A Decentralized Location and Routing Infrastructure for Fault-tolerant Wide-area Network Applications

Ph.D. Qualifying Examination Proposal

Ben Yanbin Zhao
Computer Science Division
University of California, Berkeley
ravenben@cs.berkeley.edu

Abstract

Today's Moore's Law increases in computational power and network bandwidth, combined with the increasing reach of networks into diverse environments and devices, offer new opportunities to and stretch the bounds of traditional network applications. More specifically, new challenges of scalability, fault-tolerance and manageability stretch the limits of the communication components of applications. In this work, we propose the use of a global-scale routing and location infrastructure that leverages the abundant computational and network resources to facilitate a decentralized wide-area computing model. We present an infrastructure prototype named Tapestry*, and demonstrate its usefulness with novel large-scale network applications. Finally, we evaluate it by measuring its effectiveness against current challenges, and the resource costs these benefits incur in tradeoffs.*

1 Introduction

The advent and proliferation of large scale networking has created incredible opportunities for traditional computing applications. The trends towards wireless connectivity and networking to small devices has made an ever-growing audience available to network applications. The wide reach of the Internet has also made it possible for applications to gather data and communicate them over the wide-area, in a variety of service models.

Wide-area Internet applications are facing several key challenges as a result of their scale. In addition to the scalability necessary to handle a large number of users, such applications must also be able to operate despite the increasing likelihood of failures in the wide-area, and provide mechanisms by which it can self-adapt to changing conditions in the network environment. When an application is scaling up from a centralized server model to the wide-area model, the key component affected is the communication layer. In particular, the location and routing aspects of a network application are those most prone to issues of scale, failures, and adaptivity.

Faced with these challenges, many existing services make use of existing solutions that provide partial solutions, sometimes at the cost of imposing limitations on future growth and scalability. These approaches fall under three categories, those based on the cluster-based computing model [13, 25, 14], centralized location and routing [16, 24], and optimistic location and routing [8]. As we will argue, none of these approaches completely solves the combination of issues confronting wide-area network applications.

Hypothesis

This work proposes a global-scale infrastructure for location and routing based on the use of decentralized data structures. By using novel data structures, we can provide the functionality of centralized networking infrastructure without any potential bottlenecks or single points of failure. In addition, we propose the use of redundancy leveraging current Moore's Law growth trends in computational power and network bandwidth, in order to provide improved performance and stability of performance. By stability we mean decreasing variability in system performance, easily indicated by a reduction in standard deviation. We believe by abstracting the complexity necessary to achieve these properties into an application layer, we greatly simplify the construction of network applications well-suited for wide-area operation.

Our work is driven by three key system design goals:

Scalability. First, we aim to provide unconstrainted scalability in the infrastructure. We expect that the large volume of data handled by wide-area applications implies that any point of centralization will become both a performance bottleneck and point of failure. By utilizing a completely decentralized mesh-like data structure, we avoid such centralization points and distribute load evenly across the network. The result is a system design that scales linearly with the amount of resources available, and exhibits graceful performance degradation when system-wide resources are low.

Fault-tolerance. Second, we design our infrastructure to provide redundancy at each level of operation, so that the it will utilize all resources in order to provide uninterrupted operation. In addition to surviving link and node failures, network partitions, and communication congestion, our goal is to leverage the available system redundancy to mask variations in performance, providing dependable and uniform performance to the application layer.

Self-maintenance and Adaptation. Finally, we strive to design our system to be responsive to changes in environmental conditions. Components should detect the availability and unavailability of resources, and automatically integrate or disengage individual nodes as appropriate to optimize overall system performance. Furthermore, such integration and separation should occur with minimal disturbance to the rest of the system.

We claim that such a decentralized global-scale location and routing application infrastructure is feasible, and can greatly enhance the construction of wide-area, distributed network services while providing a simple and powerful application programming interface. In this proposal, we propose a design for such a decentralized infrastructure called Tapestry, demonstrate its novelty by introducing several applications leveraging its infrastructure, and evaluate the system by how well it addresses our design goals, and the overhead costs we incur in providing those properties.

2 Motivation

Our work is chiefly motivated by the trend towards increase of scale in network applications, and the resulting implications. Today's Internet is expanding rapidly, both in reach, by providing connectivity to thin clients such as PDAs and Internet appliances, and in scale, by increasing available bandwidth to existing links. These conditions are fueling the introduction of new services enabled by wide-area large bandwidth links such as peer to peer file-sharing networks, and the expansion of existing network services such as DNS and application-level multicast.

2.1 Challenges

We believe scaling to the wide-area network poses three key challenges to applications: *scalability*, *fault-tolerance*, and *manageability*.

- Successful network applications can expect a large user base, generate large volumes of traffic and pushing the limits of servers handling requests. All components of the application must scale in order to meet such heavy demands.

- A large scale network service is likely to span a large number of infrastructure components, including routers, links, servers, and end clients. As the number of components increases, we can expect the *mean time between failures* (MTBF) of the overall system to drop. For example, an application running on a network running 200 routers, each with a MTBF of 5 years, can expect on average a router failure every nine days (5 years / 200 = 9.13 days). In other words, a successful large scale application must expect failures to be commonplace, and provide mechanisms and protocols to provide service in their presence.

- As an application scales, its overall performance becomes dependent on the individual performance of many heterogeneous devices and environmental variables. Managing such a system is near impossible. This argues for the application to detect environmental changes and self-tune for better performance. It needs to be self-managing, detecting and utilizing new resouces, and detecting the removal of resources and relocating load to adjust.

2.2 Clustering vs. Decentralization

Recent work has shown Networks of Workstations [2] can be used to provide fault-tolerance and data persistence as an application infrastructure [13, 25, 14]. The use of cluster-based application infrastructures provides an interesting solution to the previously mentioned challenges. The use of clusters abstracts away communication costs, eliminates concern for communication errors, and simplifies fault-prevention by geographically localizing nodes. It greatly simplifies application construction, presenting to the developer the abstraction of a single, incrementally scalable server.

