The Athena Service Management System

Mark A. Rosenstein
Daniel E. Geer, Jr.
Peter J. Levine

Project Athena
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139
{mar,geer,pjlevine}@ATHENA.MIT.EDU

ABSTRACT

Maintaining, managing, and supporting an unbounded number of distributed network services on multiple server instances requires new solutions. Moira, the Athena Service Management System provides centralized control of data administration, a protocol for interface to the database, tools for accessing and modifying the database, and an automated mechanism for data distribution.

1. Introduction

The purpose of Moira is to provide a single point of contact for authoritative information about resources and services in a distributed computing environment. Moira is a centralized data administrator providing network-based update and maintenance of system servers.†

- Conceptually, Moira provides mechanisms for managing servers and resources. This aspect comprises the fundamental design of Moira.
- Economically, Moira provides a replacement for labor-intensive hand-management of server configuration files.
- Technically, Moira consists of a database, a server and its protocol interface, a data distribution manager, and tools for accessing and modifying Moira data.

Moira provides coherent data access and data update. Access to data is provided through a standard application interface. Programs designed to reconfigure network servers, edit mailing lists, manage group membership, etc., all use this application interface. Applications used as administrative tools are invoked by users; applications that update servers are (automatically) invoked at regular intervals. Management reports and operational feedback are also provided.

† Care must be taken to distinguish among different senses of the word ‘‘server’’: the Moira server, which manages a database; the servers which provide system services and whose maintenance are Moira’s reason for existence; and the update servers which allow Moira to affect these system services.

1.1. Requirements

The requirements for the initial Athena Moira include:

- Management of 15,000 accounts, including individual users, course, and project accounts, and special accounts used by system services.
- Management of 1,000 workstations, timesharing machines, and network servers, including specification of default resource assignments.
- Allocation of controlled network services, such as creating and setting quotas for new users’ home directories on network fileservers, consistent with load-balancing constraints.
- Maintenance of other control information,

including user groups, mailing lists, access control lists, etc.

- Maintenance of resource directories, such as the location of printers, specialized file systems (including privately supported file systems), and other network services.

This must be accomplished with the utmost robustness.

The system must be easily expandable, both to support additional instances of a particular service and to offer additional services in the future. At this time, Moira is used to update three services (with RVD to be added shortly):

- *Hesiod:* The Athena Name Service, *Hesiod,* provides service-to-server and label translation. It can be thought of as a high-performance, read-only front-end to the Moira database. See the companion paper in this volume.[1]
- NFS: At Athena, most shared-access read-write file systems are provided by a locally modified form of the Network File System.[2] Moira manages the NFS server hosts, providing quota-based resource allocation, and load balancing. Also refer to the companion paper in this volume.[3]
- Mail Service: Athena's mail service is through a central routing hub to multiple post office servers (mail repositories) based on the Post Office Protocol[4] (POP) of the Rand Mail Handler[5] (MH) package. Moira allocates individual post office boxes to post office servers, and builds the */usr/lib/aliases* control file used on the central mail hub.
- RVD: At Athena, most shared-access read-only file systems are provided by a Remote Virtual Disk system. Moira manages the RVD server hosts, providing access control lists and server configuration files.

Each of these server hosts are controlled by some number of server-specific data files; over 50 separate files are required to support the services described above. Moira currently supports three *Hesiod* servers, 17 RVD file servers, 32 NFS file servers, one mail hub and three post offices. Each RVD server requires one file, and each server's file is different. A *Hesiod* server requires 9 separate files containing 64000 resolvable queries, but each *Hesiod* server receives the same 9 files. Each NFS server requires two files, one file identical across most NFS servers, one file different. The mail hub requires one file, */usr/lib/aliases.*

1.2. Design Points

There are five factors to be taken into account when making design decisions. In order of importance, to this project they are:

