INTEGRATING NETWORK STORAGE INTO INFORMATION RETRIEVAL APPLICATIONS

A Thesis
Presented for the
Master of Science Degree
The University of Tennessee, Knoxville

Svetlana Y. Mironova
May 2003

DEDICATION

This work is dedicated to my family, both in America and in Russia: to my husband David and my daughters Natasha and Stacey for being there for me and encouraging me in any way they could; to Babulya who taught me all invaluable lessons of life and how to stick to my goals and never give up; to my Mom for being a role model and a friend; to my sister Olga and step-Dad Basil for believing in me. I love you all.

ACKNOWLEDGEMENTS

I would like to thank all the people who helped me to complete the Master of Science degree in Computer Science. I express my gratitude to Dr. Michael W. Berry for his help and guidance through my work on my thesis project. I thank the LoCI people, especially Scott Atchley and Stephen Soltesz, for their help with the installation of the LoRS tools and for patiently answering numerous questions that I had for them. I would like to thank Ms. Wallace Mayo and Dr. David Straight for their advice and help in becoming a graduate student and Graduate Teaching Assistant in the Department of Computer Science. I would also like to thank Dr. James Plank and Dr. Micah Beck for agreeing to be on my thesis committee. I would like to thank the Department of Computer Science in general for providing me with all the valuable knowledge in the field of Computer Science and making me confident in this knowledge. This research was supported in part by the National Science Foundation under grant CISE-EIA-99-72889.

Most of all I thank my family for putting up with me during the years of school: my husband David for giving me the opportunity to return to school in the first place and taking care of our two daughters while I was busy with endless homework and lab assignments; my two precious daughters Natasha and Stacey for just being there for me, making me laugh every time they inquired, “Are you done with school yet?” and for showing their endless love to me. Without their love and support, my education would not have been possible. I would like to thank my family in Russia for giving me all the encouragement they could in achieving my goals.

ABSTRACT

The object-oriented software environment GTP (General Text Parser) with network storage capability has been designed to provide a scalable solution to index creation and query processing. GTP allows information retrieval and data mining professionals to parse a large collection of documents and create a vector space information retrieval model for subsequent concept-based query processing (GTPQUERY). The software's numerous options allow users to tune the model to their specific needs. Depending on the size of the collection, the facilitation of the model may require an enormous amount of local storage. The addition of network storage capability addresses the problem of inadequate local storage and file sharing over the network. Tools defining the Logistical Networking Testbed developed in the Logistical Computing and Internetworking (LoCI) Lab at the University of Tennessee are used to demonstrate both the creation and use of remotely stored indices. With the development of new network storage technologies, the software will be able to forgo most local file generation and will allow remote users to share the index created by GTP.

TABLE OF CONTENTS

Section **Page**

TABLE OF FIGURES

Figure **Page**

1. INTRODUCTION

The amount of textual-based information stored electronically, whether on our own computers or on the Web, is rapidly accumulating. Any desktop or laptop computer can accommodate huge amounts of data due to advances in hardware storage devices. Software companies develop products that may require megabytes of hard drive space. Without upgrading computers every few years, one cannot download one's favorite music or movies, or play the most recent computer games. Researchers and scientists involved in data mining and information retrieval are facing the same reality – an enormous amount of storage may be needed to run simulations and store their outputs. Some of the programs have to be rerun periodically with updated data. The majority of the data collections the scientists are working with are dynamic – they change with time.

Take the popular search engine Google [10] as an example: thousands of new web pages are created on the Web every day. Google's powerful crawlers have to update the stored data periodically to be able to display new pages and discard *dead* links. Google takes a *snapshot* of each page examined as it crawls the web and caches (i.e., stores) the back-up copy for use when the original page is unavailable. The cached content is the content Google uses to judge whether a page is a relevant match for the query [10]. Google owns the world's largest commercial *Linux cluster,* which consists of more than 10,000 servers that are able to store over 3 billion web documents [10]. However, Google is a business enterprise with considerable funding. Most researchers in information retrieval and data mining do not have an access to such a tremendous amount of storage. Providing an opportunity to store data on a remote network is an attempt to address the needs of novices and experts in information retrieval and modeling who deal with large text corpora on a daily basis but are subject to limited storage capabilities.

The General Text Parser (GTP) [9] software package was chosen to demonstrate the capability of providing an indexer with additional storage on a remote network. GTP is a publicly available software package developed at the Computer Science Department at the University of Tennessee. GTP is capable of parsing a large collection of documents

(text or tag separated) and creating a vector space information retrieval model for subsequent concept-based query processing. GTP provides numerous options to the user so that it is easy to use by both beginning and advanced users.

GTP utilizes Latent Semantic Indexing (LSI) for its underlying information retrieval (IR) model [6,7,8]. LSI is a concept-based retrieval method, which overcomes many of the problems evident in popular word-based retrieval systems. LSI has been shown to be 30% more effective in finding and ranking relevant items than comparable word matching methods [7]. It relies upon matrix factorization methods such as the singular value decomposition (SVD) to uncover the underlying associations between terms and documents in a large text collection. A semantic, or concept, space is constructed from the SVD factors to facilitate query matching.

During execution GTP creates several files that define the vector space IR model. Depending on the size of the text collection, those files can be quite large. The user is given an option of storing the model outputs on some set of the available Internet Backplane Protocol (IBP) servers or depots [2,11]. IBP is the foundation of the Logistical Networking Testbed developed at the Logistical Computing and Internetworking (LoCI) Lab at the University of Tennessee. This infrastructure provides a scalably sharable storage service as a network resource for distributed applications [2]. The Internet Backplane Protocol is middleware for managing and using remote storage. It was invented to support logistical networking in large scale, distributed systems and applications. IBP, as it name suggests, enables applications to treat the Internet as if it were a processor backplane [12], and allows users to share disk space or memory space over the network. Essentially, if the user chooses to store files on IBP, he can allow his colleagues to access these files as well. The storage on IBP is time-limited, i.e., if not extended, the storage will expire with time. The user has a set of capabilities with which he/she can manage allocated space and its time limits. Currently, there are 159 public IBP servers in 16 countries and 26 American states [12]. The total storage space available is approximately 9000 GB, and more space becomes available every day as more

organizations, research facilities and universities worldwide join the list of accessible depots.

2. GENERAL TEXT PARSER

General Text Parser (GTP) is a software environment developed at the University of Tennessee for text/document parsing and indexing using an IR model based on sparse matrix data structures. GTP has the ability to parse any document: raw text, an HTML document or any other tag-separated document collection via user-defined filters. This software written in C++ and Java is very flexible and can be used by novice and expert users to parse textual information with the help of numerous options and settings.

If opted by the user, GTP will create a vector-space model in which documents and queries are represented as vectors in a low-dimensional subspace. A term-by-document matrix is initially used to define the relationships between the documents in the collection and the parsed terms or keywords. The elements of the matrix are typically weighted/unweighted frequencies of terms (rows) with respect to their corresponding documents (columns) [9].

2.1 LATENT SEMANTIC INDEXING

The underlying vector-space model exploited by the GTP is *Latent Semantic Indexing* (LSI). LSI is an efficient IR technique that uses statistically derived conceptual indices rather than individual words to encode documents. LSI assumes some underlying or latent structure in word usage that is partially obscured by variability in word choice. Specifically, LSI uses the truncated singular value decomposition (SVD) of the large sparse term-by-document matrix mentioned above to build a conceptual vector space [7]. A lower-rank approximation to the original term-by-document matrix is used to derive vector encodings for both terms and documents in the same *k*-dimensional subspace. The clustering of term or document vectors in this subspace suggests an underlying (latent) semantic structure in the usage of terms within the documents.

Let t and d denote the number of terms and documents, respectively. SVD factors the original term-by-document matrix A into the product of three matrices:

$$A = U\Sigma V^T , \qquad (1)$$

where U is the $t \times t$ orthogonal matrix having the left singular vectors of A as its columns, V is the $d \times d$ orthogonal matrix having the right singular vectors of A as its columns, and Σ is the $t \times d$ diagonal matrix having the singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{\min(t,d)}$ of A, in order, along its diagonal. This factorization, which exists for any matrix A [8], reflects a breakdown of the original term-to-document relationships into linearly independent vectors or *factor values*. The use of k factors or the k-largest singular triplets is equivalent to approximating the original term-by-document matrix in k-dimensional space [8]. Equation (1) then becomes

$$A_k = U_k \Sigma_k V_k^T , \qquad (2)$$

where A_k is the best low-rank approximation of the original term-by-document matrix [8]. For retrieval purposes, the rows of the $t \times k$ matrix U_k define the term vectors for the LSI model. Linear combinations of these vectors (typically scaled by the k-leading singular values of A) are used to construct query vectors or pseudo-document vectors. Similarly, the rows of the $d \times k$ matrix V_k yield the document vectors for the model (i.e., the coordinates of each document in a k-dimensional subspace). If double precision (or 8 bytes) is used to store each coordinate in any term or document vectors, the storage requirement for the LSI model is $8k(t+d)$. If 300 factors (or singular triplets) were used to encode a collection comprising 100,000 terms and 10,000 documents, the storage requirement would be well over 250 Mbytes.

2.2 EVOLUTION OF GTP

The original version of GTP was developed in C++ for both Solaris and Linux platforms. It was based on C code first distributed by Telcordia Technologies, Inc., for LSI-based applications [9]. A parallel version was later developed in C++/MPI (Message Passing Interface), improving the SVD computational time.

The C++ version was recently ported to Java to utilize more object-oriented features. The Java version has certain limitations compared to its C++ counterpart: it is slower and it does not accept custom filters. However, it does provide an internal HTML filter. Most of the work described in this thesis was performed on the Java version of GTP. Work is under way to optimize the Java version including a graphical user interface to help the user manage the numerous options available with GTP.

2.3 GTP PROCESS

GTP is a robust software environment that allows the user to tune the parser to his/her needs through its multiple options. For example, some options allow the user to change thresholds for document and global term frequencies, specify custom filters and local/global term weighting functions, and indicate new document delimiters. For a detailed overview of the available options, see **Appendix 1**.

During parsing, GTP generates multiple files for further processing of the document collection and for deriving correlations between terms and documents. After the initial parsing of the collection, GTP creates a master index of keys, or terms, in the collection. Those terms are placed in the file *keys* together with the term's index and global weight that is calculated according to the weighting scheme selected by the user. If the SVD option is selected, a binary file called *output*, which contains term and document vectors together with the computed singular values, is generated. These files are essential to the GTP-derived vector space IR model and its utility for query processing. **Table 1** describes the most important files generated by GTP.

Table 1. Important Files Generated by GTP.

