

Tapestry: A Resilient Global-Scale Overlay for Service Deployment

B. Shao, L. Huang, J. Stribling, S. Rhea, A. Joseph, J. Kubiawicz

Presented 10.11.2004 by Joakim Koskela

Introduction

Problem:

The need for an infrastructure for distributed applications

Presented solution: **Tapestry**

- An infrastructure providing Decentralized Object Location and Routing (DOLR)
 - Key-based object locating
- Peer-to-peer overlay network focused on efficiently and fault-tolerant routing
- Real-life applications exist (OceanStore, Bayeux)

..compared with existing solutions:

- 2nd generation P2P (key-based routing)
- Takes locality into account
 - Chord / CAN's whole-network hops
 - Generally, the nearest object copy is accessed
- Does not limit the number of object replicas
 - Locating based on *pointer* mechanism

Concepts

- The actors in Tapestry are *nodes*
 - e.g. an application using the network
- Data to be retrieved – *objects* – are located on *servers*
 - e.g. a file
- Each node and object have an unique identifier in the same space
 - Node ids assigned randomly (SHA1 suggested)
- Each node has a routing table which lists other nodes that it communicates with (routes through) – *neighbors*
- An object map to a node that have the same or *similar* id – its *root*
 - The root of the object is responsible for hosting a pointer to the server serving the object
 - Most likely, there will not be a root matching the object's id. The node with the closest id will then be chosen as root – it's *surrogate*

Communication in the DOLR

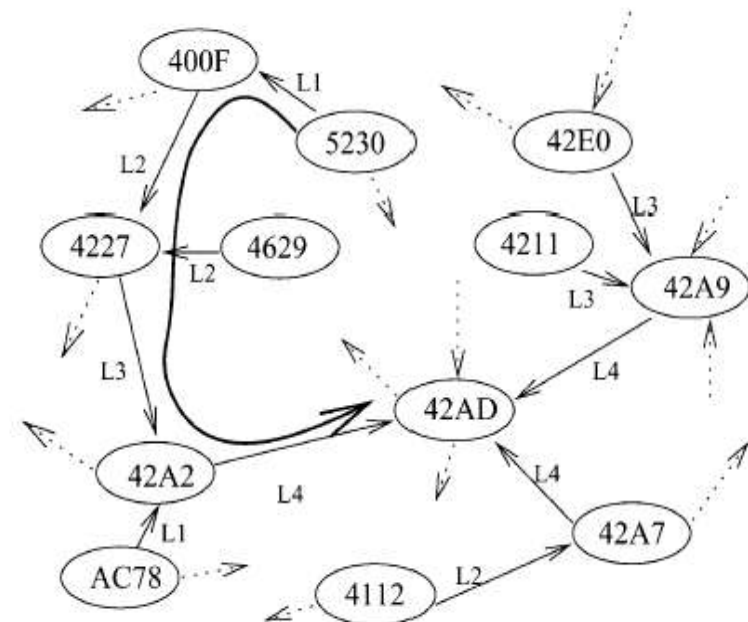
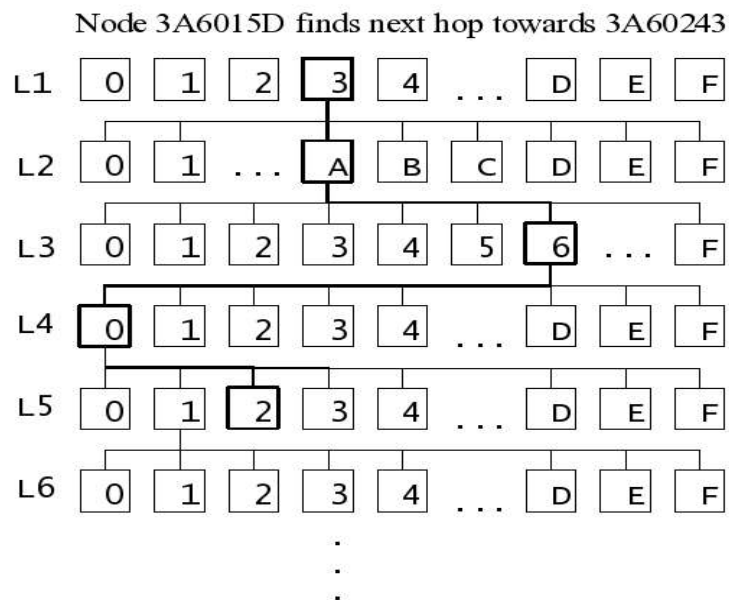
- Packet- based communication
- Objects are *published* by the server
 - Objects must be explicitly made available for the network

