Distributed Systems
Peer-to-Peer Systems

[2] P2P Systems - Goals and Definition

Goal: to enable sharing of data and resources on a very large scale by eliminating any requirement for separately-managed servers and their associated infrastructure.

Goal: to support useful distributed services and applications using data and computing resources present on the Internet in ever-increasing numbers.

- standard services scalability limited when all the hosts must be owned and managed by the single service provider,
- administration and fault recovery costs tend to dominate.

Peer-to-peer systems: applications that exploit resources available at the edges of the Internet - storage, cycles, content, human presence.

[3] P2P Systems - Features

Characteristics shared by the P2P systems:

- design ensures that each user contributes resources to the system,
- all the nodes in a P2P system have the same functional capabilities and responsibilities, although they may differ in the resources that they contribute,
- correctness of any operation does not depend on the existence of any centrally-administered systems,
- often designed to offer a limited degree of anonymity to the providers and users of resources,
- the key issues for P2P systems efficiency:
  - algorithms for data placement across many hosts and subsequent access to it,
  - key issues of these algorithms: workload balancing, ensuring availability without adding undue overheads.

[4] P2P Systems - History

Antecedents of P2P systems; distributed algorithms for placement or location of information; early Internet-based services with multi-server scalable and fault-tolerant architecture: DNS, Netnews/Usenet, classless inter-domain IP routing.

Potential for the deployment of P2P services emerged when a significant number of users had acquired **always-on**, **broadband** connections (around 1999 in USA).

Three generations of P2P systems:

1. launched by the **Napster** music exchange service,
2. **file-sharing applications** offering greater scalability, anonymity and fault tolerance (Freenet, Gnutella, Kazaa, BitTorrent).
3. P2P **middleware layers** for application-independent management of distributed resources (Pastry, Tapestry, CAN, Chord, Kademlia).

[5] P2P Middleware Introduction

Middleware platforms for distributed resources management:

- designed to place resources and to route messages to them on behalf of clients,
- relieve clients of decisions about placing resources and of holding resources address information,
- provide guarantee of delivery for requests in a bounded number of network hops,
- resources identified by **globally unique identifiers** (**GUIDs**), usually derived as a secure hash from resource's state,
- secure hashes make resources "self certifying", clients receiving a resource can check validity of the hash.
- inherently best suited to storage of immutable objects,
- usage for objects with dynamic state more challenging, usually addressed by addition of trusted servers for session management and identification.

[6] IP and P2P Overlay Routing (1)

– **Scale**:

IP: IPv4 limited to 232 addressable nodes (in IPv6 to 2128), addresses hierarchically structured and much of the space preallocated according to administrative requirements.

OR: The GUID name space very large and flat (>2128), allowing it to be much more fully occupied.

– **Load balancing**:

IP: Loads on routers are determined by network topology and associated traffic patterns.

OR: Object locations can be randomized and hence traffic patterns are divorced from the network topology.

– **Network dynamics (addition/deletion of objects/nodes)**:

IP: IP routing tables are updated asynchronously on a best-efforts basis with time constants on the order of 1 hour.

OR: Routing tables can be updated synchronously or asynchronously with fractions of a second delays.

[7] IP and P2P Overlay Routing (2)

– **Fault tolerance**:

IP: Redundancy is designed into the IP network by its managers, ensuring tolerance of a single router or network connectivity failure. n-fold replication is costly.

OR: Routes and object references can be replicated n-fold, ensuring tolerance of n failures of nodes or connections.

– **Target identification**:

IP: Each IP address maps to exactly one target node.

OR: Messages can be routed to the nearest replica of a target object.

– **Security and anonymity**:

IP: Addressing is only secure when all nodes are trusted. Anonymity for the owners of addresses is not achievable.

OR: Security can be achieved even in environments with limited trust. A limited degree of anonymity can be provided.