1. Reliability
2. Consistency
3. Flexibility
4. Time Efficiency
5. Space Efficiency

Moira must be reliable. In particular, it must be designed to allow straightforward recreation of Moira on replacement hardware from backups, should the need arise. The backup regime for Moira consists of frequent database backups in ASCII format, to redundant sites. All components of the Moira system are designed such that a duplicate configuration can be kept running as a test platform without interfering with normal, ''real,'' operations. Service updates are verified by testing the server before calling an update successful. Failed operations ring alarms that are heard, disabling parts of the system if required, but without interactions with logically unconnected parts of the service management system. Services which are duplicated for availability are updated such that the service is always available; i.e., it is not permitted for server updates to drift into unwarranted synchrony, bringing down all instances at the same time. The entire life-cycle is considered part of the package, so tight change and source-code control (reviewing each change to source, running only what can be built from source) is part of the design.

In addition to having authoritative control of the data, Moira must see that the data is kept consistent. To guarantee internal consistency, Moira clients do not touch the database directly; they do not even see the database system used by Moira. Each application uses the application library to access the database. This library is a collection of functions allowing access to the database by communicating with the Moira server using the Moira protocol. Many of the database consistency constraints are handled by the library. A number of consistency verification applications also exist. To make this consistency reliable, the protocol is designed to be tamper-proof, withstanding both denial-of-service attacks and malicious attempts to corrupt the data. Security is provided with the help of authentication by the

Kerberos private-key authentication system. See the companion paper in this volume.[3]

Beyond these goals, Moira must be flexible in both its database underpinnings and the services it supports. As discussed later in the design section, the particular Database Management System used is insulated from Moira through a Global DataBase (GDB) library,[6] making Moira plug-compatible with other database foundations. It is independent of the individual services—while each service updated by Moira has its own particular data format and structure, the Moira database stores data in one coherent format. A separate program, the Data Control Manager (DCM), converts Moira database (internal) structure to server-appropriate structure (such as a configuration file). When a new service is supported, the database can be changed and only minimal updates are necessary in the Moira server, and a new module is added to the DCM specifying the additional specific output data format to be manufactured.

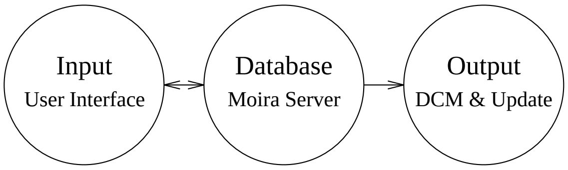
Also, in the interests of flexibility, no administrative policy decisions are coded into the design. These are determined only by the data in the database.

Simplicity of the design is more important than the speed of operation; other systems, such as the *Hesiod* name service, provide a high bandwidth read-only interface to the database. To this end, the server only supports simple queries. Processing efficiency for complex queries is maximized by local applications running on the workstation, not on the central server. Any set of changes that must be atomic to maintain database consistency are performed on the server; sets of non-atomic changes and complex lookup operations are supported in the server only through combining simpler queries.

2. System Design

There are three sides to the Moira system:

- The input side, which contains all of the user-interface programs, allowing the user to enter, examine, or modify data in the database.
- The database side, which consists of the actual database, the Moira server which manipulates the database, and utility programs to backup, restore, and verify the internal consistency of the database.
- The output side, which extracts information from the database, formats it into server-specific configuration files, and updates the various servers by propagating these files.



Various amounts of glue are required to connect these three sides. At the lowest level, there is the network protocol that client programs use to gain access to the database. This, however, depends on the actual model of how database queries are done, which is influenced by the organization of the database (but not its exact format, which is hidden through the protocol).

2.1. The Moira Protocol

The Moira protocol is the fundamental interface to Moira for client applications. It allows all clients of Moira to speak a common, network-transparent language.

The Moira protocol is a remote procedure call protocol based on the GDB library, which in turn uses a TCP stream. Each client program makes a connection to a well known port to contact the Moira server, sending requests and receiving replies over that stream. Each request consists of a major request number, and several counted strings of bytes. Each reply consists of a single status code followed by zero or more counted strings of bytes. Requests and replies also contain a protocol version number, to allow clean handling of version skew.

