Topaz: Extending Firefox to Accommodate the GridFTP Protocol

Richard Zamudio[1], Daniel Catarino[1], Michela Taufer[1],
Brent Stearn[2], Karan Bhatia[2]

[1]University of Texas at El Paso
Dept. of Computer Science
El Paso, TX 79968 USA
{rzamudio, dcatarino1, mtaufer}@utep.edu

[2]San Diego Supercomputer Center
University of California San Diego
La Jolla, CA 92093 USA
{flujul, karan}@sdsc.edu

Abstract

As grid infrastructures mature, an increasing challenge is to provide end-user scientists with intuitive interfaces to computational services, data management capabilities, and visualization tools. One novel approach, being successfully applied in the domain of Computational Chemistry, is to leverage the capabilities of the Mozilla framework to provide rich end-user tools that seamlessly integrate with remote resources such as web/grid services and data repositories. The Mozilla framework provides much of the infrastructure to build rich end-user applications, but lacks the capability to integrate with Grid protocols and APIs.

In this paper we present the design and evaluation of Topaz, a Mozilla-based component that provides GridFTP functionality to the popular Firefox browser. Topaz provides end-user scientists with a familiar and user-friendly interface with which to access arbitrary GridFTP servers by providing upload and download functionalities as well as by obtaining and managing users' grid certificates.

1. Introduction

As grid infrastructures mature, an increasing challenge is to provide end-user scientists with intuitive interfaces to access computational services, data management capabilities, and visualization tools [9]. While grid infrastructure tools are typically designed and deployed on server-class systems running a version of Unix, end-users typically have less powerful desktop or laptop computers running a desktop OS for their day-to-day work. Therefore, end-users cannot easily access the server resources because their desktops and laptops are not integrated into the grid. Furthermore, users lack a unified environment with which to access the wide range of services. Some services are accessed by web portals, others by commandline tools, still others with custom tools with little commonality. The grid user community has been driving the development of user-friendly tools that resemble familiar environments such as browsers, hence the interest in web portal frameworks [15]. While these web portals do provide a browser-based access mechanism for a variety of grid services, they do not sufficiently integrate the capability of the end-users' desktop or laptop machine. So, for example, a web portal may provide an interface for GridFTP, but the GridFTP protocol is used only between the web server and the GridFTP server: so called "last-mile" communication to the end user uses standard web-based uploading. In contrast to web-portals, our previous work with Gemstone [2, 13] suggests and supports our general idea presented in [6] of extending the Mozilla core framework with grid computing capabilities in order to provide scientists with a truly integrated environment, from desktop or laptop to the backend grid services and applications. The Mozilla framework, being developed as an open-source project by the Mozilla Foundation, provides much of the necessary infrastructure to build rich end-user applications across a very wide range of platforms. The Firefox browser, Thunderbird email client, and Sunbird desktop calendar applications are just three examples of applications built on top of the Mozilla framework. Key to its success is its ability to develop additional capabilities through new XPCOM components written in C/C++ or Java, JavaScript, Perl, Python, and Ruby, through an XPCOM language binding. Using this, grid computing APIs can be easily integrated into the framework in order to build rich desktop applications that integrate with back-end grid services. In this paper we present the design and evaluation of one component of our integrated environment that we call Topaz. Topaz is a web browser extension that allows end-users to access GridFTP servers directly from their desktop machine.

1-4244-0910-1/07/$20.00 ©2007 IEEE.

The extension works with the Mozilla Firefox browser and can potentially support all platforms that Firefox supports (our current version supports Linux and Mac OS X environments). With the Topaz extension, the Firefox browser directly supports the GridFTP protocol allowing the end-user to specify a gsiftp URL (i.e., ”gsiftp://server/file”) and provides file upload and download capabilities. In addition, Topaz supports GSI-based user authentication.

The rest of this paper is organized as follows: Section 2 gives an overview of our approach and discusses our goals of implementing a general, integrated environment based on Mozilla for accessing a wide range of application, grid, resource, and security services. Section 3 presents the design and implementation of one component of this environment: Topaz, a Firefox protocol extension for GridFTP. Section 4 looks at the overhead due to the integration of Topaz into Mozilla. Section 5 discusses critical aspects related to Topaz and introduces future work. Section 6 presents some important related work and Section 7 concludes this paper.

