Scalable Networked Information Processing Environment (SNIPE)

Graham E Fagg
Department of Computer Science
University of Tennessee
Knoxville
Tennessee 37996-1301, USA
fagg@cs.utk.edu
http://www.cs.utk.edu/~fagg

Keith Moore
Department of Computer Science
University of Tennessee
Knoxville
Tennessee 37996-1301, USA
moore@cs.utk.edu
http://www.cs.utk.edu/~moore

Jack J Dongarra
Department of Computer Science
University of Tennessee
Knoxville
Tennessee 37996-1301, USA
and
Oak Ridge National Laboratory
Box 2008, Bldg 6012
Oak Ridge
TN 37831-6367, USA
dongarra@cs.utk.edu
http://www.netlib.org/utk/people/JackDongarra

Al Geist
Oak Ridge National Laboratory
Box 2008, Bldg 6012
Oak Ridge
TN 37831-6367, USA
gst@ornl.gov

http://www.epm.ornl.gov/~geist

Abstract

SNIPE is a metacomputing system that aims to provide a reliable, secure, fault-tolerant environment for long-term distributed computing applications and data stores across the global InterNet. This system combines global naming and replication of both processing and data to support large scale information processing applications leading to better availablity and reliability than currently available with typical cluster computing and/or distributed computer environments.

Keywords: SNIPE, RCDS, MetaComputing, scalable, secure, reliable

Acknowledgements

This work was supported in part by the Office of Scientific Computing, U.S. Department of Energy, under Contract DE-AC05-96OR22464, by DARPA under Contract DAAH 04-95-1-0595, and by the National Science Foundation's Center for Research on Parallel Computation, Science and Technology Center Cooperative Agreement No. CCR-8809615.

1. Introduction

The beginning of the 21st century will present new challenges for large-scale applications involving communication with, and coordination of, large numbers of geographically dispersed information sources, supplying information to a large number of geographically dispersed consumers. Such applications will require an environment which supports long-term or continuous, reliable and fault-tolerant, highly distributed, heterogeneous, and scalable information processing.

Examples of such applications include:

- Indexing and cataloging the worldwide digital library, which will have hundreds of millions of documents, produced at millions of different locations. The collection can be expected to have a high rate of change, both in the number of new documents issued and in the locations by which the documents are accessed.
- Monitoring of weather and prediction of catastrophic conditions to provide planning and decision support for emergency relief.
- Semi-automated air-traffic control on a significantly larger scale, and with greater reliability, than exists today.
- Large-scale same-day shipping of goods over long distances, with automatic rerouting to adapt to equipment failures or weather problems, and load balancing to accommodate fluctuations in traffic.

The characteristics of such applications include: distributed data collection, distributed computation (often in significant amounts), distributed control, and distributed output. Many of these applications will require high reliability and continuous operation, even though individual nodes or links will fail or otherwise be unavailable. Such applications will be constructed out of a wide variety of computational components (including smart sensors, personal digital assistants, workstations, and supercomputers), and a wide variety of communications media (wire, optical fiber, terrestrial radio, satellite) with varying degrees of link reliability, bandwidth, and message loss. The reliability requirement means that such applications must degrade gracefully rather than fail in the presence of node or link failures, or with insufficient communications bandwidth and high message loss rates. Since some computational resources may not be available on a continuous basis, applications may have to adapt to varying computational power. The potential for hostile attack to such systems requires that they have a high degree of security, both for authentication of data and privacy of sensitive information.

To facilitate construction of such systems, we are developing a new programming environment which integrates computational, data gathering, data storage, resource management, and human-computer interaction into a common framework. The framework provides high availability and reliability through replication of both data and computational resources and by careful resource management. This programming environment, called SNIPE, is based on technology developed for the Resource Cataloging and Distribution System (RCDS) [1] and the network messaging layers from the Parallel Virtual Machine (PVM) project. [2] We present the basic concepts behind SNIPE, its design, and the current status of the project.

2. RCDS and PVM

The Resource Cataloging and Distribution System (RCDS) is designed to facilitate very scalable and fault-tolerant access to network-accessible resources and metadata, and to provide end-to-end authenticity and integrity guarantees for those resources and metadata.

RCDS accomplishes this by replicating the resources and metadata at a potentially large number of (perhaps geographically dispersed) locations. The set of locations for a resource are maintained in a highly distributed and replicated location registry. Similarly, the metadata for a resource (a list of attribute "name=value" pairs called *assertions*) are maintained in a separate distributed and replicated registry, which is indexed by the resource's Uniform Resource Locator (URL) or Uniform Resource Name (URN). The metadata for a resource is self-defining and can contain elements from arbitrary schema or data models. Subsets of metadata can also be cryptographically signed, using a variety of algorithms, and the signatures provided to RCDS clients. Authentication of resources is accomplished by the use of cryptographic hash functions (such as MD5 or SHA) which are signed by the providers of the information, and made available to clients along with the resource's other metadata.

