

HPSSQ, A Software Facility To Handle HPSS Downtime and WAN Data Transfer

M. L. Chen
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN 37831

Abstract

To improve the reliability of user access to the High Performance Storage System (HPSS) and to remedy problems associated with relatively low Wide-Area Network (WAN) transfer rates during data transfer between supercomputers and HPSS installations remote from one another, a software facility - HPSSQ - has been developed in the Computer Science and Mathematics Division at Oak Ridge National Laboratory (ORNL). This paper describes the architecture, structure, and major features of this software. The test results of this facility on an IBM SP3 Supercomputer Eagle are also reported.

Acknowledgements

HPSSQ has been developed in the ORNL/Probe research facility funded by a contractor of the U.S. Government under Contract No. DE-AC05-00OR22725. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes by DOE Office of Science. The author would like to acknowledge the support of many colleagues in the Center for Computational Sciences at ORNL. Especially I am greatly indebted to R.D. Burris for his leadership and supervision in this project. The contributions from D.L. Million to the security issues and his consistent support are highly appreciated. I would also like to thank M.K. Gleicher of Gleicher Enterprises for his valuable help on HSI software.

Introduction

Two major concerns for transferring data between supercomputers and HPSS installations are HPSS maintenance downtime handling and relatively low WAN transfer rate. If a file transfer request is made during HPSS downtime, the user gets an error return without any further information. The burden is on the user to find out the HPSS status and to resubmit the request. Furthermore, the nodes of the supercomputer where the files originate may hang because of HPSS unavailability. If the transfer involves a remote HPSS, poor WAN transfer rates will result in extended periods of workstation or supercomputer resource use. And again, if there is an unexpected network problem, users only get an error return and have to keep resubmitting until success.

HPSSQ, a software facility with decoupling and persistence features developed in the ORNL/Probe research facility and migrated to supercomputer Eagle, provides an approach to solve these problems. It saves computer resources significantly and frees users from the necessity of detecting errors and resubmitting.

Architecture

A HPSSQ server runs on a supercomputer node. Users submit "hpssq" commands (see attachment: HPSSQ User interface) interactively or through batch jobs, to request data transfers between the computer and an HPSS installation (Figs. 1 & 2, Architecture of HPSSQ). The HPSSQ client transfers the user request and necessary environment information to the server. The client's workstation or supercomputer node is released promptly after receiving the server's acknowledgement.

The server queues requests in a dynamic list. By means of the authentication information carried by the environment parameters, the server can handle the security issues and act as the user to execute the request. One thread is dispatched for each new transfer.

Requests are executed concurrently. Transfers are implemented using the Hierarchical Storage Interface (HSI), a utility whose features include recursion, wildcarding, and direct transfers between HPSS installations [1].

An interactive mode server can be created upon user's request and runs in user local directory to transfer data to HPSS installations in remote Kerberos realms.

Technical Significance

The HPSSQ server is a multithreaded process specially designed to be capable of accepting thousands of requests in seconds. This feature removes the bottleneck which causes submission failure when numerous users submit multiple requests simultaneously. HPSSQ performs multiple transfer processes concurrently. The multiplicity is only limited by the computer resources and HPSS I/O capability.

User submits the request only once. If a transfer fails, it will be automatically resubmitted at time intervals governed by environment parameters. When a request completes, a status report is emailed to the user. The request queue can be automatically recovered after an incident such as a power failure.

A sophisticated error handling system is carefully designed and implemented. Errors are coded and partitioned into different sections by their type and resource respectively. Error codes are then transferred into status words. A decoded clear text error report, if any, is sent to users by email.

This facility maintains the integrity of all HSI security features. It also provides a friendly interface to users (See attachment: HPSSQ User interface), and supports requests submitted in interactive mode or through batch jobs. It unburdens users and saves computer resources significantly.

Software structure

1. Overview

To quickly decouple users from data transfer processes, a client/server structure is established. HPSSQ employs a modularized code structure to make quick development and easy debugging possible. Six modules provide all basic function blocks. Each module can be developed and debugged by built-in testing routines individually. Client and server modules are built on top of these basic function blocks.