2. An Environment to Access Services

2.1. Overview of the unified environment

Three of the current challenges with respect to end-user accessibility to grid computing resources are: (1) the need for cross-platform end-user tools; (2) the lack of integration between the end-users' day-to-day computing environment and the grid fabric; and (3) the extreme heterogeneity of end-user tools for accessing different types of services and APIs.

Regarding the first challenge, today's end-users run a variety of laptop and desktop operating systems, including various versions of Microsoft Windows, Mac OS X, and Linux. Developing tools that not only execute on all these platforms, but also provide a simple installation procedure and automatic updates without interfering with other applications or requiring multiple dependent libraries or tools is a significant software engineering challenge. Regarding the second challenge, grid computing was originally designed for server-class hardware running server operating systems and while many approaches have been taken to include desktop systems [7, 1, 4], they all have their shortcomings. Nevertheless, the fact remains that desktops and laptops are users' primary computers, providing the most effective capabilities for visualization and data management. Regarding the third challenge, as defined by Foster et al. in [9], the grid architecture is a layered collection of services accessed through the proper APIs and SDKs. Foster et al. define four general classes of services: application, grid, resource, and security services. Different services define their own particular protocols or APIs with which users can interact. For example, users employ the UberFTP command-line tool and the GridFTP protocol to communicate with a GridFTP data service in the resource class; and security services are accessed using GSI-based tools such as GAMA [5] and MyProxy [3].

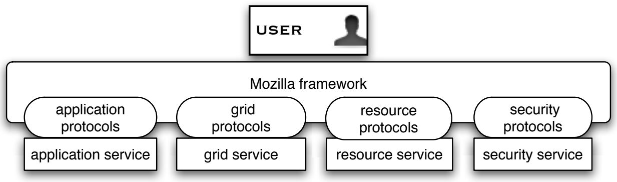

Figure 1. Our proposed environment that integrates service protocols into the Mozilla framework.

These three challenges are the key motivation of our effort towards building the rich, user-friendly environment shown in Figure 1. In our approach, the Mozilla framework serves as the unified environment for end-users to interact with the many different APIs and protocols. The advantages of this approach include: (1) Cross platform support, (2) Familiar user abstractions, (3) High level of interactivity, and (4) Integration of local desktop or laptop resources.

Currently our environment includes Gemstone [11] (a component that provides an end-user with the capability to access grid application services in the domain of computational chemistry including access to computational clusters and the Protein Databank public datasets) and Topaz (the GridFTP extension presented in this paper that allows users to access GridFTP servers) [6]. Both Topaz and Gemstone include support to access standard grid security services.

The choice of Mozilla rather than other tools such as .Net, web portals, or java applications is based on the following analysis: In contrast to *.Net*, the Mozilla framework is supported on a large number of platforms and provides a cross-platform mechanism for building user interfaces as well as packaging and deploying of applications. Mozilla indeed leverages Javascript, XUL, and XPCOM interfaces to implement cross platform support. This is related to our first challenge of having an a cross-platform environment. In contrast to *web portals* that run on the server, the Mozilla core framework provides end-users with a common desktop or laptop environment truly integrated into the grid. This is related to our second challenge of having end-users' day-to-day computing environments integrated into the grid architecture. In contrast to *Java applications*, Mozilla provides us with a single tool that accesses different types of relevant application, grid, resources, and security protocols rather than different tools for different protocols. This is related to our third challenge of having a homogenous environment at the end-user level.

2.2. The Mozilla Framework

The Mozilla project (*http://www.mozilla.org/*) was created when Netscape made its Communicator product open-source. The Mozilla Foundation continues its development and provides a suite of open-source applications including the Firefox browser, Thunderbird email client, and Sunbird calendar, all of which are built on a common core called Mozilla. From the start, the Mozilla core framework was designed to work on all modern platforms and to be highly modular in its architecture. Two key technologies used in the framework are XPCOM and XUL. XPCOM (*http://developer.mozilla.org/en/docs/XPCOM/*) is similar to CORBA and provides most of the functionality in Mozilla. Components can be written in C/C++, Python, and Javascript and are grouped into libraries that handle everything from filesystem manipulation, to security, XSLT, and rendering. XPCOM components are all cross-platform and new components can be added with a minimum of effort. XUL (*http://www.xulplanet.com/*) is used to create GUIs in Mozilla. XUL is HTML-like in its simplicity yet Java Swing-like in its power; it can be combined seamlessly with Cascading Style Sheets, Scalable Vector Graphics, Java applets and can access virtually any XPCOM component via a thin layer of Javascript. The ease of XUL and the robust, cross-platform nature of XPCOM combine to make Mozilla an ideal framework for rapid application development. Indeed, recent years have seen a proliferation of third-party applications like ActiveState's Komodo IDE that have been built using the Mozilla framework.