Network operations

- PublishObject
 - Server (publisher) sends a publish- message about the object it serves
- UnpublishObject
 - Server removes the object
- RouteToObject
 - Finds the server serving the object by routing towards the root
- RouteToNode
 - Routes according to the network logic to either the requested node, or its surrogate

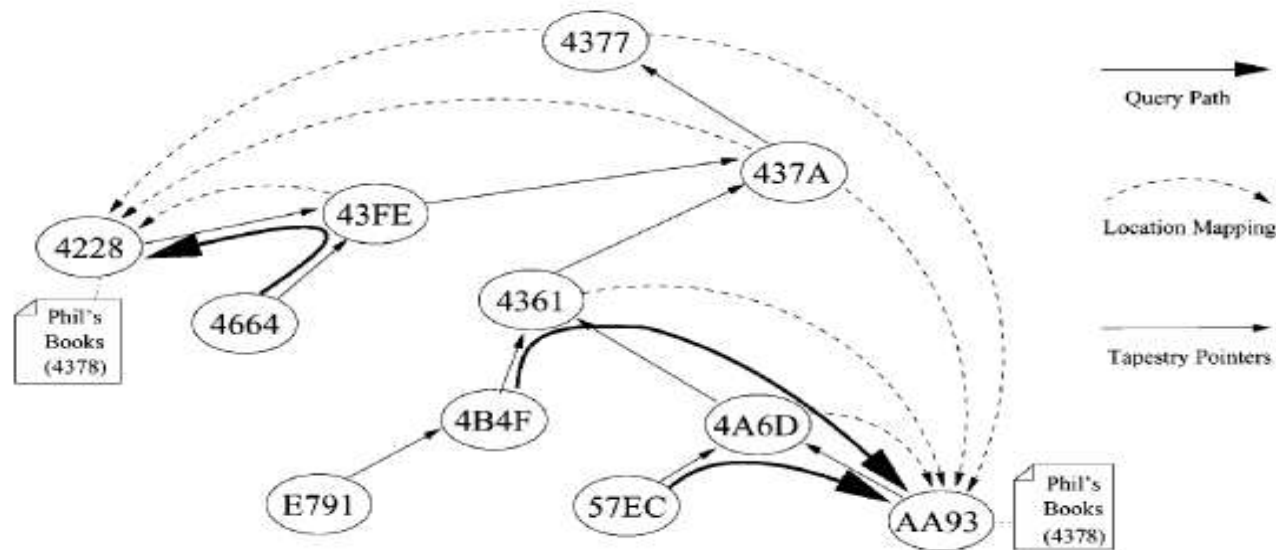
Routing

- Each node has a routing table – *mesh*
 - Entries are mapping between ids and IP-endpoints
 - Multi-leveled, each level d contains neighbors with ids that have the same prefix as the node itself up to the d :th digit
- Basic case route-table size managable
 - Backpointers
 - Resilience achieved by including alternative neighbors for each entry
- **Logic:** Packets are forwarded to the neighbor matching best the destination
 - Locating guaranteed in fault-free system
 - CIDR- like



Publishing

- Object servers periodically publishes the objects
 - A publishing message is sent through the network to the root of the object
- Each intermediate node stores the mapping of the object to its server
 - Multiple entries for the same object is sorted by network latency (locality)
 - The mapping is stored not only on the root, but throughout the path between it and the object server
- Nodes that tries to locate the object server does not have to reach the root
 - Sufficient to encounter a node in the path between the server and the root



Network management

Inserting a node

- With each new node
 - Filling the voids in other nodes' routing tables
 - Transferral of responsibility for object which root the new node is
 - Creating the mesh for the new node
 - Optimizing existing nodes' routing tables
- Although examined per-node, whole networks can be joined
 - Ids are location-independent, only the routing tables changes

Removing a node

- Transfer of root-responsibilities
- Updating the routing tables

Node insertion

Process

- The new node connects and finds the current surrogate node for it
 - Surrogate shares a prefix of d digits
- Surrogate node notifies the other nodes in its mesh sharing the same prefix
- The nodes notified contact the new node
 - Routing meshes for the existing nodes will be filled
 - The lowest level of the mesh for the new node will be filled
 - Objects for which the new node is root for, are transferred to it
- Mesh-creation: the 'Nearest neighbor search'
 - The new node queries the nodes on the lowest level of the mesh for their backpointers.
 - From these, the closest are used to construct the next level ($d-1$) in the new node's mesh
 - Iteratively, the whole mesh is constructed
 - The nodes contacted during the process can consider using the new node in their mesh for filling/optimization purposes