The following major requests are defined for Moira. It should be noted that each query defines its own signature of arguments and results. For some of these actions the server checks authorization based on the authenticated identity of the user making the request.

noop Do nothing. This is useful for testing and profiling of the server.

authenticate There is one argument, a *Kerberos*[7] authenticator. All requests received after this request are performed on behalf of the principal identified by the authenticator.

query The first argument is the name of a pre-defined query (a ‘‘query handle''), and the rest are arguments to that query. Queries may retrieve information or modify what is in the database. If the server query is allowed, any retrieved data are passed back as several return values. All but the last returned value will have a status code indicating more data, with the final one returning the real status code of the query.

access There are a variable number of arguments. The first is the name of a pre-defined query usable for the ''query'' request, and the rest are query arguments. The server returns a reply with a zero status code if the query would have been allowed, or a reply with a non-zero status code explaining why the query was rejected.

Normal use of the protocol consists of establishing a connection, providing authentication, then performing a series of queries. As long as the application only wants to retrieve data or perform simple updates, only an authenticate followed by queries are necessary. The access operation is useful for verifying that an operation with side effects will succeed before attempting it.

2.2. Queries

All access to the database by clients is provided by the application library via the Moira protocol. This interface provides a limited set of predefined, named queries, allowing tightly controlled access to database information. Queries fall into four classes: retrieve, update, delete, and append. An attempt has been made to define a set of queries that provide sufficient flexibility to meet all of the needs of the Data Control Manager as well as the individual application programs, since the DCM uses the same interface as the clients to read the database. Since the database can be modified and extended, the application library has been designed to allow the easy addition of queries.

The generalized layer of functions makes Moira independent of the underlying database. If a different database management system is required, the only change needed will be a new Moira server. It is made by linking the predefined queries to a set of data manipulation procedures provided by a version of GDB suited to the alternate DBMS.

At this time there are over 100 supported queries. See the complete technical description of Moira for a listing.[8] Some sample queries include:

get_nfs_quotas
Arguments: machine, device
Returns: list of login/quota tuples
Errors: no match, bad machine
Retrieves disk quota assignments for all users on the specified disk partition.

get_user_by_login
Arguments: login name
Returns: login, uid, shell, home, last, first, middle, status, ID number, year, expiration date, modification time
Errors: no match, not unique
Retrieves information about a particular user, searching by login name. Similar queries exist to search by last name, first name, user ID, and Registrar's ID number.

add_machine
Arguments: name, type, model, status, serial number, system type
Returns: nothing
Errors: already exists, bad type
Appends a new machine to the list of machines known by Moira.

update_server_info
Arguments: service, update interval, target file, script, dfgen
Returns: nothing
Errors: no match, not unique
Updates a service entry, allowing anything but the service name to be changed.

delete_filesys
Arguments: label
Returns: nothing
Errors: no match, not unique
Deletes the specified file system information from the database. Does not automatically reclaim the storage at this time.

Each query has two possible return status values in addition to any errors given above: **success** and **permission_denied.**

2.3. The Database Management System

The database is the core of Moira. It provides the storage mechanism for the complete system. Moira expects its database to consist of several tables of records with strings, integers, and dates. Tables are keyed on one or more fields in each record allowing efficient retrieval by key or wild cards.

The database currently in use at Athena is Ingres [9] from Relational Technology, Inc. Ingres provides a complete query system, a C library of routines, and a command interface language. Its advantages are that it is available and that it mostly works. By design, Moira does not depend on any special feature of Ingres so as to retain the option to utilize other relational database systems.

The database is an independent entity from the Moira system. The Ingres query bindings and database-specific routines are layered at the lower levels of the Moira server. All applications are independent of database-specific routines. An application passes query handles to the Moira server which then resolves the request into an appropriate database query.