2.3. The GridFTP Protocol

The Globus Toolkit has evolved into the standard for building grid systems, i.e., distributed systems that span multiple organizations incorporating heterogeneous resources while providing common security, job scheduling, data management facilities, information services, and a common runtime environment. It is the basis for a number of large-scale government sponsored scientific projects including TeraGrid, Biological Informatics Research Network (BIRN), and many others. GridFTP is one major component of the Globus Toolkit and provides file transfer capabilities from one grid node to another. It forms the basis for higher-level services such as the Globus Reliable File Transfer service (RFT) and the Data Replication Service (DRS).

GridFTP is a protocol defined by Global Grid Forum Recommendation GFD.020, RFC 959, RFC 2228, RFC 2389, and a draft before the IETF FTP working group. The GridFTP protocol provides for the secure, robust, fast, and efficient transfer of (bulk) data. GridFTP, similar to the File Transfer Protocol (FTP), provides a client/server implementation for uploading and downloading files. The GridFTP protocol supports authentication using the Grid Security Infrastructure (GSI), authorization using gridmap files as well as SAML-based tools such as the CAS, and provides high performance through support for data striping across multiple server instances and support for parallel streams for data transfer. The Globus Toolkit provides a server implementation along with a few client programs for accessing the server. The client programs include a command-line tool: "globus-url-copy url1 url2" that takes a gsiftp url1 of the form "gsiftp://server/path/to/file" and copies it to the server addressed by url2. The Globus toolkit also includes UberFTP (*http://dims.ncsa.uiuc.edu/set/uberftp/*), an interactive shell application that allows users to authenticate and upload/download files interactively. Both GridFTP clients, globus-url-copy, and UberFTP, require the Globus toolkit to be installed and hence are primarily server-based tools that can not directly be used by most end-users. Instead, many grid projects provide web-based portals through which end-users can access GridFTP repositories from their desktops or laptops. The principle advantage of such an approach is that users need only a simple web browser installed on their local machine to access the file servers. However in this case the GridFTP protocol is used only between the portal server and the GridFTP server. The connection from the portal to the user's local machine uses HTTP and hence few of the advantages of GridFTP (restart, parallel file transfers, striping, etc.) are available.

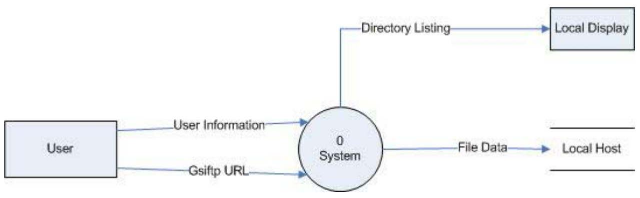

Figure 2. Level 0 DFD for download

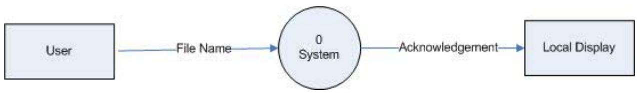

Figure 3. Level 0 DFD for upload

3. Topaz Design and Implementation

3.1. Data Flow Diagrams

In our engineering effort, rigorous software engineering tools such as Data Flow Diagrams (DFDs) are used to design, implement, and analyse the software components of Topaz. A Data Flow Diagram (DFD) is a graphical tool in software engineering used for modeling information-processing systems. Typically DFDs are built using a combination of four components: processes, data flows, data stores, and terminators [10, 14]. A process is normally represented with a circle; it represents the transformation of inputs, e.g., packets of information or chunks of data, into outputs. Arrows that go in or come out of a process represent a data flow. A data store is drawn in a DFD using two parallel lines and represents a storage location from which we can retrieve or store data into, such as hard drives. A rectangle is used to represent a terminator, which is used to identify external entities. Traditionally, external entities are those that interact with the system by either providing or receiving data. In addition, each component has a label that describes its scope. Note that DFDs differ from flowcharts because they emphasize the flow of data through the system rather than control and decision-making. A system can be represented at different levels of abstraction by organizing the DFDs in a series of levels: each DFD level captures a different level of the system abstraction. The highest DFD level (usually called Level 0) represents the overall view of the whole system, where the system is normally represented by a single process and the major inputs and outputs are given by external entities. As we increase the DFD levels, we increase the level of detail for representing the system or some of its components.