[8] **Distributed Computation (1)**

- work with the first personal computers at Xerox PARC showed the feasibility of performing loosely-coupled compute-intensive tasks by running background processes on about 100 computers linked by a local network,
- **Piranha/**Linda and adaptive parallelism,
- **SETI@home** - most widely known project
  - part of a wider project Search for Extra-Terrestrial Intelligence,
  - stream of data partitioned into 107-second work units, each of about 350KB,
  - each work distributed redundantly to 3-4 personal computers,
  - distribution and coordination handheld by a single server,
  - 3.91 million computers participated by August 2002, resulting in the processing of 221 million work units,
  - on average 27.36 teraflops of computational power,

[9] **Distributed Computation (2)**

- SETI@home didn't involved any communication or coordination between computers while processing the work units,
- although often recognized as P2P they are rather based on **client-server architecture**,
- **BOINC** – Berkeley Open Infrastructure for Network Computing.

Similar scientific tasks:

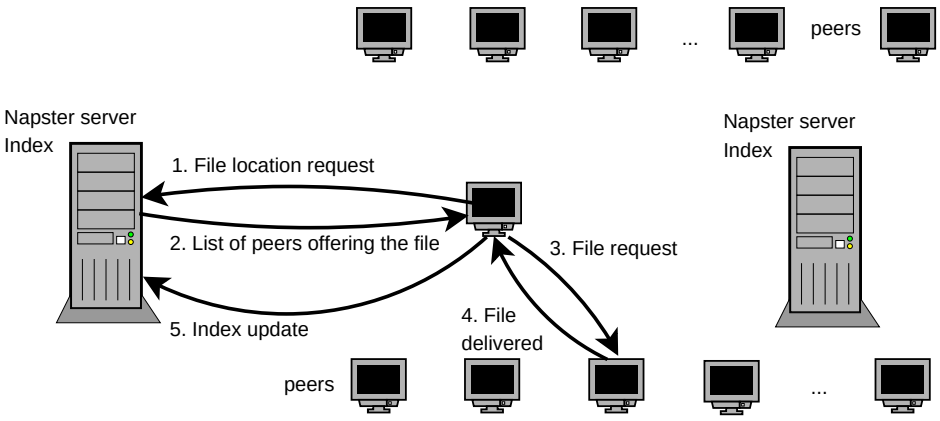
- search for large prime numbers,
- attempts at brute-force description,
- climate prediction.

Grid projects - distributed platforms that support data sharing and the coordination of computation between participating computers on a large scale. Resources are located in different organizations and are supported by heterogeneous computer hardware, operating systems, programming languages and applications.

[10] Napster – Music Files P2P (1)

- launched in **1999** became very popular for music exchange,
- architecture: **centralized replicated indexes**, but users supplied the files stored and accessed on their personal computers,
- **locality** – minimizing number of hops between client and server when allocating a server to a client requesting a file,
- taken advantage of special characteristics of the applications:
  - music files **never updated**, no need for consistency management,
  - no guarantees required concerning availability of individual files (music temporarily unavailable may be downloaded later).
- key to success: large, widely-distributed set of files available to users,
- Napster shut down as a result of legal proceedings instituted against Napster service operators by the owners of the copyright in some of the material.

[11] Napster – Music Files P2P (2)



Napster: P2P file sharing with a centralized, replicated index. In step 5. clients expected to add their own files to the pool of shared resources.

[12] P2P Middleware Requirements (1)

Function of the P2P middleware: to simplify construction of services implemented across many hosts in a widely distributed network.

Expected **functional** requirements:

- enabling clients to locate and communicate with any individual resource made available to a service,
- ability to add new resources and to remove them at will,
- ability to add hosts to the service and to remove them,
- offering simple programming interface independent of types of managed distributed resources.

[13] P2P Middleware Requirements (1)

Expected **non-functional** requirements:

- **global scalability**,
- **load balancing** - random placement and usage of replicas,
- optimization for **local** interactions between neighbouring peers,
- accommodating to highly **dynamic** host availability,
- **security** of data in an environment with heterogeneous trust,
- **anonymity**, deniability and resistance to censorship.

[14] Routing Overlays

Routing overlay
A distributed algorithm which takes responsibility for locating nodes and objects in P2P networks.

Randomly distributed identifiers (GUIDs) used to determine placement of objects and to retrieve them, thus overlay routing systems sometimes described as **distributed hash tables (DHT)**.

General tasks of a routing overlay layer:

- having given GUID routing the request,
- having given GUID publishing the resource,
- service of removal request,
- responsibility allocation depending on changing view of peers.