A primary design goal of RCDS was to facilitate world wide web access to very large and geographically distributed populations on the scale of the Internet - potentially millions of users accessing the same resource at the same time. Scalability and fault-tolerance were therefore paramount in RCDS's design. In replicated databases there are inherent tradeoffs between consistency among replicas and resource availability in the presence of node and link failures. [3, 4, 5] When the semantics of the application permit, higher availability can be obtained by using a consistency model which sacrifices strict atomicity and serializability [6]. With its clean separation between replication of data and metadata, RCDS provides a substrate upon which to implement consistency models of various strengths, according to the needs of the application.

PVM is a library and runtime system that transforms heterogeneous networks of workstations (NOWs) and massively parallel supercomputers (MPPs) into a scalable virtual parallel computer. PVM provides an easy-to-use programming interface for several high level languages. It has facilities for process creation and monitoring, inter-process message passing, multicast message passing, and asynchronous signal delivery. It has a simple facility for global registration of well-known services. It allows tasks to detect and recover from failures of other tasks. It is very portable, having been adapted to a wide variety of architectures and operating systems. PVM has been widely embellished, for example, to add security [7] and management of computational resources [8], and SNIPE also borrows technology from these efforts.

The PVM system has proven to be highly useful for supporting large-scale distributed scientific applications. However, PVM has limited flexibility, scalability, fault tolerance, and security compared to what is needed for critical information analysis applications:

- PVM allows practical scalability to tens of hosts. While larger configurations are possible, limitations in PVM's resource management and internal state management tend to make such configurations unreliable and inefficient.
- PVM can tolerate slave failures but not failure of its master host. It also cannot tolerate link failures during host table updates. It can tolerate network partitionings only in the sense that hosts that have been disconnected can rejoin the virtual machine for new computations.
- The PVM resource manager uses centralized decision making. This would be a bottleneck for a very large virtual machine. If the single resource manager fails, the default built-in allocation scheme will make inefficient use of computational resources.
- PVM lacks a global name space. Process names are valid only within a single ''virtual machine.'' While it is possible for new PVM processes to join an existing virtual machine, a process can only be a member of one virtual machine at a time. This limits PVM's applicability for data gathering and visualization applications, where the data gathered might need to be supplied to multiple computational processes, or the data presented derived from multiple computational processes, which are not specifically determined and configured in advance.
- PVM provides insufficient security for large or widely distributed applications.
- PVM lacks built-in facilities for process migration or checkpointing, though it does have low-level system hooks to support condor-based projects[9] such as Co-Check[14]. PVM also lacks facilities for redundant data storage.
- While it is possible for PVM process to run code that is loaded from other network nodes, PVM provides no protection for its hosts against malicious or errant behavior from downloaded code.

3. General Overview of SNIPE and its Components

SNIPE uses the message passing, task management, and resource management aspects of PVM, and uses RCDS as a framework for replication of resource registries and globally-accessible state. PVM's resource management has been extended to allow redundant resource management processes. These facilities have been used as a substrate onto which support for secure execution, checkpoint/restart, and migration of mobile code have been added. The resulting tools (servers and run time libraries) are intended to facilitate construction of very large decision support networks, combining data gathering, computational, data storage, resource management, and human-interaction nodes into a coherent framework. This system contains seven major components: metadata servers, file servers, per-host SNIPE daemons, client libraries, resource managers, ''playgrounds'', and consoles.

3.1 Metadata servers

The RCDS resource catalog is used to store and provide access to data needed for communication between SNIPE processes. Such metadata include:

- information about SNIPE hosts (URI-to-address mappings, host architecture and operating system, network configuration, permissions and public keys),
- location and authentication information (public keys and certificates) for SNIPE processes that require global visibility,
- location information for distributed services which provide service at multiple locations, and
- routing information for multicast groups.

In addition, SNIPE metadata may contain name-to-address bindings for replicated files, including data files consumed or produced by computational nodes, checkpoint files, and mobile code. Finally, the metadata can contain signed descriptions of mobile code, allowing playgrounds to verify the code's authenticity and integrity and to identify the resources and access rights needed for that code to operate.

Because RCDS resources are named by URLs or URNs, SNIPE processes and their metadata are addressable using a widely-deployed global name space. Instead of having isolated virtual machines as in the current PVM environment, any SNIPE process can potentially communicate (subject to access control restrictions) with any other process. Thus data gathering nodes and visualization/control nodes can communicate with a variety of computational tasks, not just those in a particular virtual machine.

