Tapestry: A Resilient Global-scale Overlay for Service Deployment

Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz

Shawn Jeffery
CS294-4 Fall 2003
jeffery@cs.berkeley.edu

What have we seen before?

- Key-based routing similar to Chord, Pastry
- Similar guarantees to Chord, Pastry
  - Log_bN routing hops (b is the base parameter)
  - $bLog_bN$ state on each node
  - $O(Log_b^2N)$ messages on insert
- Locality-based routing tables similar to Pastry
- Discussion point (for throughout presentation):
  - What sets Tapestry above the rest of the structured overlay p2p networks?

Decentralized Object Location and Routing: DOLR

- The core of Tapestry
- Routes messages to endpoints
  - Both Nodes and Objects
- Virtualizes resources
  - objects are known by name, not location

DOLR Identifiers

- ID Space for both nodes and endpoints (objects) : 160-bit values with a globally defined radix (e.g. hexadecimal to give 40-digit IDs)
- Each node is randomly assigned a nodeID
- Each endpoint is assigned a *Globally Unique IDentifier* (GUID) from the same ID space
- Typically done using SHA-1
- Applications can also have IDs (application specific), which are used to select an appropriate process on each node for delivery

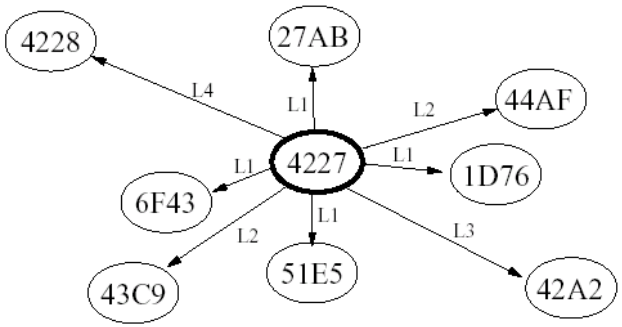
DOLR API

- PublishObject(O_G, A_{id})
- UnpublishObject(O_G, A_{id})
- RouteToObject(O_G, A_{id})
- RouteToNode(N, A_{id}, Exact)

Node State

- Each node stores a neighbor map similar to Pastry
  - Each level stores neighbors that match a prefix up to a certain position in the ID
  - Invariant: If there is a hole in the routing table, there is no such node in the network
- For redundancy, backup neighbor links are stored
  - Currently 2
- Each node also stores backpointers that point to nodes that point to it
- Creates a *routing mesh* of neighbors

Routing Mesh



Routing

- Every ID is mapped to a *root*
- An ID's root is either the node where nodeID = ID or the "closest" node to which that ID routes
- Uses prefix routing (like Pastry)
  - Lookup for 42AD: 4*** => 42** => 42A* => 42AD
- If there is an empty neighbor entry, then use *surrogate routing*
  - Route to the next highest (if no entry for 42**, try 43**)

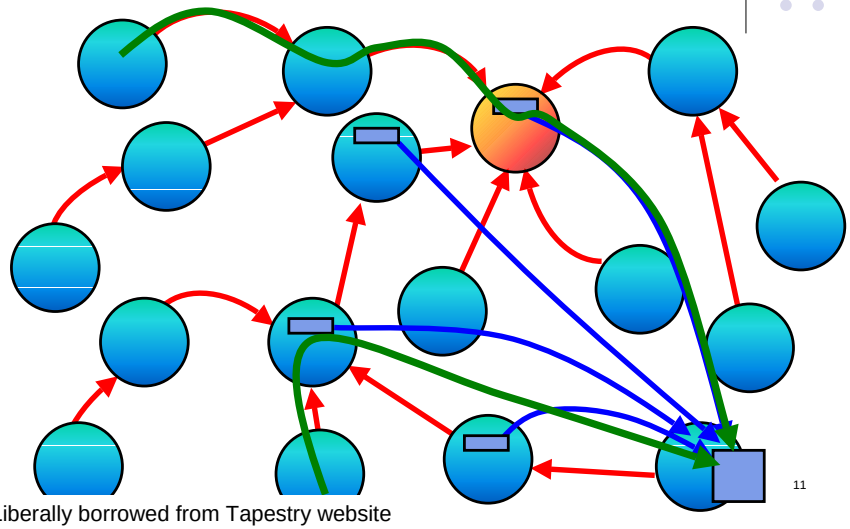
Object Publication

- A node sends a publish message towards the root of the object
- At each hop, nodes store pointers to the source node
  - Data remains at source. Exploit locality without replication (such as in Pastry, Freenet)
  - With replicas, the pointers are stored in sorted order of network latency
- Soft State – must periodically republish