Node removal

Voluntary

- The node notifies nodes in its backpointers
 - Supplies alternative nodes to fill the possible void
- Root responsibilities are moved
 - On the initiative of the new root

Involuntary

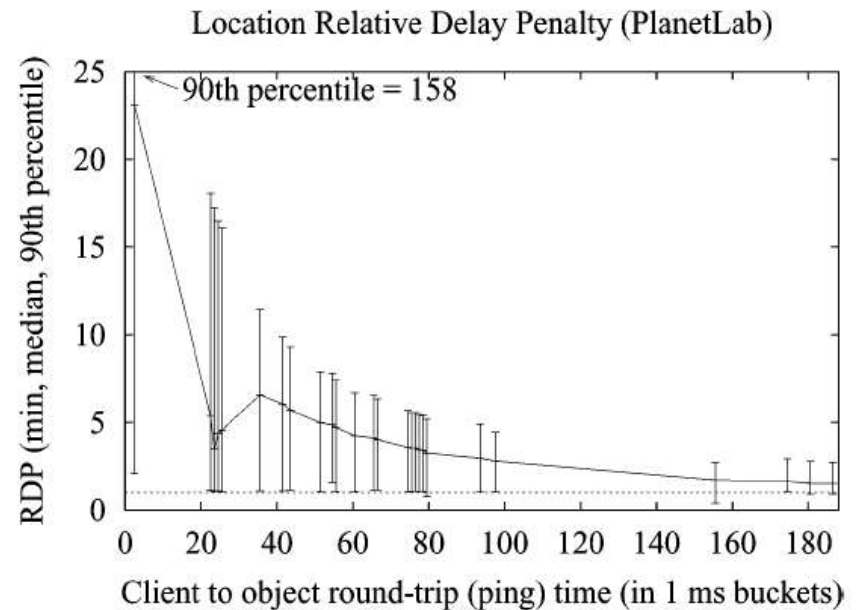
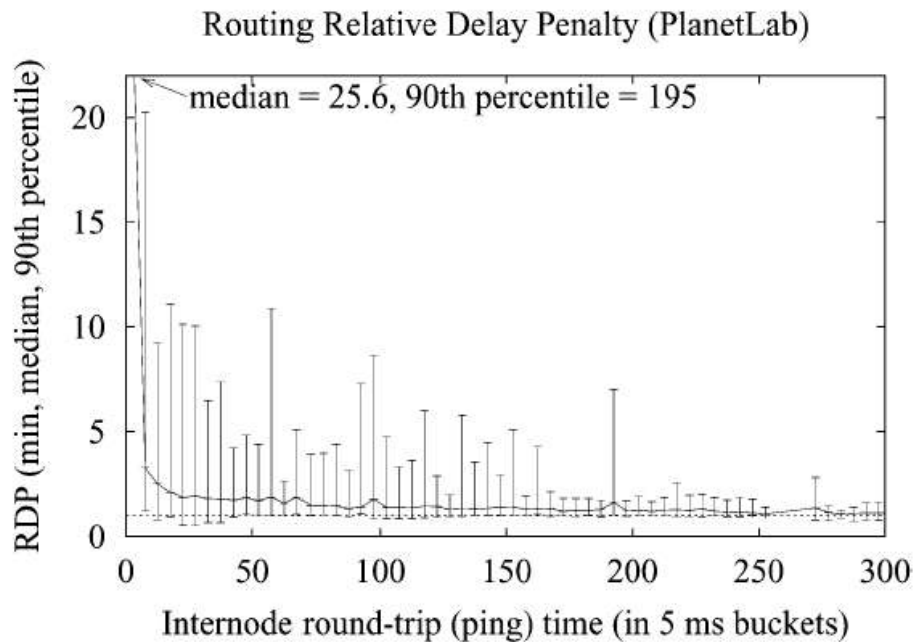
- No special mechanism
- Redundancy in routing table to minimize damage
- Periodical re-publishing

Test setup

- Design
 - Layered - neighbor-link separated from routing logic
 - Multiple applications supported on the same network
 - Some additional stabilization routines
- Implementation
 - Java-based
 - 57 kLOC, 255 source files
 - Multiple nodes per JVM
- Hardware
 - Local tests on P3 / P4 clusters
 - Wide-area tests in PlanetLab
 - A global Internet2- connected network of roughly 100 high-end servers
 - Constant load

