JPARSS: A Java Parallel Network Package for Grid Computing[1]

Jie Chen, Walt Akers, Ying Chen and William Watson III
High Performance Computing Group
Thomas Jefferson National Accelerator Facility
12000, Jefferson Ave.
Newport News, Virginia 23606, USA
chen@jlab.org

Abstract

The emergence of high speed wide area networks makes grid computing a reality. However grid applications that need reliable data transfer still have difficulties to achieve optimal TCP performance due to network tuning of TCP window size to improve bandwidth and to reduce latency on a high speed wide area network. This paper presents a Java package called **JPARSS** (Java Parallel Secure Stream (Socket)) that divides data into partitions that are sent over several parallel Java streams simultaneously and allows Java or Web applications to achieve optimal TCP performance in a grid environment without the necessity of tuning TCP window size. This package enables single sign-on, certificate delegation and secure or plaintext data transfer using several security components based on X.509 certificate and SSL. Several experiments will be presented to show that using Java parallel streams is more effective than tuning TCP window size. In addition a simple architecture using Web services to facilitate peer to peer and third party file transfer will be presented.

1 Introduction

With the rapid growth of high performance networking technologies, grid computing especially data-intensive computing has become reality among national laboratories and universities. Example of data-intensive applications include data acquisition, data analysis and simulation in various scientific or engineering fields such as high energy physics, climate modeling and aerospace simulation. These applications usually accumulate a large amount of data at one or more sites. The data then will be shared by a large community of researchers who are distributed among different computing sites. Therefore grid computing applications usually demand high TCP bandwidth in wide area networks. However achieving optimal bandwidth in practice can be difficult due to lack of automatic network tuning [1].To maximize TCP performance, the sender and the receiver of applications should adjust send/receive buffer (window) size no less than the capacity of the TCP pipe. This capacity usually is the product of transmission rate and round trip time (bandwidth*delay product). TCP performance problems arise when the bandwidth*delay product is large, where high performance wide area networks are deployed in grid computing environment, due to a default smaller send/receive buffer size on most operating systems. Therefore it is highly desirable to have a toolkit or a package that lets grid applications achieve optimal TCP bandwidth without tuning TCP window size.

Grid applications not only require efficient transfer of a large amount of data in wide area networks but also demand security and authentication services that enhance data integrity and enable data access control. In the past, user authentication required users to enter a password or pass-phrase to identify themselves to a server. This is inconvenient to users who initiate many short duration connections to the server. A single sign-on mechanism, which lets users authenticate only once to a server (e.g. when starting a grid application) and initiate additional connections to the server during a interval of time without being further prompted for identification, solves this problem. In addition a "third party" file transfer or a similar action requires delegation of user identity from one process to another so that a process can act on behalf of a user.

In recent years, Web technologies, driven by the huge and rapidly growing electronic commerce industry, provided valuable components to construct Web portals [2] allowing users to access grid servers or services through Web browsers. Since Java servlet technology is becoming the overwhelming choice for Web server programming and Java applets are the standard way to deploy sophisticated user interfaces through Web browsers, non Java solutions[3][4] will be difficult to use in Java/Web applications.

There have been a few stand alone applications or libraries [3][4] that meet some, but not all, of the above requirements.This paper focuses on a Java networking toolkit called JPARSS (**J**ava **Par**allel **S**ecure **S**tream (**S**ocket)) that enables grid applications and Java/Web services to handle encrypted or plain data transfer with opti-

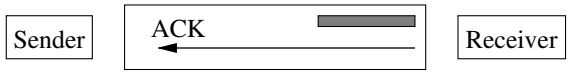
[1]Work Supported by the Department of Energy, Contract DE-AC05-84ER40150.

mal bandwidth, to access data with single sign-on mechanism and to facilitate convenient peer-to-peer and third party data transfer.

2 JPARSS

2.1 TCP Bandwidth Performance

Applications using JPARSS can achieve near optimal utilization of network bandwidth without the necessity of tuning TCP window size. The idea is to divide data into partitions that are sent over a network through several Java socket streams in parallel. This method sometimes is called network striping. Since the default socket buffer size is usually much smaller than the bandwidth*delay product in a grid environment, applications using a single socket will have sub-par performance in TCP bandwidth without tuning the buffer size. However, applications using multiple streams/sockets can overcome the limitations impacting single socket implementation.