3.2 File servers

RCDS file servers will be used to replicate files that are used by SNIPE processes, including data files, mobile code, and checkpoint files, and provide access to these files. Replication daemons on these servers communicate with one another, creating and deleting replicas of files according to local policy, redundancy requirements, and demand. Name-to-location binding for these files is maintained by metadata servers, which are informed as replicas are created and deleted. Access to the files themselves is provided by ordinary file access protocols such as HTTP, FTP, NFS, or CIFS.

3.3 SNIPE daemons

Each SNIPE daemon mediates the use of resources on its particular host. SNIPE daemons are responsible for authenticating requests, enforcing access restrictions, management of local tasks, delivery of signals to local tasks, monitoring machine load and other local resources, and name-to-address lookup of local tasks. Task management includes starting local tasks when requested, monitoring those tasks for state changes and quota violations, and informing interested parties of changes to the status of those tasks (exit, suspend, checkpoint).

3.4 Client Libraries

The SNIPE client libraries provide interfaces for resource location, communications, authentication, task management, and access to external data stores. Resource location allows the client to obtain information about named resources (hosts, processes and data files) including location, characteristics, public keys, certificates, etc. Communication includes message passing, routing (especially between different types of network media), fragmentation, data conversion (e.g. between different host architectures), and optionally encryption, as well as the ability to use different kinds of media (IP, ATM, Myrinet, etc.) Task management includes the ability to initiate tasks (either directly or via a resource manager) and monitor changes in those tasks.

3.5 Resource Managers

Resource managers are tasked with managing resources and monitoring the state of the resources they manage. Such resources can include hosts, processes, and file servers. A resource manager may manage resources for several hosts at once. For the sake of redundancy, any host may be managed by multiple resource mangers. Resource management functions include allocating resources as needed from those available, attempting to adhere to resource allocation goals, and enforcing restrictions on the use of resources. Depending on configuration, resource management may either be ‘‘passive’’ (allowing a process to reserve resources on a particular host, without actually providing the access to those resources), or ‘‘active’’ (where the resource manager acts as a proxy for the requester, allocating resources on its behalf.) In the latter mode, a resource manager may actually suspend, kill, or (if the code is mobile) migrate processes between hosts. Resource managers are also responsible for clarifying requests for resources, selecting the actual resources in response to a request. Finally, resource managers may also be used to manage RCDS metadata servers according to demand.

3.6 Playgrounds

A ‘‘playground’’ runs under the supervision of a SNIPE daemon and facilitates the secure execution of mobile code. It is a trusted environment which safely allows the execution of such code within an untrusted environment.

The playground is responsible for downloading the code from a file server, verifying its authenticity and integrity, verifying that the code has the rights needed to access restricted resources, enforcing access restrictions and resource usage quotas, and logging access violations and excess resource use. It also provides a run time environment for the untrusted process which generally allows it access to the functions of the SNIPE client library, but which enforces access restrictions.

While SNIPE playgrounds are capable of supporting native code, we anticipate that most mobile code will be written in a machine-independent language such as Java, Python, or Limbo, or some other language specifically designed to allow controlled execution of untrusted code. Implementations of such languages may also be able to arrange the allocation of program storage, in a way that facilitates checkpointing, restart, and migration. When possible, the playground provides hooks for checkpointing, restart, and process migration for use by resource managers.

3.7 Consoles

A SNIPE console is any SNIPE process which communicates with humans. Communication can be via a character-based or graphical user interface. A SNIPE process can also function as an HTTP server, allowing text and graphical output and forms and mouse-click input from any web browser. A SNIPE-based HTTP server can register a binding between a URN or URL and its current location, allowing a web browser to find it even though it may migrate from one host to another, or if the HTTP server is replicated across multiple hosts. SNIPE will make use of standards for Internet resource registration, as those standards are developed. In the meantime, a proxy server is available which allows any web browser to resolve the URI of any RCDS-registered resource (including SNIPE files and processes).

Just as in PVM, there can be multiple SNIPE consoles for any particular application. However due to the highly distributed nature of SNIPE, and the fact that there is no SNIPE "virtual machine" apart from the entire Internet, there is no way to list all SNIPE processes. The state of each process in a process group is maintained as metadata associated with that process group, which can be queried by any process with appropriate credentials. Similarly, the SNIPE processes which were initiated by the SNIPE daemon on any particular host are registered in metadata associated with that host.