The database contains several types of objects. Each object in the database has an access control list (ACL) associated with it indicating who is allowed to modify that object. Each object also has records who last modified it and when that modification was performed. The ACL's are just references to lists in the database. The database contains the following:

- User information: full name, login name, user ID, registrar's ID, login shell, home directory, class year, status, modification time, nickname, home address, home phone, office address, office phone, school affiliation, an ACL
- Machine information: name, type, model, status, serial number, system type, an ACL
- Cluster (mapping of machines to default printer and RVD server) information: name, description, location, default servers, machine assignments, an ACL
- General service information: service name to network port mapping
- File system configuration: name, type, server host, name on server host, mount point on client host, access mode, an ACL
- NFS information: host name, physical disk partition, quota by user on each partition, an ACL
- RVD information: host name, physical disk partition, virtual disks assigned to each partition, pack ID, access passwords, size, an ACL
- Printer information: name to */etc/printcap* entry mapping
- Post office location: for each user post office server and box on that server
- Lists: name, description, modification date, members (which can be users, other lists, or literal strings), attributes (mailing list, UNIX group and gid), an ACL
- Aliases: includes allowed keywords for certain fields in the database, alternate names for file systems, alternate names for services
- DCM information: services to be updated, hosts supporting each service, target files on each host, last update time, enable/override/success flags for updates
- Internal control information: next user ID, group ID, machine ID, and cluster ID to assign (these are just hints); an ACL for each query; usage statistics

2.4. Moira-to-Server Update Protocol

Moira provides a reliable mechanism for updating the servers it manages. The goals of the server update protocol are:

- Rational, automatic update for normal cases and expected kinds of failures.
- Ability to survive clean (target) server crashes.
- Ability to survive clean Moira crashes.
- Easily understood state so that straightforward recovery by hand is possible.

All actions are initiated by the Moira system. Updates of managed servers are performed such that a partially completed update is harmless. Updates not completed are simply rescheduled for retry until they succeed, or until an update can not be initiated. In the latter case, a human operator will be notified.

The update protocol is based on the GDB library just as is the Moira protocol itself. In the update protocol, there are four commands:

authenticate A *Kerberos* authenticator is sent. The update server on the target server uses this authenticator to verify that the entity contacting it is authorized to initiate updates.

transfer This command is used for sending information, usually entire files. The protocol is capable of efficiently sending a half-megabyte binary file.

instructions This sends over a command script, which when executed on the target server will install the new configuration files just sent.

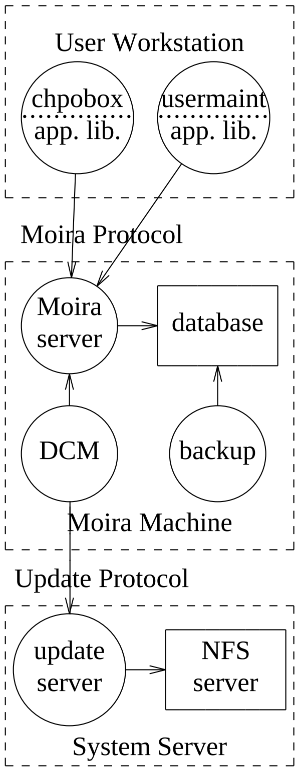
execute This instructs the update server to execute the instructions just sent.

In typical usage, all four commands are used in the order presented above. Multiple transfers may be necessary for some server types. Note that when data files are transfered, they do not directly overwrite the existing data files. Instead, they are put in a temporary position. When the update is executed, the old files are renamed to another temporary name, and the new files are given the correct names. This minimizes the interval during which a server system crash would leave an inconsistent set of configuration files. If all portions of the preparation are completed without error, the execution is then allowed. This usually consists of moving files around, then sending a signal to or restarting a server process. The update server then performs a plausibility check on the result by verifying the operation of the system server, and sends Moira an indicative return code.

3. System Components

Six software components make up the Moira system.