3.2. Topaz Design

The goal of the Topaz project is to extend Firefox with the following functionalities: (1) Download (i.e., list as regular ftp, get file by clicking on name, drag and drop to desktop); (2) Upload (i.e., put file by selecting from browser menu option, drag and drop to server); and (3) Remote upload-download (i.e., third-party transfer based on stripe transfer).

Currently our protocol extension supports download and upload functionalities. Remote upload-download is work in progress and is not covered in this paper. Figures 2 and 3 provide the highest DFD levels (Level 0) of the two major functionalities: download and upload respectively. The same functionalities are presented at a lower level of the DFDs (Level 1) in Figures 4 and 5. Figure 4 presents the DFD Level 1 of the download functionality in which the end-user provides a gsiftp URL and user information (i.e., username and password) to Firefox (if this information was not already provided in previous downloads). Firefox then forwards the user information and the URL to Topaz. If the user does not have a valid proxy credential, Topaz obtains one from an appropriate certificate authority using the end-user's certificate (how Topaz handles proxy credentials is addressed in Section 3.4). Once a valid proxy has been obtained, Topaz uses the Globus FTP client API to establish a GridFTP control connection to the GridFTP server and authenticates the user. Only once the authentication has been completed successfully, Topaz determines whether the gsiftp URL represents a directory or a file and issues the corresponding request to establish a data connection to download the content. Globus fetches raw data from the GridFTP server and forwards it to Topaz that formats the data appropriately for the request and forwards the formatted data to Firefox. For list requests, Firefox converts the formatted list data to HTML and displays the listing in the browser window (see Section 3.4). For file requests, Firefox displays the raw data in the browser window or saves it to a local file if ”Save Link As ..” is selected from the pop-up menu. In case of failure, Topaz instructs Firefox to display a proper message. Figure 5 presents the DFD Level 1 of the upload functionality in which the end-user selects the file to upload through the Firefox file dialog. The file object and current gsiftp URL from the location bar are passed to Topaz. Topaz establishes a control connection, authenticates the user, and opens a data connection for upload. Topaz also opens the file and sends the data over the data connection to the GridFTP server.

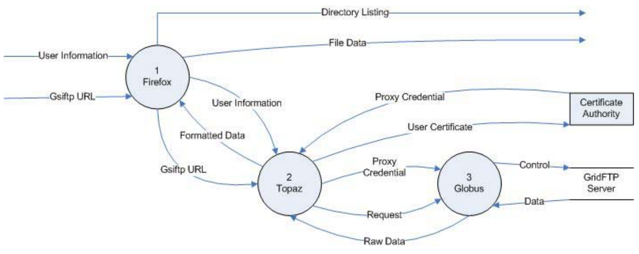

Figure 4. Level 1 DFD for download

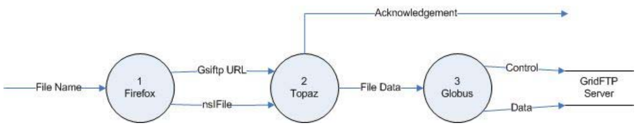

Figure 5. Level 1 DFD for upload

While the version of Topaz presented in this paper is used exclusively through the Firefox browser, it can also be packaged as a standalone commandline tool, a standalone application GUI, or as part of any other Mozilla based application.