This programming model has its limitations, however. By colocating servers geographically and in the network, we allow local events to have a large impact on overall system performance. For instance, a UPS failure, a single network partition, a fire or natural disaster will bring down the cluster, and with it the entire service. Furthermore, localized network effects such as flaky routers or active Denial of Service attacks on the outgoing link will impact the entire cluster adversely. Additionally, the scalability of clusters is limited by considerations such as power, cooling, and outgoing bandwidth. Finally, the placement of clusters determines its overall proximity to all clients in the network, which may not be desirable for network applications with a geographically distributed user group.

To gain true fault-tolerance and resilience to environmental artifacts, we believe a system needs to be redundant at every layer: storage, computation, and communication. This level of redundancy can be easily achieved by utilizing servers in a geographically and network distributed fashion. Therefore, we propose the use of a wide-area, decentralized network of workstations as an alternative to the current clustering model. We link together large number of individual servers with varying computational, storage, and network resources into a widely-distributed, well-connected virtual machine. In this model, workers that perform the same common task are identified in the application layer with identical names. Communication between nodes handling different tasks in the same application occurs with nodes first sending messages to the nearest node handling some specific task, then establishing a session. The initial location and subsequent message routing is handled by a shared location and routing layer, which also provides resilience against communication failures and reports node failures to the application layer.

Widely-distributed virtual computing has been explored in previous systems such as the Pirahna system [6] and the SETI program [26]. Applications using this programming model gain a number of interesting benefits, largely unexplored in previous systems. First, because of the distributed nature of servers, there is a good chance any client has server within some reasonable network distance. This locality can be exploited for faster response time to the client, or used for application-level local optimizations. Second, the distributed nature of servers means any external event's impact on the overall service is limited to the local nodes affected. This is a powerful property that provides resilience to both random failures and intentional attacks. Next, by spreading work out to many "worker" nodes, the overall system can treat each node as an independent entity and single unit of failure. A failure in any subcomponent is reflected in its end-to-end performance, and can be used by the overall system to load-balance on the granularity of a single request. Finally, lack of deployment constraints means more resources can be deployed more cheaply, and we assert that the application can exploit the resulting redundancy for better and more predictable performance.

2.3 Routing and Location

There are a large group of applications poised to take advantage of this fully distributed computing model. Peer to peer file sharing systems such as Gnutella [3], Freenet [8] and Napster [12], global-scale storage systems such as Farsite [5], PAST [10], and OceanStore [18] are some prominent examples. Yet it is evident that for these applications (and the decentralized application model) to succeed, a critical component, the network location and routing layer must be provide several key functions.

In the context of the decentralized application model, the location layer performs several key functions. In general, the location layer provides the "find nearest" functionality necessary to minimize communication paths between nodes and between nodes and clients in the application. Once a communication path has been established, the location layer provides fault-tolerant message routing in the absence of a network partition. Furthermore, the location and routing layer functions as a complete entity with global knowledge of the system, and transparently redirects messages and requests to functioning worker nodes whenever possible.

We now examine the impact of our wide-area challenges on an application's location and routing sublayer. By allowing workers to come and go according to resource availability and failure cases in the decentralized model, we rely on the location and routing sublayer to address issues caused by network and application scale. For instance, the system can scale to handle increasing requests by adding more nodes, and scalability is reflected in the volume of data and messages flowing across networks links. The location and routing layer provides fault-tolerance by circumventing link and router failures, and adapts to changing network latencies and intermittent network partitions. To accomplish these goals while allowing ease of deployment, we choose an overlay approach based on nodes running a routing and location protocol in the application layer.

Finally, to concretize our requirements for such an location and routing layer, we briefly examine previous approaches in service location services and optimistic location and routing. Previous work in service location [28, 15, 16, 17] has been limited in scalability to the wide-area. The Globe location system [28] and the Berkeley Wide-area Service Discovery Service [16] are two examples of hierarchical structured location services which saw data explosion at the root level servers. The Berkeley WSDS proposes lossy compression at the higher-level servers at the cost of allowing false positives on queries. We learned from these projects that as the volume of data scales, any point of centralization will become a scalability bottleneck. This leads us to believe that a completely decentralized system could provide a solution. Furthermore, the problem our proposed infrastructure tackles is an easier one, since we already have the name of the object we are searching for, whereas the service location work has an additional step of semantically searching for an unique service name given a set of attributes. We believe this simplification reduces the scalability problem to a manageable one, and we can provide a scalable solution while excluding false positives.

A somewhat related approach is optimistic routing and location, such as the scheme used in the Gnutella [3] and Freenet [8] data distribution networks. While these approaches are decentralized, they suffer other limitations. The Gnutella scheme uses local flood queries to find data, which is not scalable to the wide-area. In the Freenet scheme, queries can be forwarded in an unconstrained manner, with no guarantee of success. In comparison, we would like our location and routing layer to be decentralized, scalable with no points of centralization, and provide definitive responses within a reasonable, constrained amount of time.

3 Decentralized Routing and Location

The key infrastructure component enabling the wide-area distributed computing model is the decentralized location and routing layer. For ease of deployment, we chose to take an application-level approach. In this section, we focus in on routing and location, state the assumptions for our desired infrastructure, explain its application interface, and discuss the relevant research issues.

3.1 Assumptions

To tackle the challenges of scalability, fault-tolerance, and manageability, we have chosen to examine an approach using a decentralized, application-level routing and location layer. We make several assumptions that clarify the design space:

- The application-level infrastructure resides on a large, well-connected network backbone with subnetworks, much like the current structure of the Internet.

- There exists an underlying protocol layer such as IP which provides unreliable packet delivery, wide-area routing to specific network addresses, and mechanisms to monitor simple network characteristics (such as Ping or Traceroute on IP).

- Nodes participating in the infrastructure are *cooperative* under normal operating conditions. That is, they are willing to perform simple tasks on behalf of other nodes in the system. Such tasks may include forwarding messages or storing location mappings on other nodes' behalf.

- Node and object names are determinstic mappings of some unique characteristic into a randomly distributed namespace. The namespace should be sufficiently large to ensure names are unique with no collisions. To provide these properties, we choose a 160-bit bit sequence as the namespace, and specify that names are derived by applying a secure one-way hashing function such as SHA-1 [23] a hashkey, either the public key of the node or the content of the object. If such a hashkey is not unique to the node, then use a combination of the hashkey and a known salt.