File name	Type	Description
keys	DBM	Database of keys generated
output	Binary	SVD output
rawmatrix.Z	ASCII (Compressed)	Raw term-by-document matrix
matrix.hb.Z	ASCII (Compressed)	Term-by-document matrix (sparse format)
lao2	ASCII	Summary of SVD output (singular values)
larv2	Binary	File of right singular vectors (documents)
lalv2	Binary	File of left singular vectors (terms)
LAST_RUN	ASCII	Summary of options used during the most recent GTP run

Depending on the text collection parsed, the size of the above mentioned files could be very large – varying from kilobytes to gigabytes. A significant amount of storage is required to accommodate those files. For example, the *larv2* and *lalv2* files require 8*kd* and 8*kt* bytes, respectively, for *k* terms parsed from *d* documents by GTP. The user must also take into consideration that, in the course of his research, he might need to repeat the process of parsing the collection several times to achieve the desired IR model. Using the Internet Backplane Protocol (IBP) as described in **Section 4**, one can successfully eliminate the storage bottleneck commonly associated with IR model research and development.

2.4 QUERY PROCESS

GTP is not only capable of creating an index. It also provides users with a module for querying, i.e., determining the similarities between a query and all documents in the collection. This query-processing module requires several of the output files generated by GTP, namely *keys, output*, and *LAST_RUN* (see **Table 1** for the description of these files).

The query module (GTPQUERY) treats each query in the same manner that the GTP process treats a document in the collection. That is, the same term-based operations are applied to the query to produce a resulting query vector. Query vectors are constructed as *pseudo*-document vectors, thus allowing their projection into the original term-document vector space. It is achieved by summing the term vectors of the corresponding terms in the query (the term vectors are generated by GTP), and then by scaling each term vector dimension by the inverse of a corresponding singular value. Scaling the query vector is optimally done using the singular values produced by GTP [6,7,9]. A cosine similarity measure between the query vector and document vectors is used to determine the relevance of any or all documents to the query.

The result of the query process consists of files (one per query) with document ID and corresponding cosine similarity pairs ranked from the most relevant to the least relevant. A graphical user interface (GUI) has added the functionality of allowing the user to view the desired document in a separate window [10]. The result of the query process and the GUI's separate document window are illustrated in **Figure 1**. The entire GTP and GTPQUERY processes are summarized in **Figure 2**. GTPQUERY options are described in detail in **Appendix 2**.

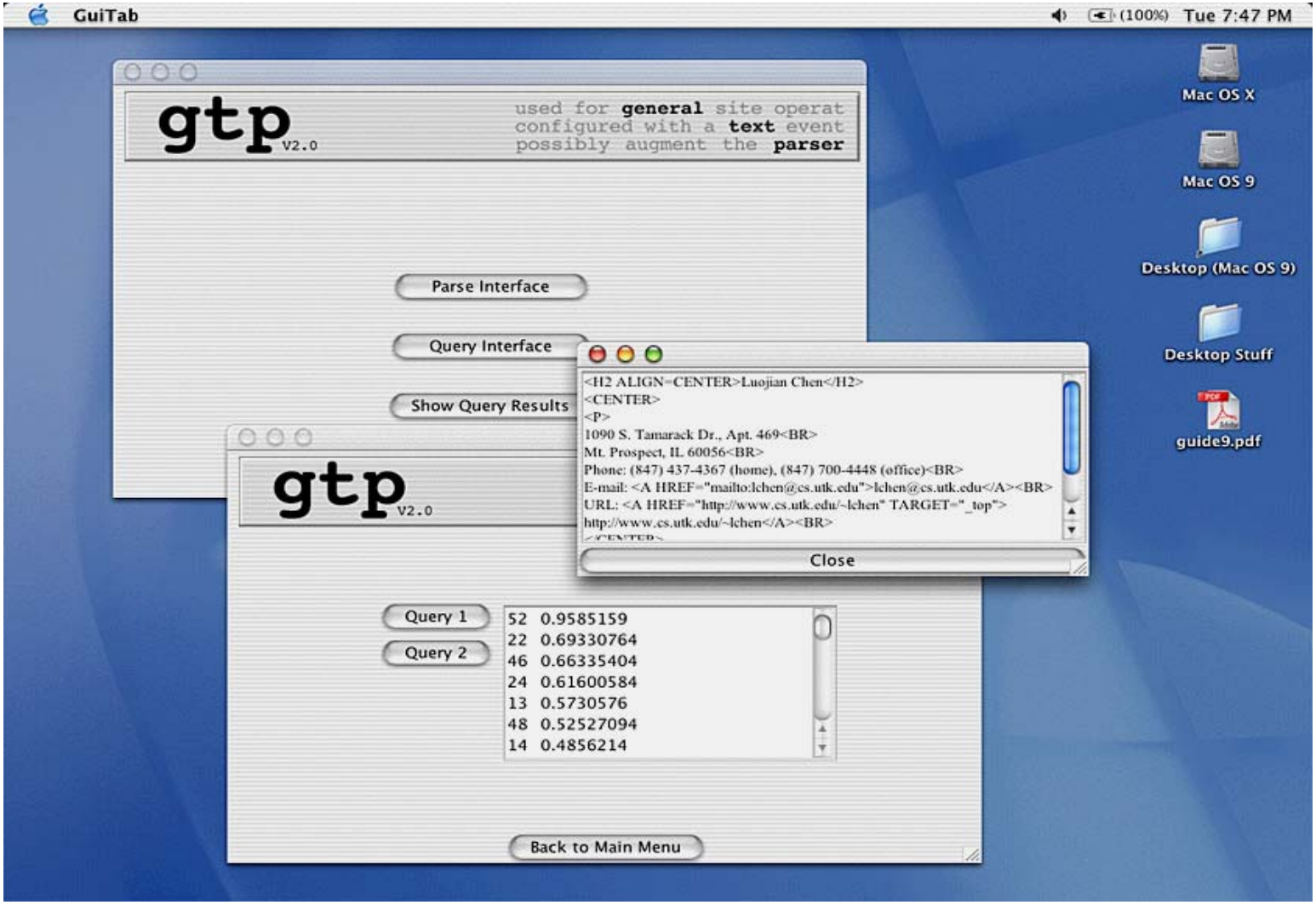

Figure 1. GTP's Graphical User Interface with GTPQUERY Results and Document Window [13].

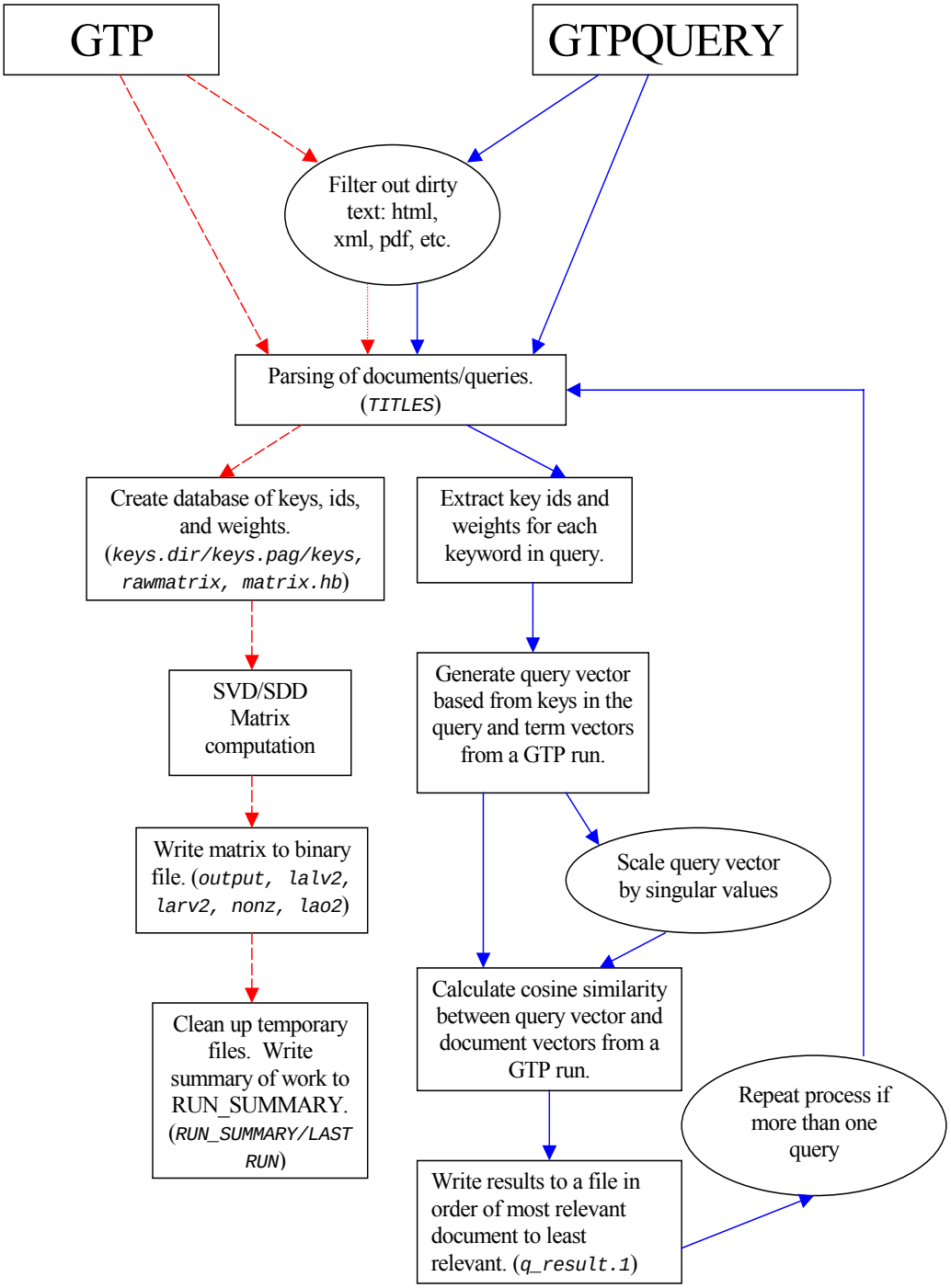

Figure 2. Process of GTP and GTPQUERY [9].

3. NETWORK STORAGE STACK

Exchange of information between producers and consumers of large datasets over a wide-area network presents a logistical challenge. Data that is generated by model simulations can be difficult to obtain since it has to be transferred through such slow services as HTTP and FTP [1]. What is needed is a more flexible framework for moving content to distribution sites, decentralized load-balancing to ensure use of all available resources while maintaining scalability, the ability to quickly add more replicas as demand requires, and improvement of throughput to end users [1]. To allow users to store data in the network and access it quickly and easily, the Logistical Computing and Internetworking Lab (LoCI) at the University of Tennessee [12] has developed the Network Storage Stack. The Network Storage Stack is modeled after the Internet Protocol (IP) Stack, and is designed to add storage resources to the Internet in a sharable and scalable manner [2]. **Figure 3** shows the organization of the Network Storage Stack.