- The database, currently built on the commercial database system RTI Ingres.
- The Moira server, a program always running on the machine containing the database. It accesses the database for the client programs.
- The application library, a collection of routines implementing the Moira protocol. It is used by client programs to communicate with the Moira server.
- The client programs, a collection of programs that make up the user interface to the system.
- The data control manager, a program run periodically by *cron* and driven by data in the database. It constructs up-to-date server configuration files and installs these files on the servers.
- The update servers, which run on each machine containing a server that Moira updates. These are contacted by the data control manager to install the new configuration files and notify the real servers being managed by Moira.



Because Moira has a variety of interfaces, a distinction must be maintained between applications called clients that directly read and write to Moira (i.e., administrative programs) and services which use information distributed from Moira (i.e. a name server). In both cases the interface to the Moira database is through the Moira server, using the Moira protocol. The significant difference is that server update is handled automatically through the Data Control Manager; administrative programs are executed by users.

3.1. Clients

Moira includes a set of specialized management tools to grant system administrators overall control of system resources. For each system service there is an administrative interface. To accommodate novice and occasional users, a menu interface is the default. For regular users, a command-line switch is provided that will use a line-oriented interface. This provides speed and directness for users familiar with the system, while being reasonably helpful to novices and occasional users. A specialized menu building tool has been developed in order that new application programs can be developed quickly. This user interface does not depend on the X Window System.[10] It must be possible for system operators to use dumb terminals to correct resource problems, i.e. it cannot be a requirement that a high level of functionality be present before the service management system can operate.

Fields in the database have associated with them lists of legal values. A null list indicates that any value is possible. This is useful for fields such as *user_name, address*, and so forth. The application programs, before attempting to modify anything in the database, request this information, and compare it with the proposed new value. If an invalid value is discovered, it is reported to the user, who is given the opportunity to change the value, or ‘‘insist’’ that it is a new, legal value. (The ability to update data in the database does not necessarily indicate the ability to add new legal values to the database.)

Applications should be aware of the ramifications of their actions, and notify the user if appropriate. For example, an administrator deleting a user is informed of storage space that is being reclaimed, mailing lists that are being modified, and so forth. Objects that need to be modified at once (such as the ownership of a mailing list) present themselves to be dealt with.

The following list of client programs are currently in use at Project Athena. These are rewrites of standard UNIX programs, and are available to regular users:

- *chfn* - change finger information
- *chsh* - change default shell

These are new programs available to regular users:

- *register* - allow new students to claim an Athena account
- *mailmaint* - allow users to add/delete themselves to mailing lists

These are used by system administrators:

- *attachmaint* - map file system names to physical server configurations
- *chpobox* - change forwarding post office for a user
- *clustermaint* - associate a machine with a set of default servers
- *dcmmaint* - update DCM table entries, including service/machine mapping
- *listmaint* - create and maintain groups, mailing lists, and access control lists
- *nfsmaint* - configure NFS file servers
- *portmaint* - maintain the list of well known contact ports
- *regtape* - enter new students from the Registrar’s tape
- *rvdmaint* - configure RVD servers
- *usermaint* - maintain user information including file service and post office location

Finally, this program is used only in debugging Moira:

- *mrtest* - perform any query manually

3.1.1. New User Registration

A specialized client is the new user registration program. A new student must be able to claim an Athena account without any intervention from Athena user accounts staff. Without this the user accounts staff would be faced with manually creating hundreds of accounts at the beginning of each academic term.

Athena obtains a copy of the official list of registered students from the MIT Registrar shortly before the start of each term. The *regtape* client adds each student on the Registrar’s tape who has not already been registered for an Athena account to the ‘‘users’’ relation of the database, and assigned a unique user ID; the student is not assigned a login name or any other resources at this time. A one-way encrypted form of the student’s ID number is stored along with the name. No other database resources are allocated at that time. This ID number and the exact spelling of the student’s full name as recorded by the Registrar are all that are needed for a student to claim an account. Thus the ID number is something of a password until a real account has been set up.

