A Directory Enabled Middleware Framework for Distributed Systems

Jehan Wickramasuriya , Qi Han & Nalini Venkatasubramanian
Dept. of Information & Computer Science
University of California, Irvine, CA 92697-3425, USA
{jwickram,qhan,nalini}@ics.uci.edu

Abstract

In order to achieve the goals of a next generation service-based Internet, we need to provide high performance, high capacity, secure and reliable services that can be rapidly scaled and managed. The ability to manage and share information on large-scale network resources with a growing number of users in a secure and efficient manner is desirable. Distributed systems allow services, endsystems and protocols to dynamically attach to and detach from the distributed environment. In turn it is desirable that these services are able to operate in a safe, concurrent manner while possibly sharing common resources; that is to say they are *composable*. The development of flexible, scalable and customizable directory services can be used as an enabling technology for the development of next generation composable middleware frameworks. In this paper we use an object-based access control framework as an example of a directory-enabled middleware architecture and discuss some of the tradeoffs involved in obtaining consistent information from the directory in an efficient manner. We also discuss implementation and performance issues pertaining to the directory service, in the context of a composable, middleware framework (CompOSE|Q) being developed at the University of California, Irvine.

1 Introduction

In the future, large scale ubiquitous computing environments will consist of diverse applications executing on heterogeneous devices, systems and networks. Managing the evolution of such large scale systems requires efficient repositories of system and application level information that can be used to allocate resources and effectively and provide application requirements such as reliability, security and QoS. Such information repositories, also termed directory services, will form the crux of effective middleware for dynamic distributed events. Directory services provide the advantage of managing resources and decreasing administrative costs by centralizing the control and management of service delivery (though the content itself may be distributed). Directory services hold system, user and object level information that can be used by various services such as resource-provisioning, information collection, security, and location management. Directory services provide seamless access of this information to both system components and end-users and go far beyond what are commonly known as 'name servers'.

In this paper we present a directory-enabled approach to designing distributed systems middleware. We illustrate several examples of how DSs can be used to manage distributed systems and networks. A key challenge of middleware for ubiquitous computing environments lies in the need to add, remove or re-locate various system components/services without interfering with ongoing services and applications. Directory services play a vital role in composing the various "ilities" (reliability, security, mobility, quality-of-service) - i.e allowing simultaneous execution of multiple protocols and services in a safe and non-interfering manner. The DS can be considered a "core" service that provides a single point of management in accessing a possibly distributed range of content. For example, directory services can be used to provide transparent access to mobile resources regardless of where they are located. In order to provide secure and efficient access to distributed resources, the DS can help track and manage user access patterns and changing security levels. This allows us to manage trust assumptions between varying security domains, filter and approximate data for information collection for adequate QoS and further optimize performance.

As a specific case study of how to manage several "ilities" using directory services, we present an adaptive security service for a mobile object environment. The directory service is a vital part of such an architecture since it becomes the repository for access control information in the system, and the timely and consistent retrieval of this information is an essential part of correct execution. As such, there are tradeoffs that must be made between accuracy of directory information and update overhead costs that must be considered in engineering directory management protocols. For example,

performing delegation and revocation in a highly mobile, distributed environment is a non-trivial task. Particularly, revoking access rights in a 'real-time' manner can be difficult to achieve, especially if the object in question is constantly migrating. Hence, providing accurate location information in order to carry out the revocation in a timely manner becomes vital; the directory accesses and caching mechanisms must be optimized. Further non-interference requirements for the access control architecture as well as details of the implementation and design can be found in [19], here we focus on the capabilities themselves and the role of the directory in enabling some access control related operations. We examine these issues in the context of CompOSE|Q a composable, customizable middleware framework being developed at the University of California, Irvine.