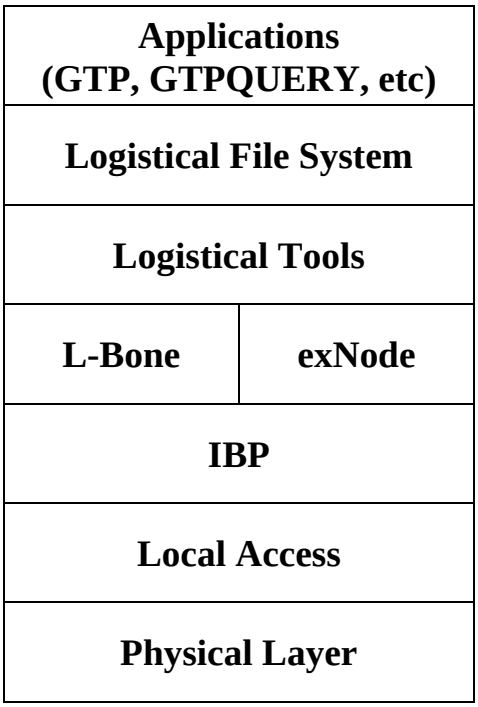

Figure 3. Network Storage Stack

3.1 IBP

The Internet Backplane Protocol (IBP) is the foundation of the Network Storage Stack. IBP's purpose is to allow users to share storage resources across networks. Its design echoes the major advantages of Internet Protocol (IP): the abstraction of the datagram delivery process, scalability, simple fault detection (faulty datagrams are dropped), and ease of access. These factors allow any participant in an IBP network to use any local storage resource available regardless of who owns it [2]. Using IP networking to access IBP storage creates a global storage service.

3.1.1 LIMITATIONS

Some limitations to the direct use of IBP storage arise from two underlying network problems. The first problem concerns a vulnerability of IP networks to Denial of Use (DoU). The free sharing of communication within a routed IP network leaves every local network open to being overwhelmed by traffic from the wide area network. The second concern is that the storage service is based on processor-attached storage, which implies strong semantics: near-perfect reliability and availability. It can be almost impossible to implement on the scale of the wide area networks [2]. These issues are resolved as follows:

IBP storage is time limited. When the time expires, the resources can be reused. An IBP allocation can also be refused by a storage facility (depot) if the user's request demands more resources than available.

IBP is a "best effort" storage service [4]. The semantics of IBP storage are weaker than the typical storage service. Since there are so many unpredictable and uncontrollable factors involved, network access to storage may become permanently unavailable (if a depot decides to withdraw from the pool, for example).

IBP storage is managed by depots or servers used by a client to perform storage operations. **Table 2** shows the IBP client calls classification.

Table 2. IBP Client Calls

Calls	Description
Allocate	Allocates requested amount of storage for requested amount of time if the depot can accommodate this request. If successful, the depot sends the user keys. The keys grant write, read, and manage privileges.
Store	Once the user has the capabilities (keys), he can write data to the allocation.
Load	Once the data is stored, it can be read from any offset within the allocated space.
Copy	Allows the user to transfer data from one allocation to the other.
Mcopy	Allows the user to transfer data from one allocation to multiple allocations.
Manage	Allows the user to change the properties of the allocation.

3.1.2 SECURITY

Security of the allocation is a major concern for any user. A user will not feel comfortable storing his data on the network if there might be a breach in security. The basis of IBP network security is the *capability,* or the *key*. Capabilities are created by the depot in response to an allocation request and returned to the client in the form of long, cryptographically secure byte strings [4]. Every subsequent request to perform any action on the allocated byte array must then present the appropriate capability. As long as capabilities are transmitted securely between client and server and the security of the depot itself is not compromised, only someone who has obtained the capability from the client can perform operations on the data stored [4]. It must be mentioned that this is the only level of security that IBP must deal with. The data encryption has to be handled in the layer(s) above IBP.

3.2 ExNode

The management of many IBP capabilities can be complicated. The exNode library was created to help the user in this task, and to automate most of the work. The exNode data structure is somewhat similar to the UNIX inode; but at the same time, it is fundamentally different. Just as inodes contain pointers to disk blocks, exNodes hold pointers to IBP allocations (or capabilities). Two major differences between exNodes and inodes are that the IBP buffers may be of any size, and their domains may overlap and be replicated [15]. Thus, the exNode allows users and applications to create network files out of time-limited and failure-prone IBP allocations in such a way that much stronger properties (e.g., fault-tolerance, longer duration) may be achieved [15]. **Figure 4** compares the exNode to the UNIX inode.

The exNode consists of two major components: arbitrary metadata and mappings. Metadata consists of <*name*, *value*, *type*> triplets where the types can be 64-bit integers, 64-bit floating-point numbers, character strings, and metadata lists. The metadata lists allow nesting of metadata [1].

Each mapping can also have a function metadata component that describes how the data was encoded. The function metadata is a nested list that describes the type of encodings and their relative order. Each function has arguments and might have metadata. If the user has encrypted and included checksums in the data, he/she can store the encryption algorithm name, the encryption key, and the checksum algorithm name using the function metadata [1].

The exNode library allows the user to create an exNode, attach a mapping to it, store IBP capabilities into the mapping, and add metadata to the mapping. The exNode can also be serialized to XML so that exNodes created on one platform can be recognized on the other supported platforms.

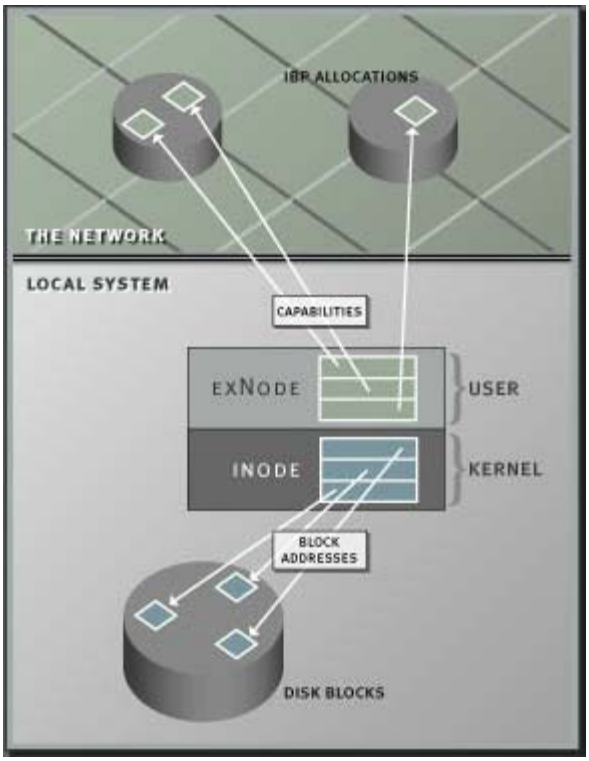

Figure 4. ExNode Compared to Inode.

The exNode makes it possible for the user to chain IBP allocations into a logical entity that resembles a network file [2]. Current IBP allocations have a limit of 2 GB; the exNode, however, allows the user to chain 2 billion IBP allocations, which equals 4 Exabytes (2^{62}) [2].

Each exNode can have multiple copies of the allocation, which provides better fault-tolerance. If a depot becomes unavailable for some reason, the user can still retrieve data from the copies stored on other depots.

3.3 L-Bone

The Logistical Backbone (L-Bone) is a distributed layer of middleware that allows access to a collection of IBP depots deployed on the Internet specifically to offer network storage to applications [16]. It is a *resource discovery service* that maintains a

list of public depots and metadata about those depots [2,3]. The metadata consists of IBP information such as *hostname*, *port*, and *allocation duration* policy, as well as recent *space availability* values. Currently, the L-Bone also maintains uptime, or availability, performance on each depot. The L-Bone server polls each depot once per hour. In addition to the IBP metadata, the L-Bone can also store geographic location information as well as machine room characteristics such as data backup policy, power backup availability, etc. [2]. The operating environment data currently kept is *connection speed*, amount of *monitoring*, the availability of *power backup*, the scheduling of *data backup* and whether the machine is behind a *firewall* [12].

The L-Bone combines both static information (such as IP addresses, zip codes, country codes) and dynamic decisions based on current network conditions to determine proximity. The L-Bone uses Network Weather Service (or NWS) [18] to monitor throughput between depots. NWS takes periodic measurements between depots, which it stores and uses to produce forecasts about network throughput, when needed [18]. As of December 2002, the L-Bone provide services of over 140 depots on five continents. **Figure 5** shows the locations of the available IBP depots [12].

3.4 LoRS

The next and final layer of the Network Storage Stack (see **Figure 3**) is the Logistical Runtime System (or LoRS). Although the L-Bone makes it easier to find depots and the ex-Node library handles IBP capabilities, the user still has to manually request allocations, store the data, create the ex-Node, attach mappings to the ex-Node, and insert the IBP allocations and metadata into the mappings [2]. The LoRS layer consists of a C API (Application Programming Interface) and a command line interface tool set that automate finding of IBP depots via the L-Bone, creating and using IBP capabilities, and creating and managing exNodes [12]. The LoRS library also provides flexible tools to deal with the lower levels of the Network Storage Stack. **Table 3** lists six network file-based functions provided by the LoRS tools [2].

Figure 5. The Map of L-Bone Depots Throughout the World.

Table 3. LoRS Functions

Function	Description
Upload	Upload data to a network file.
Download	Retrieve the data from a network file.
Augment	Add replicas to a network file.
Trim	Remove replicas from a network file.
Refresh	Modify the expiration time of a network file.
List	View the network file's metadata.

The LoRS API provides the user with more fine-grained control over the allocation. The API can store data from files or memory. Users may also use the API to implement new tools or capabilities such as multicast augments or overlay routing [2].

Both the LoRS tools and the API provide end-to-end services. To ensure that the data stored on the IBP depots was not altered in transit or while on disk, LoRS can insert MD5 checksums [2]. The MD5 (*M*essage *D*igest number 5) value for a file is a 128-bit value similar to a checksum. Its additional length (conventional checksums are usually either 16 or 32 bits) means that the possibility of a different or corrupted file having the same MD5 value as the file of interest is drastically reduced. During the download, if a block's checksum does not match, the block is discarded and the same block is downloaded from another source.

To protect data in transit or while it is stored on a depot, which is an unreliable server, LoRS provides multiple types of encryption, including DES. DES stands for Data Encryption Standard, which was adopted by NIST (National Institute of Standards and Technology) as a national standard in 1976. DES encrypts and decrypts data in 64-bit blocks, using a 64-bit key. To achieve extra security, the application may use additional

encryption algorithms and then add the algorithm type and key as function metadata to the exNode [2].