- The overall infrastructure agrees on a consistent view of algorithms used in routing and location, as well as system parameters such as base of representation of node and object names.

3.2 Application Interface

The main functionality provide by our decentralized application infrastructure is to publish/advertise objects, search for objects and route messages to them. All messages are assumed to be one way, and sent in asynchronous mode. Response messages are recognized as such by the application layer. The core API calls are very simple:

- *void PublishObject(String ObjectNameID, String SelfNodeID)*
- *void RouteClosestObject(String ObjectNameID)*
- *void RouteNode(String NodeNameID)*

We augment the core API with functions which leverage the additional benefits a decentralized location and routing layer offers. We specify methods to leverage redundancy available in the location system, and methods on integrating and disengaging nodes from the overlay network.

- *void PublishRedundantNames(String OrigObjName, Integer RedundancyLvl)*
- *void RouteClosestObject(String OrigObjID, Integer RedundancyLvl)*
- *void RouteAllObjects(String OrigObjID, Integer RedundancyLvl)*
- *void RouteSubsetObjects(String OrigObjD, Integer MinimumObjects, Integer RedundancyLvl)*
- *void IntegrateSelfNode(String DesiredNodeID)*
- *void RemoveSelfNode()*

3.3 Research Issues

We have specified a general framework for a decentralized location and routing infrastructure, and define here a number of issues to be investigated.

Overlay Routing Distance vs. Overhead As the case with any overlay network, routing over our decentralized infrastructure will not be as efficient as routing on the underlying network layer. For any decentralized scheme we choose, a key metric of evaluation is how efficient the overlay route is when compared to the closest path on the underlying network (in most cases IP). This ratio of distances is often referrred to as the *Relative Delay Penalty* or RDP, and is first introduced in [7]. There is an implicit tradeoff between the RDP of an overlay, and the state kept at each node for routing purposes. We are interested applying these metrics to different overlay schemes, in hope of finding a routing scheme which achieve both low routing storage overhead and low routing RDP.

Location Locality vs. Overhead An analogous measure of efficiency versus overhead exists in the location phase of our infrastructure. We define the metric of LRDP, *Location Relative Delay Penalty*, as the ratio of the distance traveled by a message in locating and routing to a given object to the distance between the client and the object in the underlying network layer. We believe that where scalability can be preserved, correctness is requirement of any wide-area location system. By correctness we mean guaranteed successful search for an existing object under normal (non-failure or low-failure) operating conditions, and the absence of any "false positive" matches, where the system reports the object to be at a location where it is not. Given the correctness requirement, we wish to closely examine the tradeoff between storage overhead for advertising location of objects, and the LRDP metric.

Cost of Fault-tolerance and Availability As specified in our design goals, our decentralized infrastructure needs to provide a variety of mechanisms to detect, repair, and operate in the presence of link/node failures and data corruption. For any such infrastructure, we need to measure the true level of fault-tolerance it provides, i.e. how frequent/many of different faults can the infrastructure withstand while providing degraded service, and what is the level of performance degradation suffered as a result. We wish to explore the tradeoff between the level of fault-tolerance provided and the cost in redundancy it requires. The costs involved often appear as redundant bandwidth required to detect and circumvent failures, or as redundant storage to secondary routes or replicas of location pointers.

Performance Stability Through Redundancy Given the large number of external variables which impact system performance in the growing wide-area network, including localized network congestion and intermittent faults, we expect the performance of any single request to be erratic. We propose to stabilize the expected erratic system performance as seen by the end client, by utilize the abundant computational, storage and network resources available in the near future for redundancy in our main location and routing mechanisms. By issuing redundant requests and make use of the "best" response in performance or correctness, we hope to explore the tradeoff between level of redundancy utilized and the level of performance stability gained. Furthermore, we wish to explore a variety of approaches to utilizing the redundancy, and examine where they each lie on the tradeoff between redundancy utilized and stability or fault-resilience gained.

Node Entry/Exit Overhead Finally, our network infrastructure should be self-maintaining and adaptive to changes in the network. Part of this self-maintenance involves the detection of failures and the self-repair mechanisms examined in our investigation of fault-tolerance mechanisms. The other aspect of self-maintenance applies to the process of integrating a new node into the overlay infrastructure, and the process of disengaging exiting nodes from the infrastructure. Both involve communication with other nodes in the infrastructure. We wish to study the various mechanisms and algorithms that allow seamless integration and disengagement, and evaluate them in terms of latency, resources used, and susceptibility to failures.

4 Prototype Design and Analysis

As part of our preliminary work into this area, we have designed and implemented via simulation a decentralized wide-area location and routing infrastructure we call *Tapestry*. The Tapestry infrastructure uses a decentralized data structure much like a mesh of trees rooted at every node. A similar mesh structure was first introduced by Plaxton, Rajamaran and Richa in [21]. Details on the original Plaxton mesh design and the Tapestry infrastructure can be found in our technical report [31].

4.1 Plaxton and Tapestry

To summarize, the Plaxton mesh structure specifies a routing algorithm which incrementally approaches the desti-

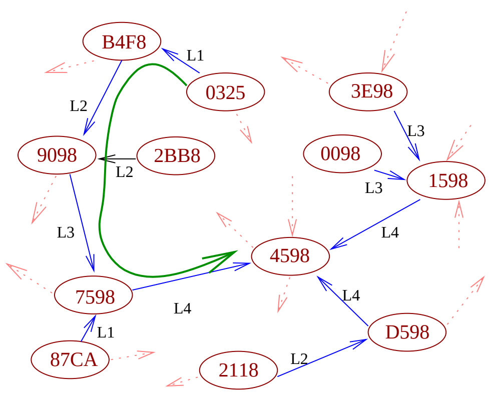

Figure 1: *Example of Plaxton mesh routing.* Node 0325 is routing to node 4598 in a mesh using hexadecimal digit representation.