CompOSE|Q: A QoS-enabled Customizable Middleware Framework

The customizable and safe distributed systems middleware infrastructure currently being developed at UC Irvine, called CompOSE|Q (Composable Open Software Environment with QoS) [16], has the ability to provide cost-effective QoS-based distributed resource management. The primary distinguishing feature of CompOSE|Q is that it provides composable distributed resource management, i.e., it allows the concurrent execution of multiple resource management policies in a distributed system in a safe and correct manner. This will allow safe integration of mechanisms for activities such as directory services, load balancing, caching, fault tolerance and end-to-end QoS. Within the CompOSE|Q framework, we will support mechanisms such as QoS-based load management, scheduling, resource discovery etc. to guarantee differentiated levels of service for various traffic streams.
CompOSE|Q is based on a semantic model, known as the Two-Level MetaArchitecture (TLAM) [18] that supports specification and reasoning about components of open distributed systems and interactions among these components. Using a model of distributed objects, Actors, we have earlier defined a two level meta-architectural framework for distributed resource management that permits customization of resource management policies. To ensure non-interference among system services, CompOSE|Q contains a group of core resource management services that are used as a basis for creating more complex protocols and services:

1. Remote Creation - used as a basis to design algorithms for migration and replication to provide fault tolerance, dependability and load balancing.

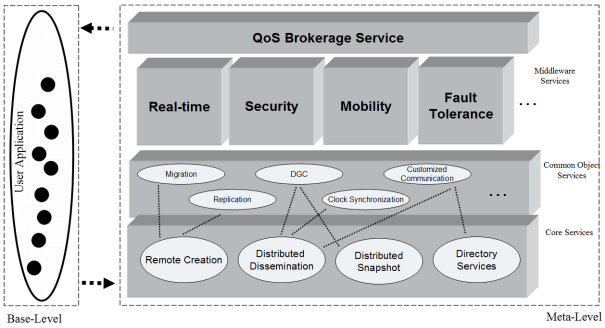


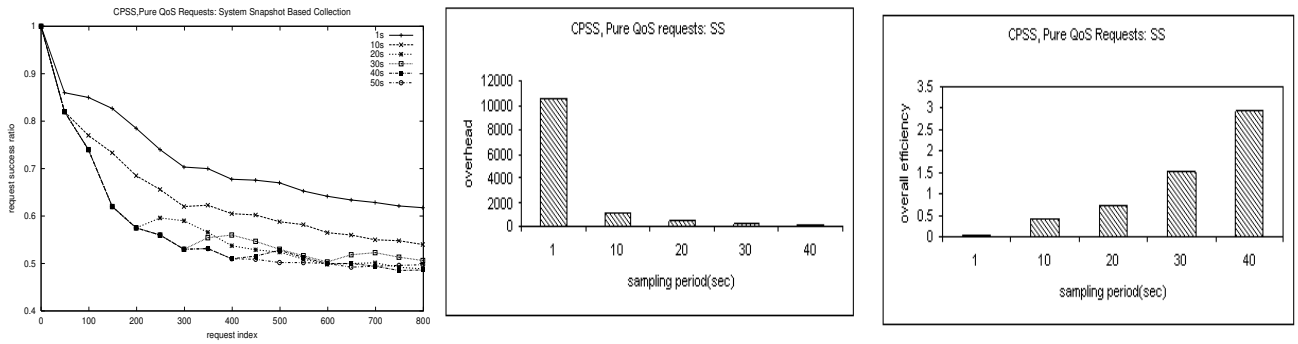
Figure 1: The CompOSE|Q high-level architecture.

2. Distributed Snapshot - used for activities such as checkpointing, distributed garbage collection.

3. Directory Services - used to define access control, naming and security services. To provide composability amongst various resource management services, CompOSE|Q ensures correctness (safety, liveness and non-interference) in interactions amongst the services. Specifically, we have studied interactions between the core services to enable efficient implementations of migration, replication and load balancing strategies while ensuring the maintenance of global state information captured by the snapshot. Figure 1. illustrates the general architecture of CompOSE|Q and shows how higher level functionality such as a QoS brokerage can be obtained by using general services within the framework.