3.3. Software Components

The Mozilla framework is modular in nature and open to modifications and additions. Firefox retains this flexibility by enabling extensions, like Topaz, to enhance its core functionality as shown in Figure 4 and Figure 5. Extensions are packaged for distribution into a special zip file called an XPI that contains compiled binaries and/or Javascript, GUI files, and extension metadata. Topaz is installed from the website with a single click via the Firefox Extension Manager that downloads the XPI and configures the browser to make use of the new protocol extension. Topaz comprises five software components: the protocol handler, the login manager, the upload manager, the channel, and the stream. The protocol handler is responsible for processing gsiftp URLs in Firefox and is used only for downloads. The stream handles data transfer from GridFTP servers (download), or to GridFTP servers (upload). For downloads, the channel handles data flow between the browser and the stream; the login manager manages the users' proxy credential. For uploads, the upload manager creates a stream and provides it with the current URL. Figures 6 and 7 show Level 2 of the data flow within the Topaz module for download and upload respectively. For each functionality, a different data flow takes place among the software components. An in-depth description of the data flow for the download and the upload is presented in Section 3.4 and Section 3.5 respectively.

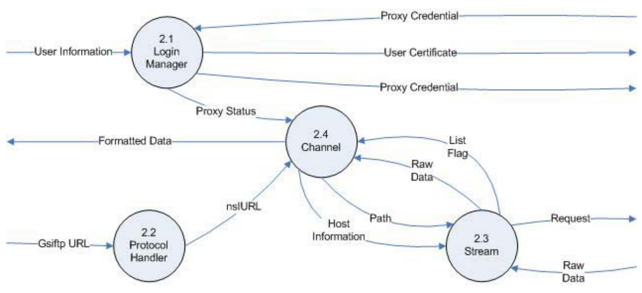

Figure 6. Level 2 DFD for download

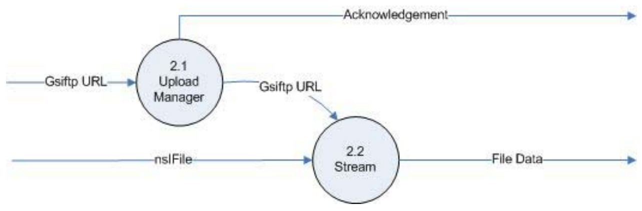

Figure 7. Level 2 DFD for upload

3.4. Download Functionality

The detailed DFD of the download functionality in Topaz is shown in Figure 6. Once Firefox creates the Topaz module, it passes the gsiftp URL to the protocol handler. The protocol handler creates an nsIURL object from the URL string. This is an XPCOM object that contains the URL information and the appropriate methods to parse it. The protocol handler creates a channel for the transfer and passes it the URL object. The channel creates a login manager to verify that the user has a valid proxy credential. If a valid proxy does not exist, the login manager presents the user with a dialog box to enter a username and password and to select a certificate authority in order to obtain a proxy. Possible authentication servers are: GAMA servers [5] and MyProxy servers. Once the proxy is obtained, it is stored in a standard location, i.e., the user's Firefox profile directory. Additionally, if a valid proxy was already available to the user through another authentication method, it can be specified using the Topaz extension preferences from the Extension Manager. After the login manager returns successfully, the channel calls nsIURL methods to parse the GridFTP server name and the path of the file or directory being requested. The channel creates a stream and passes it the host and path information. Like FTP, GridFTP uses separate connections for transmitting control information and data. Contrary to FTP, it provides additional security by allowing these connections to be encrypted. By default, the control connection is encrypted and the data connection is not. The control connection is established by the stream using the Globus function *globus_ftp_control_connect()*. The stream then calls *globus_ftp_control_authenticate()* to authenticate the user on the GridFTP server using the proxy in the user's Firefox profile. At this point, the stream obtains the size of the content being downloaded using the *globus_ftp_client_size()* call. In order to determine whether the content is a file or directory listing, the stream issues the *globus_ftp_client_get()* call to open a data connection and *globus_ftp_register_read()* to get a portion of the content. If this first call returns data then the URL corresponds to a file and the download can continue. This data is passed to the channel and then to Firefox. On the contrary, if the read call does not return any data the URL corresponds to a directory. In this case, the stream calls *globus_ftp_client_list()* in order to begin a directory listing download. The directory data is retrieved using the *globus_ftp_register_read()* function. The data is passed to the channel, along with a flag indicating that this is a directory listing. When this flag is detected, the channel creates an XPCOM nsIStreamConverterService to convert the directory listing into HTTP-index format. This format is designed to provide a machine-readable directory listing. The data is then passed to Firefox that displays it as a web page.