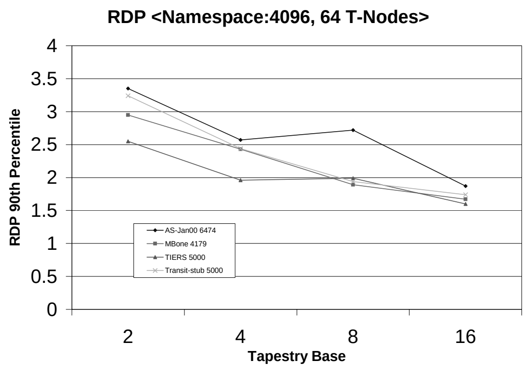

Figure 2: *RDP Simulation on 4 Topologies.* We plot here the average RDP between all node-pairs on four different topologies.

nation node ID digit by digit with each additional hop. Figure 1 shows an illustration of how the routing algorithm works. Node `0325` is routing incrementally towards node `4598`, by routing to nodes which share suffixes of increasing lengths with the destination node. The Plaxton mesh is scalable: each hop contains routing state logarithmically proportional to the size of the entire namespace; a route to node takes $Log_b(N)$ hops through the overlay network, where b is the base of representation used in the namespace, and N is the size of the overall namespace.

Plaxton et al. also introduced a decentralized location mechanism based on this mesh, whereby an object O residing on server S would advertise itself by routing an advertisement to its root node R. At each intervening hop, the advertisement would store a location mapping of $< O, S >$ on local storage. A message from a client destined for object O routes first towards R, and redirects to S when it encounters either R or one of the intervening hops storing a cache of O's location mapping. The root node R is a random node deterministically chosen to maintain a consistent and current mapping of the location of O. A simple way of determining a root node would be to choose the node in the system whose ID shares the longest suffix with object O's ID. Plaxton chooses a scheme which uses global knowledge to make the object to root node mapping.

The original scheme presented by Plaxton et al. suffers from several constraints, the most serious of which is its dependence on global knowledge. It also has little innate redundancy, and performance may be fragile in the presence of failures.

Our prototype, the Tapestry location and routing infrastructure, implements a number of enhancements above and beyond the original proposal by Plaxton et al. They include enhancements to add redundancy to both routing and location phases, protocols for detecting and self-repairing from failures, and deterministic distributed algorithms for routing and node insertion. For more details, we refer the reader to [31].

4.2 Preliminary Evaluation

We present some initial evaluation results of the Tapestry infrastructure, on several fronts including routing overhead, availability, location locality tradeoff, and performance stability. These results are derived from a centralized packet-level simulator of Tapestry running on well-known topologies.

To avoid topology specific results, we performed the experiments on a representative set of real and artificial topologies, including two real networks (AS-Jan00, MBone) and two artificially generated topologies (TIERS, Transit-stub). The AS-Jan00 graph models the connectivity between Internet autonomous systems (AS), where each node in the topology represents an AS. It was generated by the National Laboratory for Applied Network Research [NLA] based on BGP tables. The MBone graph was collected by the SCAN project at USC/ISI in 1999, and each node represents a MBone router. The TIERS graph includes 5000 nodes, and was generated by the TIERS generator. Finally, we used the GT-ITM [30] package to generate the transit-stub graph.

Routing Overhead Recall that one of the metrics we wanted to measure was overlay routing distance. Given our choice of suffix-based incremental routing, we want to measure the RDP ratio of overlay routing as compared to underlying IP. In Figure 2, we see that for all of our topologies, the RDP decreases as we increase the base of the Tapestry ID digits. This is intuitive, since a larger base

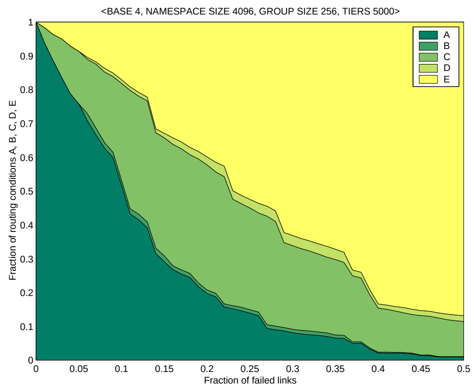

Figure 3: *Routing Redundancy in Tapestry.* A probability plot comparing the likelihood of successfully reaching the destination node as a function of link failure rate. The regions correspond to success by each routing method. Here Tapestry routing redundancy level is 3.

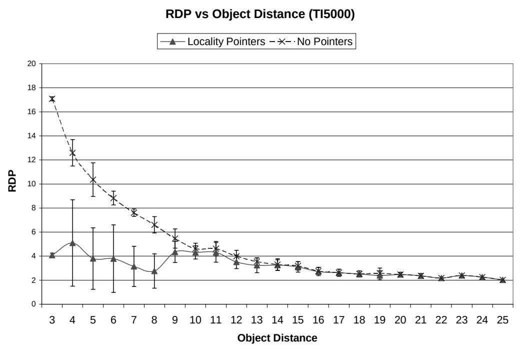

Figure 4: *Location Pointers Effect on RDP: TIERS 5000:* Ratio of network hops traveled via Tapestry (with and without intermediate location pointers) over network hops via IP.

implies fewer digits in the ID representation, which implies fewer Tapestry hops. Each Tapestry hop then goes further in the IP network space, and more closely approximates IP routing distances. This comes at a tradeoff of fewer logical hops, which more routing is left to IP, and Tapestry components make less of an impact on overall performance. We do note that for base 4, which generates a reasonable length ID, the 90th percentile of RDPs is around 2-3 for all topologies. This result is very good, since we know from Lenore Cowen's work on compact routing [9] that very few routing algorithms with sublinear storage at each node can route with worst case RDP less than 3.

Routing Redundancy We also wanted to study the effects of routing redundancy on fault-tolerance and availability. In Figure 3 we show the results of simulating a routing comparison between Tapestry routing with level 3 redundancy (each original forwarding pointer has 2 backup pointers) and IP routing. The simulation is run on a network of 5000 nodes in the TIERS topology [1], and both routing algorithms are applied to all pairwise routes in a group of 256 randomly chosen nodes. As we increase the rate of link failures in the network, we show the success rate of using Tapestry routing versus IP. By success for IP we mean whether the original path in the BGP tables can reach the destination, and by success for Tapestry we mean whether the primary route or backup routes can be taken to reach the destination node. These are so defined because BGP routing tables can potentially take on the order of minutes to converge after a failure [19], whereas Tapestry fault-tolerant protocols operate on the order of milliseconds.