2 Directory-Enabled Middleware Services

The use of directory service-based solutions as an enabling technology for emerging application (e.g. web, mobile device, desktop) areas on the Internet is growing at a rapid rate. The key reason for this is the need to support hundreds of thousands of users within these networked infrastructures. Allowing the middleware to interact with the directory service allows developers to effectively create a variety of directory-enabled applications quickly and easily. This also helps de-couple context or policy from the application logic, and allows easier handling of interactions between a variety of general purpose services within the middleware (as oppose to analysis on a per-application basis). Directory services play an important role in middleware frameworks, and are utilized in a number of different areas in the CompOSE|Q architecture.



A.request success ratio B. overhead C. overall performance efficiency

Figure 2: *Tradeoff between directory service accuracy and its maintenance overhead*

Resource Discovery for Load Management and Admission Control

An important issue in managing the directory information is that of maintaining accurate and current system state information. In highly dynamic environments, where system conditions are constantly changing, it is critical to provide effective approximations of the current system state. Integrated resource discovery and reservation mechanisms are critical to admission control and QoS-compliant load management; such mechanisms require current network and system conditions for effective performance. The directory service in CompOSE|Q maintains approximations of application and system level information such as:

- The replica set for data objects in the system with attributes that map a replica to the server location, status of an ongoing replication, popularity of replica object etc.
- Client request information with status of current ongoing requests and their resource requirements.
- Server and link load status (server load factors, delay and bandwidth on a link)
- Client location information (for mobile clients) that will help in predictive resource provisioning for moving objects.

Using the above information, we have proposed a number of policies for effective resource discovery. Utilizing current server load information and popularity of data objects, we have designed predictive data placement schemes that dynamically decide when, where and how many replicas of data objects will be created [17]. State information has previously been used in QoS routing techniques to improve the network utilization by balancing the load among the individual network links [2, 4]. Similarly, server selection policies have been developed that direct the user to the "best" replica/server while statically treating the network path leading from the client to the server as pre-determined by the routing tables, even though there may exist multiple alternative paths [17, 7]. While the above techniques can independently achieve some degree of load balancing, in multimedia environments, where the applications are highly sensitive to QoS parameters like bandwidth and delay, high-level provisioning mechanisms are required to address problems such as route selection and server selection problem in a unified way. Performance studies have revealed that integrated mechanisms can potentially achieve higher system-wide utilization, and therefore allow more concurrent users.

In a highly dynamic environment, cost-effective resource discovery and QoS provisioning techniques must be able to tolerate some information imprecision and be able to work effectively with approximate system state information held within the directory service. We have studied several techniques to address combined path and server selection (CPSS) problems in highly dynamic multimedia environments [5]. The optimized resource discovery techniques are designed to ensure effective utilization of network and server resources while tolerating imprecision in system state information. In addition, adaptive scheduling of requests to replicas using policies such as dynamic request migration, client and proxy staging use directory information to obtain request mappings, proxy server resource conditions and network resource availabilities. Components within the framework implement the optimized scheduling policies as well as collect/update the network and server parameters using a directory service. In addition, we study multiple techniques for efficiently updating the directory service with system state information and evaluate the performance of the QoS provisioning policies (e.g. CPSS) under different update mechanisms.