When the student decides to register with Athena, he or she walks up to any workstation and logs in using the username of ‘‘register’’. This produces a form-like interface prompting for the user’s first name, middle initial, last name, and student ID number. The *register* program does not talk to the Moira server directly, but goes through a registration server first. The registration server deals with access control to the Moira server and the *Kerberos* administration server. Register encrypts the ID number, and sends a *verify_user* request to the registration server. The server responds with **already_registered, not_found,** or **OK.** After this the register server will do work on behalf of the user; the user still cannot contact Moira directly until he or she obtains a login name and user ID.

If the user has been validated, *register* then prompts the student for a choice in login names. It then goes through a two-step process to verify the login name: first, it simulates a login request for this user name with *Kerberos;* if this fails (indicating that the username is free and may be registered), it then sends a **grab_login** request. On receiving a **grab_login** request, the registration server then proceeds to register the login name with *Kerberos;* if the login name is already in use, it returns a failure code to *register.* Otherwise, it allocates a home directory for the user on the least-loaded fileserver, sets an initial disk quota for the user, builds a post office entry for the user on the least-loaded post office server, and returns a success code to *register.* Moira keeps track of loading on servers in the database, incrementing and decrementing its estimate of the load as it allocates and deallocates resources on each server.

Register then prompts the user for an initial password. It sends a **set_password** request to the registration server, which decrypts it and forwards it to *Kerberos.* At this point, pending propagation of information to the *Hesiod* name service, the central mail hub, and the user's home fileserver, the user's account has been established. These updates may take up to 12 hours to complete.

3.2. The Moira Server

As previously stated, all remote communication with the Moira database is done through the Moira server, using the Moira protocol. The Moira server runs as a single UNIX process on the Moira database machine. It listens for connections on a well known service port and processes remote procedure call requests on each connection it accepts.

GDB, through the use of BSD UNIX non-blocking I/O, allows the programmer to set up a single-process server that handles multiple simultaneous TCP connections. The Moira server will be able to make progress reading new RPC requests and sending old replies simultaneously, which is important if a reasonably large amount of data is to be sent back. The RPC system from Sun Microsystems was also considered for use in the RPC layer, but was rejected because it cannot handle large return values, such as might be returned by a complex query.

A major concern for efficiency in some DBMS's is the time it takes to begin accessing the database, sometimes requiring starting up a back-end process. Since this is a heavyweight operation, the Moira server will do this only once when it starts.

The server performs access control checks on all queries. An access control list (ACL) is associated with each query handle, and with many objects within the database. For instance, to add someone to a list, it is sufficient to either be on the ACL associated with the **add_member_to_list** query, or to be on the ACL of the list in question. In addition, lists, users, machines, and file systems have lists of additional users who are allowed to modify them. The concept of an all powerful database administrator is not necessary with Moira, although could be implemented by having the same ACL for all queries that affect the database.

Because one of the requests that the server supports is a request to check access to a particular query, it is expected that many access checks will have to be performed twice: once to allow the client to find out that it should prompt the user for information, and again when the query is actually executed. It is expected that some form of access caching will eventually be worked into the server for performance reasons.

3.2.1. Input Data Checking

Without proper checks on input values, a user could easily enter data of the wrong type or of a nonsensical value for that type into Moira. For example, consider the case of updating a user's mail address. If, instead of typing ''athena-po-1'' (a valid post office server), a user accounts administrator typed in ''athena-po1'' (a nonexistent machine), all the user's mail would be returned to sender as undeliverable.

Input checking is done by both the Moira server and by applications using Moira. Each query supported by the server may have a validation routine supplied which checks that the arguments to the query are legitimate. Queries that do not have side effects on the database do not need a validation routine.

Some checks are better done in applications programs; for example, the Moira server is not in a good position to tell if a user's new choice for a login shell exists. However, other checks, such as verifying that a user's home directory is a valid file system name, are conducted by the server. An error condition will be returned if the value specified is incorrect. The list of predefined queries defines those fields which require explicit data checking.

3.3. Backup

It is not critical that the Moira database be available 24 hours a day; what is important is that the database remain internally consistent and that the data never be lost. With that in mind, the database backup system for Moira has been set up to maximize recoverability if the database is damaged. Backup is done in a simple ASCII format to avoid dependence on the actual DBMS in use.