In the Figure, region A denotes the probability that both IP and Tapestery successfully reach a reachable destination. Region B represents the probability that IP succeeds, where Tapestry fails. Region C shows cases where IP fails, and Tapestry succeeds. Region D shows where both fail to reach a reachable destination, and region E is the probability that the destination is unreachable by any route. As the figure shows, where the destination is reachable from the client, Tapestry almost always succeeds in reaching the destination. The improvement in availability over native IP is very significant. Also note that even when half of all nodes have failed, Tapestry successfully routes to 10% of all possible routes. This demonstrates that a redundancy of 3 (2 backup routes) is more than sufficient to provide near-optimal fault-tolerance against link failures.

Location Locality vs. Overhead We continue with experiments [2] which examine the effectiveness of the location pointers stored at intermediate hops between the storage server and the root node of the object. While Plaxton, Rajamaran and Richa prove that locating and routing to an object with these location pointers incurs a small linear factor of overhead compared to routing using the physical layer [21], we would like to confirm the theoretical results. We ran our experiments on a packet level simulator of Tapestry using unit-distance hop topologies, and measured *Location Relative Delay Penalty (LRDP)*, the ratio of distance traveled via Tapestry location and routing, versus that traveled via direct routing to the object.

[1] We chose to use the TIERS topology because it offered longer route paths than the alternatives.

[2] All error bars shown as part of graphs in this section show one stardard deviation above and below the data point.

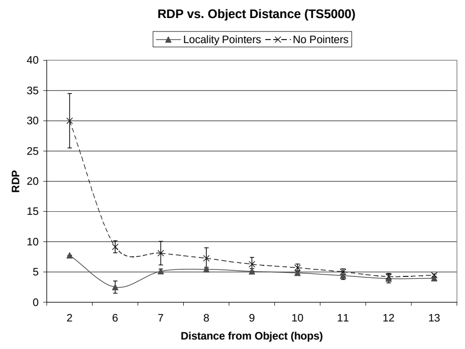

Figure 5: *Location Pointers Effect on RDP: Transit-stub 5000:* Location RDP of Tapestry (with and without intermediate location pointers).

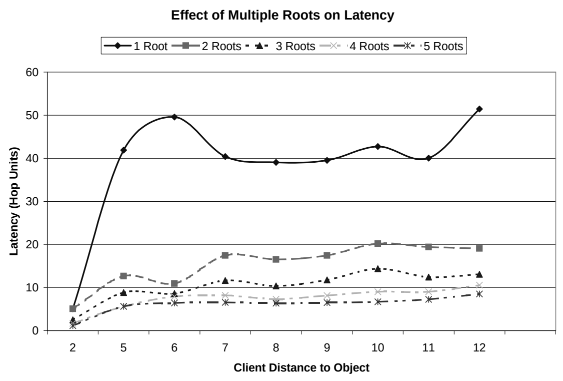

Figure 6: *Effect of Multiple Roots on Location Latency:* Time taken to find and route to object as a function of client distance from object's root node and number of parallel requests made. (Error bars omitted for clarity.)

The results from experiments on all four topologies show the same trend. Here we show results from the two representative topologies, TIERS 5000 nodes and Transit-stub 5000 nodes, in Figures 4 and 5. The results largely confirm what we expected, that the presence of locality pointers helps maintain the LRDP at a small relatively constant factor, and their absence results in a large number of hops to the root node and large LRDP values.

Performance Stability Through Redundancy Finally, we present here some preliminary results on exploiting redundancy for performance stability in the use of multiple root nodes for object location. To remove the single point of failure that a root node presented in Plaxton location, Tapestry advertises each object with several different IDs, derived from hashing the original object ID together with several salts (e.g. natural numbers 1, 2, 3). Object

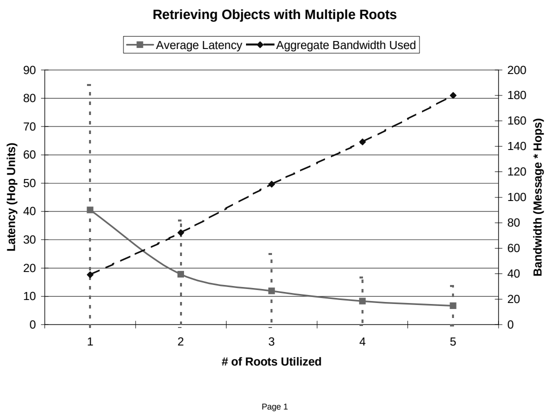

Figure 7: *Multiple Roots and Aggregate Bandwidth Used:* Time to find and route to object graphed with aggregate bandwidth as a function of number of parallel requests made.

references gain the fault-tolerance of multiple roots while reusing the same routing infrastructure. Incoming queries can utilize this redundancy by sending out multiple requests in parallel, greatly improving the chances that one of them will return in a timely and predictable manner. We show the result of simulating such redundant requests here.

We used the packet level simulator on 4096 Tapestry nodes on a 5000 node Transit-stub topology, applying a memoryless distribution to hop latencies. We assign 5 random IDs to a randomly placed object, and publish its presence using each ID. For the first experiment, we show latency taken to find and route to the object from randomly placed clients as a function of both number of parallel requests issued and distance between clients and the root object. The resulting Figure 6 shows several key features. First, for all clients not immediately next to the root node, increasing the number of parallel requests drastically decreases response time. Second, the most significant latency decrease occurs when two requests are sent in parallel, with the benefit deceasing as more requests are added. Finally, a smaller number of requests shows a more jagged curve, showing their vulnerability to random effects such as long hop latencies. In contrast, those factors are hidden in a smoothed curve when all five requests are issued in parallel.

In our second experiment shown in Figure 7, we present a new perspective on the same experiment in conjunction with aggregate bandwidth used. We plot the location and routing latency and aggregate bandwidth against the number of parallel requests. The chart confirms artifacts observed in Figure 6, including the significant decrease in latency and variability with each one additional request. Furthermore, by plotting the aggregate bandwidth used on the secondary Y-axis, we see that the greatest benefit in latency and stability can be gained with minimal bandwidth overhead.

4.3 Alternate Approaches

4.3.1 Pastry

PAST [10] is a recent project begun at Microsoft Research focusing on peer-to-peer anonymous storage. The PAST routing and location layer, called Pastry [11], is a location protocol sharing many similarities with Tapestry. Key similarities include the use of prefix/suffix address routing, and similar insertion/deletion algorithms, and similar storage overhead costs.