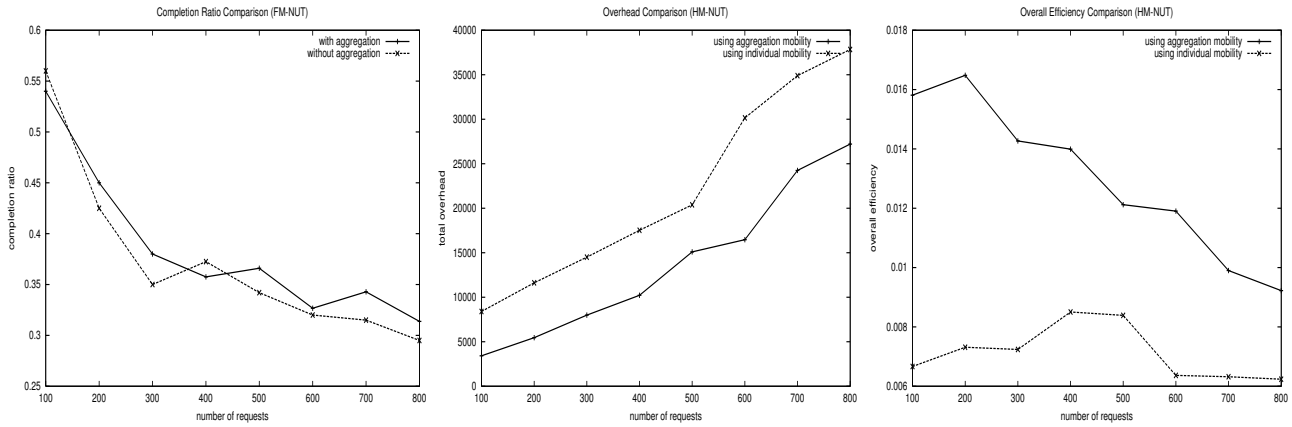


Figure 3: **Storing Aggregate vs. Individual Mobility Information in Directory Service**

Figure 2 illustrates the tradeoff between system performance and the directory service maintenance overhead. Here, we depict an instantaneous snapshot-based information collection mechanism where the directory service is updated periodically with current sampled state values. We notice that a shorter update period results in a higher request success ratio (due to a more accurate directory service) than a larger update period. However, a shorter update period implies that more probes are initiated for sampling. This causes increased sampling and directory update overheads resulting in low performance efficiency. With a larger update period, the snapshot policy causes decreased request success ratios since the policy retains a single (inaccurate) value for a larger duration of time.

A number of resource discovery techniques (request scheduling, route selection, CPSS etc.) are based on a given network topology and replica map, assuming knowledge of link and server load information. Effectiveness of admission control and resource scheduling mechanisms rely on maintaining accurate and current system state information. To address the tradeoff between the accuracy of directory information and the update overhead costs, we develop simple and efficient policies to collect and approximate the current system state information in dynamic multimedia environments. We have developed adaptive parameter collection protocols using the notion of a dynamic range to describe the residual capacities of resources being provisioned with a simple algorithm to relax and tighten the range [5, 14]. We also develop a metric to statistically measure the effectiveness of using the collected range to perform admission control. We apply selective update mechanisms that execute only when necessary, reducing the locking overhead incurred during directory updates. Finally, we apply these adaptive techniques to improve the performance of resource provisioning mechanisms, which allows load balancing not only between replicated servers [17], but also among network links to maximize the request success ratio and system throughput [5]. Our performance evaluation results indicate that adaptive range based parameter collection is more cost-effective than static approaches that use instantaneous values of system state.

System Information Collection and Update

In mobile environments, constant mobility of end-users could cause significant variation in available network resources making it difficult to ensure QoS to applications executing on the end-user devices. While keeping track of individual user mobility patterns can support advance reservation of resources needed to guarantee QoS, maintaining accurate location information in the directory service can entail very high overhead. Our work explores the possibility of maintaining aggregate user mobility information in the directory service to determine if advance reservation is required or not. In this environment, the directory service maintains the mobile host population in a certain region at a certain time [15]; this information can be obtained from wireless access points to the fixed network, thereby eliminating the need for constant monitoring of individual mobile host location.

Such coarse level mobility information can be used to support QoS-based resource allocation mechanisms,e.g CPSS in mobile environments. The CPSS process, that determines the optimal path and server for a given request (at a given time) based on current resource availability information in the directory service must now trigger the necessary re-routing of network packets

and/or rescheduling of the session on a new (more accessible) server. Furthermore, this may happen at multiple points during the servicing of a request. Figure 3 illustrates the advantage of using the aggregate mobility information in the CPSS algorithm as compared to using individual host mobility information; we observe higher request completion ratios and lower overheads by using the coarse-grained mobility information.