3.5. Upload Functionality

The DFD Level 2 of the upload functionality is shown in Figure 7. In the upload, the check for a valid proxy is done a-priori by doing a directory listing on a GridFTP server using Topaz as described in Section 3.4. A file upload is initiated when a user selects ”Upload File” from the ”File Menu”. This menu item is added to the existing Firefox "File Menu" using a XUL overlay. The Topaz module creates an upload manager to process the request. The upload manager gets the current URL string from the location bar of the current browser window. It then creates a stream and passes it the URL string. At this point, the stream creates an XPCOM nsIFilePicker to prompt the user for the file to upload. If a valid file is given, it is opened for reading. The stream then creates an nsIURL object from the URL string and parses the GridFTP server name and path where to upload the file. The stream uses the *globus_ftp_control_connect()* call to open a control connection to the specified server. The end-user is authenticated on the GridFTP server with the *globus_ftp_control_authenticate()* call using the proxy in the users Firefox profile. If the authenticate request is successful, the stream issues a *globus_ftp_client_put()* call to establish a data connection to the GridFTP server; otherwise a proper error message is generated. The data is read from the file and sent to the GridFTP server using the *globus_ftp_client_register_write()* call.

4. Overhead Evaluation

Since Topaz is built on top of Globus libraries and is integrated into the Firefox browser, we introduce an additional layer of abstraction in the communication. We should expect that any overhead due to this layer does not affect the data exchange between the user and the GridFTP server. To address this requirement, we compared the transfer times in seconds for uploading (i.e., put) and downloading (i.e., get) data between a client at the University of Texas at El Paso and a GridFTP server at the San Diego Supercomputer Center using different transmission tools: Topaz, the globus-url-copy command in Globus 4.0.1, UberFTP 1.13, LFTP version 3.2.1-2, and the Secure Copy Protocol command scp. We used a set of files with different sizes (i.e., 32KB, 256KB, 2MB, 16MB, 128MB, and 1024MB) and we ran the different upload and download experiments three times for each protocol and each file size. The numbers reported in this section are the average values of those measurements. For each tool, we measured the time from when the command is issued by the user until the completion of the transfer. In particular, for Topaz, we measured the time from when the channel is created until the stream is finished with the transfer for downloads (Figure 8) and from when the stream receives the filename until it is finished with the transfer for uploads (Figure 9). For globus-url-copy we measured the time from when the command is issued until its completion. UberFTP, LFTP, and scp give the transfer time after each operation and this is the time reported in this paper. While running our performance measurements, we used default configurations for the several tools and no optimization has been applied: this is because we wanted to emulate the usage conditions for most scientists that normally have neither the needed expertise nor the time to configure these tools with high levels of optimization. Since Topaz deploys GridFTP, it can benefit from the several optimizations available for this protocol, e.g., multi-streaming, buffer size tuning. This paper focuses on the additional overhead that our extension can cause rather than its and other protocols' optimizations. The results of our comparisons for the download and upload transfer times are presented in Figures 8 and 9 respectively. In particular, Figure 8 shows the average transfer time (y-axis, in logarithmic scale) measured for downloading files with different sizes (x-axis) and Figure 9 presents the same measurement, the transfer time, for the upload of files with different sizes. Both of the figures show that although Topaz is characterized by an initial transfer time that is larger than the other tools, in both cases, for the download and the upload, this gap tends to decrease as the size of the file increases and for files larger than 16MB the time becomes the same due to the saturation of the network bandwidth. For small files, less than 2MB, the time gap between globus-url-copy command and Topaz is constant. The difference for the download times is equal to 1.8 seconds. The time gap between upload times is smaller and equal to 0.7 seconds. The reason for the additional overheads in the Topaz functionalities can be found in Figures 4 and 5 that show the additional layer of abstraction of Topaz on top of the Globus libraries. The DFDs in Figures 6 and 6 depict the reason for a larger overhead in the download functionality. Note in these figures how the download functionality has a more complex transfer structure than the upload functionality. Indeed the download requires the generation of a channel by the protocol handler as well as the generation of the login manager and the stream by the channel. On the contrary, for uploads, the upload manager generates only the stream.