There are several key differences that distinquish Pastry from Tapestry. First, objects in PAST are replicated without control by the owner. Upon "publication" of the object, it is replicated and replicas are placed on several nodes whose nodeIDs are closest in the namespace to that of the object's objectID. Second, where Tapestry places references to the object location on hops between the server and the root, Pastry assumes that clients use the objectID to attempt to route directly to the vicinity where replicas of the object are kept. Placing actual replicas at different nodes in the network brings storage overhead at multiple servers, and brings up questions of security, confidentiality, and consistency. Also, Pastry routing's analogy of Tapestry's "surrogate routing" algorithm (see Section 3.3 in [31]) provides weaker analytic bounds on the number of logical hops taken. In Tapestry, we have analytically proven, well-defined, probabilistic bounds in routing distances, and are guaranteed to find an existing reachable object (see Section 3 in [31]). Finally, Pastry's location mechanism calls for a sequence of routes, even if objects are nearby the client. This means that the system does not exploit locality well, and may incur a high value of RDP for local objects.

4.3.2 Chord

Several recent projects at MIT are closely related to Tapestry, Pastry and CAN. The Chord [27] project provides an efficient distributed lookup service, and uses a logarithmic-sized routing table to route object queries. The focus is on providing hashtable-like functionality of resolving key-value pairs. For a namespace defined as a sequence of m bits, a node keeps at most m pointers to nodes which follow it in the namespace by 2^1, 2^2, and so on, up to 2^{m-1}, modulo 2^m. The i_{th} entry in node n's routing table contains the first node that succeeds n by at least 2^{i-1} in the namespace. Each key is stored on the first node whose identifier is equal to or immediately follows it in the namespace. Chord provides similar logarithmic storage and logarithmic logical hop limits as Tapestry, but provides weaker guarantees about worst-case performance. The main distinction worthy of note is that there is no natural correlation between overlay namespace distance and network distance in the underlying network, opening the possibility of extremely long physical routes for every close logical hop. This problem is partially alleviated by the use of heuristics.

Several other projects from MIT are also relevant. First, Karger et. al. presented a decentralized wide-area location architecture for use with geographic routing in GRID [20]. GRID uses a notion of embedded hierarchies to handle location queries scalably, much like the embedded trees in Tapestry. Second, the Intentional Naming System (INS) [1] combines location and routing into one mechanism. Finally, Resilient Overlay Networks [4], leverages the GRID location mechanism and the semantic routing of the Intentional Naming System (INS) [1] to provide fault-resilient overlay routing across the wide-area. Because of the scalability of INS, however, the RON project focuses on networks of size less than 50 nodes.

4.3.3 CAN

The "Content Addressable Networks" (CAN) [22] work is being done at AT&T Center for Internet Research at ICSI (ACIRI). In the CAN model, nodes are mapped onto a N-dimensional coordinate space on top of TCP/IP in a way analogous to the assignment of IDs in Tapestry. The space is divided up into N dimensional blocks based on servers density and load information, where each block keeps information on its immediate neighbors. Because addresses are points inside the coordinate space, each node simply routes to the neighbor which makes the most progress towards the destination coordinate. Object location works by the object server pushing copies of location information back in the direction of the most incoming queries.

There are several key differences between CAN and Tapestry. In comparison, Tapestry's hierarchical overlay structure and high fanout at each node results in paths from different sources to a single destination converging quickly. Consequently, compared to CAN, queries for local objects converge much faster to cached location information. Furthermore, Tapestry's use of inherent locality paired with introspective mechanisms means it allows queries to immediately benefit from query locality, while being adaptive to query patterns and allowing consistency issues to be handled at the application layer. CAN assumes objects are immutable, and must be reinserted once they change their values. Finally, Tapestry node organization uses local network latency as a distance metric, and has been shown to be a reasonable approximation of the underlying network. CAN, however, like Chord, does not attempt to approximate real network distances in their topology construction. As a result, logical distances in

CAN routing can be arbitrarily expensive, and a hop between neighbors can involve long trips in the underlying IP network. The main advantage a CAN has is that because of the simplicity of the node addition algorithm, it can better adapt to dynamically changing environments such as sensor networks.

In summary, Pastry, Chord and CAN are very similar to Tapestry in their functionality and run-time properties. In particular, Pastry is the closest analogy offering "locating and routing" to an object, where Chord and CAN both focus on providing distributed hashtable functionality. Because Pastry controls replica placement, and Chord and CAN are not optimized for large objects, Tapestry is the only system which allows the user to control the location and consistency of the original data, allowing the system to manipulate and control only references to the object for performance. It is also noteworthy that Tapestry and Pastry have natural correlation between the overlay topology and the underlying network distance, while CAN and Chord may incur high physical hop counts for every logical hop. Finally, Pastry's lack of location pointers means requests do not exploit locality, and requests to local objects may incur high RDP overhead.

5 Applications

The main goal of building a decentralized location infrastructure is to allow applications on top to transparently inherit its desirable properties. We have implemented several novel applications on top of our prototype Tapestry to test this hypothesis, including a scalable and fault-tolerant application level multicast system and a wide-area archival system.

Bayeux Bayeux [32] is an efficient application-level multicast system that scales to thousands of members, incurs minimal delay and bandwidth penalties, and handles faults in both links and routing nodes. It uses a single source model, and leverages the hierarchical nature of Tapestry routing to distribute packets with minimal packet duplication. Each node in the dissemination tree handles a partition of the receiver group, and sends at most one packet to each of its forward routes. For example, a node handling all listeners ending in `1` forwards a packet to neighbors who represent one more digit (e.g. `21`, `41`), iff there are listeners with that suffix. In addition, Bayeux packet delivery leverages Tapestry's routing redundancy to circumvent failures. Bayeux also utilizes the "find nearest" semantics of Tapestry location to allow replication of multicast root nodes. Nodes joining any multicast session can utilize Tapestry to transparently subscribe to the nearest root node.