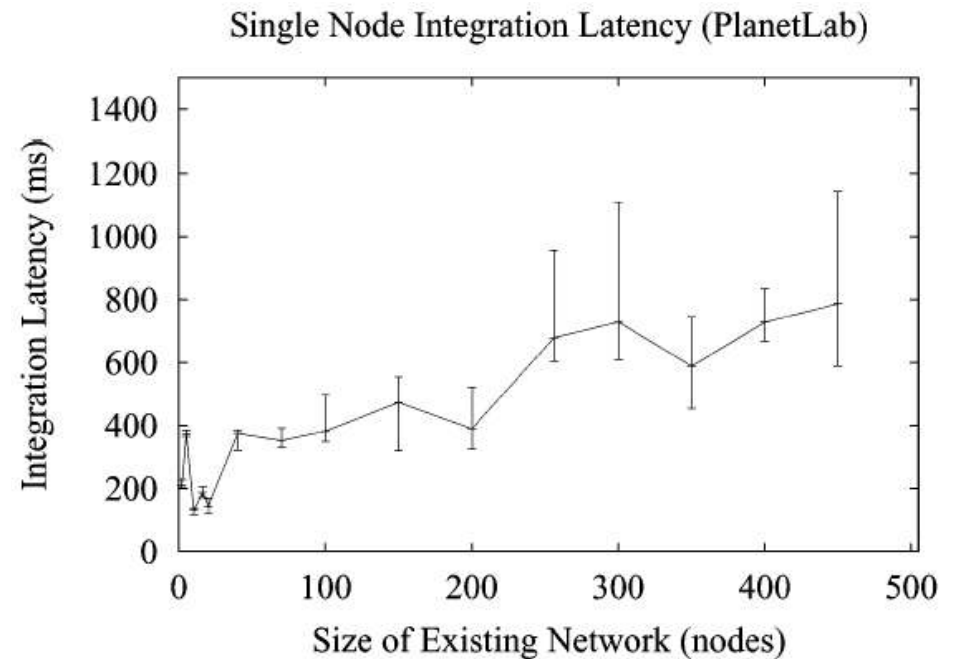
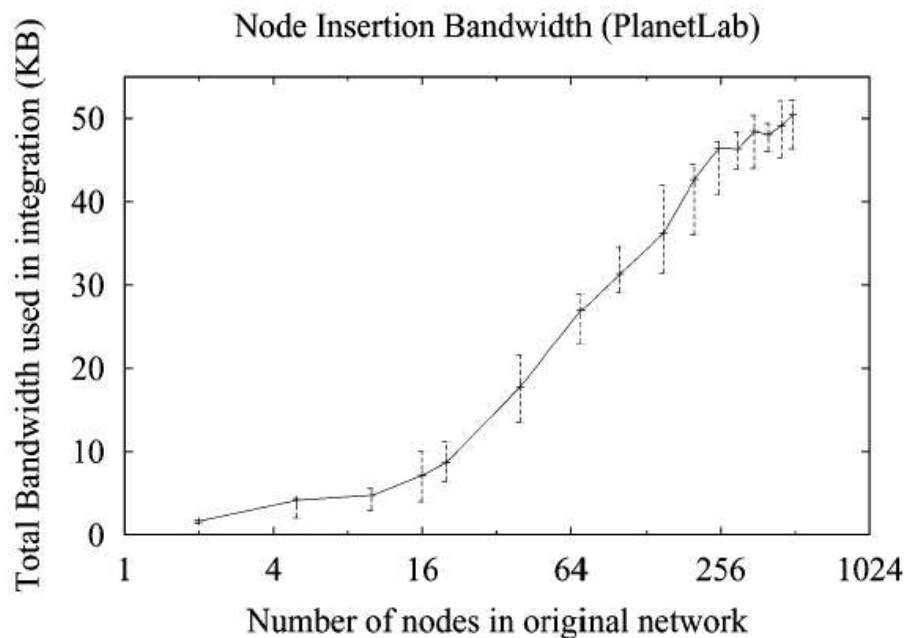
Results

- Throughput good
 - P4 capable of processing 7100 packets / s when packet size 4kb
- Relative roundtrip penalty decreases with network size toward 1
 - Locality properties
 - Optimization possible (redundant publishing)