4 Implementation Details

4.1 Host Environment

SNIPE nodes can vary in power from personal digital assistants to supercomputers. The only minimum requirement is an Internet Protocol (IP) implementation, though other protocols can be used -- either via a gateway (for non-IP capable hosts), or between IP-capable hosts that also share a faster communications medium. While all SNIPE host environments provide communications and therefore the ability to access and manipulate SNIPE-registered resources, a preemptive multitasking operating system with reasonable security is generally required for implementation of SNIPE file servers, daemons, resource managers, and playgrounds. Due to a lack of computational resources, less powerful host environments may lack some security features.

4.2 RCDS Data Structures

4.2.1 Host Metadata

A host is a resource on which processes can be spawned. It may have one or more CPUs, and one or more network interfaces. There may be restrictions on the use of certain CPUs and/or interfaces, which are enforced by the host daemon. The actual management of host resources may be performed by the host daemon or one or more broker processes.

So the RCDS metadata for a host includes:

- The distinguished URL for the host
- The number and types of CPUs available on that host
- The number and types of network interfaces available on the host
- The data formats supported by the host
- The protocols supported by the host daemon
- The URL of the host daemon
- The URLs of any brokers which manage this host’s resources
- Authentication credentials -- public keys and key certificates to be used to authenticate the host to potential clients.

The network interface metadata includes such things as protocol (IPv4, IPv6, Myranet, raw ATM), addresses, per-message latency, and bandwidth. For IP networks, the netmask is also included; for non-IP networks, a "net name" (which is shared by all hosts on that network, but otherwise globally unique) is included. This information is used by the message routing library to choose an efficient path to the destination, taking advantage of fast, private, and/or non-IP networks where available. It is also used to determine where to establish multicast routers.

4.2.2 File Server Metadata

A file server is a host which is capable of spawning ‘‘file sinks’’, which accept data from SNIPE processes to be stored in files, and make that data available to other processes. The files thus stored may be replicated to other locations, and made available by multiple protocols such as http and ftp. The RCDS metadata for a file server includes:

- A URL for that file server
- The protocols via which data are accepted
- The protocols via which data are provided

A single host may provide both computational and file storage services, in which case both kinds of metadata are used.

4.2.3 Process Metadata

A process runs on one or more hosts and provides computational or other services. The process metadata allows other processes to monitor it or communicate with it. The process also has a ‘‘notify list’’ of other processes which wish to be notified if a process changes state. This metadata includes:

- The distinguished URN for that process
- The supervisor LIFN [15] for the process
- The communications addresses for the process (including interface characteristics such as netmask or net name)
- The ‘‘notify list’’ for that process. Each member of the notify list is a URN of another process.

4.2.4 Multicast Group Metadata

A multicast group is a named group of processes, to which one can send a message as if it were a single process. The actual routing of multicast messages is performed by host daemons which elect themselves as multicast routers on a per-group basis. The RCDS metadata for a multicast group exists to allow other hosts to find the multicast routers for a particular group, and thus join or leave the group. It includes:

- The name of the multicast group (a URN or URL)
- The URLs of multicast routers for the group
- A ‘‘notify list’’ of processes that wish to be notified if the set of multicast routers changes.

4.3 Unicast Message Routing

Unicast message routing is performed using the RCDS metadata for the destination process, and the RCDS metadata for the host on which that process currently resides. If the source and destination are on a common private network or common IP subnet, the message is sent using the fastest of those. Otherwise, the message is sent using the host’s normal IP routing.

4.4 Multicast Message Routing

Multicast messages are sent to one or more host daemons which are acting as routers for that particular multicast group. Each router is responsible for relaying messages to a subset of the processes in the group, and to other routers which have not received the message. Whenever a process joins a multicast group, its host daemon heuristically determines (based on the presence or absence of other routers in the group, and the networks to which those routers are attached) whether it should become a router for that group.

For the sake of fault-tolerance, each process wishing to participate in a multicast group may register its membership in the group with multiple multicast routers. Each router which adds itself to the group also registers itself with more than half of the other routers for that group, and any message sent to that group is initially sent to more than half of the routers for that group. This is intended to ensure that there is at least one path from the sending process to each recipient process.

4.5 Spawning Processes

A request to spawn a process is made relative to a particular host, or more generally, to a set of resources named by a URL (of which a URL for a host is a special case). The request is accompanied by a specification of the program to be run and the environment which the program requires. For instance, the program may require direct access to a particular network or resources, it may run only on certain CPU types, it may require a certain amount of memory or CPU time or local disk space. If the program must be run on a particular host, that is also part of the environment specification.

If the RCDS metadata for a host contains a list of brokers, the request to spawn is sent to one of the brokers for that host. Otherwise, the request is sent to the host daemon. The host daemon may handle the request itself, or refer the request to a broker.

