Tapestry: Decentralized Routing and Location

System Seminar S '01
Ben Y. Zhao
CS Division, U. C. Berkeley

Challenges in the Wide-area

- Trends:
  - Exponential growth in CPU, b/w, storage
  - Network expanding in reach and b/w
- Can applications leverage new resources?
  - Scalability: increasing users, requests, traffic
  - Resilience: more components → inversely low MTBF
  - Management: intermittent resource availability → complex management schemes
- Proposal: an infrastructure that solves these issues and passes benefits onto applications

Cluster-based Applications

Advantages

- Ease of fault-monitoring
- Communication on LANs
  - Low latency
  - High bandwidth
  - Abstract away comm.
- Shared state
- Simple load balancing

Limitations

- Centralization as liability
  - Centralized network link
  - Centralized power source
  - Geographic locality
- Scalability limitations
  - Outgoing bandwidth
  - Power consumption
  - Physical resources (space, cooling)
- Non-trivial deployment

Global Computation Model

- A wish list for global scale application services
- Global self-adaptive system
  - Utilize all available resources
  - Decentralize all functionality
    no bottlenecks, no single points of vulnerability
  - Exploit locality whenever possible
    localize impact of failures
  - Peer-based monitoring of failures and resources

Driving Applications

- Leverage proliferation of cheap & plentiful resources: CPU's, storage, network bandwidth
- Global applications share distributed resources
  - Shared computation:
    - SETI, Entropia
  - Shared storage
    - OceanStore, Napster, Scale-8
  - Shared bandwidth
    - Application-level multicast, content distribution

Key: Location and Routing

- Hard problem:
  - Locating and messaging to resources and data
- Approach: wide-area overlay infrastructure:
  - Easier to deploy than lower-level solutions
  - Scalable: million nodes, billion objects
  - Available: detect and survive routine faults
  - Dynamic: self-configuring, adaptive to network
  - Exploits locality: localize effects of operations/failures
  - Load balancing

Talk Outline

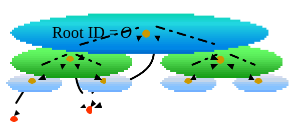
- Problems facing wide-area applications
- Previous work: Location services & PRR97
- Tapestry: mechanisms and protocols
- Preliminary Evaluation
- Sample application: Bayeux
- Related and future work

Previous Work: Location

- Goals:
  - Given ID or description, locate nearest object
- Location services (scalability via hierarchy)
  - DNS
  - Globe
  - Berkeley SDS
- Issues
  - Consistency for dynamic data
  - Scalability at root
  - Centralized approach: bottleneck and vulnerability

Decentralizing Hierarchies

- Centralized hierarchies
  - Each higher level node responsible for locating objects in a greater domain
- Decentralize: Create a tree for object *O* (really!)
  - Object *O* has its own root and subtree
  - Server on each level keeps pointer to nearest object in domain
  - Queries search up in hierarchy



Directory servers tracking 2 replicas

What is Tapestry?

- A prototype of a *decentralized, scalable, fault-tolerant, adaptive* location and routing infrastructure
- Network layer of OceanStore *(Zhao, Kubiatowicz, Joseph et al. U.C. Berkeley)*
- Suffix-based hypercube routing
  - Core system inspired by Plaxton, Rajamaran, Richa (SPAA97)
- Core API:
  - *publishObject(ObjectID, [serverID])*
  - *sendmsgToObject(ObjectID)*
  - *sendmsgToNode(NodeID)*

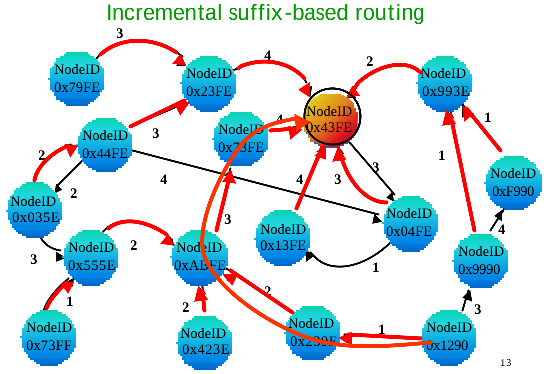
PRR (SPAA 97)

- Namespace (nodes and objects)
  - large enough to avoid collisions (~2^{160}?) (size *N* in $Log_2(N)$ bits)
- Insert Object:
  - Hash Object into namespace to get ObjectID
  - For ($i=0$, $i<Log_2(N)$, $i+j$) { //Define hierarchy
    - *j* is base of digit size used, ($j = 4$ → hex digits)
    - Insert entry into nearest node that matches on last *i* bits
    - When no matches found, then pick node matching ($i – n$) bits with highest ID value, terminate

