The handbook of gStore System

Edited by gStore team [1]

August 6, 2017

[1]The mailing list is given in Chapter 13.

Contents

Preface

The RDF (*R*esource *D*escription *F*ramework) is a family of specifications proposed by W3C for modeling Web objects as part of developing the semantic web. In RDF model, each Web object is modeled as a uniquely named *resource* and denoted by a URI (*U*niform *R*esource *I*dentifier). RDF also uses URIs to name the properties of resources and the relationships between resources as well as the two ends of the link (this is usually referred to as a "triple"). Hence, an RDF dataset can be represented as a directed, labeled graph where resources are vertices, and triples are edges with property or relationship names as edge labels. For more details, please go to RDF Introduction

To retrieve and manipulate an RDF graph, W3C also proposes a structured query language, SPARQL (*S*imple *P*rotocol *A*nd *R*DF *Q*uery *L*anguage), to access RDF repository. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports aggregation, subqueries, negation, creating values by expressions, extensible value testing, and constraining queries by source RDF graph. Similar to RDF graphs, a SPARQL query can also be modeled as a graph, which is a query graph with some variables. Then, evaluating a SPARQL query is equivalent to finding subgraph (homomorphism) matches of a query graph over an RDF graph. You can have a better understanding of SPARQL at SPARQL Introduction.

Although there are some RDF data management systems (like Jena, Virtuoso, Sesame) that store the RDF data in relational systems, few existing systems exploit the native graph pattern matching semantics of SPARQL. **Here, we implement a graph-based RDF triple store named gStore, which is a joint research project by Peking University, University of Waterloo and Hong Kong University of Science and Technology.**

The system is developed and maintained by the database group in Institute of Computer Science and Technology, Peking University, China. A detailed description of gStore can be found at our papers [Zou et al., VLDB 11] and [Zou et al., VLDB Journal 14] in the Publication chapter. This HELP document includes system installment, usage, API, use cases and FAQ. gStore is a open-source project in github under the BSD license. You are welcome to use gStore, report bugs or suggestions, or join us to make gStore better. It is also allowed for you to build all kinds of applications based on gStore, while respecting our work.

Please make sure that you have read Legal Issues before using gStore.

Part I

Start

Chapter 00: A Quick Tour

Gstore System(also called gStore) is a graph database engine for managing large graph-structured data, which is open-source and targets at Linux operation systems. The whole project is written in C++, with the help of some libraries such as readline, antlr, and so on. Only source tarballs are provided currently, which means you have to compile the source code if you want to use our system.

Getting Started

This system is really user-friendly and you can pick it up in several minutes. Remember to check your platform where you want to run this system by viewing System Requirements. After all are verified, please get this project's source code. There are several ways to do this:

- download the zip from this repository and extract it
- fork this repository in your github account
- type `git clone git@github.com:Caesar11/gStore.git` in your terminal or use git GUI to acquire it

Then you need to compile the project, just type `make` in the gStore root directory, and all executables will be ok. To run gStore, please type `bin/gbuild database_name dataset_path` to build a database named by yourself. And you can use `bin/gquery database_name` command to query a existing database. What is more, `bin/gconsole` is a wonderful tool designed for you, providing all operations you need to use gStore. Notice that all commands should be typed in the root directory of gStore.

A detailed description can be found at Chapter 04 How to use in this document.

Advanced Help

If you want to understand the details of the gStore system, or you want to try some advanced operations(for example, using the API, server/client), please see the chapters below.

- Basic Introduction: introduce the theory and features of gStore
- Install Guide: instructions on how to install this system
- How To Use: detailed information about using the gStore system
- Socket API Explanation: guide you to develop applications based on our Socket API
- HTTP API Explanation: guide you to develop applications based on our HTTP API
- Project Structure: show the whole structure and sequence of this project
- Publications: contain essays and publications related with gStore
- Update Logs: keep the logs of the system updates
- Test Result: present the test results of a series of experiments

Other Business

We have written a series of short essays addressing recurring challenges in using gStore to realize applications, which are placed in Recipe Book.

You are welcome to report any advice or errors in the github Issues part of this repository, if not requiring in-time reply. However, if you want to urgent

on us to deal with your reports, please email to to submit your suggestions and report bugs to us by emailing to . A full list of our whole team is in Contributors.

There are some restrictions when you use the current gStore project, you can see them on Limitations.