In addition to replication as a means for additional fault-tolerance, LoRS tools allow coding blocks to be stored as well. These coding blocks are similar to the parity blocks used in RAID storage systems. The addition of coding blocks can greatly improve fault-tolerance [2]. It gives an opportunity to restore the “lost” block of data from the remaining data and the coding blocks. To reduce the amount of data stored, LoRS also supports compression.

4. GTP WITH NETWORK STORAGE

The creation and maintenance of an index for a large text collection usually involves many modifications. These modifications may include the addition of new documents, the deletion of documents that are no longer needed, or the updating of existing documents. In any case, before the final index is created, several revisions may be needed thereby requiring the user to parse the collection multiple times. In some cases, the collection is dynamic, as is the case with web pages (HTML), so the parsing of such a collection has to be done on a regular basis in order to monitor updates. In other cases, the user will want to try different weighting schemes, or perhaps different methods of matrix decomposition. If the user decides to keep all the files generated by GTP and GTPQUERY after each parsing, the subsequent output files will take up an exhaustive amount of local disk storage.

Fortunately, the concept of network storage can alleviate the local disk storage burden: the user can clean up his hard drive and store the information produced by the parser on a remote network. The Logistical Networking Testbed developed at LoCI [12] appears to have the right set of tools to facilitate the temporary storage of these large files on a remote network (Internet) along with immediate retrieval when needed. The storage and retrieval processes are transparent to the user with insignificant time overhead.

Since the storage provided by the Internet Backplane Protocol (IBP) is temporary, if the user is not satisfied with the parser results, he might choose not to extend the time the files are stored on the network. Thus, when the allocation expires, the storage will be automatically reused. If, on the other hand, the user wants to store the results of the parser permanently, he can either make sure that the time limits for the storage depot do not expire or he can download the files back to his personal machine and then write them to other media, e.g., a CD-ROM.

4.1 OVERVIEW

The execution of GTP creates two large files: *keys* (the database of the terms parsed) and *output* (a binary file, containing vector encodings generated by the SVD) (see **Figure 2**). These files are essential to the GTP and GTPQUERY. If the user chooses to use network storage after the files *keys* and *output* are generated, they are automatically uploaded to IBP depot(s); and a set of capabilities is returned to the user in the form of XML files (one XML file for each file uploaded), which are stored in a designated directory. If the upload is successful, the files *keys* and *output* are deleted form the user's space. Currently, there is no automatic tool to provide time extension of the *.xnd* files, which expire after a certain number of days. Each depot provides storage for a designated number of days (1-28) [12]. It is the user's responsibility to make sure that allocated storage does not expire. If the allocation gets reused by the depot, all the data becomes unrecoverable.

When the user wants to query into the collection, the files are downloaded back to the user's space prior to execution of the query process using the information stored in XML files. The LoRS tools, described in detail in **Section 3.4**, are used to facilitate upload and download processing.

The processes of upload and download are made as transparent to the user as possible. The software provides the default upload and download, but the user is encouraged to supply additional information about the desired location of the allocation to speed up the process. The entire process of GTP and GTPQUERY with incorporated network storage is represented in **Figure 6**.

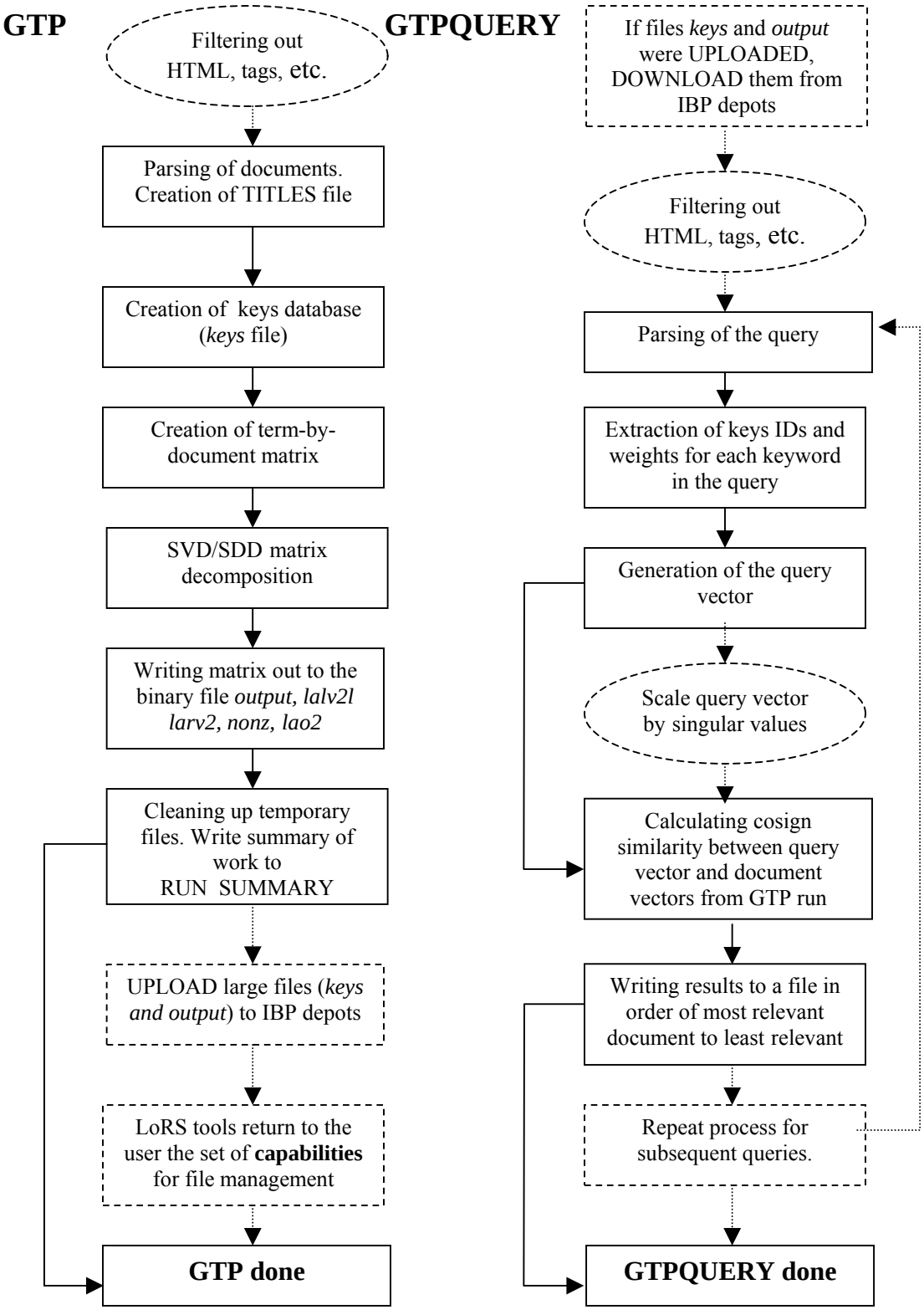

Figure 6. GTP and GTPQUERY Process with Embedded Network Storage. Dotted lines represent optional execution.

4.2 GTP AND UPLOAD

As seen from **Figure 6**, the process of GTP must complete creation of the files *keys* and *output* before they can be uploaded to the remote network. IBP requires knowledge of the file's size before uploading it. Our goal is to stream the data to network storage while it is being generated without creating local files.

The upload process requires as little or as much information from the user as he is willing to provide. This information helps to optimize the performance of the upload tools. Below is a short list of the fields the user can specify.

Location allows the user to enter keyword and value pairs to determine where he wants storage and minimum environmental criteria. The user may specify as many or as few keyword/value pairs as she wants. The location pointer can even be NULL if location and environment are unimportant. She can specify *hostname*, *zip*, *state*, *city*, *country* and *airport*. It is strongly recommended to indicate some sort of location to improve performance. If the user resides in Tennessee, it does not make sense to store the data in France or Australia. Since IBP relies on the performance of the network during the time of the upload, it might happen that the data will be stored in an undesired or distant location.

Duration is the maximum number of days that the user will need the space. The user can also specify partial day amounts. For example, if 0.5 is requested, data will be stored on the network for 12 hours. Each depot has maximum number of days the data can be stored. This information can be obtained from L-Bone's list of depots (see http://loci.cs.utk.edu/lbone). If a longer time period is required, the user becomes responsible for extending the time of the allocation. If the allocation is allowed to expire, the space it occupies will be reused by the network. A set of tools that automates the process of extending the duration of the storage has been developed by LoCI for some platforms.

The ***Fragments*** option allows the user to subdivide a file into partitions of equal size and to store those partitions on different depots. Available depot space is used more efficiently and the performance of the download can be greatly improved.

The ***Copies*** option allows the user to specify how many copies of the original file to store. Users are encouraged to store several copies of the data. As was mentioned in **Section 3.1**, there is always the possibility the data could be temporarily unavailable due to numerous uncontrollable circumstances. Subdividing the file into several fragments and storing multiple copies of the file can prevent an undesired loss of data. If during the download process some fragments cannot be found, LoRS tools will automatically check for all the copies of this fragment and will deliver the first available one.

If the upload process is successful, the LoRS tools will return to the user a file with an *.xnd* extension. This file contains XML encoded information needed by the user and IBP to keep track of the file, retrieve the file, and perform LoRS operations described in **Section 3.4**. GTP stores the XML files (one per uploaded file) in a directory, and will automatically delete the files being uploaded and conserve local disk storage. If on the other hand, the upload fails, the files will be saved on the user's machine and the user will be notified of the failure.

4.3 DOWNLOAD AND GTPQUERY

If IBP is used to store the GTP-generated index, a query into the document collection requires that the files *keys* and *output* be downloaded from the network. The download process depends solely on the XML files produced during the upload process. The exNode files (having the .xnd extension) store the location of the user's data within IBP. If these files do not exist, the download will fail, and the recovery of the data will be impossible. The LoRS download tool uses multiple threads to retrieve small blocks of data from the network, and then it reassembles the blocks into a complete file at the client. It uses an adaptive algorithm that retrieves more blocks from "faster" depots (depots with higher throughput to the client). Each active thread selects a different block

of the file to download, and all threads start downloading. When a thread is finished with its block, it selects a new block that is not being downloaded by any other thread. If a download fails, then the failed block becomes available again and another thread may attempt to download it. If some depots are much slower than others, the download tool can automatically try getting lagging blocks from other depots that have the same data [15]. The download tool is capable of starting from a specified offset and can process a prescribed bytecount of data. All GTP output files, however, are downloaded in their entirety.

After the download process is complete, the user will have the files necessary to perform any query on the collection. See **Figure 6** for the entire process of GTPQUERY with network storage.

