Scalable Networked Information Processing Environment (SNIPE)

Graham Fagg[1]
*Keith Moore*0
Jack Dongarra[2]
Al Geist[3]

Extended Abstract

1. The Challenge

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 accomodate fluctations 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 authentation of data and privacy of sensitive information.

To facilitate construction of such systems, we are developing a new programing 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

---

3

[1] The University of Tennessee

[2] The University of Tennessee, Oak Ridge National Laboratory

[3] Oak Ridge National Laboratory

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 asychronous 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 computional 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 mutliple 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 such as Co-Check [9]. 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. SNIPE

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 facilitiate 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 requestor, 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. Inplementations 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 appliation. 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. 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.

5. 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.

6. Implementation Status

A prototype of SNIPE is currently (May 1997) under construction. This prototype uses the PVM message passing libraries to communicate between processes, 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.

7. 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 enviroment provides facilities for process initiation (inculding 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.

8. 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.