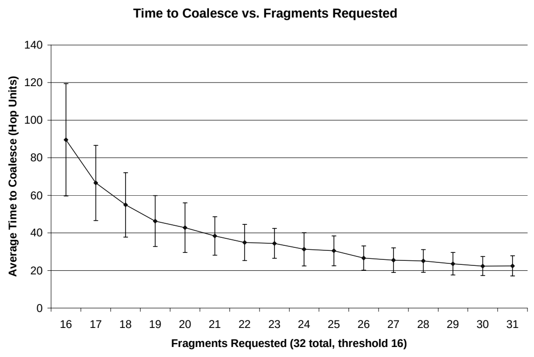

Figure 8: *Reconstruction Latency*: This chart shows latency required to receive enough fragments for reconstruction in a Transit-stub network of 4096 Tapestry nodes.

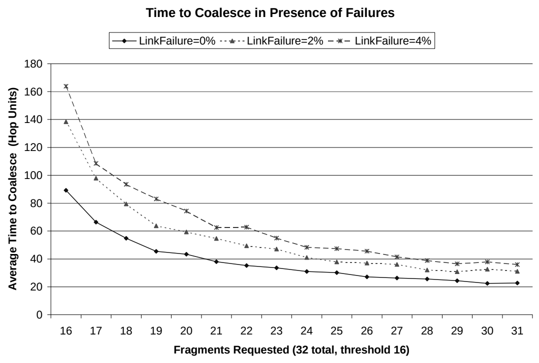

Figure 9: *Reconstruction with Failures*: This chart shows simulated time necessary for block reconstruction in a Transit-stub network of 4096 Tapestry nodes, retrying after link failures.

Silverback The Silverback archival system [29] is a key component of the OceanStore global-scale storage architecture. Its key contribution is the use of erasure-codes to intelligently disseminate persistent data into widely distributed fragments for extreme durability. Erasure codes can efficiently encode a document into small blocks, such that some relatively small threshold of them can be used to reconstruct the original document. Silverback names all fragments from the same block with a single block ID, and utilizes Tapestry to efficiently locate any n blocks for quick reconstruction of archival data.

The kind of redundancy that erasure-coding offers on fragments allows us to explore trading bandwidth via redundancy for performance and performance stability. In Figure 8, we show that in a packet level simulator of Tapestry with memoryless distribution on hop latencies and randomly placed queuing delays, when we request more fragments than the necessary reconstruction threshold, we see the expected decrease in time to receive the first threshold fragments, as well as a significant decrease in performance variability. Furthermore, when we begin to allow for link failures, we see that a small level of redundancy in requests reaps a great gain in error-prone networks, such as those we expect in the future wide-area.

6 Research Methodology and Plan

We will evaluate the success of our work on several factors. First, we want to measure the effectiveness of the decentralized wide-area computing model by building additional applications on top of our Tapestry prototype. Second, we will use simulation on large topologies to further explore the tradeoffs between Tapestry system parameters and various performance metrics. We will also use our Tapestry prototype and its applications as a platform to test our proposal of trading available resources for performance stability. Finally, we seek to understand the relationship between Tapestry and its alternatives Chord, CAN and Pastry, and to map out a taxonomy of the decentralized routing and location design space.

We will design and implement new wide-area applications on top of the Tapestry infrastructure, and demonstrate how these applications can leverage Tapestry properties to better support wide-area operation. We seek to design novel applications newly enabled by a decentralized routing and location layer, such as a network-embedded service discovery service. We also want to demonstrate that with the help of Tapestry, traditional services such as those running on clusters or single nodes can be reconfigured for wide-area operation with minimal effort.

We also seek to further explore through simulation various tradeoffs exposed by the decentralized wide-area computing model. First we need to fully understand the impact of system parameters on the Tapestry infrastructure. Then we will utilize Tapestry applications to explore our proposed tradeoff of utilizing plentiful resources via redundancy in order to gain improved performance and performance stability. We also want to see how applicable this desired tradeoff is to wide-area applications in general.

We have also begun to study the alternatives in decentralized routing structures such as Chord and CAN. By further study, experimentation, and cooperation with other researchers, we will better understand the relationship between design differences in these systems and their resulting performance characteristics. We hope to make progress towards a taxonomy of wide-area routing and location techniques, and place the current systems into their place in the design space.

6.1 Outside the Scope of This Work

Our proposal focuses on providing an application infrastructure that addresses the issues of scalability, fault-tolerance, and manageability. While such an infrastructure can provide a natural basis for addressing issues of security attacks, and in particular Denial of Service attacks, it is beyond the scope of this work. Nor do we intend to address the issues of streaming media and content distribution. Finally, while the Tapestry infrastructure, like Chord, Pastry and CAN, is ideal for building Peer to Peer systems, we will save any such efforts for future work.

6.2 Research Timeline

Phase 1 (0-4 months)

- Continue to explore the performance of the Tapestry prototype, examining in detail how system parameters such as base of the ID representation and the level of redundancy affect performance metrics including: locality effects, routing overhead, and performance stability.
- Make concrete two approaches to exploiting routing redundancy, and quantify their performance characteristics via simulation.
- Finish an implementation of the dynamic insertion algorithm
- Consider additional mechanisms as inspired by the work on consistent hashing, in order to simplify the insertion algorithm and gain additional analytical properties.

Phase 2 (4-8 months)

- Leverage the Markov-chain based link characterization work done by Konrad et al. to implement fault-tolerance via link behavior prediction.
- Design and implement the network-embedded service discovery protocol
- Evaluate the dynamic insertion algorithm and its implementation
- Begin writing dissertation

Phase 3 (8-12 months)

- Finish writing dissertation
- Travel and attend employment interviews
- Graduate

7 Conclusion

The decentralized wide-area location and routing infrastructure is essential to the wide-area distributed computing model. By abstracting the complexities of fault-tolerance, scalability and self-maintenance into the application layer, our Tapestry prototype allows easy construction of network applications which operate well in the wide-area network. Initial measurements show that Tapestry successfully provides these properties to their applications with reasonable overhead. Much remained to be explored within the space of decentralized location and routing. In the near future, we seek to build additional wide-area applications, utilize them to better understand system tradeoffs, and make progress towards understanding a taxonomy of the decentralized location and routing research space.