Sometimes you may find some strange phenomena(but not wrong case), or something hard to understand/solve(don't know how to do next), then do not hesitate to visit the Frequently Asked Questions page.

Graph database engine is a new area and we are still trying to go further. Things we plan to do next is in Future Plan chapter, and we hope more and more people will support or even join us. You can support in many ways:

- watch/star our project
- fork this repository and submit pull requests to us
- download and use this system, report bugs or suggestions
- ...

People who inspire us or contribute to this project will be listed in the Thanks List chapter.

Chapter 01: System Requirements

We have tested on linux server with CentOS 6.2 x86_64 and CentOS 6.6 x86_64. The version of GCC should be 4.4.7 or later.

Item	Requirement
operation system	Linux, such as CentOS, Ubuntu and so on
architecture	x86_64
disk size	according to size of dataset
memory size	according to size of dataset
glibc	version >= 2.14
gcc	version >= 4.4.7
g++	version >= 4.4.7
make	need to be installed
readline	need to be installed
readline-devel	need to be installed
openjdk	needed if using Java api
openjdk-devel	needed if using Java api
realpath	needed if using gconsole
ccache	optional, used to speed up the compilation

Table 1: software requirement

NOTICE:

1. The name of some packages may be different in different platforms, just install the corresponding one in your own operation system.

2. To install readline and readline-devel, just type `dnf install readline-devel` in Redhat/CentOS/Fedora, or `apt-get install libreadline-dev` in Debian/Ubuntu. Please use corresponding commands in other systems. If you use ArchLinux, just type `pacman`

`-S readline` to install the readline and readline-devel.(so do other packages)

3. You do not have to install realpath to use gStore, but if you want to use the gconsole for its convenience, please do so by using `dnf install realpath` or `apt-get install realpath`.

4. Our programs use regEx functions, which are provided by GNU/Linux by default. You do not need to have to install boost and boost-devel for more powerful regEx libraries.

5. ANTLR3.4 is used in gStore to produce lexer and parser code for SPARQL query. However, you do not need to install the corresponding antlr libraries because we have merged the libantlr3.4 in our system.

6. When you type `make` in the root directory of the gStore project, the Java api will also be compiled. You can modify the makefile if you do not have JDK in your system. However, you are advised to install openjdk-devel in your Linux system.

7. To install ccache, you need to add epel repository if using CentOS, while in Ubuntu you can directly install it by 'apt-get install ccache' comand. If you can not install ccahe(or maybe you do not want to), please go to modify the makefile(just change the CC variable to g++).

8. Any other questions, please go to FAQ page.

Chapter 02: Basic Introduction

The first essay to come up with Gstore System is gStore_VLDBJ, and you can find related publications in Publications.

What Is gStore

gStore is a graph-based RDF data management system(or what is commonly called a "triple store") that maintains the graph structure of the original RDF data. Its data model is a labeled, directed multi edge graph, where each vertex corresponds to a subject or an object.

We represent a given SPARQL query by a query graph Q. Query processing involves finding subgraph matches of Q over the RDF graph G, instead of joining tables in relational data management system. gStore incorporates an index over the RDF graph (called VS-tree) to speed up query processing. VS-tree is a height balanced tree with a number of associated pruning techniques to speed up subgraph matching.

The gStore project is supported by the National Science Foundation of China (NSFC), Natural Sciences and Engineering Research Council (NSERC) of Canada, and Hong Kong RGC.

Why gStore

After a series of test, we analyse and keep the result in Test Results. gStore runs faster to answer complicated queries(for example, contain circles) than other database systems. For simple queries, both gStore and other database systems work well.

In addition, now is the big data era and more and more structured data is coming, while the original relational database systems(or database systems based on relational tables) cannot deal with them efficiently. In contrast, gStore can utilize the features of graph data structures, and improve the performance.

What is more, gStore is a high-extensible project. Many new ideas of graph database have be proposed, and most of them can be used in gStore. For example, our group is also designing a distributed gstore system, which is expected to be released at the end of 2016.

Open Source

The gStore source code is available as open-source code under the BSD license. You are welcome to use gStore, report bugs or suggestions, or join us to make gStore better. It is also allowed for you to build all kinds of applications based on gStore, while respecting our work.

