Peer-to-Peer Systems

- Consider a file storage application provided by a client/ server system
  - Architecture
    - Single server, cluster, remote replication
  - File location
    - Well known location(s)
  - Scalability
    - Storage, bandwidth
  - Availability
    - What proportion of requests are satisfied
  - Manageability / control
    - How well can we control the service
  - (Total) Cost
    - Purchase, operation, administration, maintenance

Peer-to-Peer Systems

- Peer-to-Peer systems offer an alternative to the traditional client/server approach
  - "exploit resources available at the edges of the Internet"
  - commodity personal computers or workstations
  - becoming increasingly possible and desirable
- What resources?
  - storage capacity
  - stored content
  - CPUs
  - users?
- Examine:
  - characteristics and challenges
  - middleware
  - applications

Characteristics

- Each peer contributes resources to the system
  - the aggregated set of resources is used to provide the service
- All peers in the system have the same capabilities
  - although peers may differ over time in the roles they perform or the resources they provide
- No dependence on the availability of a centralized component or service
  - What well known P2P application violated this characteristic?
- A key issue in the design of a peer-to-peer system is the placement of migratable resources in the system (and the subsequent location of these resources)
  - balancing workload, availability, overheads (e.g. latency)

Challenges

- Volatility
  - owners/managers/users of peers (hosts/nodes) in a peer-to-peer system will usually not (or cannot) provide guarantees for the availability of individual peers, leading to ***churn***
- Therefore, a peer-to-peer service cannot guarantee the availability of a resource
  - although a system can be designed to make the probability of failure (I.e. unavailability) small
- This is an issue called ‘harvest’
  - what proportion of the normally available resources can currently be accessed?

Peer-to-Peer Middleware

- 3rd Generation Peer-to-Peer systems
  - 1st Generation: e.g. Napster
  - 2nd Generation: e.g. Freenet, Kazaa, Gnutella, BitTorrent
- examples: **Pastry**, Tapestry, CAN, Chord, Kademlia
- Peer-to-peer middleware systems
  - place resources (files, data objects, …) on a set of computers that are widely distributed throughout the Internet
  - route messages to these resources on behalf of clients
  - relieve client applications of this responsibility
- Resources are identified by globally unique IDs (GUIDs)
  - a secure hash from some or all of the object's state
- (Overlay) Routing takes place in the GUID space, not the IP address space

Middleware Requirements

- Assign resources to nodes on behalf of applications
  - Add new resources and remove existing ones
  - Load balancing: achieve an even/fair distribution of load across peers through random placement and replication of resources
- Subsequently, locate resources on behalf of applications
- Optimise for local interactions between nearby clients (in the network space, **not** the GUID space)
- Add new hosts and remove existing ones
  - Hosts in a peer-to-peer system will usually not provide guarantees about availability in the same way that centralised servers do
  - Accommodate volatile host availability (churn)
  - Redistribute resources / load
- Other issues of less interest to us: security, anonymity, freedom

Pastry

- A. Rowstron and P. Druschel, "Pastry: Scaleable, decentralized object location and routing for large-scale peer-to-peer systems", 2001
  - **Required reading**
- Nodes and objects are assigned 128-bit GUIDs
  - Computed using a secure hash function
  - Nodes in close proximity w.r.t. the nodeId space are likely to be distant w.r.t. the network, geographic, admisistrative, … spaces
- Pastry uses ***prefix routing*** to route a message with destination *D* to the node that is numerically closest to *D*
  - in each routing step, a message is (usually) forwarded to a node B that shares with D a nodeId prefix that is b bits (if b=4, then 1 hex digit) longer than that shared between A and D, for example:

D	4	6	7	C	4	…

nodeId A

D	4	6	9	5	3	…

nodeId B

•••

D	4	6	9	E	2	…

nodeId D

Pastry Routing State Information

- Each Pastry node maintains routing state information
- ***Routing Table*** R
  - $\log_{2^b} N$ rows
  - 2^b - 1 entries in each row
  - each entry contains nodeId and associated IP address
  - each entry in row n refers to a node whose nodeId shares the same first n digits but whose n+1th digit has one of the 2^b-1 other possible values other than the n+1th digit in this node's nodeId
  - usually there will be many nodes that could be placed in a given entry of the routing table, but we only record one of them
  - in practice, an effort is made to ensure that the node selected for a given entry in R is ***close*** to the current node (according to some definable proximity metric, e.g. network latency)
  - if no suitable node for a given entry is known, then the entry is left blank
  - the maximum number of rows in the routing table is 128/b but in practice only (approx) $\log_{2^b} N$ rows will be populated

Pastry Routing State Information

- Routing table example
  - b=2, nodeId=23102103
  - each entry also contains the IP address of the associated node but this information is not shown

n=0	02010321	11030210	2	32010302
n=1	20132010	21302013	22013202	3
n=2	23021032	1	23202132	23321021
n=3	0	23112012	23121021	23130021
n=4	23100101	23101230	2	23103202
n=5	23102012	1	23102203	23102300
n=6	0		23102121	
n=7				3

Pastry Routing State Information

- ***Leaf Set*** L
  - Contains the set of nodes numerically closest to the current nodeId (i.e. w.r.t. nodeId, not the proximity metric)
  - |L| / 2 closest nodes with smaller nodeIds and |L| / 2 closest nodes with larger nodeIds
  - Used during message routing (more later)

- ***Neighbourhood Set*** M
  - Contains the nodeIds and IP addresses of the |M| closest to the current node, w.r.t the chosen proximity metric (e.g. network latency)
  - Not normally used for routing messages but is used to maintain/improve routing locality properties (more later)