Whichever host daemon or broker actually the process will also create a distinguished URL for the process and associate the per-process RCDS metadata with that URL. This makes the new process globally visible so that other processes can find it and communicate with it.

4.6 Process Migration

Process migration is performed as follows:

1. A process wishing to migrate arranges for a ‘‘clone’’ of itself to be spawned on some other host.
2. The original process suspends all normal processing and saves its state.
3. The original process transfers its state to the clone. The state consists of the process’s variables as well as the state of its message routing routines - including messages which have been received for the process but have not yet been read, messages queued for writing but not yet written, messages written but not yet acknowledged, taskid-to-URL mappings, multicast routing tables, etc.
4. The original process tells the clone process to restore the saved state and resume processing.
5. The original process sends redirects to all processes with which it has an open connection, to inform them of the new location of the process.
6. The original process changes its RCDS metadata to reflect that the location of the clone is now the location of the process. In particular, the supervisor URL and communications URL are changed.
7. The original process notifies each of the processes on the ‘‘notify list’’ that the process has moved to a new location.
8. If possible, the original process remains for a time to forward messages to the new location of the process, and/or to send redirect messages to processes that continue to send messages to the old location of the process.

Some programming environments are designed to make it easy for a process to be migrated from one host to another without explicit code in the program to perform that operation. For such programming environments, the details of process migration may be arranged by the host daemon, but otherwise the procedure is the same.

If a process migrates while other processes are communicating with it, those other processes will need to be notified of the new location of the process. This can happen in any of several ways:

- If a process has an open communications channel with a process that is migrating, it will receive a redirect message from the migrating process.
- If a process is registered on the ‘‘notify list’’ of a migrating process, it will receive a redirect from the process when it migrates.
- If process *A* attempts to send a message to the old location of process *B*, and process *B* still has remained in place to forward messages, process *A* will receive a redirect from process *B*.
- If process *A* detects an error while sending to the former location of process *B*, or fails to receive an acknowledgment from process *B*, process *A* will then obtain the new address of process *B* via RCDS, and begin routing messages to the new location.

4.7 Replicated Processes

Several kinds of replicated processes are supported by SNIPE:

- If several computational processes are run concurrently, provided with the same input, and expected to produce the same result, a multicast group can be created to provide input to all of those processes. RCDS metadata can then be created for the new pseudo-process, consisting of all of the processes in the group, and with the multicast group listed as the communications URL. All data sent to the pseudo-process will then be transmitted to each member of the group. However, if multiple processes send data to that multicast group, there is no assurance that each of those replicated processes will receive the data in the same order. Also, any time a message is sent to that pseudo-process, the process sending the message may receive one or more error messages resulting from failure to deliver the message to particular processes, even though the message may have been delivered to other processes in the group. This will confuse a sender process which expects the message delivery to either succeed or fail.
- If it is desirable to provide a service at multiple locations, using multiple protocols, or at multiple hosts, a LIFN can be created for that service, and each of the service locations (URLs) associated with that LIFN. Any process attempting to communicate with that service will then see multiple service locations (URLs) from which to choose.

4.8 Playgrounds

A process may be executed on a host subject to certain restrictions. The host daemon is responsible for enforcing those restrictions. If the restrictions are severe, the host daemon may execute the process within a playground. A playground is an environment which enforces restrictions that cannot easily be provided via the normal operating system. It may, for instance, limit access to local files, or to the machine’s network interfaces, or the amount of cpu time or memory used.

A playground may provide a restricted environment for the execution of ‘‘native’’ programs, or it may provide an environment for the execution of programs believed to be ‘‘safe’’, such as Java bytecode, safe-tcl,etc. In either case the playground is responsible for verifying the authenticity and integrity of the program, and checking the credentials of the process making the request to ensure that the process has the appropriate permissions.

Implementation of playgrounds varies widely from one platform to another, and not all platforms are capable of imposing the restrictions which may be required without modification to the operating system. Native code playgrounds are complex to implement and difficult to verify. A playground’s capabilities are therefore advertised as RCDS metadata, which can be used by a process or resource manager in scheduling mobile code.

4.9 File Servers, Sinks, Sources and I/O

SNIPE file servers provide the ability for SNIPE processes to store data in files and retrieve the data from those files, using the normal message passing routines used to send messages between processes. A ‘‘file sink’’ process reads SNIPE messages sent to it and stores them into a file. A ‘file source’’ process reads a file consisting of SNIPE messages and sends them to a SNIPE address. Opening a file for writing thus consists of spawning a file sink process which will store its output in such a way that it can be accessed by another process via its URN or URL. Opening a file for reading is implementing by spawning a file source process which reads a particular file (named by a URN or URL) and transmits that to a particular SNIPE address.