Chapter 03: Install Guide

You are advised to read init.conf file, and modify it as you wish. (this file will configure the basic options of gStore system)

gStore is a green software, and you just need to compile it with one command. Please run `make` in the gStore root directory to compile the gStore code, link the ANTLR lib, and build executable "gbuild", "gquery", "gserver", "gclient", "gconsole". What is more, the api of gStore is also built now.

If you want to use API examples of gStore, please run `make APIexample` to compile example codes for both C++ API and Java API. For details of API, please visit API chapter.

Use `make clean` command to clean all objects, executables, and use `make dist` command to clean all objects, executables, libs, datasets, databases, debug logs, temp/text files in the gStore root directory.

You are free to modify the source code of gStore and create your own project while respecting our work, and type `make tarball` command to compress all useful files into a .tar.gz file, which is easy to carry.

Type `make gtest` to compile the gtest program if you want to use this test utility. You can see the HOW TO USE for details of gtest program.

Chapter 04: How To Use

gStore currently includes five executables and others.

All the commands of gStore should be used in the root directory of gStore like bin/gconsole, because executables are placed in bin/, and they may use some files whose paths are indicated in the code, not absolute paths. We will ensure that all paths are absolute later by asking users to give the absolute path in their own systems to really install/configure the gStore. However, you must do as we told now to avoid errors.

0. gconsole gconsole is the main console of gStore, which integrates with all functions to operate on gStore, as well as some system commands. Completion of commands name, line editing features and access to the history list are all provided. Feel free to try it, and you may have a wonderful tour!(spaces or tabs at the beginning or end is ok, and no need to type any special characters as separators)

```
[bookug@localhost gStore]$ bin/gconsole
Gstore Console(gconsole), an interactive shell based utility to communicate with
gStore repositories.
usage: start-gconsole [OPTION]
-h,--help              print this help
-s,--source            source the SPARQL script
For bug reports and suggestions, see https://github.com/Caesar11/gStore

notice that commands are a little different between native mode and remote mode!
now is in native mode, please type your commands.
please do not use any separators in the end.

gstore>help
```

```
gstore>help drop
drop            Drop a database according to the given path.

gstore>connect 127.0.0.1 3305
now is in remote mode, please type your commands.

server>disconnect
now is in native mode, please type your commands.

gstore>build lubm_10 ./data/LUBM_10.n3
...
import RDF file to database done.

gstore>unload

gstore>load lubm_10
...
database loaded successfully!

gstore>show
lubm_10

gstore>query ./data/LUBM_q0.sql
...
final result is :
?x
<http://www.Department0.University0.edu/FullProfessor0>
<http://www.Department1.University0.edu/FullProfessor0>
<http://www.Department2.University0.edu/FullProfessor0>
<http://www.Department3.University0.edu/FullProfessor0>
<http://www.Department4.University0.edu/FullProfessor0>
```

```
<http://www.Department5.University0.edu/FullProfessor0>
<http://www.Department6.University0.edu/FullProfessor0>
<http://www.Department7.University0.edu/FullProfessor0>
<http://www.Department8.University0.edu/FullProfessor0>
<http://www.Department9.University0.edu/FullProfessor0>
<http://www.Department10.University0.edu/FullProfessor0>
<http://www.Department11.University0.edu/FullProfessor0>
<http://www.Department12.University0.edu/FullProfessor0>
<http://www.Department13.University0.edu/FullProfessor0>
<http://www.Department14.University0.edu/FullProfessor0>

gstore>query "select distinct ?x ?y where { ?x <rdf:type>
<ub:UndergraduateStudent> .
?x <ub:takesCourse> ?y . ?y <ub:name> <FullProfessor1> . }"
final result is :
?x      ?y
[empty result]

gstore>unload

gstore>quit
```

Just type `bin/gconsole` in the root directory of gStore to use this console, and you will find a `gstore>` prompt, which indicates that you are in native mode and can type in native commands now. There are another mode of this console, which is called remote mode. Just type `connect` in the native mode to enter the remote mode, and type `disconnect` to exit to native mode.(the console connect to a gStore server whose ip is '127.0.0.1' and port is 3305, you can specify them by type `connect gStore_server_ip gStore_server_port`)

You can use `help` or `?` either in native mode or remote mode to see the help information, or you can type `help command_name` or `? command_name` to see

the information of a given command. Notice that there are some differences between the commands in native mode and commands in remote mode. For example, system commands like `ls`, `cd` and `pwd` are provided in native mode, but not in remote mode. Also take care that not all commands contained in the help page are totally achieved, and we may change some functions of the console in the future.