5. Discussion and Future Work

There are several advantages in using Firefox to create a general environment to access services such as those provided by a GridFTP client. The browser provides end-users with an easy to use interface that most scientists are already familiar with. Firefox provides a number of features that are available to the protocol such as drag and drop support and dialog boxes for saving files and providing passwords. Since Firefox handles the user interface, it makes it easier and faster to develop a GridFTP client across multiple platforms. Firefox also provides standard means to package and deploy extensions as well as an update notification system to upgrade software when new versions are available. There are also some challenges in integrating Topaz into the Mozilla framework. Mozilla is still largely under development and has a number of unfrozen interfaces that may be

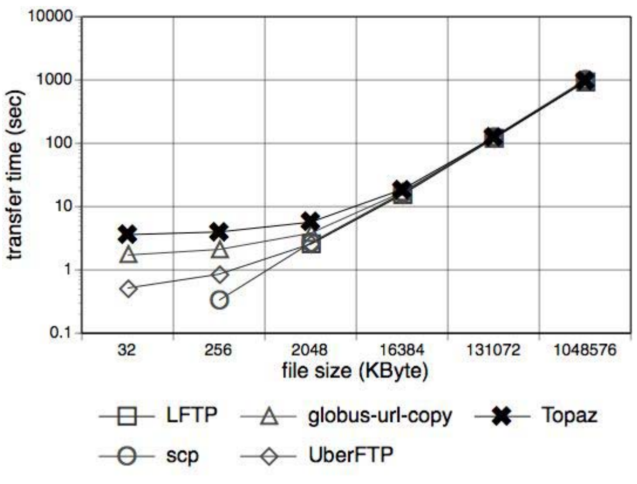

Figure 8. Transfer rate of download (get)

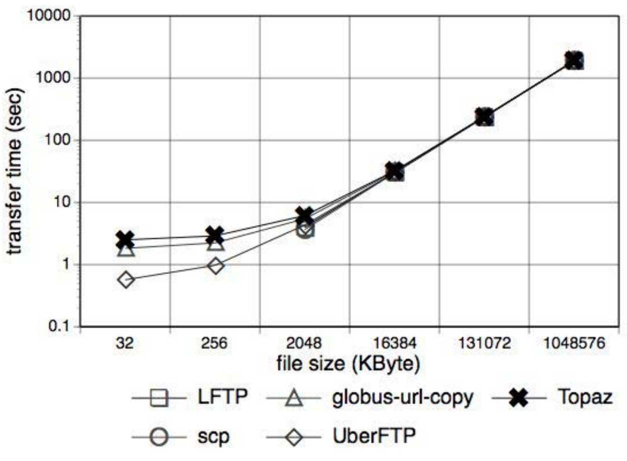

Figure 9. Transfer rate of upload (put)

changed in future releases. Because of this, interface references within Topaz will need to be maintained. Moreover, the dynamic nature of Mozilla can make it challenging to find good documentation.

Topaz is an in-progress, open-source project. In order for end-users to take full advantage of the GridFTP protocol, several additional advanced features of the protocol need to be integrated into Topaz. Among these features are: implementation of third party transfers (i.e., to connect to two GridFTP servers simultaneously and initiate, from the end-user desktop, a direct file transfer between GridFTP servers), parallel data transfers (i.e., to allow multiple TCP streams to be used for data transfers between a GridFTP server and a desktop, as well as between two GridFTP servers), and partial file transfers (i.e., to allow an arbitrary portion of a file to be specified for transfer); a portability extension to support Windows PCs (currently Topaz supports the Mac and Linux platforms); an authentication extension to include a wider range of grid authentication methods (including the MyProxy tool for creating and managing X.509 credentials on the server and client).

6. Related Work

So far, efforts have been focused on single APIs or SDKs tailored to access single application, grid, resource, or security services but little attention has been paid to build integrated environments that allow end-users to access a wide range of services. One approach taken in [8] is to develop compatible grid protocols for desktop operating systems such as Windows so that these desktop resources become first class grid resources. The work in [8] is based on a new implementation of the GridFTP protocol based on the Microsoft .NET framework for Windows. It includes a GridFTP client with an interactive mode. The .NET GridFTP implementation is designed to be compatible with the Globus Toolkit 4 [12] implementation, except that it does not currently support data channel authentication and the striped data transfer implementation does not interoperate with GT4. An alternative approach is to develop lightweight portable tools that will extend the grid onto the desktop fully under the control of the end-user. This latter approach may be advantageous, as it does not require the full grid stack to be developed and maintains control of the machine for the end-user. One general challenge to build an integrated environment is the integration of data management capabilities across the desktop to the server: the data that an end-user employs is typically on his or her own local machine and used in day-to-day work such as email, writing reports, basic analysis using spreadsheets, or other desktop software. Within the grid, this data can be accessed using high-performance and secure grid protocols such as GridFTP [15]. However, from the desktop (which is outside of the formal grid) the end-user must employ different protocols to upload the data to a repository. After the grid calculation, again the grid infrastructure uses one protocol to move the data to a repository and another protocol for downloading it to the desktop.