SNIPE file servers can also be used to access ordinary data files via URLs and LIFNs, and to export data to files which can then be accessed by external programs using common protocols such as HTTP. This requires that the SNIPE process create the file on a SNIPE file server which also supports the HTTP protocol.

5. Current Testbed status

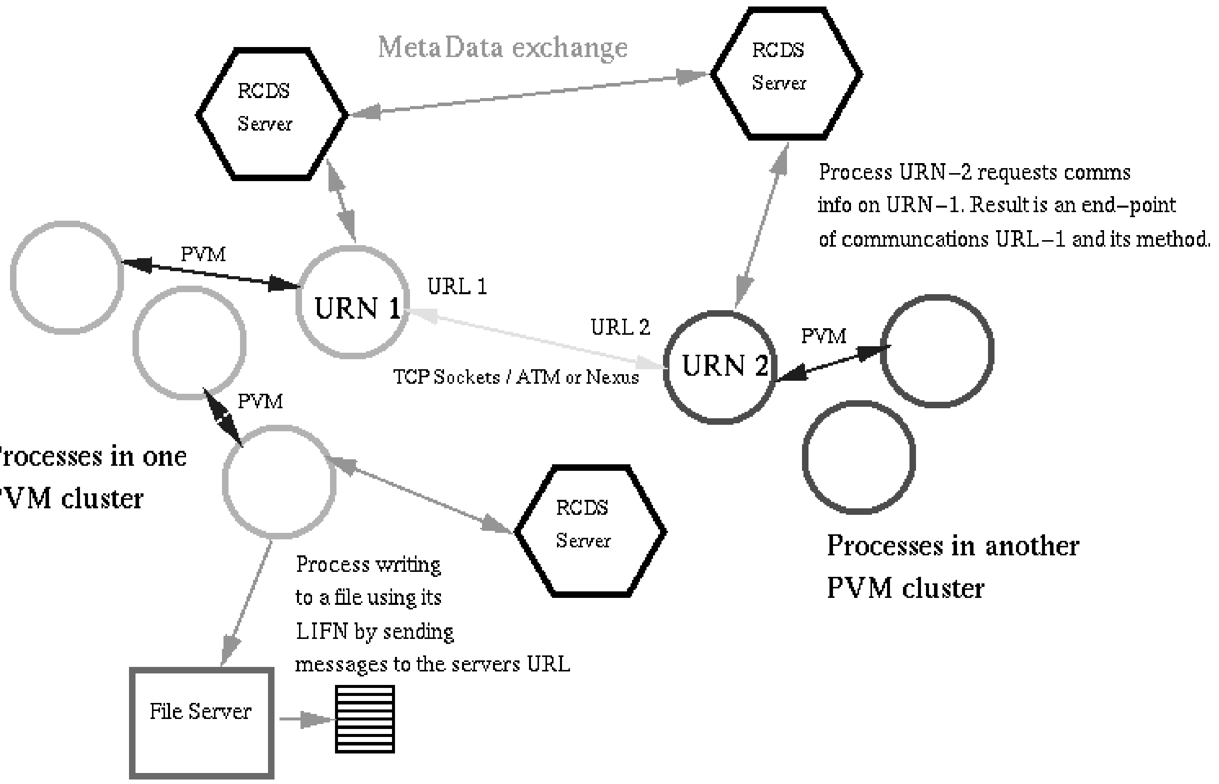
5.1 Implementation Status

A prototype of SNIPE is currently (August 1997) under construction. This prototype uses the PVM message passing libraries to communicate locally between processes within the same cluster, and the RCDS metadata server to store globally accessible state such as the locations and characteristics of processes and files. HTTP servers are used for file storage and retrieval. Per-host process management is accomplished by a modified PVM daemon. The PVM library is being modified to use RCDS calls for process registration and routing, and to allow sending PVM messages to files, and receiving PVM messages from files. A secure execution environment developed for NetSolve [13] is used as a playground.

5.2 Internals of Testbed

The testbed consists of multiple clusters of machines, where each cluster belongs to a single PVM virtual machine. A number of clusters together form the overall SNIPE system, and allow the testing of true global naming, inter-cluster communication, multiple communication methods, process replication, and resource management techniques etc.

Each cluster uses PVM functions to communicate within itself, as the "*PVM method*" is listed as the default local communication method for any processes within a cluster. Inter-cluster communication is identified by processes with differences in their communication addresses in much the same way as IP systems use differences between local addresses and subnet masks. Once non-PVM communications is detected, then a default TCP socket library is utilized, unless other "faster" methods are available to both parties such as Nexus for example. The following figure shows two clusters, a file server and two processes that are inter-communicating using a method and address obtained from a nearby RCDS server.