What we have done is enough to bring you much convenience to use gStore, just enjoy it!

1. gbuild gbuild is used to build a new database from a RDF triple format file.

```
bin/gbuild db_name rdf_triple_file_path
```

For example, we build a database from LUBM_10.n3 which can be found in example folder.

```
[bookug@localhost gStore]$ bin/gbuild LUBM10 ./data/LUBM_10.n3
gbuild...
argc: 3 DB_store:LUBM10      RDF_data: ./data/LUBM_10.n3
begin encode RDF from : ./data/LUBM_10.n3 ...
```

2. gquery gquery is used to query an existing database with files containing SPARQL queries.(each file contains exact one SPARQL query)

Type `bin/gquery db_name query_file` to execute the SPARQL query retrieved from query_file in the database named db_name.

Use `bin/gquery --help` for detail information of gquery usage.

To enter the gquery console, type `bin/gquery db_name`. The program shows a command prompt("gsql>"), and you can type in a command here. Use `help` to see basic information of all commands, while `help command_t` shows details of a specified command.

Type `quit` to leave the gquery console.

For `sparql` command, input a file path which contains a single SPARQL query. (*answer redirecting to file is supported*)

When the program finish answering the query, it shows the command prompt again.

gStore2.0 only support simple "select" queries(not for predicates) now.

We also take LUBM_10.n3 as an example.

```
[bookug@localhost gStore]$ bin/gquery LUBM10
gquery...
argc: 2 DB_store:LUBM10/
loadTree...
LRUCache initial...
LRUCache initial finish
finish loadCache
finish loadEntityID2FileLineMap
open KVstore
finish load
finish loading
Type `help` for information of all commands
Type `help command_t` for detail of command_t
gsql>sparql ./data/LUBM_q0.sql
... ...
Total time used: 4ms.
final result is :
<http://www.Department0.University0.edu/FullProfessor0>
<http://www.Department1.University0.edu/FullProfessor0>
<http://www.Department2.University0.edu/FullProfessor0>
<http://www.Department3.University0.edu/FullProfessor0>
<http://www.Department4.University0.edu/FullProfessor0>
<http://www.Department5.University0.edu/FullProfessor0>
<http://www.Department6.University0.edu/FullProfessor0>
```

```
<http://www.Department7.University0.edu/FullProfessor0>
<http://www.Department8.University0.edu/FullProfessor0>
<http://www.Department9.University0.edu/FullProfessor0>
<http://www.Department10.University0.edu/FullProfessor0>
<http://www.Department11.University0.edu/FullProfessor0>
<http://www.Department12.University0.edu/FullProfessor0>
<http://www.Department13.University0.edu/FullProfessor0>
<http://www.Department14.University0.edu/FullProfessor0>
```

Notice:

- "[empty result]" will be printed if no answer, and there is an empty line after all results.
- readline lib is used, so you can use arrow key in your keyboard to see command history, and use and arrow key to move and modify your entire command.
- path completion is supported for utility. (not built-in command completion)

3. ghttp ghttp is a daemon. It should be launched first when accessing gStore by HTTP protocol. It uses port 9000.

Just type `bin/ghttp` to start server. After the server is started, you can access it by visit the url in a browser or use the Restful API in your program. You can press Ctrl-C to stop the server. (Multiple connections are supported in HTTP server)

```
[bookug@localhost gStore]$ bin/ghttp
the current settings are as below:
key : value
------------------------------------------------------------
BackupTime : 2000          # 4 am (GMT+8)
```

```
buffer_maxium : 100
db_home : .
db_suffix : .db
debug_level : simple
gstore_mode : single
operation_logs : true
thread_maxium : 1000

enter initialize.
server port: 9000 database name:
```

URL rules are listed blow:

parameters: operation, db_name, ds_path, format, sparql

NOTICE: do URL encoding before sending it to database server.

operation: build, load, unload, query, monitor, show, checkpoint

- db_name: the name of database, like lubm
- format: html, json, txt, csv
- sparql: select ?s where ?s ?p ?o .
- ds_path in the server: like /home/data/test.n3

Examples:

- to build a database from a dataset:
  http://localhost:9000/?operation=build&db_name=[db_name]&ds_path=[ds_path]
- to load a database:
  http://localhost:9000/?operation=load&db_name=[db_name]
- to query a database:
  http://localhost:9000/?operation=query&format=[format]&sparql=[sparql]