PRR97 Object Lookup

- Lookup object
  - Traverse same relative nodes as insert, except searching for entry at each node
  - For ($i=0$, $i<Log_2(N)$, $i+n$) {
    - Search for entry in nearest node matching on last *i* bits
- Each object maps to hierarchy defined by single root
  - *f* (ObjectID)= RootID
- Publish / search both route incrementally to root
- Root node = *f* (*O*), is responsible for “knowing” object’s location

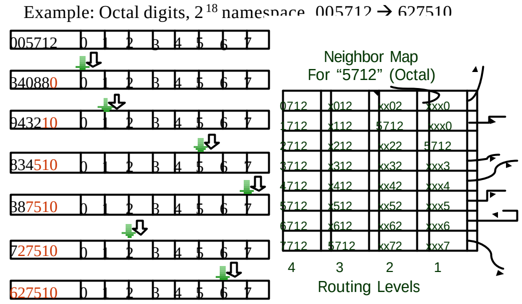
Basic PRR Mesh
Incremental suffix-based routing



13

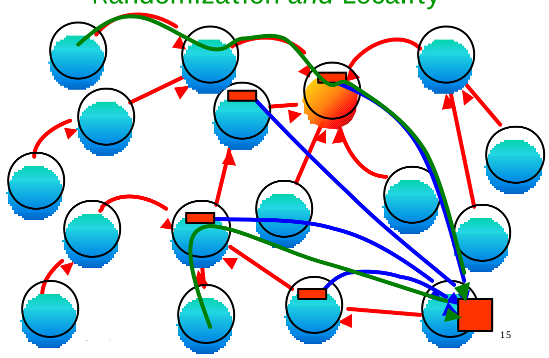
PRR97 Routing to Nodes

Example: Octal digits, 2^{18} namespace, 005712 → 627510



Ben Zhao - Tapestry @ U. W. 590 S'01 14

Use of Plaxton Mesh
Randomization *and* Locality



15

PRR97 Limitations

- Setting up the routing tables
  - Uses global knowledge
  - Supports only static networks
- Finding way up to root
  - Sparse networks: find node with highest ID value
  - What happens as network changes
    - Need deterministic way to find the same node over time
- Result: good analytical properties, but fragile in practice, and limited to small, static networks

Ben Zhao - Tapestry @ U. W. 590 S'01 16

Talk Outline

- Problems facing wide-area applications
- Previous work: Location services & PRR97
- Tapestry: mechanisms and protocols
- Preliminary Evaluation
- Sample application: Bayeux
- Related and future work

Ben Zhao - Tapestry @ U. W. 590 S'01 17

Tapestry Contributions

PRR97

- Benefits inherited by Tapestry:
  - Scalable: state: $bLog_b(N)$, hops: $Log_b(N)$ b=digit base, N= |namespace|
  - Exploits locality
  - Proportional route distance
- Limitations
  - Global knowledge algorithms
  - Root node vulnerability
  - Lack of adaptability

Tapestry

- A *real* System!
  - Distributed algorithms
    - Dynamic root mapping
    - Dynamic node insertion
  - Redundancy in location and routing
  - Fault-tolerance protocols
  - Self-configuring / adaptive
  - Support for mobile objects
- Application Infrastructure

Ben Zhao - Tapestry @ U. W. 590 S'01 18

Fault-tolerant Location

- Minimized soft-state vs. explicit fault-recovery
- Multiple roots
  - Objects hashed w/ small salts → multiple names/roots
  - Queries and publishing utilize all roots in parallel
  - P(finding Reference w/ partition) = $1 - (1/2)^n$ where n = # of roots
- Soft-state periodic republish
  - 50 **million** files/node, daily republish, $b = 16$, $N = 2^{160}$, 40B/msg, worst case update traffic: 156 kb/s,
  - expected traffic w/ 2^{40} real nodes: 39 kb/s

Fault-tolerant Routing

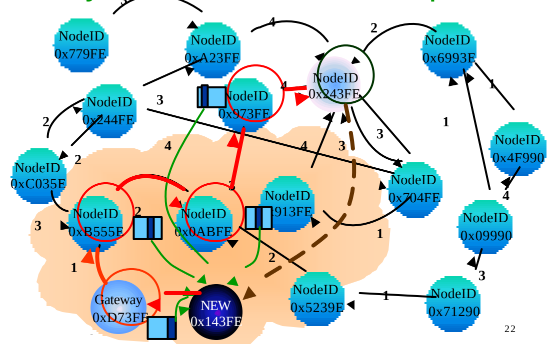
- Detection:
  - Periodic probe packets between neighbors
  - Selective NACKs
- Handling:
  - Each entry in routing map has 2 alternate nodes
  - Second chance algorithm for intermittent failures
  - Long term failures → alternates found via routing tables
- Protocols:
  - Reactive Adaptive Routing
  - Proactive Duplicate Packet Routing