2. Server

Accepting thousands of requests in seconds and handling multiple transfer operations concurrently imposes stringent technical requirements on running speed, efficiency, reliability, and so on. A software structure has been designed to specifically meet these challenges. The server, a multithreaded process, executes concurrent tasks such as request handling, buffer management, queue control, HPSS access and resubmission, file management, etc. It provides two layers of buffering, a circular buffer and a dynamic list, to ensure satisfactory decoupling functions. (Fig. 3, Server software structure of HPSSQ)

Two dynamic queues are controlled by three major threads - action manager, retry manager, and queue manager. One queue contains data blocks of all submitted requests. The action thread scans this queue downwards quickly and dispatches one thread to execute each request. In this way, requests are executed concurrently and newly submitted requests can be executed promptly. Status of each transfer is monitored and recorded. In case of failure, the request will be re-submitted into this queue once more for a quick retry. If the quick retry fails again, the pointer of this request block together with a control parameter structure will be appended to the other queue named retry-Q. The retry thread dispatches one thread for each request listed in retry-Q to resubmit the transfer according to a time schedule, which is governed by environment parameters, until success. The queue manager thread manages the queue, provides queue access to users upon request, and emails users upon completion.

According to the software design, there is ideally no limit on the total number of concurrent threads except the one given by thread library (32767). However a practical machine dependent limit exists because of limited available resources. A thread management method is designed to control the thread creation, monitoring, and the resource re-collection.

A large circular buffer managed by the circular-buffer thread offers the capability of accepting thousands of requests within a very tiny delta T. The server is therefore able to allow users submit requests essentially simultaneously.

3. Client

Client software can run in three modes - daemon server mode, interactive server mode, and direct access mode (Fig. 4, Client software structure of HPSSQ). Mode selection can be specified by switches of “hpssq” command.

The daemon server mode is the default mode and uses Kerberos authentication method. The HPSSQ client process communicates with the server, which runs constantly and manages the transactions. The client is released promptly after receiving acknowledgement from the server.

The HPSSQ daemon server has the capability to handle remote site transfers. Since authentication methods are widely diverse between different HPSS sites, the authentication of remote sites is still awkward.

The interactive server mode provides a way to meet the authentication challenge. In case of a remote site configured with a different authentication method, upon user request, the client process forks a server as a child process to receive the user requests and to manage the transaction. The client is released after the server receives the request. This server

runs in the user's local directory and carries the user's authentication regardless of the authentication method used. The interactive server will be shut off when transfers are completed.

The client process can also provide prompt direct access to HPSS upon user request.

Test results

HPSSQ was installed onto ORNL's "Eagle" supercomputer. Test programs were developed to simulate various, including multi-user, running conditions. By means of this test program, test run in different conditions were performed. Test runs with one daemon server and two local servers running simultaneously were done successfully. About 8k requests submitted in each run were all handled properly. Long-term runs in daemon mode lasted up to 7 days (6k requests per-day with a peak request rate of 12 requests per second) with no error. Test runs during HPSS down times show that HPSSQ handles the resubmission process successfully.

Conclusion

Test results show satisfactory performance and good reliability of this software facility. It unburdens users and saves computer resources significantly. Long-term test runs prove its reliability. HPSSQ is ready for production.

This work was presented in the SC2002 poster session [2]. The following comment was made by the referee: "HPSSQ is very interesting and can be used by attendees".

Attachment: HPSSQ User Interface

HPSSQ can run either interactively or in batch jobs. For users' convenience, hpssq command arguments are the same as HSI's wherever it is possible.

