinite Systems Company*, 1998.

[10] A. Friedlander, A. Mankin, W.D. Maughan, and S.D. Crocker. DNSSEC: a protocol toward securing the internet infrastructure. *Communications of the ACM*, 50(6):50, 2007.

[11] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. DNS Performance and the Effectiveness of Caching. *IEEE/ACM Transactions on Networking (TON)*, 10(5):589–603, 2002.

[12] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM, 1997.

[13] A. Kumar, J. Postel, C. Neuman, P. Danzig, and S. Miller. Common DNS implementation errors and suggested fixes. Technical report, RFC 1536, USC/Information Sciences Institute, 1993.

[14] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. *Peer-to-Peer Systems*, pages 53–65, 2002.

[15] J. Mirkovic and P. Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[16] P.V. Mockapetris and K.J. Dunlap. Development of the domain name system. *ACM SIGCOMM Computer Communication Review*, 25(1):112–122, 1995.

[17] M. Naor and U. Wieder. Novel architectures for P2P applications: the continuous-discrete approach. *ACM Transactions on Algorithms (TALG)*, 3(3):34, 2007.

[18] L.B. Pei. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM*, pages 126–134. Citeseer, 1998.

[19] CG Plaxton, R. Rajaraman, and AW Richa. Accessing nearby copies of replicated objects in a distributed environment. *Theory of Computing Systems*, 32(3):241–280, 1999.

[20] V. Ramasubramanian and E.G. Sirer. Beehive: Exploiting power law query distributions for O (1) lookup performance in peer to peer overlays. In *Symposium on Networked Systems Design and Implementation, San Francisco CA*. Citeseer, 2004.

[21] V. Ramasubramanian and E.G. Sirer. The design and implementation of a next generation name service for the Internet. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 331–342. ACM, 2004.

[22] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, page 172. ACM, 2001.

[23] E. Rescorla and N. Resonance. Introduction to Distributed Hash Tables. *IAB plenary, IETF*, 65.

[24] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, volume 11, pages 329–350. Citeseer, 2001.

[25] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, page 160. ACM, 2001.

[26] P. Thurrott. Microsoft suffers another DoS attack, 2001.

[27] PF Tsuchiya. The landmark hierarchy: a new hierarchy for routing in very large networks. In *Symposium proceedings on Communications architectures and protocols*, pages 35–42. ACM, 1988.

[28] G. Urdaneta, G. Pierre, and M. van Steen. A Survey of DHT Security Techniques. *ACM Computing Surveys*, 2009.

[29] C. Wills and H. Shang. The contribution of DNS lookup costs to web object retrieval. *Worcester Polytechnic Institute Technical Report TR-00-12*, 2000.

[30] B.Y. Zhao, J. Kubiatowicz, and A.D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *Computer*, 74:11–20, 2001.