To improve performance the local library system caches as much information as possible, to avoid excessive loading upon the RCDS servers. Different types of MetaData become stale in different ways: Communication end-points can become stale when communication calls timeout or return errors, or if redirects or notify messages are received. Likewise, process identifiers can become stale when processes or their hosts crash. In these events, RCDS servers are polled to identify the state of the stale entities and then default polices are followed such as re-establishing communication using different communication methods, respawning of processes or returning an error to the users application.

The only modifications to PVM currently effect only the PVM daemons. They now register new processes with RCDS and obtain URNs. Perform multicasting via new multicast processes (instead of by the other PVM daemons) and all process creation operations such as spawn and connection of non-spawned processes have to be approved via a SNIPE resource manager before allocation of a URN. The SNIPE resource manager used is based on the PVM General Resource Manager(GRM)[8].

The current system has demonstrated process creation, suspension and migration as well as on-the-fly communication method re-configuration such as switching from ATM (155Mb/sec) to 10Mb ethernet dynamically while maintaining correct message ordering transparently without the user application having to intervene.

Initial timing tests of process creation across clusters compared to within a single cluster, and the subsequent communication between processes appears encouraging. In a finial system any additional SNIPE overhead will be negligible compared to any strict form of authentication that maybe imposed.

6. Future Plans

A final system will be developed from scratch utilizing the lessons learnt from the initial testbed. The testbed itself will be used to develop a better understanding of algorithms, methods and policies that work (or don't work) on a number of key exemplar large scale applications such as Netsolve and PVMPI. From these lesions the internal policies, protocols and security methods will be tuned before the final system is released.

This is expected to include:

- steamlining SNIPE daemons
- non-RPC based RCDS servers
- faster and more efficient MetaData transfer methods between RCDS servers
- more intelligent resource management polices
- automatic data replication of files to improve system throuhput

Inter-operation between SNIPE and other MetaComputing systems such as Legion and Globus is also being considered, even if this is just at the MetaData (resource information) sharing level.

7. Comparison with related work

The Legion project at the University of Virginia aims to provide an architecture for designing and building system services that afford the illusion of a single worldwide virtual machine. [10] The Legion virtual machine is intended to provide secure shared objects and shared name spaces, application adjustable fault-tolerance, and good performance. The Legion goals are similar to those of SNIPE. However, the SNIPE approach is to use self defining interface definitions that allow various modules, rather than objects, to interact. Facilities of the Legion system that are incorporated into Legion core objects will be supported in SNIPE by separately loadable modules. SNIPE will support all the core functions planned for Legion, including a scalable, distributed architecture with a single name space, user-selectable levels of security, and fault tolerance.

Globus is a project of Argonne National Laboratory that aims to provide the basic software infrastructure for computations that integrate geographically distributed computational and information resources. [11] The Globus system offers applications a number of services that include security, resource management, and communication. As compared to SNIPE, Globus has only a very limited form of fault tolerance, and is less able to adapt dynamically to application needs.

A part of the Globus project is the Metacomputing Directory Service (MDS), which is intended to provide efficient and scalable access to distributed information about computers, networks, and other resources [12] The MDS data representation and application programming interface are adapted from the Lightweight Directory Access Protocol (LDAP), with wrappers for other information access protocols such as NIS, NIS+, DNS, and SNMP. MDS is intended to facilitate the development of flexible and efficient distributed computing services and applications. MDS has similar goals to the RCDS extensions for SNIPE. The principal difference between RCDS and MDS is that RCDS is designed for very high scalability, fault tolerance, and ability to quickly propagate frequently changing metadata. RCDS (and therefore SNIPE) can use URLs as resource names and DNS to locate RCDS servers, thus taking advantage of the widely deployed DNS infrastructure.

8. Summary

SNIPE's goal is to facilitate easy construction of large scale, highly distributed, secure, and reliable information processing applications. To this end, the SNIPE programming environment provides facilities for process initiation (including initiation of redundant processes), secure interprocess communication (unicast and multicast), process monitoring, redundant data storage, checkpointing, process migration, resource registration and location, and resource management. Computational processes, files, and services can all be replicated to provide increased fault-tolerance; the locations of replicated processes, files, and services are maintained in a database which is itself replicated and can be accessed by any SNIPE process. The degree of replication (and therefore fault-tolerance) can be adjusted according to the needs of the application.

SNIPE has ambitious goals. In the short term, SNIPE provides vastly improved flexibility, scalability, fault-tolerance, and security as compared with PVM. Eventually, we hope to provide an environment reliable enough to support applications such as those described in section 1. In the meantime, SNIPE serves as a research platform for investigation of highly distributed, fault-tolerant, secure information processing systems.