With the emergence of distributed real-time environments (DRE), seamlessly providing timely access to dynamically changing data is gaining importance. Directory services must reflect changes in the DRE as closely as possible; this process can consume significant computation and communication resources thereby causing violations of real-time requirements. Addressing the timeliness/accuracy/cost tradeoff in maintaining directory services for real-time applications will require the development of algorithms that exploit the accuracy and latency margins to ensure that most applications receive information at the desired levels of quality and timeliness while minimizing resource utilization.

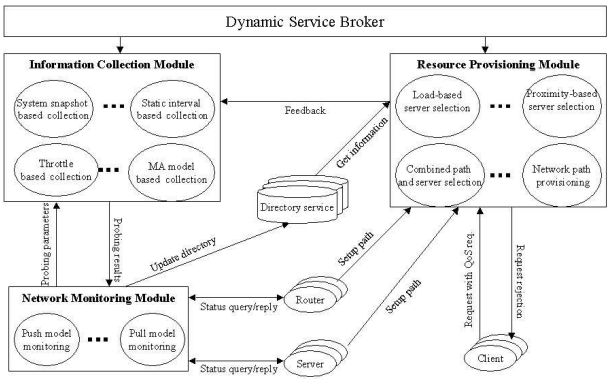


Figure 4: The AutoSeC Dynamic Service Broker Framework

As an example, we illustrate an adaptive parameter collection process [6, 14] where the directory service issues periodic probes to network routers and servers to collect the system state information such as current residual bandwidth capacity and delay bounds. The probes consolidate the collected sample values and update a directory service with the current system image. The probe doesn't update the directory until a new change is confirmed. The current system image consists of a range of values for each entry with upper and lower bound values. Since the system image isn't necessarily accurate, we expect that the network or servers will reject some requests during the subsequent resource reservation phase from the clients even though they were assigned to the clients by the provisioning module. Implementation details of the parameter collection process such as threshold and range manipulation are discussed in [6, 14]. A management framework called AutoSeC (Automatic Service Composition) [14] has been designed to dynamically select an appropriate combination of information collection and resource provisioning policies based on current system conditions and user requirements. One of the main components of AutoSeC is the directory service (depicted in Figure 4), which holds system information. An important issue in managing the directory service is to cost-effectively maintain accurate system state information. In this article, we study a family of information collection strategies that use varying data representations and diverse sampling policies. We also study resource provisioning algorithms for dealing with QoS-sensitive requests. Specifically, we analyze the traditional server selection mechanisms (load-based and proximity-based) and the more complex CPSS (combined path and server selection) mechanism1 that takes both path and server status into consideration.

3 Case Study: Directory Enabled Access Control

In this section we discuss the use of an access-control framework for dynamic, concurrent objects as a case-study in the use of directory services in the CompOSE|Q middleware framework. The distinct separation of access control policies and mechanisms is an important aspect of devising a framework that is able to provide both developers and end-users with the flexibility to adapt it to their needs (in terms of security requirements and performance). The interaction and composability of security mechanisms with other services in the CompOSE|Q framework may pose constraints on security policies that address a variety of threats such as timely location of objects, key management and secure communication.