Examples of interactive mode with local Kerberos authentication:

1. Online help (commands and simple examples)

➤ *hpssq -help*

2. To submit requests to server queue and to retry in case of failure

To put myFile into myDir of ORNL production HPSS and to use default log file [request record remains after completion]:

➤ *hpssq -q "cd myDir; put myFile" [-keepList]*

To get myStoredFile from myDir of ORNL production HPSS and to set log file path = myLogPath:

➤ *hpssq -q "cd myDir; get myStoredFile" -O "myLogPath"*

3. Queue control and information querying commands

To query current status of submitted requests and to specify log file for full [or request cliId#] information dumping:

➤ *hpssq -accessq "get status [cliId#]" -O "detailLogPath"*

To remove records of requests submitted with -keepList flag:

➤ *hpssq -accessq "rm cliId#1 cliId#2 cliId#3"*

4. Direct access to HPSS

➤ *hpssq -direct -q "cd myDir; ls"*

Examples of batch job with local Kerberos authentication:

Batch job command file example

```
#!/bin/csh -fxv
```

```
#@ job_type      = serial
```

```
#@ class        = batch
```

```
#@ wall_clock_limit = 0:3:00
```

```
#@ output       = $(host).$(jobid).out
```

```
#@ error        = $(host).$(jobid).out
```

```

#@ queue
#
echo $LOADL_PROCESSOR_LIST
echo $KRB5CCNAME
# write/read a file to/from ORNL production HPSS
hpssq -q "cd $mssd; put $file_1" [-O "logpath"]
echo $status (= returned cliId)
hpssq -q "cd $mssd; get $file_2" [-O "logpath"]
echo $status (= returned cliId)
# Query current status of a submitted request
hpssq - accessq "get status [cliId#]" [-O "logpath"]
echo $status (= current status of request cliId#)
exit

```

Accessing HPSS with authentication in remote Kerberos realms:

1. Online help (commands and simple examples)

➤ *hpssq -remote -help*

2. To submit requests to server queue and to retry in case of failure

To put myFile into myDir of a remote site HPSS and to use default log file
[request record remains after completion]:

➤ *hpssq -remote -s remSiteName -q "cd myDir; put myFile" [-keepList]*

To get myStoredFile from myDir of a remote site HPSS and to set log file
path = myLogPath:

➤ *hpssq -remote -s remSiteName -q "cd myDir; get myStoredFile"
-O "myLogPath"*

3. Queue control and information querying commands

To query status of submitted request and to specify log file for full
[or specified request] information dumping:

➤ *hpssq -remote - accessq "get status [cliId#]" -O "detailLogPath"*

To remove records of requests submitted with -keepList flag:

➤ *hpssq -remote - accessq "rm cliId#1 cliId#2 cliId#3"*

References

1. "HSI: Hierarchical Storage Interface," www.sdsc.edu/storage/hsi .
2. M.L. Chen, R.D. Burris, M.K. Gleicher, D.L. Million, "A Software Facility for Persistent and Asynchronous Access to HPSS," Nov. 16-22, 2002, Baltimore, MD, SC2002.

