

On the Management of Large-Scale Clusters: A Progress Report

Hardik Shukla
East Tennessee State University

Abstract

Oak Ridge National Labs (ORNL) is studying the problem of large-scale cluster administration. The work at ORNL seeks to develop public domain tools that allow individual system administrators to effectively administer clusters of thousands of machines.

This paper describes recent work done by the author related to the ORNL cluster computing initiative.

The first part of this paper discusses a new, public domain implementation of the rsh protocol for Microsoft Windows. Rsh, or remote shell, is a protocol for using accounts on remote machines to execute commands. Rsh is the basis for a variety of tools that support distributed and cluster computing, including PVM^[8] and C3^[14]. Other current Windows implementations of rsh are third-party commercial products, provided by companies other than Microsoft.

The second part describes research related to the development of the efficient and robust distribution of files within clusters. Currently, the tool of choice for distributing files across the cluster is rdist^[13]—a freeware tool for maintaining identical copies of a file across the network. This paper describes progress towards an alternative implementation of rdist that distributes a file across a cluster-using multicast^[3].

1 Introduction

A *cluster* is a group of servers and other resources that act like a single system and enable high availability and—in some cases—load balancing and parallel processing^[1]. Cluster computing is the use of a cluster's shared resources to solve a specialized task. Because of the significant reduction in the costs of workstations and high speed networking devices, cluster computing has a potential to offer the benefits of supercomputers at significantly low costs.

For optimal performance, a cluster's resources must co-operate efficiently. It should be possible to add new resources to the cluster or remove any failed resource from the cluster dynamically, distribute jobs dynamically to any resource within a cluster, or share information among a cluster's resources. Cluster maintenance is a subset of cluster computing that deals with the above-mentioned issues.

In order to distribute a job to any resource within a cluster, it is necessary to contact the resource and start the job on that resource. Remote Shell, popularly known as rsh, is a tool based on the rsh protocol for starting a job on a remote machine. Currently Microsoft does not support the rsh tool completely. The first part of this paper describes the development and implementation of a rsh tool for the Windows Operating system (WinRshTool) under the guidance of Dr. Stephen Scott (High Performance Computing Division^[2], Oakridge National Labs).

Sharing information within a cluster is another cluster maintenance problem. File distribution is one way to share information within a cluster's resources. The second part of this describes research done by the author related to the development of an efficient and robust file distribution tool based on multicast^[3].

2 WinRshTool

Popular cluster computing tools like PVM^[5] and MPI^[6] use the rsh protocol to manage jobs on remote machines. Rsh is a protocol for remote command execution that is implemented in the standard “client-server”^[4] fashion. A program that sends the local user’s commands to a remote machine is called an rsh client. A program that executes commands for an rsh client and returns their results is called an rsh server.

Rsh servers are implemented as daemon processes— programs that run continuously, handling periodic requests for service^[5]. When contacted by a remote user, the rsh daemon verifies the remote user’s identity, and starts a new process to execute valid requests. The standard input, output and error streams of the new process are mapped to the remote client’s standard input, output and error streams. The new process takes the request (commands) from the remote user, processes the request under the remote user’s security context, and returns sends the results to the user.

Heretofore, the ease with which rsh can be obtained has depended on the platform in use. Rsh client and server programs are part of the standard distribution of Unix and all Unix-like operating systems. The Windows Operating System distribution, on the other hand, provides only the client program freely with its standard distribution. Rsh servers have been available, but not for free: users of tools like PVM have had purchase rsh servers from third party vendors like Ataman^[3] or PC Consult^[4] or obtain them by subscribing to Microsoft’s MSDN network.

The author has developed a freeware Rsh implementation, WinRshTool, for the Windows operating system. This implementation will be released with the next PVM release through Oak Ridge National Labs. The rest of this section describes the implementation, and details concerns related to the implementation of WinRshTool.

2.1 WinRshTool Design Considerations

A) Rsh server is implemented as a service

The WinRshTool server, a Windows *service application*, is the Windows equivalent of a Unix daemon process. The server is started by the operating system before any user logs on; runs continuously; and services the user’s requests until stopped, either by the administrator or at system shutdown.

B) Equivalent of `setuid ()` on Win32 Platform

The rsh protocol recommends that the remote user’s commands be executed under the remote user’s security context. In the Unix environment the rsh daemon runs with administrator privileges and uses the “`setuid()`” API to become the remote user. The rsh daemon then execute the remote user’s commands under the security context of that remote user.