Dynamic Insertion

Operations necessary for N to become fully integrated:

- Step 1: Build up N's routing maps
  - Send messages to each hop along path from gateway to current node N' that best approximates N
  - The i^{th} hop along the path sends its i^{th} level route table to N
  - N optimizes those tables where necessary
- Step 2: Move appropriate data from N' to N
- Step 3: Use back pointers from N' to find nodes which have null entries for N's ID, tell them to add new entry to N
- Step 4: Notify local neighbors to modify paths to route through N where appropriate

Dynamic Insertion Example



Summary

- Decentralized location and routing infrastructure
  - Core design from PRR97
  - Distributed algorithms for object-root mapping, node insertion
  - Fault-handling with redundancy, soft-state beacons, self-repair
- Analytical properties
  - Per node routing table size: $bLog_b(N)$
    - N = size of namespace, n = # of physical nodes
  - Find object in $Log_b(n)$ overlay hops
- Key system properties
  - Decentralized and scalable via random naming, yet has locality
  - Adaptive approach to failures and environmental changes

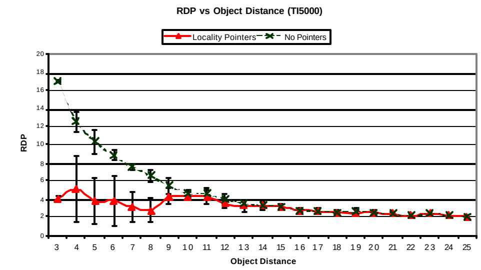
Talk Outline

- Problems facing wide-area applications
- Previous work: Location services & PRR97
- Tapestry: mechanisms and protocols
- Preliminary Evaluation
- Sample application: Bayeux
- Related and future work

Evaluation Issues

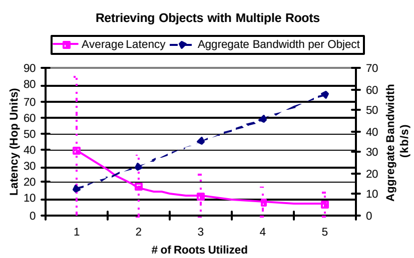
- Routing distance overhead (RDP)
- Routing redundancy → fault-tolerance
  - Availability of objects and references
  - Message delivery under link/router failures
  - Overhead of fault-handling
- Optimality of dynamic insertion
- Locality vs. storage overhead
- Performance stability via redundancy

Results: Location Locality



Measuring effectiveness of locality pointers (TIERS 5000)

Results: Stability via Redundancy



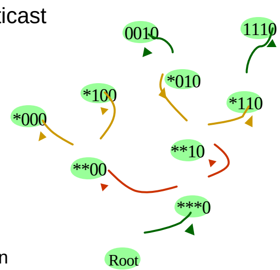
Parallel queries on multiple roots. Aggregate bandwidth measures b/w used for soft-state republish 1/day and b/w used by requests at rate of 1/s.

Talk Outline

- Problems facing wide-area applications
- Previous work: Location services & PRR97
- Tapestry: mechanisms and protocols
- Preliminary Evaluation
- Sample application: Bayeux
- Related and future work

Example Application: Bayeux

- Application-level multicast
- Leverages Tapestry
  - Scalability
  - Fault tolerant data delivery
- Novel optimizations
  - Self-forming member group partitions
  - Group ID clustering for better b/w utilization



Related Work

- Content Addressable Networks
  - Ratnasamy et al., (ACIRI / UCB)
- Chord
  - Stoica, Morris, Karger, Kaashoek, Balakrishnan (MIT / UCB)
- Pastry
  - Druschel and Rowstron (Rice / Microsoft Research)

Future Work

- Explore effects of parameters on system performance via simulations
- Explore stability via statistics
- Show effectiveness of application infrastructure
  Build novel applications, scale existing apps to wide-area
  - Silverback / OceanStore: global archival systems
  - Fault-tolerant Adaptive Routing
  - Network Embedded Directory Services
- Deployment
  - Large scale time-delayed event-driven simulation
  - Real wide-area network of universities / research centers