(a) Single socket stream with default buffer size.

(b) Single socket stream with buffer size = capacity.

(c) Multiple socket streams with default buffer size.

Figure 1: Single or multiple TCP socket streams with different buffer sizes.

JPARSS opens multiple socket streams between the sender and the receiver for an application. Data needed by the application are then partitioned into segments. The number of segments is equal to the number of streams. The segments are sent through streams in parallel by different threads and are then reassembled by the receiver.The final data are presented to the application as if they were transmitted through a single socket. Figure 1 illustrates why sending partitioned data through multiple socket streams can achieve near optimal bandwidth. In the figure the shaded areas represent socket buffer size and the large empty rectangles represent TCP pipe capacity (bandwidth*delay).

Several experiments have been conducted between Jefferson Lab in Newport News (Virginia) and Florida International University in Miami (Florida). Both sites are connected to the Internet via OC3 connection. The link is not only limited by the 100 Mb/s Fast-Ethernet connection on host machines (PIII 700MHz), but also by numerous intermediate hops. Experimental host machines are running RedHat Linux 6.2 with kernel 2.2.16 and using the Java Virtual machine version 1.2. Figure 2 and figure 3 summarize the results of TCP bandwidth performance for JPARSS under various conditions along with the results from a well known Iperf [5] that is a C utility using one socket stream.

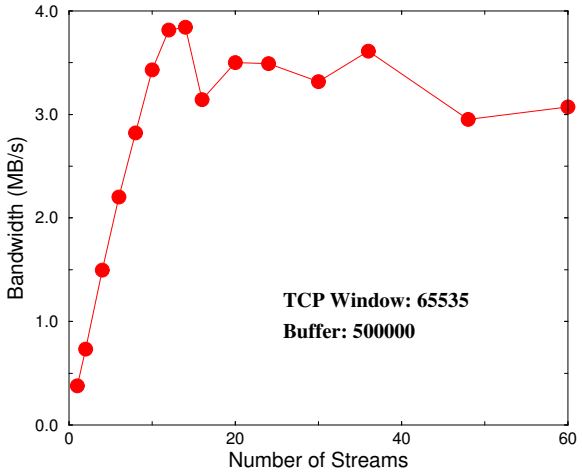


Figure 2: TCP bandwidth for different parallel streams.

Figure 2 shows that TCP bandwidth increases as the number of socket streams increases from 1 to 16 with a fixed TCP window size and a fixed sending buffer size. The maximum bandwidth is achieved when the number of streams is around 16. The TCP bandwidth starts dropping when the number of streams is above 20 because of overheads introduced with extra threads for parallel streams.

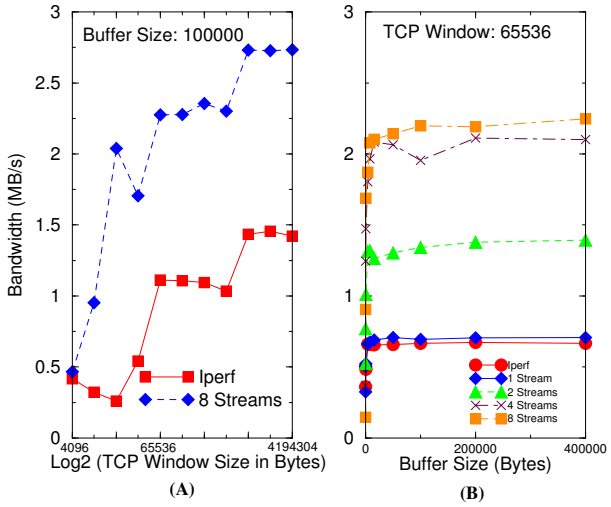


Figure 3: TCP bandwidth for different TCP window sizes and different data sizes.

Figure 3 (A) highlights the improvements of TCP bandwidth using JPARSS over a traditional single socket stream C implementation with varying TCP window size and a fixed buffer size. The Java implementation using 8 parallel streams consistently outperforms Iperf for all

TCP window sizes ranging from 4KB to 4 MB. Anticipated equality in bandwidth between JPARSS and Iperf does not materialize when TCP window size is beyond bandwidth*delay product of 2.15 MB for the connection. This may be caused by driver of network interface cards or the network itself [4].