[15] Routing Overlay – Identifiers

GUIDs – opaque identifiers, reveal nothing about locations of objects to which they refer. Computed with usage of hash function (such as SHA-1) from all or part of the state of an object, unique. Uniqueness verified by searching for another object with the same GUID.

Prefix routing - narrowing the search for the next node along the route by applying a binary mask that selects an increasing number of hexadecimal digits from the destination GUID after each hop.

[16] Routing Overlay – DHT

---

put(GUID, data)
The data is stored in replicas at all nodes responsible for the object identified by GUID.

remove(GUID)
Deletes all references to GUID and the associated data.

value = get(GUID)
The data associated with GUID is retrieved from one of the nodes responsible for it.

Basic programming interface for a **distributed hash table (DHT)** as implemented by the PAST API over Pastry.

[17] Routing Overlay – DOLR

---

publish(GUID)
GUID can be computed from the object (or some part of it, e.g. its name). This function makes the node performing a *publish* operation the host for the object corresponding to GUID.

unpublish(GUID)
Makes the object corresponding to GUID inaccessible.

sendToObj(msg, GUID, [n])
Following the object-oriented paradigm, an invocation message is sent to an object in order to access it. This might be a request to open a TCP connection for data transfer or to return a message containing all or part of the object's state. The final optional parameter [n], if present, requests the delivery of the same message to n replicas of the object.

Basic programming interface for **distributed object location and routing (DOLR)** as implemented by Tapestry.

[18] Routing Overlay – Routing and Location

DHT:

- when data submitted to be stored with its GUID DHT layer takes responsibility for choosing a location, storing it (with replicas) and providing access,
- data item with GUID X stored at the node whose GUID numerically closest to X and moreover at the r hosts with GUIDs numerically closest to it, where R is a replication factor chosen to ensure high availability.

DOLR:

- locations for the replicas of data objects decided outside the routing layer,
- host address of each replica notified to DOLR using the publish() operation.

[19] Routing Overlay – Prefix Routing

Prefix routing:

- both Pastry and Tapestry employ **prefix routing** to determine routes,
- prefix routing is based on applying a binary mask that selects increasing number of hexadecimal digits from the destination GUID after each hop (similar to CIDR in IP).

Other possible routing schemes:

- based on numerical difference between the GUIDs of the selected node and the destination node (**Chord**),

- usage of distance in a d-dimensional hyperspace into which nodes are placed (**CAN**),
- usage of the XOR of pairs of GUIDs as a metric for distance between nodes (**Kademlia**).

[20] **P2P - Human-readable Names**

- GUIDs are not human-readable, some form of indexing service using human-readable names or search requests required,
- weakness of centralized indexes evidenced by Napster,
- example: **indices on web pages** in **BitTorrent**. Definitions: **seed** – peers with complete copy of the torrent still offering upload; **swarm** – all peers including seeds sharing a **torrent**,
- in BitTorrent a web search index leads to a stub file containing details of the desired resource. The **torrent** file contains metadata about all the files it makes downloadable, including: names, sizes, checksums of all pieces in the torrent, address of a *tracker* that coordinates communication between the peers in the *swarm* ,
- **tracker** – server that keeps track of which **seeds** and peers are in the **swarm**, not directly involved in the data transfer, does not have copies of data files.
- clients report information to the tracker periodically and in exchange receive information about other clients that they can connect to.

[21] **Pastry - Introduction**

Pastry: message routing infrastructure deployed in several applications including **PAST**, an archival (immutable) file storage system implemented as a distributed hash table with DHT API and in **Squirrel**, a P2P web caching service.

- 128-bit GUIDs (hash function such as SHA-1) randomly distributed in the range $0 \div 2^{128} - 1$,
- in a network with N participating nodes, Pastry routing algorithm correctly route a message addressed to any GUID in $O(log_N)$ steps,
- if a target node is active, message is delivered, otherwise message delivered to active node which is numerically closest to it.