4.4 USER INTERFACE

In order to make the usage of GTP and GTPQUERY as user-friendly as possible, a graphical user interface (GUI) [13] was designed (**Figure 7**). The defaults for every option were incorporated into the interface to make user's task of performing parsing and querying even easier. Network storage has its own panel (labeled "Network Storage"), that allows the user to specify the location, the duration of storage, and the number of fragments and copies needed (see **Section 4.4**). He will also be able to extend the time of the allocation and view all the details about the files stored on the network. When upload or download processes are activated, a special panel monitors the progress (**Figure 8**) and a map of the available depots provides visual information on where the files are being stored (**Figure 9**). Arrows indicate where the fragments and copies of the file are being uploaded or downloaded.

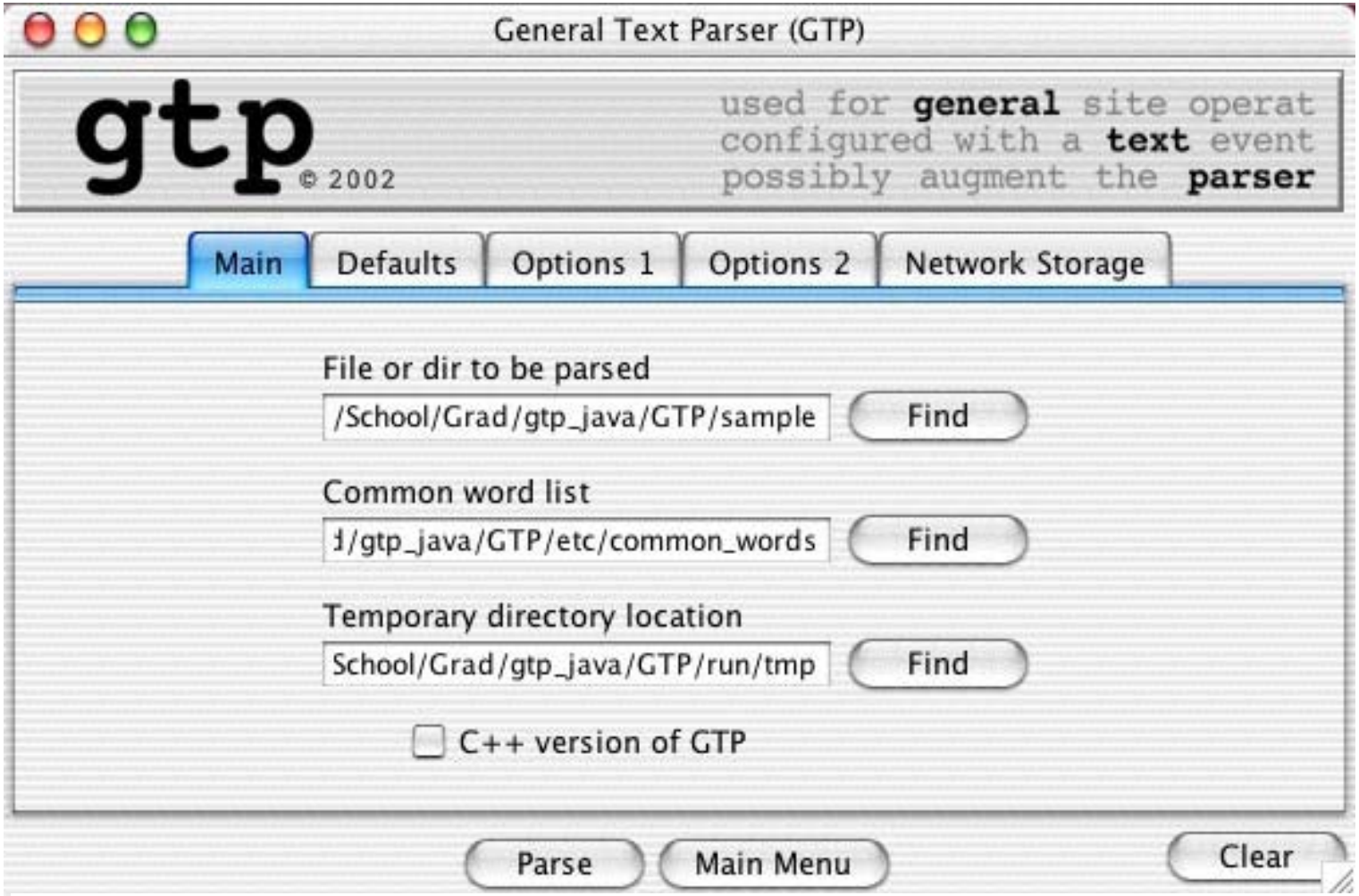

Figure 7. GTP's Graphical User Interface

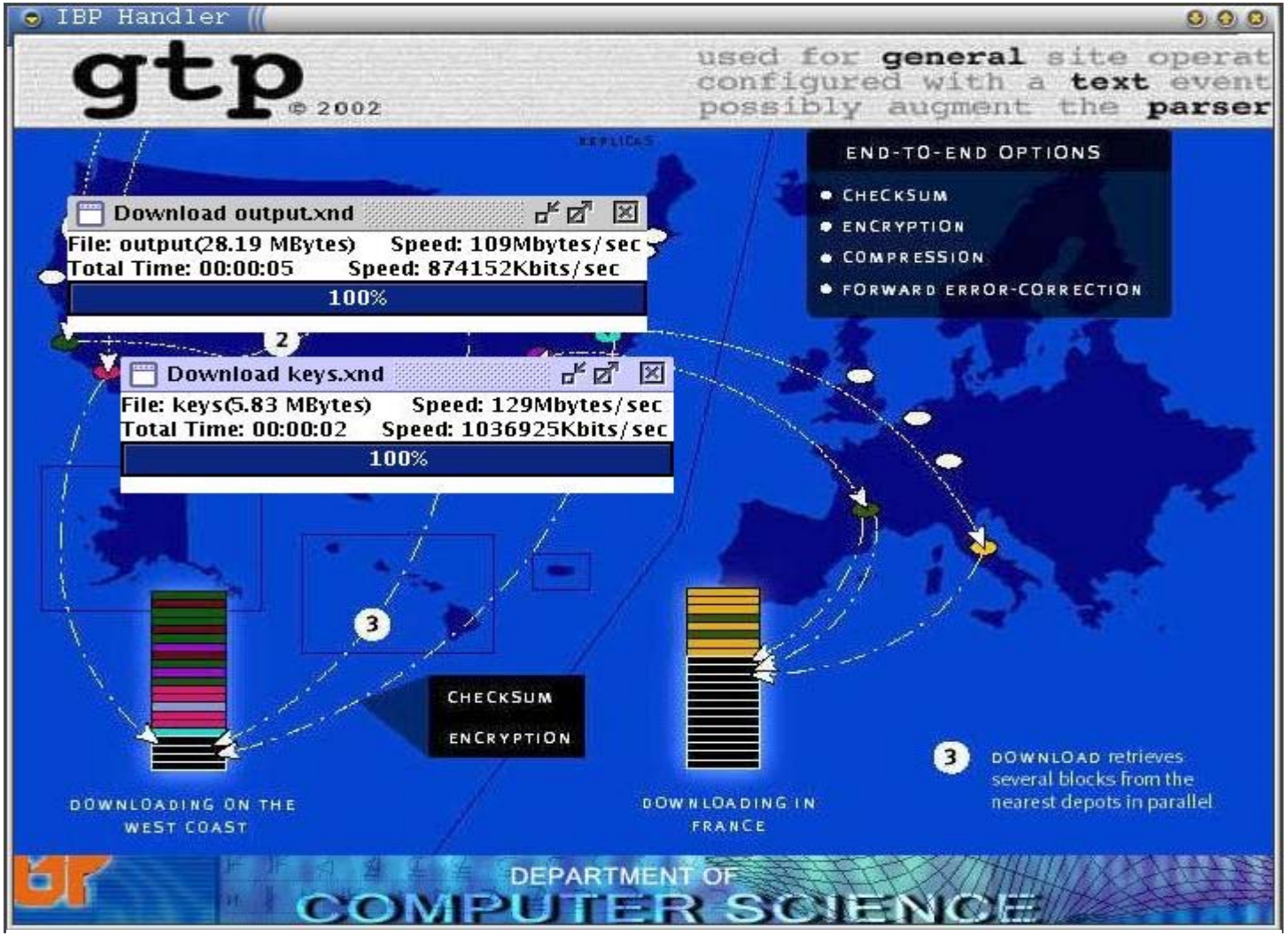

Figure 8. Network Storage Panel.

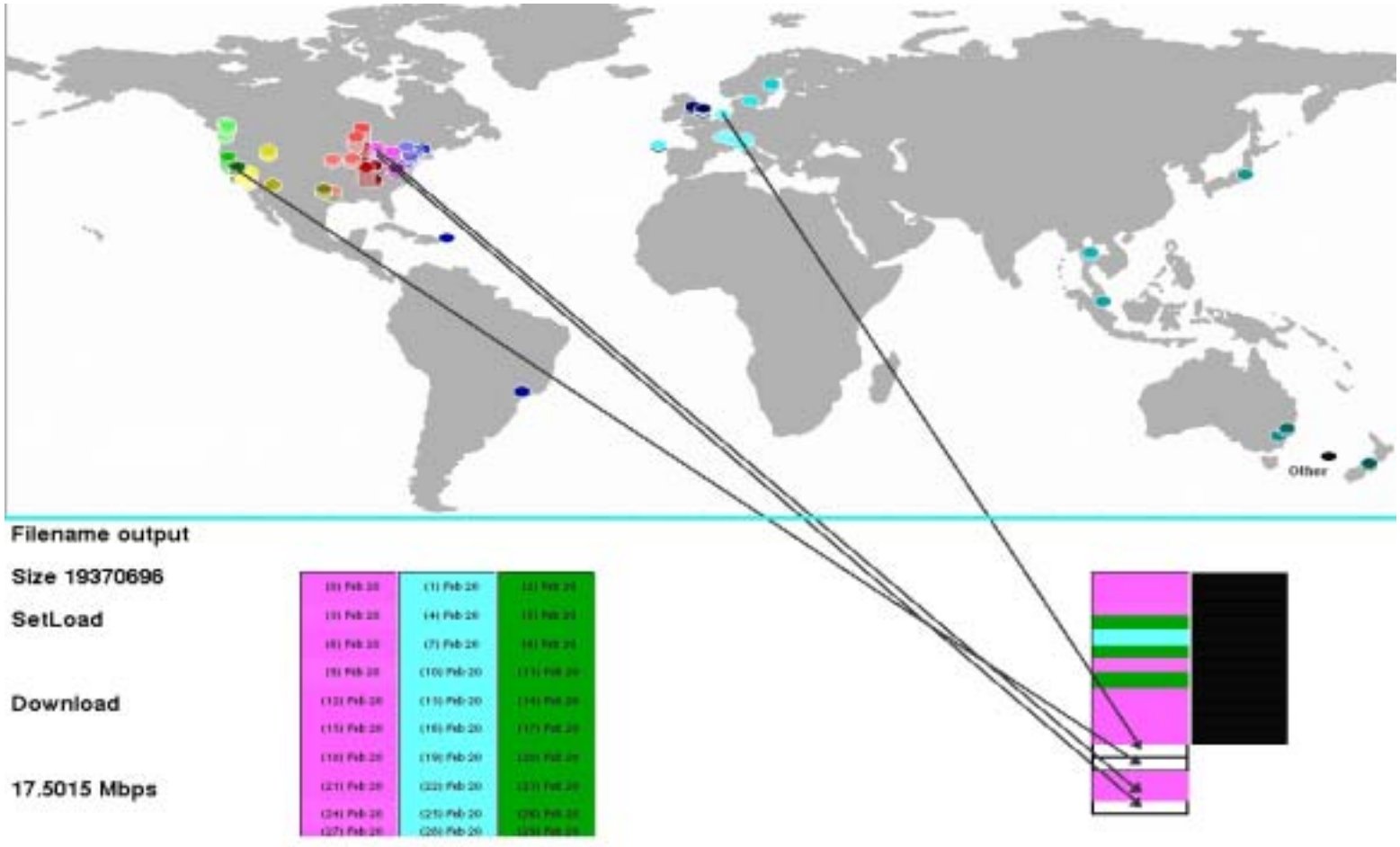

Figure 9. Map of the World with Depots Used by Network Storage.