Two programs *(mrbackup* and *mrrestore)* are generated automatically (using an *awk* script) from the database description file. *mrbackup* copies each relation of the current Moira database into an ASCII file. *Mrbackup* is invoked nightly by a command file that maintains the last three backups on-line. These backups are put on a separate physical disk drive from the drive containing the actual database and copied over the network to other locations. *Mrrestore* does the inverse of *mrbackup,* taking a set of ASCII files and recreating the database. These backups by themselves provide recovery with the loss of no more than roughly a day's transactions. To obtain complete recovery, it is necessary to examine the log files of the servers. An automated procedure to do this has not been written since it is complicated and has not been needed yet.

3.4. The Data Control Manager

The Data Control Manager is a program responsible for distributing information to servers. The DCM is invoked by *cron* at regular, frequent intervals. The data that drives the DCM is read from the database, rather than being coded into the DCM or kept in separate configuration files. Each time the DCM is run, it scans the database to determine which servers need updating. Only those that need updating because their configuration has changed and their update interval has been reached will be updated.

The determination that it is time to check a service is based on information about that service in the database. Each service has an update interval, specifying how often providers of that service should be updated, and an enable flag. For each server/host tuple, the database stores the time of the last update attempt, whether or not it was successful, and an override flag. If the previous update attempt was not successful, the override flag will indicate when to try again. The administrative user's interface provides a mechanism to set the override flag manually, so as to update a server sooner than it otherwise would be updated.

Locking is also provided since an update may still be in progress the next time the DCM is invoked. Without this locking the new DCM process would attempt to update the same service that the older process is still working on.

If it is necessary to update a server, a separate program (also named in the database) is invoked to extract the information from the database and format into the server-specific files. The DCM then contacts the update server on the machine with the target server, sends the necessary data files and the shell script that is invoked on the server machine to install the new files. The success flag is set or cleared based on whether the update attempt succeeded. If the attempt failed, the override flag is set, requesting that another attempt be made to update this server sooner than indicated by the default update interval.

For performance reasons, some parts of the DCM currently touch the database directly rather than going through the Moira server. Nothing is being done that could not be done through the server. However, bypassing the server makes extracting large amounts of information much faster and avoids slowing down the server for these operations.

4. System Performance

The system is more reliable than the one previously in use at Athena. The old version had an update mechanism more suited to the scale of tens of timesharing hosts rather than thousands of workstations. System crashes have been rare. The speed of the system is fair, being fast enough to use interactively, although some queries do take a while to complete. The database currently occupies about 13 megabytes on the server.

Most of the system as described here has been in use for over six months. A few of the points mentioned here are just now being implemented and put into service. Note that this is the only management tool for 5000 active users, 650 workstations, and 65 servers.

4.1. Availability

The Moira server has nearly always been accessible. Some queries, such as listing all publicly accessible mailing lists, will tie up the server for a short period of time. Regular users are prohibited from the longer queries such as listing all users, which will lock up the server for several minutes. The server is occasionally down for safety when an Moira administrator wishes to modify the database directly through Ingres.

4.2. Security

Kerberos authentication on all network access and physical security of the machine have been sufficient to prevent breakins. However, the system has not had enough exposure to believe it is really resistant to concerted attacks.

One problem with the current implementation has to do with security and access control lists. It is difficult to administer the numerous ACL's in the system, and it is not always obvious when different queries are used to predict the effects of changing an ACL. Currently this problem is avoided by having two classes of queries: those that anyone can do, and those that only the database administrators can perform. There need to be more intermediate levels. For instance, it would be useful to allow some system operators to change the information describing public workstations, but not the timesharing machines and service hosts.