- to unload a database:
  http://localhost:9000/?operation=unload&db_name=[db_name]
- to monitor the server:
  http://localhost:9000/?operation=monitor
- to show the database used:
  http://localhost:9000/?operation=show
- to save the database currently:
  http://localhost:9000/?operation=checkpoint

4. gserver gserver is a daemon. It should be launched first when accessing gStore by gclient or API. It communicates with client through socket.

```
[bookug@localhost gStore]$ bin/gserver -s
Server started at port 3305

[bookug@localhost gStore]$ bin/gserver -t
Server stopped at port 3305
```

You can also assign a custom port for listening.

```
[bookug@localhost gStore]$ bin/gserver -p 3307
Port changed to 3307.
```

Notice: Multiple threads are not supported by gserver. If you start up gclient in more than one terminal in the same time, gserver will go down.

5. gclient gclient is designed as a client to send commands and receive feedbacks.

```
[bookug@localhost gStore]$ bin/gclient
ip=127.0.0.1 port=3305
```

```
gsql>help
help - print commands message
quit - quit the console normally
import - build a database for a given dataset
load - load an existen database
unload - unload an existen database
sparql - load query from the second argument
show - show the current database's name
gsql>import lubm data/LUBM_10.n3
import RDF file to database done.
gsql>load lubm
load database done.
gsql>sparql "select ?s ?o where { ?s <rdf:type> ?o . }"
[empty result]

gsql>quit
```

You can also assign gserver's ip and port.

```
[bookug@localhost gStore]$ bin/gclient 172.31.19.15 3307
ip=172.31.19.15 port=3307
gsql>
```

We can use these following commands now:

- `help` shows the information of all commands
- `import db_name rdf_triple_file_name` build a database from RDF triple file
- `load db_name` load an existing database
- `unload db_name` unload database, but will not delete it on disk, you can load it next time

- `sparql "query_string"` query the current database with a SPARQL query string(quoted by “”)
- `show` displays the name of the current loaded database

Notice:

- at most one database can be loaded in the gclient console
- you can place ‘ ’ or ‘\t’ between different parts of command, but not use characters like ‘;’
- you should not place any space or tab ahead of the start of any command

6. test utilities A series of test program are placed in the test/ folder, and we will introduce the two useful ones: gtest.cpp and full_test.sh

gtest is used to test gStore with multiple datasets and queries.

To use gtest utility, please type `make gtest` to compile the gtest program first. Program gtest is a test tool to generate structural logs for datasets. Please type `./gtest --help` in the working directory for details.

Please change paths in the test/gtest.cpp if needed.

You should place the datasets and queries in this way:

```
DIR/WatDiv/database/*.nt
```

```
DIR/WatDiv/query/*.sql
```

Notice that DIR is the root directory where you place all datasets waiting to be used by gtest. And WatDiv is a class of datasets, as well as LUBM. Inside WatDiv(or LUBM, etc. please place all datasets(named with .nt) in a database/ folder, and place all queries(corresponding to datasets, named with .sql) in a query folder.

Then you can run the gtest program with specified parameters, and the output will be sorted into three logs in gStore root directory: load.log/(for database loading time and size), time.log/(for query time) and result.log/(for all query results, not the entire output strings, but the information to record the selected two database systems matched or not).

All logs produced by this program are in TSV format(separated with '\t'), you can load them into Calc/Excel/Gnumeric directly. Notice that time unit is ms, and space unit is kb.

full_test.sh is used to compare the performance of gStore and other database systems on multiple datasets and queries.

To use full_test.sh utility, please download the database system which you want to tats and compare, and set the exact position of database systems and datasets in this script. The name strategy should be the same as the requirements of gtest, as well as the logs strategy.

Only gStore and Jena are tested and compared in this script, but it is easy to add other database systems, if you would like to spend some time on reading this script. You may go to test report or Frequently Asked Questions for help if you encounter a problem.

7. gadd gadd is used to add triples in a file to an existing database.

Usage: `bin/gadd db_name rdf_triple_file_path`.

```
[bookug@localhost gStore]$ bin/gadd lubm ./data/LUBM\_10.n3
...
argc: 3 DB_store:lubm   insert file:./data/LUBM_10.n3
get important pre ID
...
insert rdf triples done.
inserted triples num: 99550
```

8. gsub gsub is used to remove triples from an existing database.