Figure 3 (B) illustrates the tremendous effect on bandwidth using multiple socket streams with varying sending buffer size and a fixed TCP window size. TCP bandwidth using parallel streams increases when number of parallel streams increases from 1 to 8 across the range of data transfer buffer sizes. The JPARSS test using a single stream actually performs as well as the test of Iperf. There are no obvious benefits to throughput when transfer buffer size exceeds 16 KB.

2.2 Security Components

2.2.1 Proxy Certificate

A certificate is an electronic document used to identify an individual or an application and to associate that identity with a public key. The content of a certificate supported by SSL [6] and many other softwares is organized according to the X.509 v3 certificate specification, which has been recommended by the International Telecommunications Union (ITU), an international standards body, since 1988. A X.509 certificate issued by a certificate authority (CA) binds a particular public key to a distinguished name (DN) the certificate identifies. Only the public key certified by the certificate will work with the corresponding private key possessed by the entity identified by the certificate. To authenticate a user or an application, a related application digitally signs a randomly generated piece of data using the private key and sends both the certificate and the signed data across the network. The server uses techniques of public-key cryptography to validate the signature and confirm the validity of the certificate. The DN of a user then can be mapped to a local identity (e.g. Unix hosts have a file containing DN and username pair).

The safety of a user private key that corresponds to the user's X.509 certificate is critical to the security of grid computing. If another person were to gain access of the private key, the second person would be able to impersonate the owner at will until the certificate is revoked by the CA. To limit this potential danger, a private key is usually stored as a file encrypted by a password or a pass-phrase.

In a grid environment, users may need to authenticate themselves multiple times in a relatively short period of time. Each authentication requires a user to type pass-phrase to decrypt the private key. Multiple pass-phrase inputs not only are inconvenient for users but also are opportunities of compromising security. JPARSS solves this problem using proxy certificate similar to the approach of the globus [7] . A proxy certificate is a short-lived certificate that is created by a user and signed by the user's regular long-lived certificate. The proxy certificate has its own non-encrypted private key stored in a file protected by file permission, so it can be used multiple times without typing a pass-phrase. In short, a proxy certificate is a short-lived binding of the user's DN to a different pair of private and public key. At runtime a JPARSS application first checks whether a proxy X.509 exists at an well known location on a host when the application starts. If the certificate exists and is valid, it will be used to authenticate the application to a potential server. Otherwise a new X.509 proxy certificate , that will be valid for a short period of time (usually 24 hours) and has a non-encrypted private key, will be generated, signed by the user's permanent certificate and stored in the well known location. Once a JPARSS server receives the X.509 proxy certificate, it will verify it and map the certificate onto a local user on the host. This scheme allows single sign-on, protects user credentials because of the short lived certificate, enables interoperability with local security solutions due to the mapping from a certificate to a local user, and provides interoperability with secure Web servers because of the standard of X.509.

2.2.2 Delegation of Proxy Certificate

In a grid environment, various processes need to perform secure operations on a user's behalf and therefore the user must delegate his/her identity. Since a proxy is a short-lived identifier for a user, a user can delegate his/her proxy certificate to a remote process which in turn may delegate the proxy certificate to a different process. JPARSS uses simple protocols to delegate proxy certificates to web servers or JPARSS daemons over secure SSL connections.

2.2.3 Secure and Plain Text Data Transfer

A JPARSS application connects to a JPARSS server using a proxy certificate and establishes a secure single socket stream via SSL. All subsequent parallel connections can be either secure streams or regular socket streams depending on client requests. The parallel secure streams can be used to transfer sensitive data and the regular parallel socket streams can be used to achieve optimal bandwidth.

2.3 Java Implementation

Applications using JPARSS need only deal with four socket classes: *PClientSocket*, *PSocket*, *PClientSSLSocket* and *PServerSocket*. The first two are derived from *Java.net.Socket*, the third one is derived from the first one and the final one is derived from *Java.net.ServerSocket*. The internal parallel streams can be obtained from *getInputStream* or *getOutputStream* of the derived socket classes. All I/O operations are handled asynchronously via threads to improve I/O efficiency. Client connections are established without callbacks from servers to the clients to avoid problems introduced when the clients are behind firewalls.