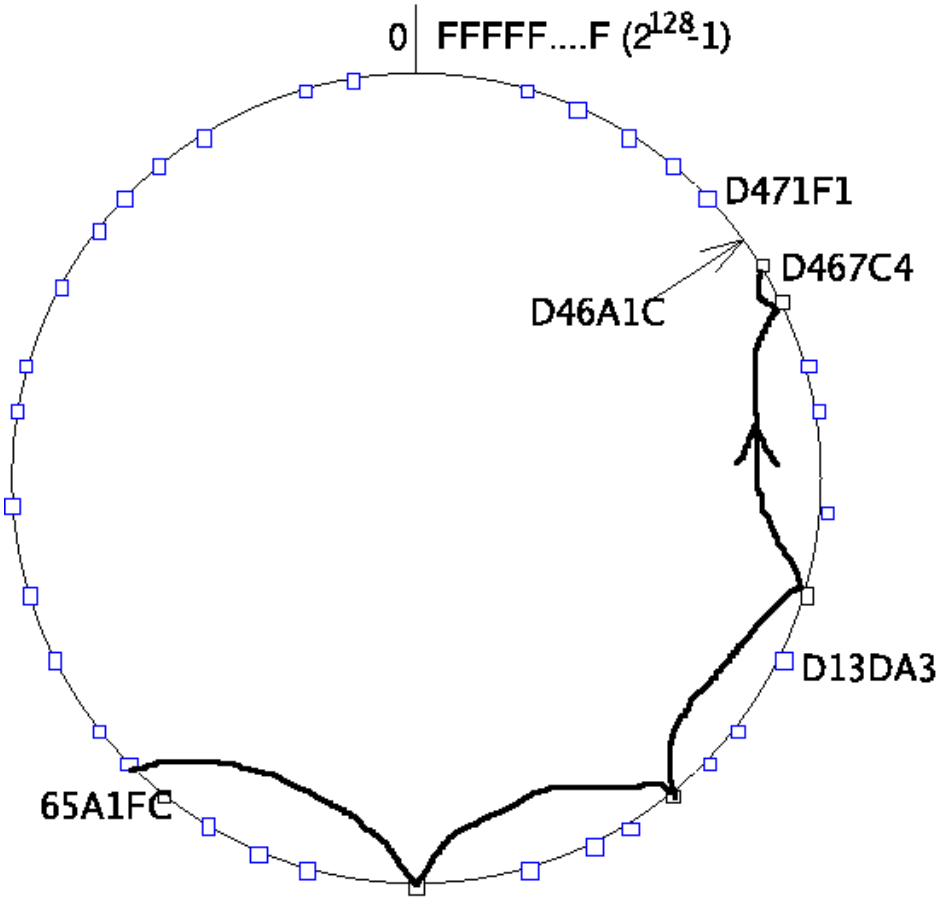
- active nodes take responsibility for processing requests addressed to all objects in their numerical neighbourhood,
- moreover Pastry uses a locality metric based on network distance in the underlying network to select appropriate neighbours,

[22] **Pastry - Routing**

Routing, simplified approach:

- each active node stores a leaf set – a vector L (of size $2l$) containing the GUIDs and IP addresses of the nodes whose GUIDs are numerically closest on either side of its own (above and below),
- leave sets maintained by Pastry as nodes join and leave,
- any node A that receives a message M with destination address D routes the message by comparing D with its own GUID A and with each of the GUIDs in its leaf set and forwards M to the node amongst them that is numerically closest to D,
- inefficient, requires about $N/2l$ hops to deliver a message.

[23] **Circular Routing**



Black color depicts live nodes. The space is considered as circular: node 0 is adjacent to node ($2^{128} - 1$). The diagram illustrates the routing of a message from node 65A1FC to D46A1C using leaf set information alone, assuming leaf sets of size 8 ($l = 4$, in Pastry usually 8). This is a degenerate type of routing that would scale very poorly; it is not used in practice.

[24] **Pastry Routing**

- efficient routing due to routing tables,
- each node maintains a **tree-structured routing table** of nodes spread throughout the entire address range, with increased density of coverage for GUIDs numerically close to,
- the routing process at any node uses the information in its routing table and leaf set to handle each request from an application and each incoming message from another node,

– new nodes use a joining protocol and compute suitable GUIDs (typically by applying the SHA-1 to the node's public key, then it make contact with a nearby (in network distance) Pastry node.