5. IMPLEMENTATION CHALLENGES

Initial integration of network storage into GTP was achieved through the *xnd tools* – LoRS predecessors with more or less the same functionality. The x*nd tools* were written in C, while the target version of GTP was in Java. Fortunately, the *xnd* library included an *xnd_server* that could be used as a bridge between Java and C APIs. The GTP application had to stream specifically formatted data, required by the requested operation, through a socket directly to the *xnd_server*. The *xnd_server* then invoked the appropriate utility (upload or download) with the parameters passed from GTP. After completion of the operation, the *xnd_server* returned to GTP a byte string denoting either a success or a failure. If the operation was a success, the software received a byte stream with the parameters of the file followed by the file stream (XML encoded file for upload or the original file for download). At that point, such operations as *upload*, *download*, *refresh* and *list* were implemented. For the detailed description of these operations, see **Table 3**.

After the release of the LoRS tools in December 2002, the GTP software had to be adjusted to work with the new tools. A different server, *lors_server*, was also released with the tools. This server did not have an implementation of *refresh* or *list*, and the corresponding modules in GTP ceased to work. It was difficult to keep track of all the stored files and their expiration dates. At this stage, the capability of displaying the map of the world (with the depots) during upload/download process was added. The user could now visualize where the files' copies and fragments were stored, and obtain information on the expiration dates of the files.

To initiate an upload/download of several files at a time, the execution of these processes was accomplished with threads. Threading was intended to make the process of network storage faster and more efficient. When threading was added, a download of more than one file often failed. Extensive debugging helped to pinpoint and fix a race condition in the LoRS code.

The original Java code for GTP and GTPQUERY did not contain any graphical components. The panel for network storage was designed to visualize the process (**Figure 8**), and forced other tradeoff considerations. One such issue arose between the GUI component and GTP execution. For example, closing the storage panel accidentally during GTP execution terminated the GTP run. As a result, the decision was made to wait until GTP finishes creating its output to proceed with the upload. This issue should no longer exist when network storage is fully integrated with the main GUI.

The issues of extending the allocation time of the file and viewing it through the interface have not been solved successfully. As mentioned above, these features were implemented and functioned well with the *xnd tools* but are not supported by the new *lors_server.* The user could specify the number of days preferred for the allocation (usually longer than the depot's limit) since most depots have rather short storage periods (1-28 days). The code ran in the background daily to extend the allocation time by one day, until the allocation time requirement was met. The user could also view the file's metadata, including hostnames, storage expiration dates, file/fragment status, etc., in a comprehensible format. An appropriate set of tools is being developed by the LoCI researchers to enable such features for GTP users.

6. PERFORMANCE

The GTP software has been frequently tested and evaluated. The results described in this section were achieved using the Java version of GTP. The machine used in all experiments has following specifications:

- Dual Intel® Xeon™ 2.4GHz processors with 512KB advanced transfer L2 Cache
- 2GB of dual-channel, ECC, DDR 266MHz SDRAM memory
- 2GB available swap space
- 20GB ATA/100 7200 RPM hard drive

Benchmarks were produced on the three FBIS (Foreign Broadcast Information Service) sub collections from TREC-5 [17]. Specifications of the sub collections are described in **Table 4**.

Figures 10-12 illustrate the timing results for a GTP run for each of the collections with uploads to France (FR), California (CA), and Tennessee (TN) with the server located in Tennessee. Each GTP run was executed using the following command line options (for a detailed description of the options see **Appendix**):

UNIX> java GTP [collection name] -c [common word list] -t [temporary directory] -h -z svd1 test2 -R [name of the run] -O -I -w log entropy

Table 4. Collections Used for Benchmarking.

Name	Size	Documents	Distinct Terms	File *output*	File *keys*
FBIS 5K	17.8MB	5,000	22,558	11MB	2.78MB
FBIS 10K	32MB	10,000	31,667	18MB	3.5MB
FBIS 20K	63MB	20,000	46,488	28MB	5.8MB

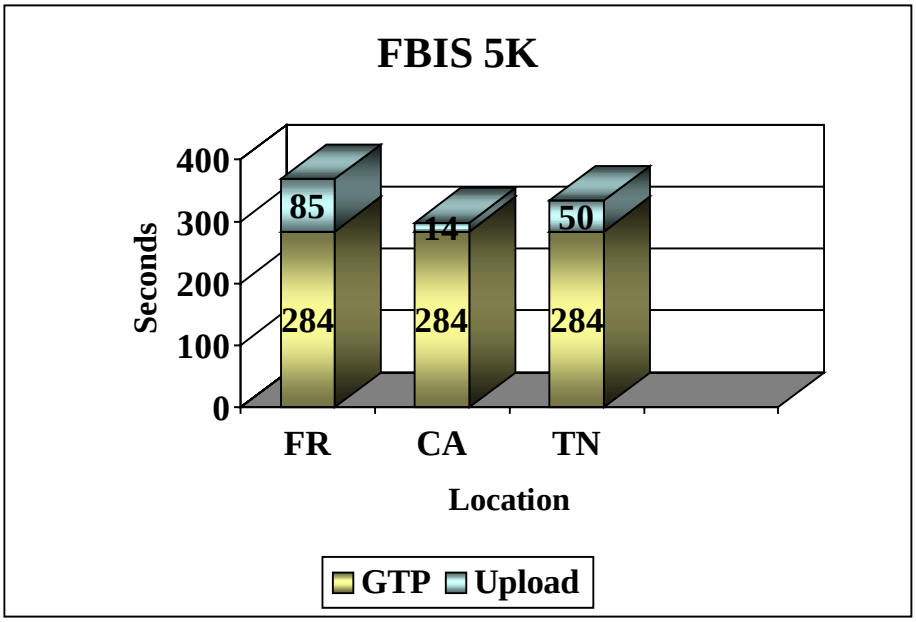

Figure 10. GTP Upload Benchmarks for FBIS 5K.

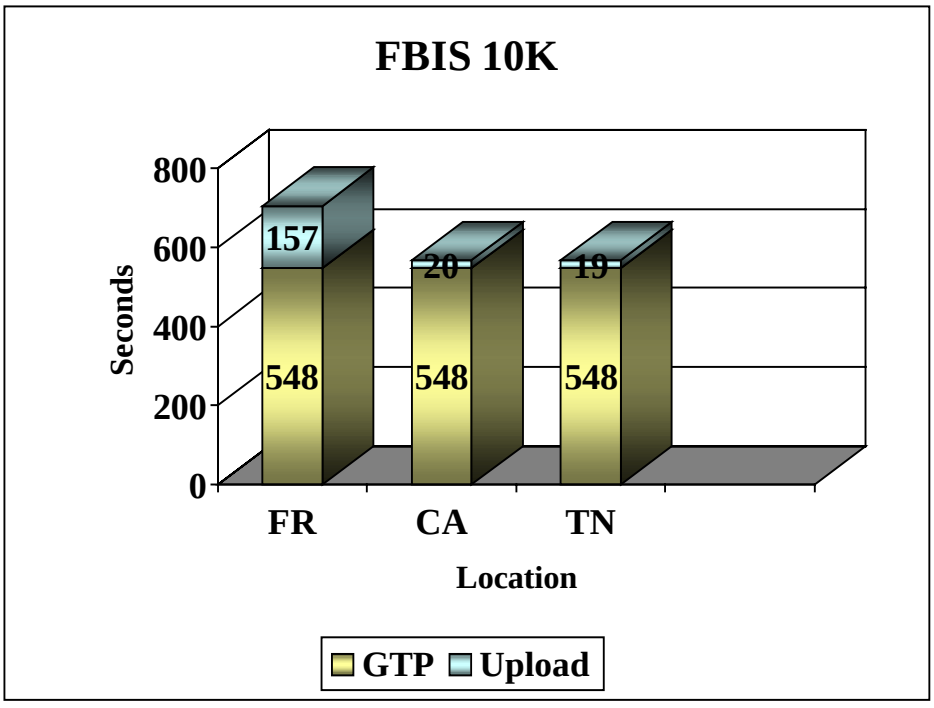

Figure 11. GTP Upload Benchmarks for FBIS 10K.

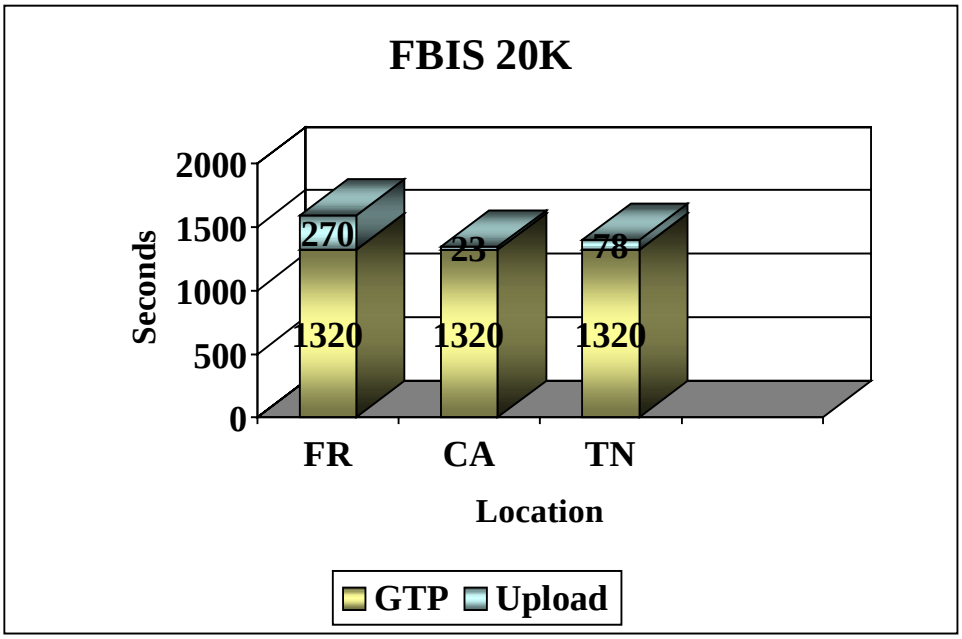

Figure 12. GTP Upload Benchmarks for FBIS 20K.