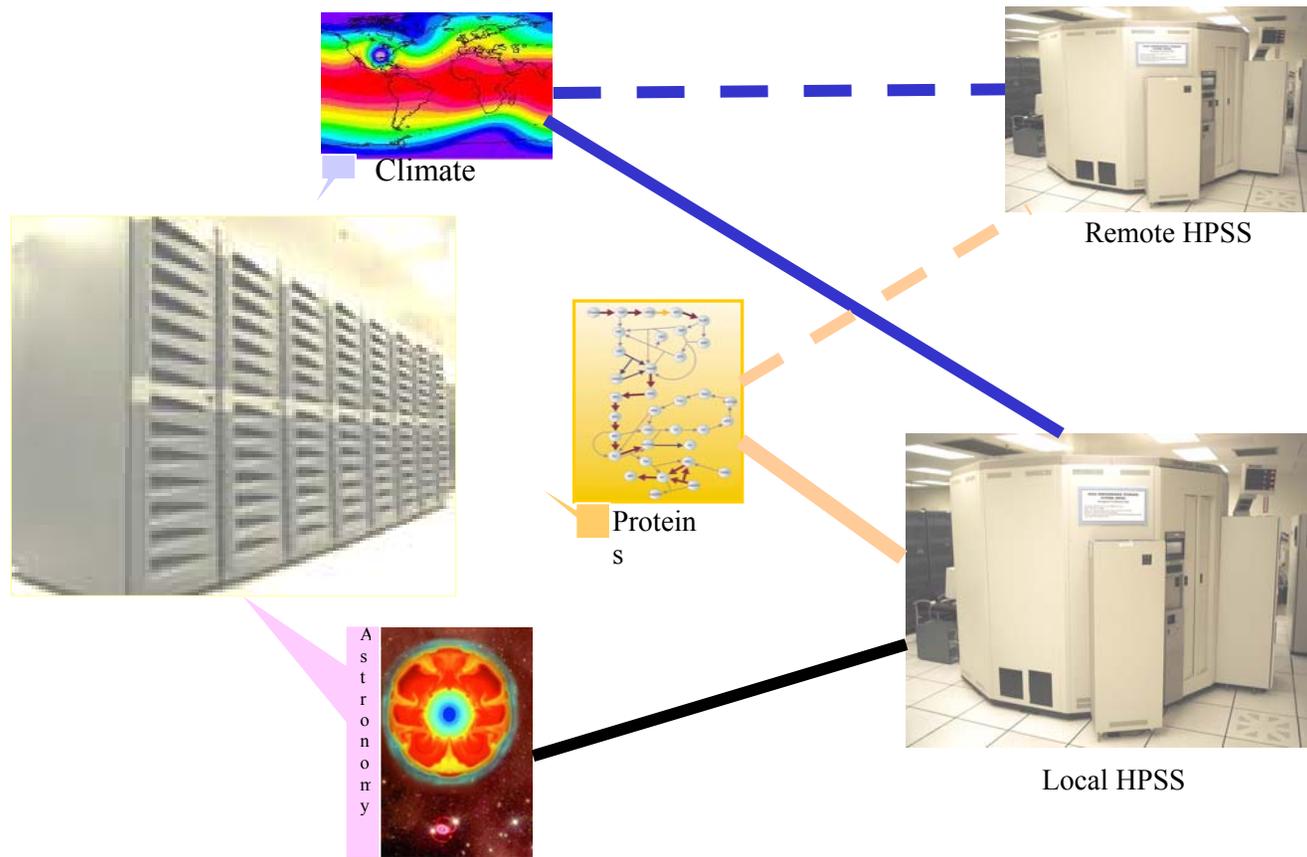


Fig.1 Architecture of HPSSQ-1

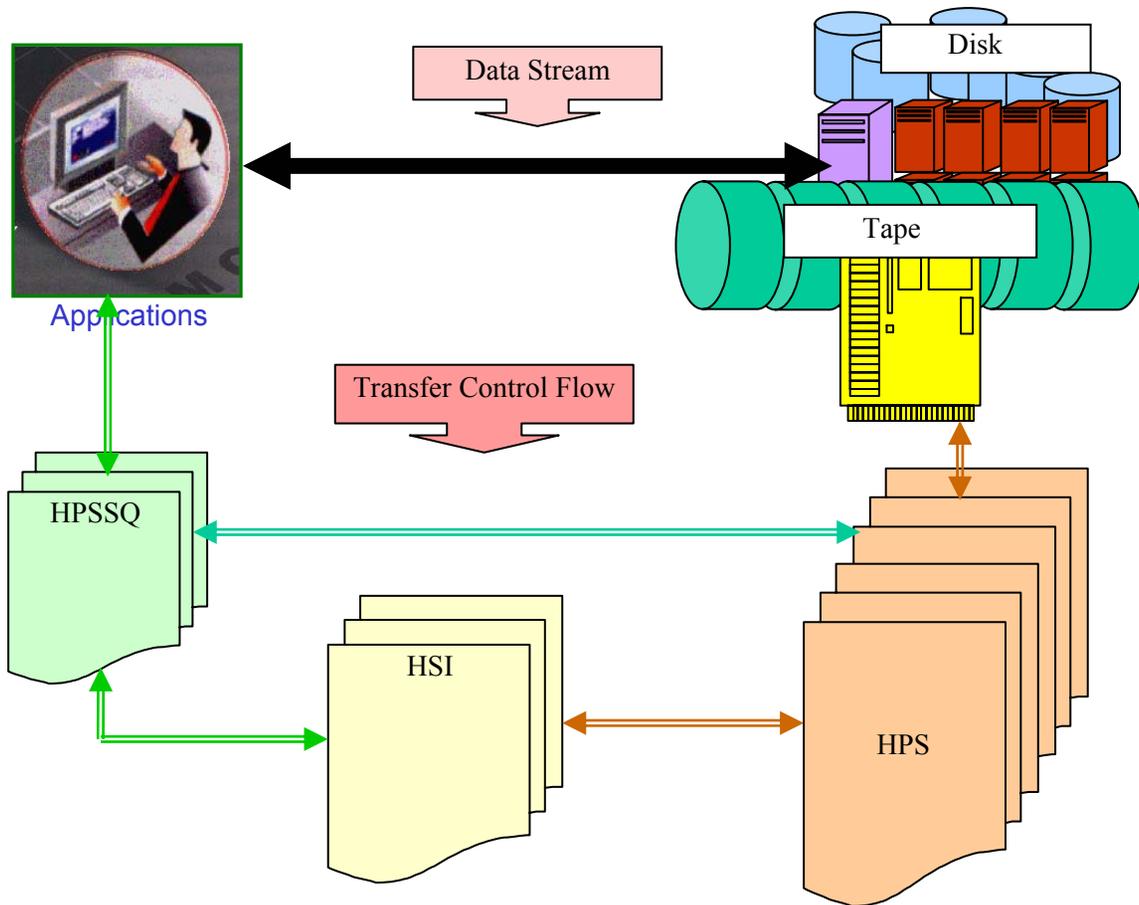