2.4 Applications

Several file transfer applications using JPARSS are implemented and deployed in the Jefferson Lab/MIT Lattice QCD grid environment. The achieved bandwidth of these Java parallel file transfer utilities is 3 to 4 times better than that of regular ftp. These applications include a file transfer server and a simple command line file transfer utility that imitates *scp* syntax but without requiring the user to enter a password every time he/she transfers files. The file server also accepts a delegated proxy certificate from network to enable third party file transfers.

2.5 Web Services

A simple Web service is implemented to enable peer-to-peer and third party transfer via Web browsers. Users can initiate, monitor and control the transfers all through Web browsers. A simple modification to Apache mod-ssl enables single sign-on using proxy certificates to a Web service which delegates the proxy certificates to a file transfer daemon. There is a communication channel between the daemon and the Web service so that the file transfer can be monitored and controlled. Figure 4 illustrates the architecture of third party file transfer through Web services.

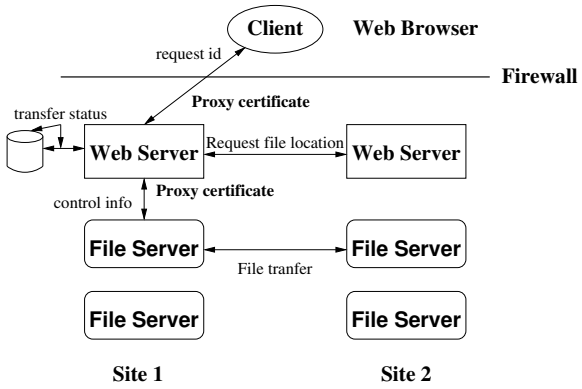


Figure 4: Architecture of third party transfer using Web services.

2.6 Comparisons to other works

JPARSS has similarity to some components in the globus toolkit. For example, both support efficient, reliable data transfer, both support proxy certificate and certificate delegation, both could handle file transfer through Web services (portals). But they differ from each other in several aspects. First of all globus file transfer server is implemented in C and its Java client does not support parallel socket streams. In contrast the JPARSS file server and the client APIs are implemented in Java and support parallel socket streams. Secondly globus only uses SSL style handshake to authenticate users and uses regular socket to transfer data in plain text form. But JPARSS offers applications choices to transfer data either in encrypted or in plain text from. Finally globus manages proxy certificate in a central server which is contacted by each grid portal to retrieve a stored proxy certificate. A user will be prompted to enter a password when he/she wishes to access a different portal. In contrast JPARSS delegates a user's proxy certificate to portals directly so that the owner of the proxy certificate will be allowed to access all portals without entering a password again.

3 Conclusions

This paper presents a pure Java network package called JPARSS. It helps grid applications to achieve near optimal TCP bandwidth without tuning TCP window size, provides simple Java classes as building blocks for rapid development using applets or servlets, offers a single sign-on mechanism, facilitates certificate delegation, and reduces the complexity of development for secure connection between communication peers. Finally the source code and classes can be obtained through ftp://ftp.jlab.org/pub/cdev/jparss.tar.gz

References

[1] Jeffery Semke, Jamshid Mahdavi and Mattew Mathis. Automatic TCP Buffer Tunning. *ACM SIGCOMM'98/Computer Communication Review*, Vol. 28, No.4, Oct. 1998.

[2] G. von Laszewski, I. Foster. Grid Infrastructure to Support Science Portals for Large Scale Instruments. In *Proceedings of the Workshop Distributed Computing on the Web*. 1999.

[3] Ann Chervenak, Bill Alcock, Joe Bester, John Bresnahan, Ian Foster, Carl Kesselman, Sam Meder, Veronika Nefedova, Darcy Quesnel, Steven Tuecke. Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. In *Proceedings of 18th IEEE Symposium on Mass Storage Systems*. San Diego, California. April, 2001.

[4] H. Sivakumar, S. Bailey and R. L. Grossman. PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks. In *Proceedings of SuperComputing 2000*. Dallas, Texas. Nov. 2000.

[5] Ajay Tirumala, Jim Ferguson. http://dast.nlanr.net/Projects/Iperf/index.html, May 2001.

[6] K. Hickman and T. Elgamal. The SSL protocol. Internet draft, Netscape Communication Corp., June 1995. Version 3.0.

[7] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke. A Security Architecture for Computational Grids. In *Proceedings 5th ACM Conference on Computer and Communications Security Conference*, pg. 83-92, 1998.