[25] **Pastry's Routing Table**

p =	GUID prefixes and corresponding nodehandles n															
0	0 n	1 n	2 n	3 n	4 n	5 n	6 n	7 n	8 n	9 n	A n	B n	C n	D n	E n	F n
1	60 n	61 n	62 n	63 n	64 n	65 n	66 n	67 n	68 n	69 n	6A n	6B n	6C n	6D n	6E n	6F n
2	650 n	651 n	652 n	653 n	654 n	655 n	656 n	657 n	658 n	659 n	65A n	65B n	65C n	65D n	65E n	65F n
3	65A0 n	65A1 n	65A2 n	65A3 n	65A4 n	65A5 n	65A6 n	65A7 n	65A8 n	65A9 n	65AA n	65AB n	65AC n	65AD n	65AE n	65AF n

First four rows of a Pastry routing table located in a node whose GUID begins with 65A1.

– each "n" element represents [GUID, IP address] pair specifying next hop to be taken by messages addressed to GUIDs that match each given prefix.

– grey-shaded entries indicate that the prefix matches the current GUID up to the given value of *p: the next row down or the leaf should be examined to find a route,*

– *although there are a maximum of 128 rows in the table, only* $log_{16}N$ *rows will be populated on average in a network with N active nodes.*

[26] **Pastry's Routing Algorithm**

If $R[p, i]$ means the element at column i in the row p of the routing table and L means leaf set. To handle a message M addressed to a node D:

if ($L_{-l} < D < L_l$) {

forward M to the element L_i of the leaf set with GUID closest to D or the current node A.

} else {

find p, the length of the longest common prefix of D and A.

find i, the $(p + 1)^{th}$ hexadecimal digit of D.

if ($R[p, i] \neq null$) {

forward M to $R[p, i]$,

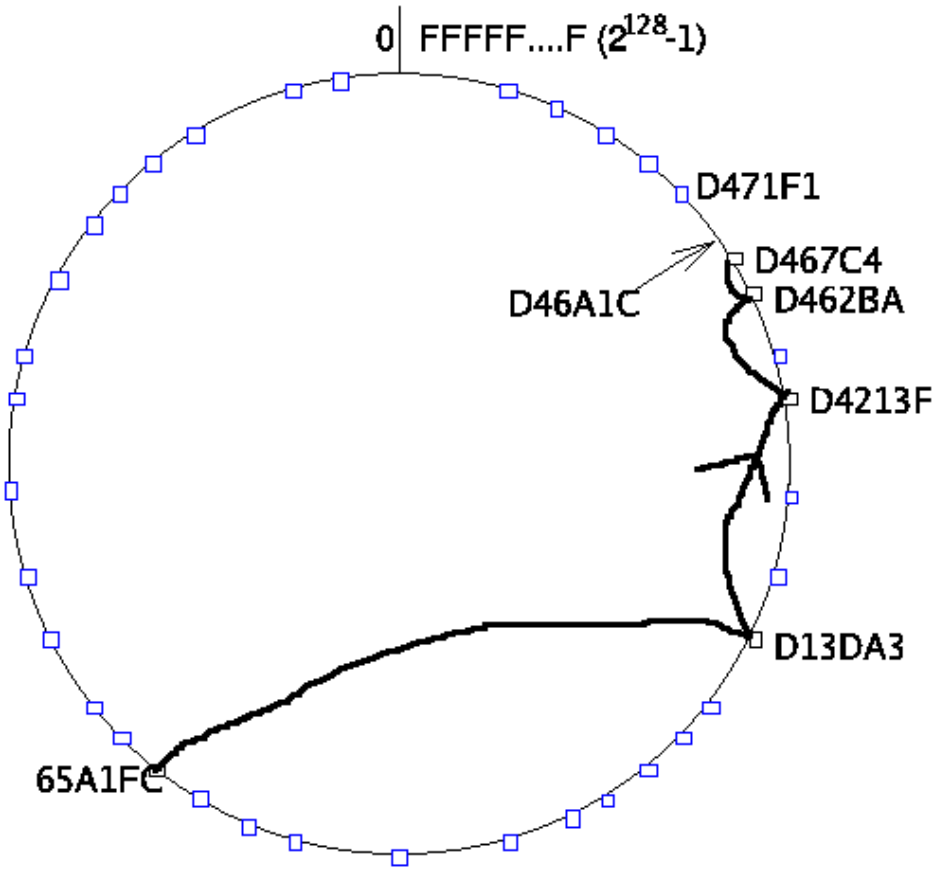
} else {

forward M to any node in L or R with a common prefix of length i, but a GUID that is numerically closer.