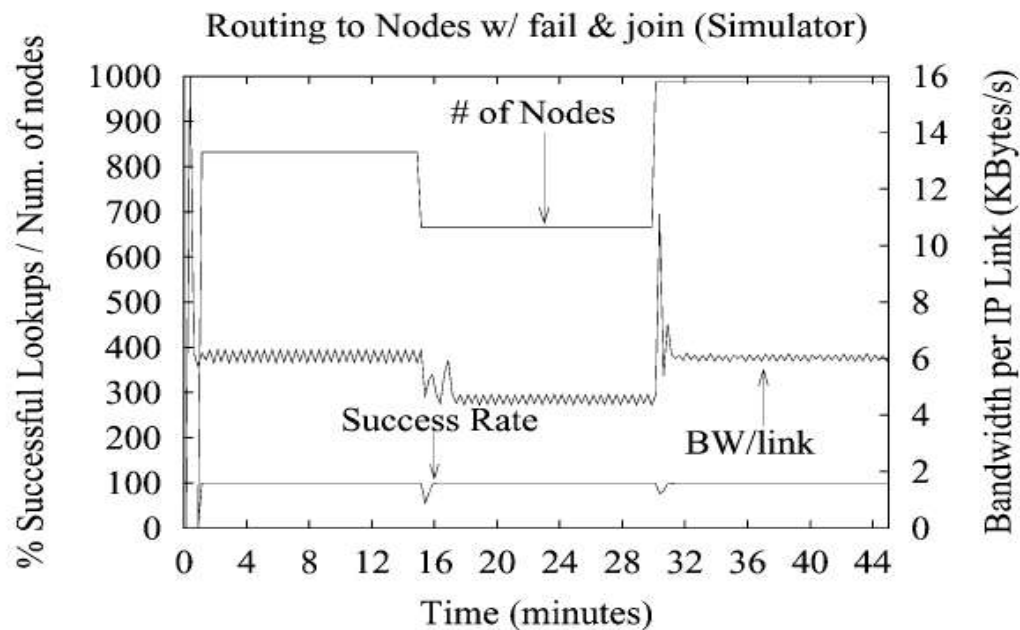
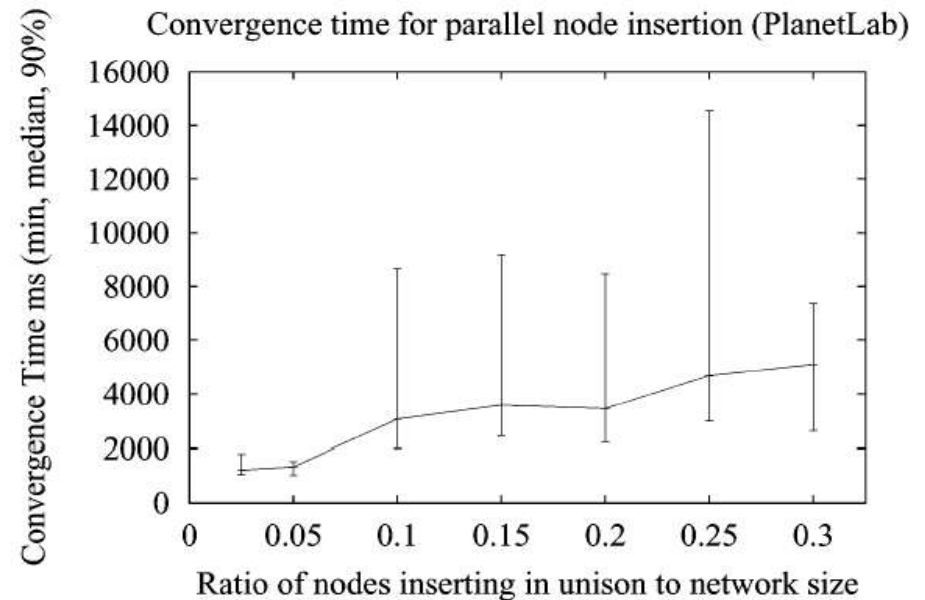
Results

- Node insertion
 - As expected, the bandwidth and time used increases with network size
 - Slowly, though



Results

- Multiple parallel inserts
 - Performs *usually* well
- Massive fluctuations
 - Stability restored quickly



Conclusions

- Scalable
 - Better performance in larger network
- Efficient
 - Routing light-weight
- Resilient
 - Redundancy protects from occasional failures
 - Fast recover from massive failures
- Observations
 - Special routing-hardware possible
 - Object-mapping lookup a problem
 - Heuristics could be used