For More Information

Tapestry:
  http://www.cs.berkeley.edu/~ravenben/tapestry
OceanStore:
  http://oceanstore.cs.berkeley.edu
Related papers:
  http://oceanstore.cs.berkeley.edu/publications
  http://www.cs.berkeley.edu/~ravenben/publications

ravenben@cs.berkeley.edu

Backup Nodes Follow…

Dynamic Root Mapping

- Problem: choosing a root node for every object
  - Deterministic over network changes
  - Globally consistent
- Assumptions
  - All nodes with same matching suffix contains same null/non-null pattern in next level of routing map
  - Requires: consistent knowledge of nodes across network

PRR Solution

- Given desired ID *N*,
  - Find set *S* of nodes in existing network nodes *n* matching most # of suffix digits with *N*
  - Choose S_i = node in *S* with highest valued ID
- Issues:
  - Mapping must be generated statically using global knowledge
  - Must be kept as hard state in order to operate in changing environment
  - Mapping is not well distributed, many nodes in *n* get no mappings

Tapestry Solution

- Globally consistent distributed algorithm:
  - Attempt to route to desired ID N_i
  - Whenever null entry encountered, choose next "higher" non-null pointer entry
  - If current node *S* is only non-null pointer in rest of route map, terminate route, $f(N) = S$
- Assumes:
  - Routing maps across network are up to date
  - Null/non-null properties identical at all nodes sharing same suffix

Analysis

Globally consistent deterministic mapping

- Null entry → no node in network with suffix
- ∴ consistent map → identical null entries across same route maps of nodes w/ same suffix

Additional hops compared to PRR solution:

- Reduce to coupon collector problem
  Assuming random distribution
- With $n * ln(n) + cn$ entries, P(all coupons) = $1-e^{-c}$
- For $n=b$, $c=b-ln(b)$,
  P(b^2 nodes left) = $1-b/e^b = 1.8 * 10^{-6}$
- # of additional hops ≅ $Log_b(b^2) = 2$

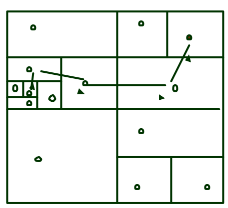
Distributed algorithm with minimal additional hops

Dynamic Mapping Border Cases

- Two cases
  - A. If a node disappeared, and some node did not detect it.
    - Routing proceeds on invalid link, fails
    - No backup router, so proceed to surrogate routing
  - B. If a node entered, has not been detected, then go to surrogate node instead of existing node
    - New node checks with surrogate after all such nodes have been notified
    - Route info at surrogate is moved to new node

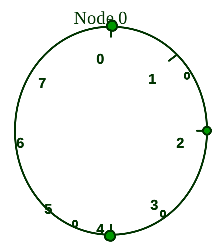
Content-Addressable Networks

- Distributed hashtable addressed in *d* dimension coordinate space
- Routing table size: $O(d)$
- Hops: expected $O(dN^{1/d})$
  - N = size of namespace in d dimensions
- Efficiency via redundancy
  - Multiple dimensions
  - Multiple realities
  - Reverse push of "breadcrumb" caches
  - Assume immutable objects



Chord

- Associate each node and object a unique ID in *uni-dimensional space*
- Object O stored by node with highest ID < O
- Finger table
  - Pointer for next node 2^i away in namespace
  - Table size: $Log_2(n)$
  - n = total # of nodes
- Find object: $Log_2(n)$ hops
- Optimization via heuristics



Pastry

- Incremental routing like Plaxton / Tapestry
- Object replicated at *x* nodes closest to object's ID
- Routing table size: $b(Log_bN)+O(b)$
- Find objects in $O(Log_bN)$ hops
- Issues:
  - Does not exploit locality
  - Infrastructure controls replication and placement
  - Consistency / security

Key Properties

- Logical hops through overlay per route
- Routing state per overlay node
- Overlay routing distance vs. underlying network
  - Relative Delay Penalty (RDP)
- Messages for insertion
- Load balancing

Comparing Key Metrics

Properties	Tapestry	Chord	CAN	Pastry
– Parameter	Base b	None	Dimen d	Base b
– Logical Path Length	$Log_b N$	$Log_2 N$	$O(d*N^{1/d})$	$Log_b N$
– Neighbor-state	$bLog_b N$	$Log_2 N$	$O(d)$	$bLog_b N+O(b)$
– Routing Overhead (RDP)	$O(1)$	→ $O(1)$	$O(1)$?	$O(1)$?
– Messages to insert	$O(Log_b^2 N)$	$O(Log_2^2 N)$	$O(d*N^{1/d})$	$O(Log_b N)$
– Mutability	App-dep.	App-dep	Immut.	???
– Load-balancing	Good	Good	Good	Good

Designed as P2P Indices