}

}

[27] **Pastry Routing Example**



Routing a message from node 65A1FC to D46A1C. With the aid of a well-populated routing table the message can be delivered in $log_{16}(N)$ hops.

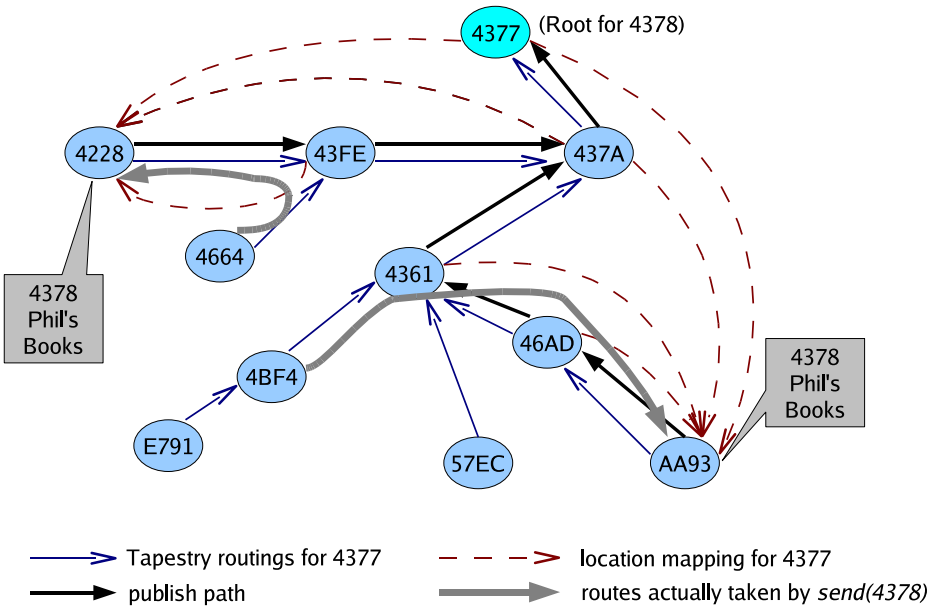
[28] **Pastry - Host Failure and Fault Tolerance**

- nodes may fail or depart without warning, node considered failed when its immediate neighbours (in GUID space) can no longer communicate with it,
- to repair leaf set, the node looks for a live node close to the failed one and requests a copy of its leaf set (one value to replace),
- repairs to routing tables made on a 'when discovered' basis,
- moreover all nodes send **heartbeat messages** to neighbouring nodes in their leaf sets,
- to deal with any remaining failures or malicious nodes, small degree of randomness introduced into the route selection algorithm. Possible usage of a routing from an earlier row with less optimal but different routing.

[29] **Tapestry**

- nodes holding resources periodically use the **publish(GUID)** primitive to make them known to Tapestry, holders responsible for storing resources, replicated resources published with the same GUID,
- 160-bit identifiers used to refer both to objects and to nodes that perform routing actions,
- for any resource with GUID G unique root node with GUID R_G numerically closest to G,
- on each invocation of *publish(G)* publish message routed towards R_G,
- on receipt R_G enters mapping between G and the sending host's IP, (G, IP_H) in its routing table, the same cached along publication path.

[30] **Tapestry Routing**



Replicas of the file *Phil's Books* (G=4378), hosted at nodes 4228 and *AA*93. Node 4377 is the root node for object 4378. Shown routings are some of the entries in routing tables. The location mapping (cached while servicing publish messages) are subsequently used to route messages sent to 4378.

[31] **Squirrel Web Cache (1)**

– developed by authors of Pastry P2P web caching service for use in local networks,

Web caching in general:

– browser cache, proxy cache, origin web server,

– metadata stored with an object in a cache: date of last modification T, time-to-leave t or *eTag* (hash computed from the object contents),

– conditional GET (cGET) request issued to the next level for validation,

– cGET request types: *If-Modified-Since*, *If-None-Match*,

– in response either the entire object or *not-modified* message.