The parameters passed to the LoRS tools were the location strings: *zip=37966* for TN, *state=CA* for CA, and *country=FR* for France.

The GTP process time is fairly consistent (among the three different uploads) since the calculations are performed on the local machine. As can be seen from the results, the time required to run GTP is directly proportional to the collection size. Java is not the optimum language for performing intensive numerical computations such as the singular value decomposition (SVD).

Current benchmarks indicate that the additional time/overhead for upload is not significant compared to the total elapsed time. The time of the upload depends on multiple factors: how far the location of the upload is from the user's location, the network bandwidth, the time of day, the size of the file to be uploaded, and the number of copies requested. The results may vary from one run to another. The status of the depots at the time of the upload also greatly affects the timing results. If some depots in the area

requested for the upload are down, other depots in the area receive more traffic and the process slows down. The LoRS tools attempt to store the files in the location with optimal bandwidth. Depending on network conditions, the files might be uploaded to a location different from the one requested. During the benchmarking of the collections FBIS 5K and FBIS 20K, files intended for upload in Tennessee, were frequently uploaded to New Zealand and Australia (**Figures 10** and **12**).

While all the preprocessing is done by GTP during parsing and construction of the model, the GTPQUERY process simply projects the query into the term-document vector space. A 100-dimensional vector space was generated for the three different subsets of the FBIS for the collections listed in **Table 4**. Query vectors are generated as scaled linear combinations of the term vectors, the left singular vectors of the original term-by-document matrix. All queries were processed using the following command line options (for a detailed description of the options see **Appendix**):

UNIX> java GTPQUERY [query file] -c [common word list] -S -I -n 15

By default, GTP uses 100 SVD factors, i.e., all term and document vectors are of length 100. In the experiments shown below, only the first (or dominant) 15 singular triplets were used in the querying process (*-n 15* option). As mentioned above, each query vector was built from linear combinations of term vectors whose dimensions were scaled by the corresponding singular value (*-S* option).

The query file used for the three FBIS sub-collections in **Table 4** consisted of three queries, separated by a blank line:

Yugoslavia Croatia Bosnia-Herzegovina
Russia embassy FIS
Nissan Motor

Prior to querying, the files *keys* and *output* were downloaded from network storage. Metadata files with *.xnd* extension generated by the GTP's upload were streamed by GTPQUERY to the LoRS download routine.

Figures 13-15 demonstrate that, in most cases, the download takes up the greater portion of the run time. The time of the GTPQUERY depends mostly on the number of queries requested by the user. Download time, on the other hand, depends on many circumstances. The most important factors in the download process are the location of the file's fragments and copies and the current conditions of the network.

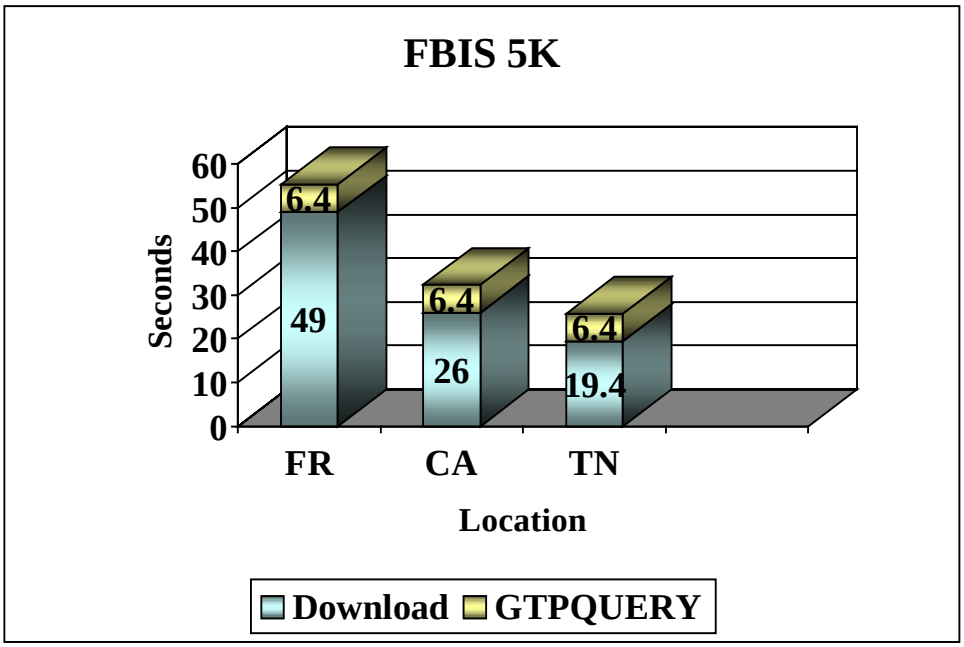

Figure 13. GTPQUERY Download Benchmarks for FBIS 5K.

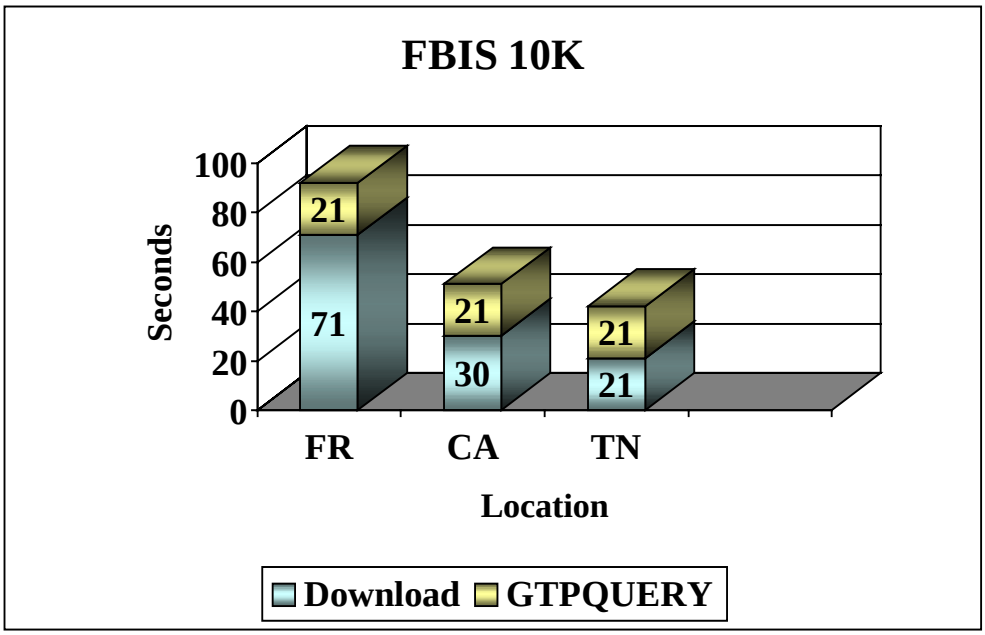

Figure 14. GTPQUERY Download Benchmarks for FBIS 10K.

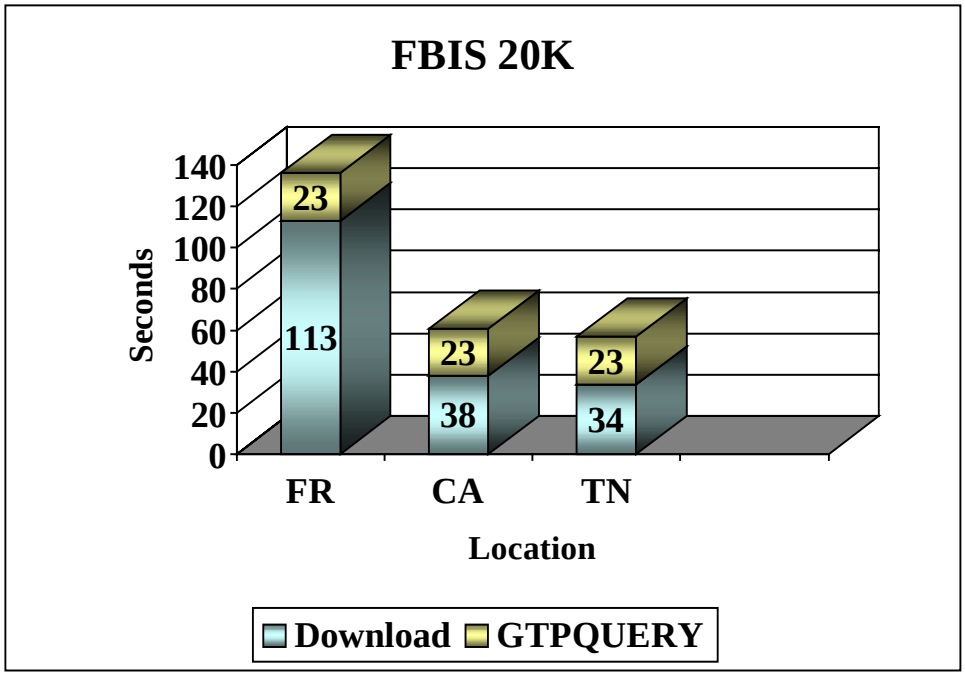

Figure 15. GTPQUERY Download Benchmarks for FBIS 20K.

7. FUTURE WORK

As researchers are trying to find ways to improve Java's performance in handling scientific calculations, new optimization techniques can be added to the Java version of the GTP to improve its performance during matrix creation and SVD calculations.

A graphical user interface (GUI) was recently developed to guide the user in selecting the numerous options for a customized IR model. The GTP GUI is still being refined and updated (**Figure 7**). Eventually, the GUI will display all available options. The Network Storage tab of the GUI is still in development and ultimately will be able to guide the user through the upload/download process, collect necessary information like location, duration, number of copies and fragments, and provide the user with the feedback on the upload/download process. The user will be able to view the information about the files stored on IBP and extend their storage time through the interface. The user will simply be able to press the "Parse" button and GTP will run, taking into account all the options and parameters specified.