9. Glossary of terms and systems

- DNS - Domain Name Service.
- File Server - A process that stores, accesses and updates files for other SNIPE processes.
- File Source - A short lived process that exists only to read / export files.
- File Sink - A short lived process that exists only to collect input messages (possibly from multiple sources such as cloned processes) and produce output data files.
- GRM - General Resource Manager. A resource manager developed to allow PVM to interface with batch and queuing systems to facility the process control of PVM and MPI applications.
- LDAP - Lightweight Directory Access Protocol.
- LIFN - Location Independent File Name. A way of specifying a file so that it can be accessed in a uniform way independently of its location and that of the its requester.
- MDS - MetaComputing Directory Service.
- Multicast Group - A collection of processes that can be communicated with by sending a message to

a single URN or by relaying a message via Multicast Routers URL.

- NetSolve - A system that allows complex mathematical problems to be solved from a desktop machine by powerful compute servers remotely located across the InterNet.
- NIS - Network Information System.
- Process - Unix level process that is globally identified by a URN.
- Process Group - A collection of processes that have an affinity and can by accessed by a single URN.
- Playground - A safe execution environment where access to facilities/services is strictly controlled.
- PVM - Parallel Virtual Machine.
- PVMPI - An MPI intercommunication system that allows vendor specific MPI applications to interoperate using the PVM infrastructure without losing local MPP communication performance.
- RCDS - Resource Cataloging and Distribution System.
- RCDS Server - A process that manages MetaData on behalf of the SNIPE system.
- URI - Uniform Resource Identifier. A generalized term for either URN, URL or LIFN.
- URN - Uniform Resource Name. Used by SNIPE to identify entities such as hosts, processes etc.
- URL - Uniform Resource Locator. Specifies a location where a service or end-point of communication (port) may exist. Used for non-tangible entities.

10. References

[1] K. Moore, S. Browne, J. Cox, and J. Gettler. The Resource Cataloging and Distribution System. Technical report, Computer Science Dept., University of Tennessee, December 1996.

[2] V. Sunderam, J. Dongarra, A. Geist, and R. Manchek. The PVM concurrent computing system: Evolution, experiences, and trends. *Parallel Computing*, 20(4):531-547, April 1994.

[3] D. Skeen and M. Stonebraker. A formal model of crash recovery in a distributed system. *IEEE Transactions of Software Engineering*, SE-9(3):219-228, May 1983.

[4] S. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in partitioned networks. *Computing Surveys*, 17(3):341-370, September 1985.

[5] K. Ramarao. Transaction atomicity in the presence of network partitions. In *Proc. Fourth International Conference on Data Engineering*, pp 512-519. IEEE Computer Society Press, February 1988.

[6] M. Herlihy. Dynamic quorum adjustment for partitioned data. *ACM Trans. Database Syst.*, 12(2):170-194, June 1987.

[7] T. H. Dunigan and N. Venugopal. Secure PVM. Technical Report ORNL/TM-13203, Oak Ridge National Laboratory, August 1996.

[8] G. E. Fagg, K. S. London, and J. J. Dongarra. Taskers and general resource managers: PVM supporting DCE process management. in *Third European PVM Users' Group Meeting*, Munich, Germany, October 1996.

[9] J. Pruyne and M. Livny. Managing Checkpoints for Parallel Programs. http://www.cs.wisc.edu/condor/doc/ckpt_mgmt.ps

[10] A. S. Grimshaw and W. A. Wulf. The Legion vision of a worldwide virtual computer. *CACM*, 40(1):39-45, January 1997.

[11] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, (to appear) 1997. Available at ftp://ftp.mcs.anl.gov/pub/nexus/reports/globus.ps.Z

[12] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A directory service for configuring high performance distributed computations. In *HPDC '97*, 1997. Available at ftp://ftp.mcs.anl.gov/pub/nexus/reports/hpdc97_mds.ps.Z

[13] H. Casanova and J. Dongarra. Netsolve: A network server for solving computational science

problems. In *Supercomputing '96*, Pittsburgh, PA, November 1996.

[14] Georg Stellner and Jim Pruyne. Resource Management and Checkpointing for PVM. *Proc EuroPVM95*, pp. 130-136, Hermes, Paris, 1995.

[15] Shirley Browne, Jack Dongarra, Stan Green, Keith Moore, Theresa Pepin, Tom Rowan, Reed Wade, and Eric Grosse. Location-Independent Naming for Virtual Distributed Software Repositories. University of Tennessee, Department of Computer Science Tech Report ut-cs-95-278, February 1995.