[32] **Squirrel Web Cache (2)**

- SHA-1 hash function applied to the URL of each cached object to produce a 128-bit Pastry GUID, GUID not used to validate content,
- in the simplest implementation: the node whose GUID numerically closest to the GUID of an object becomes the object's *home node*, responsible for holding any cached copy of the object,
- Squirrel routes a Get or a cGet request via Pastry to the home node.

Evaluation, two real working environments within Microsoft, 105 active clients (Cambridge), 36000 active clients (Redmond):

- reduction in total external bandwidth: caches 100MB, 37% (Cambridge), 28% (Redmond), hit ratio for centralized servers: 38% and 29% respectively,
- local latency perceived by users for access web objects: neglectable,
- computational and storage load: low and likely to be imperceptible to users.

[33] **OceanStore File Store**

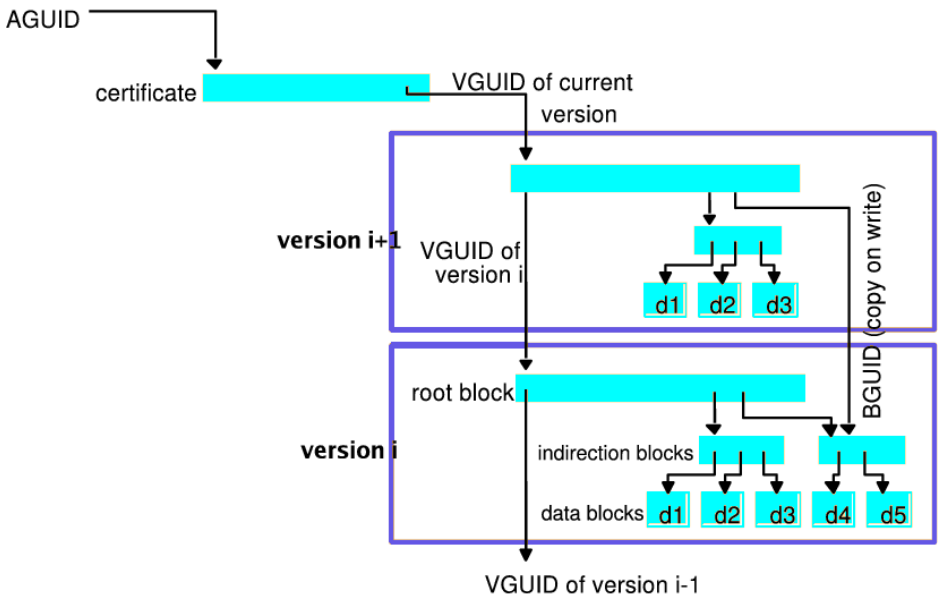
- **OceanStore** – unlike Past, supports the storage of **mutable files**,
- goal: very large scale, scalable persistent storage facility for mutable data objects with long-term persistence and reliability in changing network and computing resources environment,
- privacy and integrity achieved through **encryption of data** and use of a **Byzantine agreement** protocol for updates to replicated objects – because trustworthiness of individual hosts cannot be assumed,
- **Pond** – OceanStore prototype implemented in Java, uses **Tapestry** routing overlay to place blocks of data at distributed nodes and to dispatch requests to them,
- data stored in a set of blocks, data blocks organized and accessed through a metadata block called **root block**,
- each object represented as an **ordered sequence of immutable versions** kept for ever, versions share unchanged blocks (copy-on-write technique),

[34] **Ocean Store - Storage Organization (1)**

Name	*Meaning*	*Description*
BGUID	block GUID	Secure hash of a data block
VGUID	version GUID	BGUID of the root block of a version
AGUID	active GUID	Uniquely identifies all the versions of an object

- several replicas of each block stored at peer nodes selected accordingly to locality and storage availability criteria,
- data blocks GUIDs published (with *publish()*) by each of the nodes that holds a replica, Tapestry can be used by clients to access the blocks,
- AGUID stored in directories against each file name,
- association between an AGUID and the sequence of versions of the object recorded in signed certificate stored and replicated by primary copy replication scheme,
- trust model for P2P requires construction of each new certificate being agreed amongst small set of hosts called the **inner ring**.