Fig. 2 Architecture of HPSSQ – 2

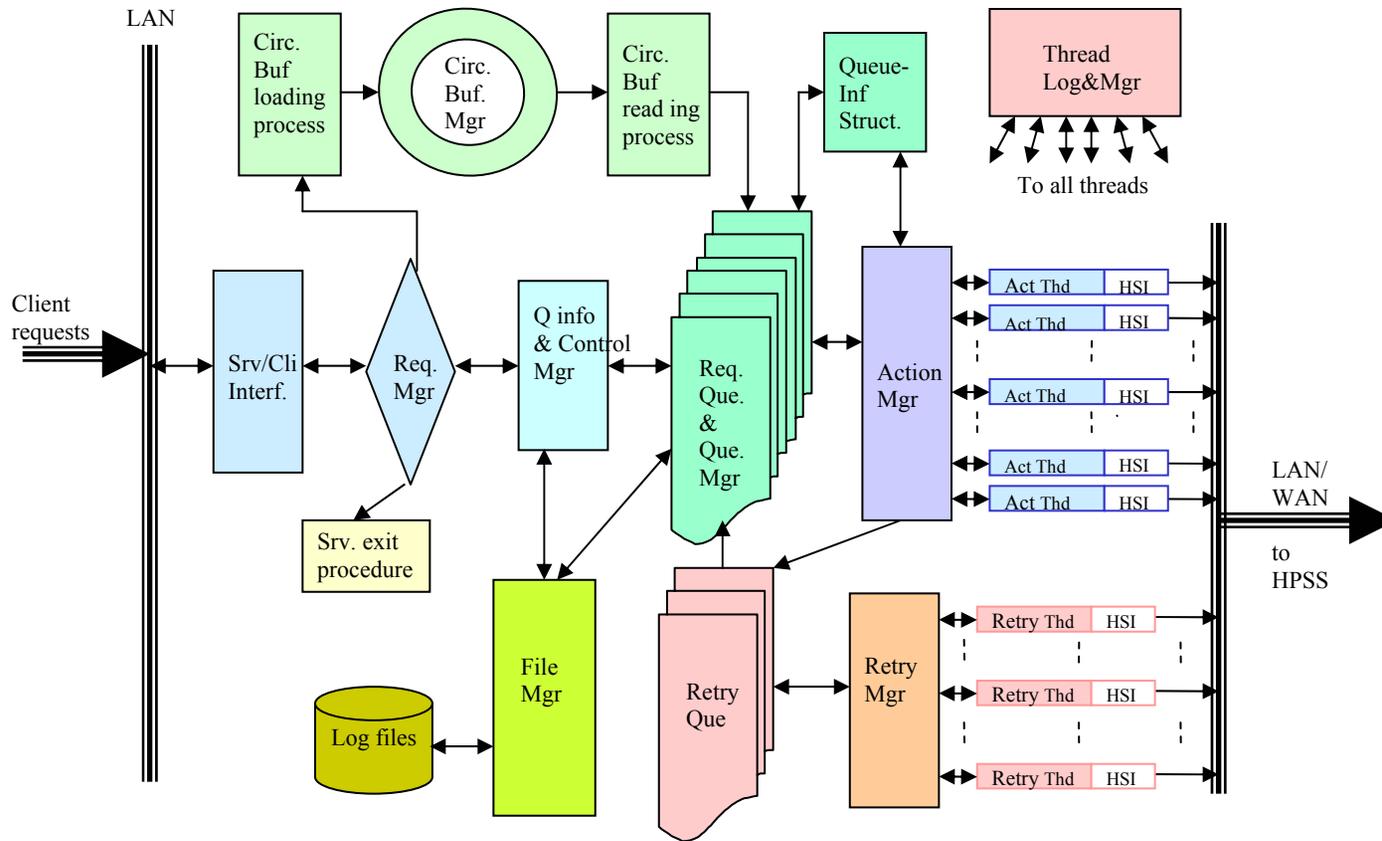


Fig. 3 Server software