Object Location

- Client sends message towards object's root
- Each hop checks its list of pointers
  - If there is a match, the message is forwarded directly to the object's location
  - Else, the message is routed towards the object's root
- Because pointers are sorted by proximity, each object lookup is directed to the closest copy of the data

Use of Mesh for Object Location



Liberally borrowed from Tapestry website

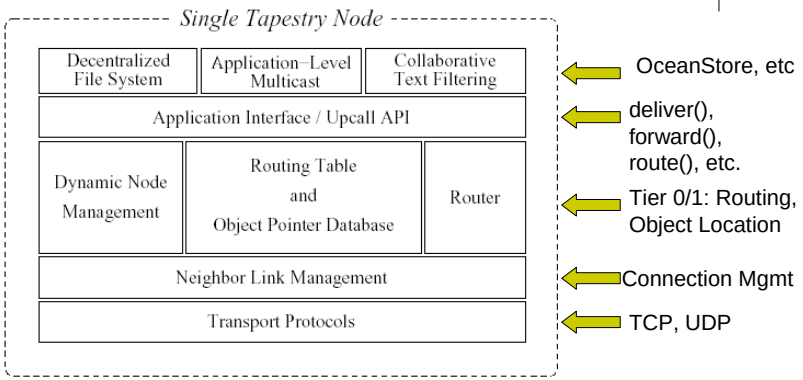
Node Insertions

- A insertion for new node N must accomplish the following:
  - All nodes that have null entries for N need to be alerted of N's presence
    - *Acknowledged mulitcast* from the "root" node of N's ID to visit all nodes with the common prefix
  - N may become the new root for some objects. Move those pointers during the mulitcast
  - N must build its routing table
    - All nodes contacted during mulitcast contact N and become its neighbor set
    - Iterative nearest neighbor search based on neighbor set
  - Nodes near N might want to use N in their routing tables as an optimization
    - Also done during iterative search

Node Deletions

- Voluntary
  - Backpointer nodes are notified, which fix their routing tables and republish objects
- Involuntary
  - Periodic heartbeats: detection of failed link initiates mesh repair (to clean up routing tables)
  - Soft state publishing: object pointers go away if not republished (to clean up object pointers)
- Discussion Point: Node insertions/deletions + heartbeats + soft state republishing = network overhead. Is it acceptable? What are the tradeoffs?

Tapestry Architecture



- Prototype implemented using Java

Experimental Results (I)

- 3 environments
  - Local cluster, PlanetLab, Simulator
- Micro-benchmarks on local cluster
  - Message processing overhead
    - Proportional to processor speed - Can utilize Moore's Law
  - Message throughput
    - Optimal size is 4KB

Experimental Results (II)

- Routing/Object location tests
  - Routing overhead (PlanetLab)
    - About twice as long to route through overlay vs IP
  - Object location/optimization (PlanetLab/Simulator)
    - Object pointers significantly help routing to close objects
- Network Dynamics
  - Node insertion overhead (PlanetLab)
    - Sublinear latency to stabilization
    - O(LogN) bandwidth consumption
  - Node failures, joins, churn (PlanetLab/Simulator)
    - Brief dip in lookup success rate followed by quick return to near 100% success rate
    - Churn lookup rate near 100%

Experimental Results Discussion

- How do you satisfactorily test one of these systems?
  - What metrics are important?
  - Most of these experiments were run with between 500 - 1000 nodes. Is this enough to show that a system is capable of global scale?
  - Does the usage of virtual nodes greatly affect the results?

Best of all, it can be used to deploy large-scale applications!

- Oceanstore: a global-scale, highly available storage utility
- Bayeux: an efficient self-organizing application-level multicast system
- We will be looking at both of these systems

Comments? Questions? Insults?

jeffery@cs.berkeley.edu