Our access control architecture [19] is based on capabilities, which provide minimalistic, name-based protection of the objects in the system. This access control information is stored in the directory, and allows a mapping to be maintained between actors on a given node and the capabilities they currently hold. Capabilities are tied to the directory entries associated with an object. Access control information in the DS is represented as a triple: *ActorID x Access Rights x CapInf* as illustrated in Figure 3. *ActorID* uniquely identifies an actor object within the system; Access rights specify a set of valid operations for a given object; *CapInf*, allows additional information such as *expiration* to be embedded in the capability. To manage access rights, we introduce the Meta-Level Security Actor (MSA), which

resides on each node and encapsulates and enforces the chosen access control policy (which is built into the behavior of the MSA). With actors, message passing is the only means of inter-object communication, hence controlling the sending and receiving of messages to particular objects allow control over the information flow in the system. In addition, read and write access are normally encapsulated by the *execute* method of an actor. Effectively the MSA acts as a message filter [10] that tags and checks messages based on the capabilities stored in the directory service. On creation of a new actor or migration of existing actors, the DS needs to be consulted to obtain the most up to date information about an actor's location.

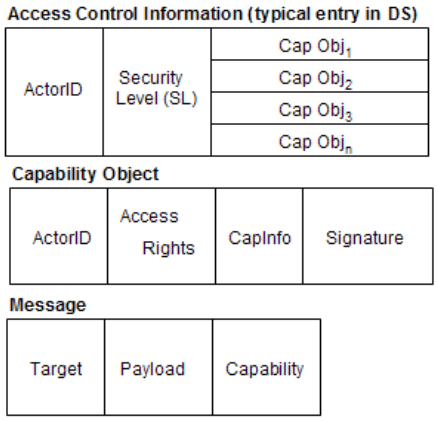


Figure 5: Modeling Capabilities.

Object migration introduces complications in the access control process. The DS maintains information about an actor's current location and hence is often consulted. The specific implementation of object migration can have significant impact on the implementation and performance of access control policies;

- DS-based migration management: location information is maintained by the directory, update overhead is high.
- Forwarder-based policies: establish a forwarder to an object when it migrates, directory will usually point to the first link in the forwarder chain. Can become expensive to traverse after many migrations.
- Hybrid Policy: a combination of the two schemes above

Even with a hybrid policy (which we implement), there are tradeoffs between the consistency of location information in the DS and the overhead incurred by the communication and access control processes. Further, delegation and revocation of access control rights of migrating objects is an issue. In performing revocation (as described in [19]), the issue of having to revoke an object's access rights in 'real-time' becomes an issue in a highly mobile environment. Here, each node's location information for an object (or more specifically an MSA's information) will be inconsistent when the object is constantly migrating. Since each local MSA will need to check for the objects location in trying to revoke its access rights, there is the possibility of repeated directory accesses as the object keeps migrating from node to node. We have proposed two preliminary solutions to this problem; (a) broadcast-based invalidation: here an 'invalidation message' is sent to the MSAs on all nodes to effectively freeze the object, enabling access rights to be revoked. This needs to be done in a secure manner (effectively a secure group broadcast), and concurrent broadcasts (to different or same objects) must be prioritized; (b) revocation authority monitor: this is an event-level monitor which is assigned to each object with revocable access rights and provides an extra layer of abstraction which checks if the object's rights are revoked before granting access to a resource. Both techniques are viable solutions, with (a) possibly introducing more network overhead and (b) being more computationally intensive if many revocations are being performed in the system. Current work consists of comparison between these techniques to obtain performance results and further optimize the algorithms.

Implementation & Performance: For the prototype implementation of the directory service in CompOSE|Q we used OpenLDAP's slapd LDAP server was used together with the Berkeley Sleepy Cat Database as the backend for slapd. The components consisted of the server, database backend and related scripting and an interface which provided access to the directory and implemented the functionality of the directory service. We evaluated a number of DS design choices, including; (a) Hierarchical vs flat design; (b) Distributed vs centralized servers; (c) optimizations improving query and update efficiency of the DS. Traditionally, hierarchical models have been used for most DS-based applications [8, 9]. Implementations consist of centralized directories with optimizations for caching directory query results for better performance. Hierarchical models, however, suffer from scalability and performance issues in highly distributed, dynamic and autonomous environments. We utilized a directory service architecture that is essentially flat in structure as opposed to a hierarchical directory service. This kind of design allows the system to be dynamically configurable, as new servers containing new information can be dynamically added to the environment without any