structure of HPSSQ

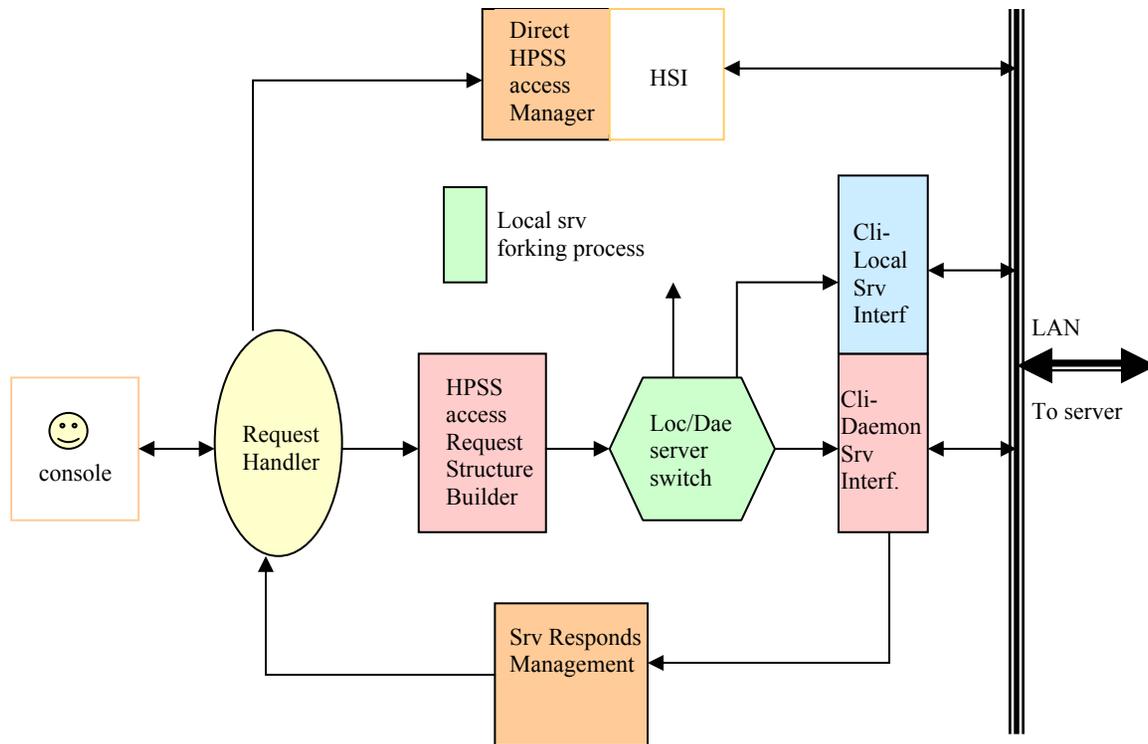


Fig. 4 Client software structure of HPSSQ