- Typical values for both |L| and |M| are 2^b or 2^{b+1}

Pastry Routing Algorithm

- Routing algorithm describes the actions taken by a node A on receipt of a message D to forward the message to the nest node along the route (to D)

- Some notation
  - R^i_l is the routing table entry at column i and row l
  - L_i is the i^{th} closest nodeId in the leaf set, with –ve values indicating smaller nodeIds and +ve values indicating larger nodeIds
  - D is the GUID of the message destination
  - D_l is the value of the l^{th} digit in D
  - shl(A,B) is the length of the prefix shared between A and B, measured in digits

Pastry Routing Algorithm

if $(L_{-\lfloor |L|/2 \rfloor} \leq D \leq L_{\lfloor |L|/2 \rfloor})$ {
    // D is within range of our leaf set
    forward to L_i, s.th. $|D - L_i|$ is minimal;
} else {
    // use the routing table
    Let $l = shl(D, A)$;
    if $(R_l^{D_l} \neq null)$ {
        forward to $R_l^{D_l}$;
    }
    else {
        // rare case
        forward to $T \in L \cup R \cup M$, s.th.
            $shl(T, D) \geq l$,
            $|T - D| < |A - D|$
    }
}

Pastry API

- Joining an existing Pastry network (or creating a new one if this is the first node to join)
  - `nodeId = pastryInit(Credentials, Application)`
  - `Credentials`: Application specific – used for authentication
  - `Application`: Object used for application callbacks
  - Returns `nodeId` (GUID) for the joining node
- Routing a message to a destination
  - `route(msg, key)`
  - Routes `msg` through the Pastry network to the node with the nodeId numerically closest to the `key`
- Application object must provide
  - `deliver(msg, key)`: called when a `msg` is received and the current nodeId is closest to the `key`
  - forward(msg, key, nextId): called by Pastry before a `msg` with destination `key` is routed to the next node in the route identified by `nodeId` – the current node may modify the message or stop it
  - `newLeafs(leafSet)`: informs application of a change in the `leafSet`

Pastry node arrival

- Arriving nodes need to populate their routing table, neighbourhood set and leaf set
- Procedure
  - Initially, an arriving node with nodeId X contacts an existing node A (which it is told about or can find, e.g. through multicast/broadcast)
  - Node A routes a **join** message to Z – the existing node that is numerically closest to the new node X – through the Pastry network
  - Each node in the route A..Z along the Pastry route to X sends its state information to X which uses the received information to initialize its state
- Initializing neighbourhood set
  - It is assumed that A was in close proximity to X so A’s neighbourhood set is used by X
- Initializing leaf set
  - Z will be numerically close to X (last step in the Pastry route) so Z’s leaf set is used by X

Pastry node arrival

- Initializing routing table
  - Entries in the first row of X's routing table (X_0) must be in close proximity to X but are independent of X's nodeId so A_0 is a good choice of entries for X_0
  - For row X_1, where each entry has a prefix of length 1 in common with X, we can use the entries in B_1 – the second row of B's routing table, where B is the second step in the route from A .. Z
  - Similarly, for row X_2, where each entry has a prefix of length 2 in common with X, we can use the entries in C_2 – the third row of C's routing table, where C is the third step in the route from A .. Z
  - We continue like this until we have populated the routing table
- Finally
  - X sends its new state to every node in R, L and N to allow them to update their own state using X's information

Pastry node departure

- A pastry node is considered failed when its immediate neighbours can no longer communicate with it
  - Requires repair of leaf sets in which the failed node appeared
  - Node that discovers failure will obtain leaf set from a node close to the failed node and replace the failed node with one from this other leaf set
  - Works unless |L| / 2 nodes fail simultaneously

  - To repair a routing table entry R_l^d, R_l^i is contacted ($i \neq d$) and sends its own R_l^d
  - If this fails to produce a valid replacement, R^i_{l+1} is contacted ($i \neq d$), and so on

  - To repair the neighbourhood set, request Neighbourhood set from remaining neighbours and update using new information

Locality of Pastry Routing

- Entries in the Pastry routing table are chosen to be in close proximity to a node
  - Each step in a pastry route forwards a message to a node that shares a prefix with the message that is greater than or equal to that shared with the present node
  - As a result, each step moves a message closer to the destination in the nodeId space, while travelling the least possible distance (ideally?) in the proximity space
- Pastry does not route messages to achieve a globally minimum routing distance
  - Each routing decision is based on local information only
  - But the routes that are produced are “good”
  - Why?

Locality of Pastry Routing

- If a message is routed from a node A to a node B at a distance d, the message cannot subsequently be routed to a node at a distance of less than d from A
  - Follows from routing procedure and (correct?) maintenance of routing information
- The expected distance travelled by a message during each successive routing step increases exponentially from one step to the next
  - Each entry in row l of the routing table is chosen from a set containing $N/2^{bl}$ nodes
- So, a message tends to make larger steps towards its destination in each successive routing step

Pastry performance

- Routing distance (nodeId space)
  - $\mathrm{ceil}(\log_{2^b} N)$ is the expected maximum number of steps in a Pastry route
  - verified experimentally
- Effect of maintaining only partial routing tables
  - 30%-40% longer routes than complete (global) routing tables
- Replica routing
  - Many application use Pastry to route to the nearest among the k nodes numerically closest to a given destination GUID
  - Pastry locates the nearest node 76% of the time and one of the two nearest nodes 92% of the time