changes being required to the existing servers. An interface to the DS provides clients with a uniform view of the entire system and abstracts the queries made by the clients. We ensure authenticated access to the DS by using a secure bind, natively supported by LDAP [8]. The Distributed Directory Service (DDS) architecture, in contrast to a centralized structure, requires client information to be shared between all the distributed servers. The schema is extended to store information about the clients at a single level below the root. Below is a subset of the schema information pertaining to the access control architecture.

```
attribute DomainName cis
attribute ActorID cis
attribute NodeID cis
attribute ActorClassName cis
attribute Priority cis
attribute SecurityLevel cis
attribute CapabilityObject cis
attribute DomainSecurityLevel cis
attribute SerializedObject cis
attribute NodeManagerActorID cis
attribute NodeIPAddress cis
attribute NodeActive cis
```

A single piece of information is stored for each client, which uniquely identifies a client and can be modified and queried. By utilizing this structure, searches can be sped up and reachability of the client can be improved. Additionally, the environment can be deemed more fault-tolerant by virtue of the information being distributed across multiple servers. Once the interface receives a query for a particular client, it searches the local directory server for information on that client. If the client information is unavailable locally, it sends a multicast request to all the other servers. The server that has information about the client eventually responds with the client information. This is picked up by the interface that made the broadcast and is sent back to the client. When an interface receives a multicast request from another interface, it queries its local directory server for information on the particular client. Upon success it responds with another multicast packet containing the appropriate information, else the request is ignored.

The performance of the DS was very flexible in that we could tune it to the application, depending on whether it was query heavy or update heavy. The update response showed the benefits of using a distributed architecture for the directory service. The DDS scales much better and hence is preferable to a centralized service when the number of updates is significantly higher than queries. We intend to explore hybrid architectures that balance the search vs. update overhead of directories. A summary of the key performance results for our implementation of the DS is shown in Figure 6, as well as the result of the optimizations.

	Time (µs)	
Directory Operation	Before Optimization	After Optimization (%)
Adding Actor	137,844	27,463 (80)
Adding Actor Attribute	59,086	5875 (90)
Searching for Attribute	42,165	2814 (93)

Figure 6: Directory service operations

Based on the configuration and setup of the directory server as described above, tuning was done to ensure faster performance for both access and updates. On our server (Sun Sparc 5, running Solaris 2.7), priority paging was enabled to enhance system response since the file system was being used heavily on the machine. This allowed the system to keep cachefree pages of memory on the freelist, however only freeing up file system pages while free memory is between given limits. This was particularly useful since the system had ample memory and this reduced the effect of the file system I/O paging out significant portions of important applications heap and stack address space. In terms of improving performance on the directory side, attribute indexing was used to improve search times since we were primarily querying on an ID (location). Thus indexing this attribute enhanced performance significantly. To speed up writes, synch on DB write was turned off. When on, this synchronized the DB after each modification by default. Due to our fairly small data store and frequency of updates this would improve performance at the cost of a small possible risk to data integrity. The use of slave servers and subsequently running synch solved this problem. Caching was also used, both for the entries and index file. These parameters were tuned to the system, and a variety of tests done to determine the optimal settings.

4 Future Research Directions

There has been a large body of work in the general area of directory services, in both the database community and operating systems community. Moving object databases deal with the modelling and tracking of a moving object in the database; information collection solutions have been proposed [12] in this context that directly compare the cost of information imprecision with the cost of the message passing and make decisions based on the difference. Various approaches for directory caching have been studied extensively in the context of LDAP. Most of the work has been focused on keeping the cache information close to the client applications that access the directory information [13, 3].