Windows NT and Windows 2000, in accordance with C3 security guidelines, does not support a “change identity” API comparable to `setuid()`. Instead, Win32 API uses a technique known as *impersonation*^[6] to allow a service application to assume a user’s identity. Before a service application S may impersonate a user U, S’s host system must verify that S is authorized to impersonate U. Authorization can be accomplished in one of two ways:

- A) Using delegation^[7]. In delegation, U sends an object known as an *access token* to S that gives S the right to impersonate U. S then passes U’s access token to S’s host system, which verifies the token’s authenticity. In order for U’s token to be accepted, either of two conditions must hold:

- Authentication will succeed if U and S belong to the same *domain*—a set of machines that is configured and administered as a unit.
 - If U and S are part of different domains, then authentication will succeed if U belongs to a domain that S's domain *trusts*. Trust is a binary relation on Windows domains that is configured by a domain's administrator, using standard domain administration tools. Trust, incidentally, is asymmetric and intransitive: saying that domain A trusts domain B says nothing about whether domain B trusts domain A, or whether A trusts any other domain trusted by B.
- B) Using passwords. S can also establish its right to impersonate U by providing S's host system with U's password. Passwords must be used to support impersonation when authenticating a service application's right to execute commands on behalf of a user from a remote, untrusted domain.

PC Consult's Rsh package stores a remote user's password on a local machine and uses the stored password to authenticate remote requests. Unfortunately, storing passwords on a machine exposes those passwords to theft by anyone who has physical access to the machine, and the time to reverse-engineer the scheme used to camouflage the passwords.

One alternative to storing passwords is bypassing impersonation through the use of a default account. This approach is either inefficient or unsafe, according to the level of privilege held by the special account. Ataman's Rsh package uses this approach and executes any remote user's commands under the administrator account, thereby giving any remote user full access to the machine.

The WinRshTool uses delegation, where possible, to support the remote execution of user commands. WinRshTool supports the remote execution of commands from untrusted domains using either stored passwords or default accounts, according to the installer's discretion. Stored passwords are encrypted using blowfish, and the passwords stored are decrypted using blowfish to impersonate the user.

c) Evaluation of Environment Variables

The Unix rsh server expands the environment variables in the client's request. Thus, any platform independent implementation of rsh server should be capable of expanding environment variables in the client requests.

WinRshTool translates incoming calls which contain %variable-name% or \$ variable-name primitives to their expanded environment variables. For this, WinRshTool first checks registry key HEX_KEY/SOFTWARE to find an environment variable's value. If this fails, WinRshTool loads the user's registry and checks for the environment variable's value.

2.2 Current Status of WinRshTool

The author has completed the implementation and small scale testing of WinRshTool. After WinRshTool is tested rigorously with PVM, it will be released as a part of the next PVM release.

3 SRFTT (Scalable and Reliable File Transfer Tool)

Scalable and Reliable File Transfer Tool (SRFTT) is a tool developed to simplify information sharing among a cluster's nodes. SRFTT guarantees reliable data delivery for bulk data-transfer applications like file transfer, Web caches preloading, software and software upgrades distribution.

SRFTT implementation combines the following two strategies for error management to achieve reliability:

- 1) Forward Error Correction (FEC)^[10]: In this approach recovery packets are encoded and decoded using software FEC techniques. Data transfer tool like FCAST^[12] and RMDP^[10] are based on this approach.
- 2) Multiple multicast Channels^[11]: In this approach, recovery packets are retransmitted on multiple retransmission channels to reduce receiver reception time.

The rest of this section discusses implementation issues related to SRFTT. Section 3.1 describes SRFTT operation and Section 3.2 describes SRFTT's component software. .

3.1 SRFTT Operation

SRFTT makes four basic assumptions about its operating environment —existence of a multicast link between the sender and the receivers, static nature of receiver set, fixed size source data and heterogeneous nature of receivers.

SRFTT is session based. Each session has an identifier that uniquely identifies the session. Each session has a primary transmission channel to transmit the data packets and multiple retransmission channels to transmit the recovery packets. All the retransmission channels are implemented as distinct multicast channels.

SRFTT operation is divided into the following three steps:

1. The sender transmits session information to all receivers. Session information includes session id, primary transmission channel address, file name, file size and file checksum.
2. The sender transmits the file on the session's primary transmission channel. The receivers join the primary transmission channel to receive the file.
3. The sender transmits recovery packets on the session's retransmission channels. The receivers subscribe to the appropriate retransmission channel to receive the lost packets.

3.2 SRFTT Building Blocks

SRFTT is architected as a sender-receiver application, where the sender acts as a server, and the receivers as clients. The architecture of SRFTT senders and receivers is described in Sections 3.2.1 and 3.2.2 respectively. The SRFTT configuration file, which drives an SRFTT session, is described in Section 3.2.3.

3.2.1 SRFTT Sender

SRFTT is sender-initiated: i.e. the sender initiates the file transfer. The sender divides each file to be transmitted into B groups, with each group having K data packets. The parameter K is fixed and pre-configured. The value of B depends upon the file size and the value of K. A detailed description of B and K is given in Section 3.2.3.