The network storage option has been currently implemented only for the Java version of GTP – which presented some challenges, since all IBP and LoRS tools were originally implemented in C. The merge was possible due to a special LoRS server. In the future, it would be desirable to forgo the LoRS server altogether, and call the appropriate tool directly. This can be made possible through the usage of Java Native Interface (JNI), which allows an invocation of native methods like C functions within Java.

Work is in progress to integrate network storage into the C++ and parallel versions of the GTP. However, the integration should not present any difficulties since the LoRS C tools can be called directly from those versions.

In collaboration with LoCI Lab [12], refinements of the network storage procedure itself are underway. Upgrades include adding interactive maps and utilities to

allow the user to see more information about the files stored on IBP, extend storage time with a click of a button, and share files over the network.

LoCI researchers are working on the possibility of streaming data directly from a LoRS Java (or C) client to IBP depots as it is generated. Currently, streaming can only be performed using the LoRS C library or the UNIX command line tools. This would eliminate local file generation, which will greatly improve GTP's performance and storage requirements.

8. CONCLUSIONS

The amount of data processed in simulations by research scientists worldwide is rapidly accumulating. The lack of local storage is becoming a growing concern among the scientific community. The addition of a network storage capability to the General Text Parser software environment attempts to address the problem of inadequate storage and file sharing over the network for the purposes of information retrieval. Currently, large files cannot be sent via electronic mail. GTP with network storage gives users an opportunity to create a user-specific IR model, place the files (index) generated by GTP on a sharable network so that all the participants in a project can have access to them. The availability of the software[1] and its ease of use make it an invaluable tool in the hands of information retrieval and data mining professionals. The software is constantly being updated and augmented with innovative tools like network storage. The benchmarking results described in this paper provide motivation for further development of network storage capability for GTP as a solution to the shortage of local disk storage and file sharing.

1 GTP is public domain software available for downloading from *http://www.cs.utk.edu/~lsi.*

LIST OF REFERENCES

REFERENCES

[1] Atchley, S., Beck, M., Hagewood, H., Millar, J., Moore, T., Plank, J.S. and Soltesz, S. Next Generation Content Distribution Using the Logistical Networking Testbed. Technical Report No. UT-CS-02-498, Department of Computer Science, University of Tennessee, December 2002.

[2] Atchley, S., Beck, M., Millar, J., Moore, T., Plank, J.S. and Soltesz, S. The Logistical Networking Testbed. Technical Report No. UT-CS-02-496, Department of Computer Science, University of Tennessee, December 2002.

[3] Bassi, A., Beck, M. and Moore, T. Mobile Management of Network Files. Proceedings of the Third International Workshop on Active Middleware Services, San Francisco, CA, August, 2001.

[4] Bassi, A., Beck, M., Fagg, G., Moore, T., Plank, J., Swany, M. and Wolski, R. The Internet Backplane Protocol: A Study in Resource Sharing. In Proceedings of the second IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2002), Berlin, Germany, May 21-24, 2002.

[5] Beck, M., Moore, T. and Plank, J.S. An End-to-End Approach to Globally Scalable Network Storage. ACM SIGCOMM 2002 Conference, Pittsburgh, PA, August, 2002.

[6] Berry, M.W. and Browne, M. Understanding Search Engines: Mathematical Modeling and Text Retrieval, SIAM, Philadelphia, PA, 1999.

[7] Berry, M.W., Dumais, S.T. and O'Brien, G.W. Using Linear Algebra for Intelligent Information Retrieval. *SIAM Review*, 37:4:573-595, 1995.

[8] Berry, M.W., Drmač, Z. and Jessup, E.R. Matrices, Vector Spaces, and Information Retrieval. *SIAM Review*, 41:2:335-362, 1999.

[9] Giles, J.T., Wo, L. and Berry, M.W. GTP (General Text Parser) Software for Text Mining. Statistical Data Mining and Knowledge Discovery, H. Bozdogan (Ed.), CRC Press, Boca Raton, FL, 2003.

[10] Google Search Engine. http://www.google.com.

[11] Levy, S. A World According to Google. "Newsweek" magazine, December 16, 2002.

[12] Logistical Computing and Internetworking (LoCI) Lab, February 2003, http://www.loci.cs.utk.edu.

[13] Lynn, Patrick A. Evolving the General Text Parser (GTP) Utility into a Usable Application via Interface Design. Master's Thesis, Department of Computer Science, University of Tennessee, December 2002.

[14] Mironova, S.Y., Berry, M.W., Atchley, S. and Beck, M. General Text Parser (GTP) with Network Storage. Proceedings of the Text Mining Workshop, SIAM Third International Conference on Data Mining, San Francisco, CA, May 1-3, 2003.

[15] Plank, J.S., Atchley, S., Ding, Y. and Beck, M. Algorithms for High Performance, Wide-Area, Distributed File Downloads. Technical Report No. UT-CS-02-485, Department of Computer Science, University of Tennessee, October 2002.

[16] Plank, J.S., Bassi, A., Beck, M., Moore, T., Swany, M. and Wolski, R. Managing Data Storage in the Network. *IEEE Internet Computing*, 5:5:50-58, 2001.

[17] Proceedings of the Fifth Text Retrieval Conference (TREC-5), D. Harman and E.M. Voorhees (Eds.), NIST Special Publication 500-238, Department of Commerce, National Institute of Standards and Technology, Gaithersburg, MD, 1997.

[18] Wolski, R., Spring, N. and Hayes, J. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computer Systems.* Elsevier, vol. 15, pp. 757-768, 1999.

APPENDIX

APPENDIX 1

Options Available for GTP [9].

Option	Description	Dependency
-help	Summarize all the options.	
-q	Suppress progress summary.	
-h	Create the Harwell-Boeing compressed matrix. Default is to not create it.	Required if using –z option.
-u	Keep the Harwell-Boeing compressed matrix in an uncompressed file (on output) if the matrix is created.	-h
-N	Include numbers as keys.	
-D	Do not create Unix compatible dbm key file *keys*. Default is to generate it.	Do not use if you are to perform queries.
-K	Keep the keys file created in the temporary directory specified by the "-t temp_dir" argument.	
-T	Consider the first line of each document (up to 200 characters) to be the title of the document. Before this line is parsed, it will be written to the file TITLES in the current directory. Each title line in this file will exist on it's own line.	
-s	Normalize the document length. This ensures a unit length for columns in the term-by-document matrix.	
-m	Set a new minimum key length for the parser. The default minimum length is 2.	Must be an integer.
-M	Set a new maximum key length for the parser. The default maximum length is 20.	Must be an integer.

-L	Specify a new maximum line length. If any record being parsed exceeds this number of characters, the user is informed and the portion of the record that caused the overrun is printed to the screen. The default maximum is 10,000.	Must be an integer.
-S	Set the maximum number of common words to use. The default value is 1000.	Must be an integer.
-I	Use network storage. The files *output* and *keys* will be uploaded to the remote network.	
-d	Change the threshold for document frequency of any term. Default is 1.	Must be an integer.
-g	Change the threshold for global frequency of any term. Default is 1.	Must be an integer.
-e	Specify a string of characters, each of which will be considered a valid character, in addition to all default characters, when tokenizing keys.	
-f	Specify filters to pass each file through before the parser looks at it. If a filter has options, it needs to be surrounded by quotes. Works only for C++ version. Java has an internal HTML filter.	
-o	Specify that the key, id#, global frequency, document frequency, and weight of all keys are to be written to "filename".	
-B	Specify that a new document delimiter is needed. The new delimiter must be alone on a line in the file and must match exactly for GTP to recognize it. It can be up to 198 characters. Default is a blank line.	Cannot be used if -x is being used.

-x		Indicate that there is to be no delimiter other than the end of file.	Cannot be used if -B is being used.
-w	**local** **global**	Specify a custom weighting scheme. Local and global refer to local and global weighting formulas. Local can be tf (term frequency), log, or binary. Global can be normal, idf, idf2, or entropy. Default local is tf and global is not calculated.	
-R		Specify a name for the current run of GTP.	Must be a valid file name.
-z	**sdd** **rank** **inner_loop_criteria** **tolerance**	Specify the decomposition method.	Cannot use if using -z svd1. Have to use –h.
	svd1 **desc** **lanmax** **maxprs**		Cannot use if using -z sdd. Have to use -h.
-O		Specify that the output file is to be in one binary file for SVD. This is needed to use GTPQUERY.	-z svd1
-Z		Specify if parse procedure should be skipped so that an available matrix can be decomposed via SVD or SDD.	h -z svd1 … (or) -z sdd …

APPENDIX 2

Options Available for GTPQUERY [9].

Options		Description
-help		Summarize options.
-S		Scale the query vector by the singular values before calculating cosine similarity.
-n		Set the number of factors to use. Default is value of nfact found in LAST_RUN file generated by GTP.
-u		Set the upper threshold value for query results returned. If the upper threshold is set to 0.75, then all query results returned will be equal to or less than 0.75 (default is 1).
-l		Set the lower threshold value for query results returned. If the lower threshold is set to 0.25, then all query results returned will be greater than or equal to 0.25 (default is –1).
-r		Do not print query completion messages to stdout.
-I		Use network storage to download files from IBP.
-d	**[doc#,doc#,doc#] or [doc#:doc#]**	Specify which documents or range of documents for individual queries using document ids (integers). A new query is generated for each usage of the -d option. If there are several queries specified in filename, then all instances of the -d option are applied to each query in filename. For example, 2 queries in filename along with -d [1,2,3] -d [4:5], would result in the generation of four different queries.
-a		Specify that all query vectors should be normalized by the number of term and/or document vectors used in their construction. For example, a single query in filename with -d [1,2,3] -a would require that the final query vector have its elements divided by 4.
-k		Set the number of results returned to integer (default is all).
-f		Specify filters to pass each file through before the parser looks at it. If a filter has options, it needs to be surrounded by quotes.

-B	Specify that a new query delimiter is needed. New delimiter must be alone on a line in the file and must match exactly for the query processing to recognize it. It can be up to 198 characters, and the default delimiter is a single blank line. Cannot be used in conjunction with the –x option.
-x	Indicate that there is to be no delimiter other than the end of file. This cannot be used in conjunction with the –B option.

VITA

Svetlana Y. Mironova was born in Moscow, Russia on December 3, 1973. She was raised in Mytishchi, Russia where she attended elementary, middle, and high school. After receiving her high school diploma in 1991, she attended the Moscow State Linguistic University formerly the Maurice Thorez Moscow State Pedagogical Institute of Foreign Languages. She received her MA degree with Honors in Linguistics and Intercultural Communication in 1996. Svetlana received her MS degree in Computer Science from the Department of Computer Science at the University of Tennessee, Knoxville in May 2003.