4.3. Consistency

The current database suffers from decay. The database grows indexes and reference counts that are wrong, post office boxes that do not belong to any user, and groups with no members. These are assumed to be caused by coding errors in the client library and clients. Any problem is potentially compounded by the very *raison d'être* of Moira. With hundreds of workstations in the field there is no guarantee that Moira client programs installed on them are all at current revision level. There are also a few problems suspected in the database system itself (Ingres), but these are difficult to pinpoint. However, the various data extraction programs used by the DCM are robust; they skip over any inconsistent records so these cause few problems.

A database consistency checker, in the spirit of *fsck*, is run regularly, but only for informational purposes. Because of the dangers involved in modifying the database outside the Moira server, it is not modified by this checker at this time.

A shortcoming of the current system is that it is occasionally necessary to use Ingres directly to make some modification to the database. For example, it was recently necessary to modify every instance of a particular login shell in the user account information. A special client could have been written to do this, but it would only be used once, so the operation was done interactively with the Ingres query interpreter. A large number of similar operations to be executed would normally imply a need for a generalized batch processor. This is is too complicated for our needs, hence such operations are either done by hand through the regular clients, or edited into a script that can be run through the raw Ingres interpreter. Availability of this interpreter makes such operations relatively easy; yet there is the temptation to fix too many things that way, bypassing the checks built in to the client library and Moira server.

Care must be taken to avoid hardcoding into the Moira design current policy decisions about accounts and resources here at Athena. Yet by maintaining this flexibility, it sometimes is too easy to break the rules. This is more often an error than an intentional breaking of the rules. For instance, there have been users who did not have a post office box because of administrative mistakes; they were unable to receive mail.

5. Conclusion

Systems for managing the otherwise unmanageable need to be designed well, burned in realistically, and provided with seamless upgrades. With the Athena Moira, we have an example of a system that is working for the scale of 1,000 workstations, but needs significant refinement to go to the 10,000 workstation level. The initial vision has proven correct; the remaining question is whether the design as we now have it will be capable of the next leap in scale. We believe it will and will be back to report to you with our results.

Acknowledgments

The authors would like to acknowledge the following people from M.I.T. Project Athena for their help in making Moira a reality: Michael R. Gretzinger, a former Systems Programmer, and William E. Sommerfeld, Jean Marie Diaz, Ken Raeburn, José J. Capó, and Mark A. Roman, students working for Athena System Development, who helped with the design and implementation of the system; and Jerome Saltzer, Technical Director of Project Athena, and Jeffery Schiller, Manager of Operations at Project Athena, for invaluable critiques of the design of the system and this paper.

References

1. S. P. Dyer, ‘‘Hesiod,’’ in *Usenix Conference Proceedings*, M.I.T. Project Athena (Winter, 1988).
2. R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, ‘‘Design and Implementation of the Sun Network Filesystem,’’ in *Usenix Conference Proceedings* (Summer, 1985).
3. J. G. Steiner, B. C. Neuman, and J. I. Schiller, ‘‘Kerberos: An Authentication Service for Open Network Systems,’’ in *Usenix Conference Proceedings*, M.I.T. Project Athena (Winter, 1988).
4. M. T. Rose, *Post Office Protocol (revised)*, University of Delaware (1985). (MH internal)
5. Rand Corp., *The Rand Message Handling System: User’s Manual*, U.C.I. Dept. of Information & Computer Science, Irvine, California (November, 1985).
6. N. Mendelsohn, *A Guide to Using GDB*, M.I.T. Project Athena (1987). Version 0.1
7. S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer, *Section E.2.1: Kerberos Authentication and Authorization System*, M.I.T. Project Athena, Cambridge, Massachusetts (December 21, 1987).
8. P. J. Levine, M. R. Gretzinger, J. M. Diaz, W. E. Sommerfeld, and K. Raeburn, *Section E.1: Service Management System*, M.I.T. Project Athena, Cambridge, Massachusetts (1987).
9. Relational Technology, Inc., *Ingres Reference Manual*, Release 5.0, Unix, 1986.
10. R. W. Scheifler and J. Gettys, ‘‘The X Window System,’’ *ACM Transactions On Graphics* **5**(2), pp. 79-109 (April, 1987).