7. Conclusion

Scientists have a need for user-friendly environments that allow them to access a wider range of application, grid, resource, and security services. To address the scientists' need we have extended the grid infrastructure to the desktop environment through the use of a lightweight, cross platform framework such as the Mozilla framework. In previous work we have built Gemstone on top of the Mozilla framework to provide end-user scientists with access to a set of biomedical applications [6]. In this paper, we present Topaz, a Firefox protocol extension for GridFTP, that provides end-users with a simple and secure access to grid technology. Rigorous software engineering tools such as Data Flow Diagrams guided its design and implementation. Topaz allows end-users to easily upload and download files from GridFTP servers guaranteeing the required security through the use of security infrastructures such as GAMA. The overhead due to the additional layer of abstraction introduced by Topaz becomes insignificant as the size of the transferred files increases, i.e., above 16MB. This makes Topaz a well-suited, user-friendly, and secure tool for large file transfers.

Acknowledgements

This work was supported in part by the National Science Foundation under Grants #SCI-0438430, SCI: NMI Development: The Computational Chemistry Prototyping Environment; #EIA-0080940, MII: Graduate Education for Minority Students in Computer Science and Engineering: Extending the Pipeline; and #SCI-0506429, DAPLDS - a Dynamically Adaptive Protein-Ligand Docking System based on multi-scale modeling.

References

[1] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *Proceedings of GRID'04*, 2004.

[2] K. Baldridge, J. Greenberg, W. Sudholt, K. Bhatia, S. Mock, C. Amoreira, Y. Potier, and M. Taufer. The computational chemistry prototyping environment. *Proceedings of the IEEE Special Issue on Grid Computing*, 93(3):510 – 512, 2005.

[3] J. Basney. Myproxy protocol, 2005.

[4] K. Bhatia. Peer-to-peer requirements on the ogsa framework, 2005.

[5] K. Bhatia, S. Chandra, and K. Mueller. Gama: Grid account management architecture. In *Proceedings of e-Science'05*, 2005.

[6] K. Bhatia, B. Stearn, M. Taufer, R. Zamudio, and D. Catarino. Extending grid protocols onto the desktop using the mozilla framework. In *Proceedings of GCE'06*, 2006.

[7] A. Chien, B. Calder, S. Elbert, and K. Bhatia. Entropia: architecture and performance of an enterprise desktop grid system. *J. of Parallel and Distributed Computing*, 63(5):597–610, 2003.

[8] J. Feng, L. Cui, G. Wasson, and M. Humphrey. Toward seamless grid data access: Design and implementation of gridftp on .net. In *Proceedings of GRID'05*, 2005.

[9] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Int. J. Supercomputer Applications*, 15(3), 2001.

[10] C. Gane and T. Sarson. *Structured System Analysis*. Prentice Hall, 1978.

[11] Gemstone - grid enabled molecular science through online networked environments. http://grid-devel.sdsc.edu/gemstone/.

[12] Globus - an open-source software toolkit used for building grid systems and applications. http://www.globus.org/.

[13] J. P. Greenberg, S. Mock, K. Bhatia, M. Katz, G. Bruno, F. Sacerdoti, P. Papadopoulos, and K. K. Baldridge. Incorporation of middleware and grid technologies to enhance usability in computational chemistry applications. *Future Generation Computer Systems*, 21(1):3–10, 2005.

[14] J. Herard and et al. Validation of communication in safety critical control system. Technical Report 543, Nordtest, 2003.

[15] S. Krishnan, K. K. Baldridge, J. Greenberg, B. Stearn, and K. Bhatia. An end-to-end web services-based infrastructure for biomedical applications. In *Proceedings of GRID'06*, 2005.