In the first round of transmission, the sender transmits the session information to all the receivers using a reliable data delivery mechanism (TCP/IP). In the next round, the sender multicasts the data packets on the session's primary transmission channel. After sending the data packets, the sender multicasts multiple "stop" packets on the session's primary transmission channel. A "stop" packet is a special packet that signals the end of transmission on the session's primary channel. In subsequent rounds

of transmission, the sender multicasts the recovery packets on all of the session's retransmission channels until a certain degree of redundancy has been achieved. The recovery packets are encoded using software FEC techniques^[10].

The number of retransmission channels is fixed and pre-configured (cf. §3.2.3). If a receiver experiences more losses for a group than the network's pre-determined loss-rate, the receiver requests more recovery packets for the group and the sender responds to the receiver's request by increasing the number of recovery packets for that group.

3.2.2 SRFTT Receiver

On receiving the session information from the sender, the receiver joins the session's primary transmission channel to receive the file. The file is divided in $B * K$ packets, where B = number of groups and K = number of packets per group. (cf. §3.2.3). If a receiver receives all data packets correctly, the receiver leaves the primary transmission channel and reassembles the file from the received packets.

If a receiver fails to receive all the data packets correctly, the receiver joins the appropriate retransmission channels to recover the lost packets; receives the required number of recovery packets—i.e., the number of data packets lost for that group; leaves all retransmission channels; decodes the parity packets; and reassembles the file. To recover packets lost from a group i , the receiver joins the retransmission channel $i \% T$, where T = total number of retransmission channels.

When a receiver determines that it is unable to receive sufficient recovery packets for any group, the receiver sends an “increase packet” request to the sender, requesting more packets for that group. To prevent the sender from being flooded with “increase packet” requests, every receiver waits a random amount of time before sending an “increase packet” request. The receiver aborts a pending request for more packets if one of the following occurs:

- 1) The receiver receives “increase packet” request from any receiver that can satisfy the receiver's need for extra recovery packets the receiver needs
- 2) The sender starts transmitting sufficient number of recovery packets because of “increase packet” from another receiver.

3.2.3 SRFTT configuration file:

SRFTT uses a configuration file to configure parameters like packet size, group size (number of packets per group), number of retransmission channels and network loss. For optimal SRFTT performance, these parameters must be tuned to the given application or environment. This sub-section is a detailed description of the SRFTT configuration parameters.

A) Packet size: IP multicast is built on the top of IP protocol and IP multicast packets are transmitted as IP datagrams. The network infrastructure de-fragments a packet in multiple IP datagrams if the packet size exceeds a certain limit (normally 512 or 1024 bytes). Increasing the packet size helps to transmit the file in fewer packets and, thus reduces overhead, while increasing the cost of fragmentation and reassembly.

B) Group size: Group size is defined as number of packets in a group. As the number of packets in a group increases, the robustness of the code (in terms of erasure recovery capability), but it also increases the encoding and decoding costs^[10].

C) Number of retransmission channels: If the number of SRFTT retransmission channels is less than the number of groups in a file, some retransmission channels must carry recovery packets for two or more groups. In this scenario, multicast groups that support multiple retransmission channels multiplex among their assigned channels, delaying retransmission, but decreasing the overall load on the network.

D) Loss rate: A network's loss rate is defined as the number of packets lost by the network per 100 packets transmitted. SRFTT initially transmits FEC packets depending upon a pre-decided loss rate of the network. Different networks have different loss rates. If the initial loss rate is configured below the network loss rate, the network load increases because of the retransmission requests by multiple receivers [refer 4.2.2]. On the other hand, if the initial loss rate is configured above the network loss rate the network load increases because of the sender transmitting unnecessary FEC recovery packets.

3.3 Current Status of SRFTT

The author has completed the implementation of SRFTT. After SRFTT is tested rigorously, it will be released as a freeware sometime in May.

References:

- 1) http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci211796,00.html
- 2) <http://csm.ornl.gov/~sscott>
- 3) http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci212610,00.html
- 4) http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci211796,00.html
- 5) http://searchsolaris.techtarget.com/sDefinition/0,,sid12_gci211888,00.html
- 6) http://msdn.microsoft.com/library/default.asp?url=/library/en-us/security/security/client_impersonation.asp
- 7) http://msdn.microsoft.com/library/default.asp?url=/library/en-us/com/security_0dri.asp
- 8) www.epm.ornl.gov/pvm
- 9) <http://www-unix.mcs.anl.gov/mpi/>
- 10) A Reliable Multicast data Distribution Protocol based on software FEC techniques (RMDP) by Luigi Rizzo and Lorenzo Vicisano
- 11) S. Kaser, J. Kurose, and D. Towsley, "Scalable Reliable Multicast Using Multiple Multicast Channels"
- 12) <http://research.microsoft.com/barc/mbone/fcast.asp>
- 13) www.tac.eu.org/cgi-bin/man-cgi?rdist+1
- 14) <http://www.csm.ornl.gov/torc/C3>