[35] Ocean Store - Storage Organization (2)



Version $i+1$ has been updated in blocks d1, d2 and d3. The certificate and the root blocks include some data not shown. All unlabelled arrows are BGUIDs.

[36] **Pond Performance**

	LAN		*WAN*		*Predominant operations in benchmark*
Phase	*Linux NFS*	*Pond*	*Linux NFS*	*Pond*	
1	0.0	1.9	0.9	2.8	Read and write
2	0.3	11.0	9.4	16.8	Read and write
3	1.1	1.8	8.3	1.8	Read
4	0.5	1.5	6.9	1.5	Read
5	2.6	21.0	21.5	32.0	Read and write
Total	4.5	37.2	47.0	54.9	

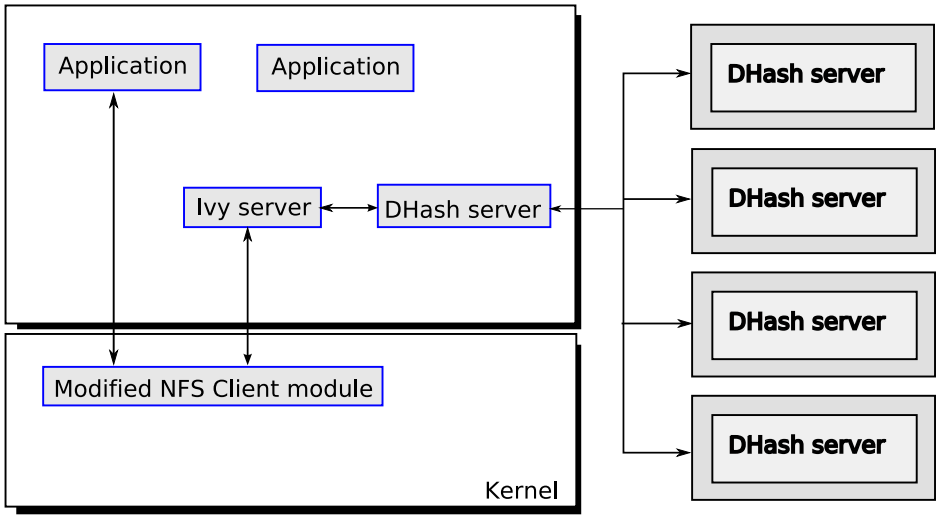
Times in seconds to run different phases of the *Andrew benchmark*. (1) recursive subdirectory creation, (2) source tree copying, (3) status only examining of all the files in the tree, (4) every data byte examining in all the files, (5) compiling and linking the files.

[37] **Ivy File System**

- read/write file system emulating a Sun NFS server,
- stores the state of files as logs of the file update requests issued by Ivy clients,
- log records held in DHash distributed hash-addresses storage service (160-bit SHA-1),
- **version vectors** to impose a **total order** on log entries when reading from multiple logs,
- potentially very long *read* time reduced by use of a combination of local caches and *snapshots*,
- shared file system seen as a result of merging all the updates performed by (dynamically selected – views) set of participants,

– possible continuing operations during partitions in the network, conflicting updates to shared files resolved similar like in **Coda** file system.

[38] Ivy Architecture



Ivy system architecture.

[39] Ivy – Performance

Each participant maintains a mutable DHash block (called **log-head**) that points to a participant's most recent log record. Mutable blocks are assigned a cryptographic public key pair by their owner. The contents of the block are signed with the private key. Any participant that has the public key can retrieve the log-head and use it to access all the records in the log.

Performance:

– execution times mostly two times (for some operations three times) larger than for NFS,

– in WAN 10 times slower than in LAN, similar to NFS – still NFS not designed for usage in WAN,

Primary contribution of Ivy: novel approach to the management of security and integrity in an environment of partial trust (in networks spanning many organizations and jurisdictions).

[40] P2P – Summary

The benefits of P2P:

- ability to exploit unused resources (storage, processing) in the host computers,
- ability to support large numbers of clients and hosts with adequate balancing of the loads on network links and host computer resources,
- self-organizing properties of the middleware platforms lead to to costs largely independent of the numbers of clients and hosts deployed.

Weaknesses and subjects of research:

- relatively costly as storage solution for mutable data compared to trusted centralized service solutions,
- still lack of strong guarantees for client and host anonymity.