This allows effective scaling of client-server database architecture. Solutions such as query template schemes and that of reusing cached LDAP directory entries for answering LDAP queries have been proposed as viable solutions. Other techniques, such as flexible management of lists (distribution lists, access control lists etc.) in directories have been studied by making efficient and evaluable extensions to the LDAP query language for the location and expansion of lists [11]. Formalizing the behavior of directory services will help in reasoning about the semantic implications of implementation details such as consistency-based updates to caches and qualitative cache coherence. We intend to provide extensible implementations of directory management and reason about them using formal approaches (e.g. rewriting logic).

References

[1] G.A. Agha: *ACTORS: A Model of Concurrent Computation in Distributed Systems.* MIT Press, Cambridge, Massachusetts, 1986.

[2] G. Apostolopoulos, R. Guerin, S. Kamat & S.K. Tripathi: *QoS Routing: A Performance Perspective.* SigComm'98

[3] S. Cluet, O. Kapitskaia & D. Srivastava: *Using LDAP Diectory Caches.* Proc. of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats, 1999.

[4] S. Chen & K. Nahrstedt: *Distributed Quality-of-Service Routing In Ad-Hoc Networks.* IEEE JSAC, Special Issue on Ad-Hoc Networks, 1999.

[5] Z. Fu & N. Venkatasubramanian: *Directory-Based Composite Routing and Scheduling Policies for Dynamic Multimedia Environments.* Proc. of IPDPS 2001.

[6] Z. Fu & N. Venkatasubramanian: *Adaptive Parameter Collection in Dynamic Distributed Environments.* Proc. of IEEE ICDCS 2001.

[7] J.D.Guyton & M.F.Schwartz: *Locating Nearby Copies Of Replicated Internet Servers.* Proc. of ACM SIGCOMM, August 1995.

[8] T.A. Howes & M.C. Smith: *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol.* MacMillan Technology Series, 1997.

[9]] T.A. Howes, M.C. Smith & G.S. Good: *Understanding & Deploying LDAP Directory Services.* MacMillan Network Architecture and Development Series, 1st Ed. 1999.

[10] S. Jajodia & B. Kogan: *Integrating an object-oriented data model with multilevel security.* In Proc. of the IEEE Computer Society Symposium on Security and Privacy, pp.76-85, 1990.

[11] H. V. Jagadish, M.A. Jones, D. Srivastava and D. Vista: *Flexible List Management In A Directory.* Proc. of the Seventh Intl. Conf. on Information and Knowledge Management (CIKM), 10-19, 1998.

[12] O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, & G. Mendez: *Cost and Imprecision In Modeling The Position Of Moving Objects.* Proc. of ICDE 1998.

[13] O. Kapitskaia, R.T. Ng & D. Srivastava: *Evolution And Revolutions In LDAP Directory Caches.* Proc. of the Intl. Conf. on Extending Database Technology (EDBT), 202-216, 2000.

[14] Q. Han & N. Venkatasubramanian: *AutoSeC: An Integrated Middleware Framework For Dynamic Service Brokering.* ACM Middleware 2001.

[15] Q. Han & N. Venkatasubramanian: *Aggregation based Information Collection in Mobile Environments.* Journal of High Speed Networks, Vol.11, No. 3-4, 2002.

[16] N. Venkatasubramanian, M. Deshpande, S. Mohapatra, S. Gutierrez-Nolasco & J Wickramasuriya: *Design and Implementation of a Composable Reflective Middleware Framework.* IEEE ICDCS 2001.

[17] N. Venkatasubramanian & S. Ramanathan: *Load Management For Distributed Video Servers.* Proc. of IEEE ICDCS 1997.

[18] N. Venkatasubramanian & C.L. Talcott: *Reasoning about Meta Level Activities In Open Distributed Systems.* Proc. of ACM PODC 1995

[19] J. Wickramasuriya & N. Venkatasubramanian: *A Middleware Approach To Access Control For Mobile, Concurrent Objects.* Submitted for publication, May 2002

[20] M. Wahl, T. Howes & S. Kille: *Lightweight Directory Access Protocol (v3).* IETF RFC 2251, December 1997.