References

[1] ADJIE-WINOTO, W., SCHWARTZ, E., BALAKRISHNAN, H., AND LILLEY, J. The design and implementation of an intentional naming system. In *Proceedings of ACM SOSP* (December 1999), ACM.

[2] ANDERSON, T. E., CULLER, D. E., AND PATTERSON, D. A. A case for NOW (network of workstations). *IEEE Micro 15*, 1 (February 1995), 54–64.

[3] ANONYMOUS. What is gnutella? `http://www.gnutellanews.com/information/what_is_gnutella.shtml`.

[4] BALAKRISHNAN, H., KAASHOEK, M. F., AND MORRIS, R. Resilient overlay networks. `http://nms.lcs.mit.edu/projects/ron/`.

[5] BOLOSKY, W. J., DOUCEUR, J. R., ELY, D., AND THEIMER, M. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *Proc. of SIGMETRICS* (2000), ACM, pp. 34–43.

[6] CARRIERO, N., GELERNTER, D., KAMINSKY, D., AND WESTBROOK, J. Adaptive parallelism with piranha. Tech. Rep. 954, Yale University, February 1993. `http://www.cs.yale.edu/Linda/ap.html`.

[7] CHU, Y., RAO, S. G., AND ZHANG, H. A case for end system multicast. In *Proceedings of ACM SIGMETRICS* (June 2000), pp. 1–12.

[8] CLARKE, I., SANDBERG, O., WILEY, B., AND HONG, T. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability* (New York, 2001), H. Federrath, Ed., Springer.

[9] COWEN, L. J. Compact routing with minimum stretch. In *Proceedings of ACM-SIAM SODA* (January 1999), ACM.

[10] DRUSCHEL, P., AND ROWSTRON, A. Past: Persistent and anonymous storage in a peer-to-peer networking environment. Submission to ACM HOTOS VIII, 2001.

[11] DRUSCHEL, P., AND ROWSTRON, A. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. Submission to ACM SIGCOMM, 2001.

[12] FANNING, S. Napster. `http://www.napster.com`.

[13] FOX, A., GRIBBLE, S. D., CHAWATHE, Y., BREWER, E. A., AND GAUTHIER, P. Cluster-based scalable network services. In *Proceedings of SOSP* (Saint-Malo, France, October 1997), vol. 16, ACM.

[14] GRIBBLE, S. D., WELSH, M., VON BEHREN, R., BREWER, E. A., CULLER, D., BORISOV, N., CZERWINSKI, S., GUMMADI, R., HILL, J., JOSEPH, A., KATZ, R. H., MAO, Z. M., ROSS, S., AND ZHAO, B. The ninja architecture for robust internet-scale systems and services. *IEEE Computer Networks* (2000). Special Issue on Pervasive Computing.

[15] GUTTMAN, E., PERKINS, C., VEIZADES, J., AND DAY, M. Service Location Protocol, Version 2. IETF Internet Draft, November 1998. RFC 2165.

[16] HODES, T. D., CZERWINSKI, S. E., ZHAO, B. Y., JOSEPH, A. D., AND KATZ, R. H. An architecture for a secure wide-area service discovery service. *Wireless Networks* (2000). Special Issue of Selected Papers from MobiCom 1999, Under Revision.

[17] HOWES, T. A. The Lightweight Directory Access Protocol: X.500 Lite. Tech. Rep. 95-8, Center for Information Technology Integration, U. Mich., July 1995.

[18] KUBIATOWICZ, J., BINDEL, D., CHEN, Y., EATON, P., GEELS, D., GUMMADI, R., RHEA, S., WEATHERSPOON, H., WEIMER, W., WELLS, C., AND ZHAO, B. OceanStore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS* (November 2000), ACM.

[19] LABOVITZ, C., AHUJA, A., ABOSE, A., AND JAHANIAN, F. An experimental study of delayed internet routing convergence. In *Proceedings of SIGCOMM* (August 2000), ACM.

[20] LI, J., JANNOTTI, J., COUTO, D. S. J. D., KARGER, D. R., AND MORRIS, R. A scalable location service for geographic ad hoc routing. In *Proceedings of ACM MOBICOM* (August 2000), ACM.

[21] PLAXTON, C. G., RAJARAMAN, R., AND RICHA, A. W. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of ACM SPAA* (June 1997), ACM.

[22] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SCHENKER, S. A scalable content-addressable network. Submission to ACM SIGCOMM, 2001.

[23] ROBSHAW, M. J. B. MD2, MD4, MD5, SHA and other hash functions. Tech. Rep. TR-101, RSA Laboratories, 1995. version 4.0.

[24] ROSENBERG, J., SCHULZRINNE, H., AND SUTER, B. Wide area network service location. IETF Internet Draft, November 1997.

[25] SAITO, Y., BERSHAND, B. N., AND LEVY, H. M. Manageability, availability and performance in porcupine: a highly scalable, cluster-based mail service. In *Proceedings of SOSP* (December 1999), ACM, pp. 1–15.

[26] SPACE SCIENCES LABORATORY, UNIVERSITY OF CALIFORNIA AT BERKELEY. Search for extraterrestrial intelligence at home. `http://setiathome.ssl.berkeley.edu`.

[27] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. Submitted to ACM SIGCOMM, 2001.

[28] VAN STEEN, M., HAUCK, F. J., HOMBURG, P., AND TANENBAUM, A. S. Locating objects in wide-area systems. *IEEE Communications Magazine* (January 1998), 104–109.

[29] WEATHERSPOON, H., WELLS, C., EATON, P. R., ZHAO, B. Y., AND KUBIATOWICZ, J. D. Silverback: A global-scale archival system. Submitted for publication to ACM SOSP, 2001.

[30] ZEGURA, E. W., CALVERT, K., AND BHATTACHARJEE, S. How to model an internetwork. In *Proceedings of IEEE INFOCOM* (1996).

[31] ZHAO, B. Y., KUBIATOWICZ, J. D., AND JOSEPH, A. D. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB/CSD-01-1141, University of California at Berkeley, Computer Science Division, April 2001.

[32] ZHUANG, S. Q., ZHAO, B. Y., JOSEPH, A. D., KATZ, R. H., AND KUBIATOWICZ, J. D. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the 11th International Workshop on Network and Operating System Support for Digital Audio and Video* (June 2001), ACM.