Usage: `bin/gsub db_name rdf_triple_file_path`.

```
[bookug@localhost gStore]$ bin/gsub lubm data/LUBM\_10.n3
...
argc: 3 DB_store:lubm  remove file: data/LUBM\_10.n3
...
remove rdf triples done.
removed triples num: 99550
```

9. gmonitor After starting ghttp, go into gStore/bin/ and type `./gmonitor ip port` to check current status of gStore.

```
[bookug@localhost bin]$ ./gmonitor 127.0.0.1 9000
parameter: ?operation=monitor
request: http://127.0.0.1:9000/%3Foperation%3Dmonitor
null--->[HTTP/1.1 200 OK]
Content-Length--->[127]
database: lubm
triple num: 99550
entity num: 28413
literal num: 0
subject num: 14569
predicate num: 17
connection num: 7
```

10. gshow After starting ghttp, go into gStore/bin and type `./gshow ip port` to check loaded database.

```
[bookug@localhost gStore]$ ./gshow 127.0.0.1 9000
parameter: ?operation=show
request: http://127.0.0.1:9000/%3Foperation%3Dshow
null--->[HTTP/1.1 200 OK]
```

```
Content-Length--->[4]
lubm
```

Part II

Advanced

Chapter 05: Socket API Explanation

This Chapter guides you to use socket API for accessing gStore, which can be used when the server runs gserver. We also provide HTTP API for ghttp, please see HTTP API Explanation .

Easy Examples

We provide JAVA, C++, PHP and Python API for gStore now. Please refer to example codes in `api/socket/cpp/example`, `api/socket/java/example`, `api/socket/php` and `api/socket/python/example`. To use the four examples to have a try, please ensure that executables have already been generated. Otherwise, for Java and C++, just type `make APIexample` in the root directory of gStore to compile the codes, as well as API.

Next, **start up a gStore server by using `./gserver` command.** It is ok if you know a running usable gStore server and try to connect to it, but notice that **the server ip and port of server and client must be matched.**(you don't need to change any thing if using examples, just by default) Then, for Java and C++ code, you need to compile the example codes in the directory gStore/api/socket/. We provide a utility to do this, and you just need to type `make APIexample` in the root directory of gStore. Or you can compile the codes by yourself, in this case please go to gStore/api/socket/cpp/example/ and gStore/api/socket/java/example/, respectively.

Finally, go to the example directory and run the corresponding executables. For C++, just use `./example` command to run it. And for Java, use `make run` command or `java -cp ../lib/GstoreJavaAPI.jar:. JavaAPIExample` to run it. For PHP, use `php ./PHPAPIExample`. For python, use `python`

`./PythonAPIExample`. All these four executables will connect to a specified gStore server and do some load or query operations. Be sure that you see the query results in the terminal where you run the examples, otherwise please go to Frequently Asked Questions for help or report it to us.(the report approach is described in README)

You are advised to read the example code carefully, as well as the corresponding Makefile. This will help you to understand the API, specially if you want to write your own programs based on the API interface.

API structure

The API of gStore is placed in api/socket/ directory in the root directory of gStore, whose contents are listed below:

- gStore/api/socket/
  - cpp/ (the C++ API)
    - src/ (source code of C++ API, used to build the lib/libgstore-connector.a)
      - GstoreConnector.cpp (interfaces to interact with gStore server)
      - GstoreConnector.h
      - Makefile (compile and build lib)
    - lib/ (where the static lib lies in)
      - .gitignore
      - libgstoreconnector.a (only exist after compiled, you need to link this lib when you use the C++ API)
    - example/ (small example program to show the basic idea of using the C++ API)
      - CppAPIExample.cpp
      - Makefile

- java/ (the Java API)
  - src/ (source code of Java API, used to build the lib/GstoreJavaAPI.jar)
    - jgsc/GstoreConnector.java (the package which you need to import when you use the Java API)
    - Makefile (compile and build lib)
  - lib/
    - .gitignore
    - GstoreJavaAPI.jar (only exist after compiled, you need to include this JAR in your class path)
  - example/ (small example program to show the basic idea of using the Java API)
    - JavaAPIExample.cpp
    - Makefile
- php/ (the PHP API)
  - GstoreConnector.php (source code of PHP API, you need to include this file when you use the PHP API)
  - PHPAPIExample.php (small example program to show the basic idea of using the PHP API)
- python/ (the Python API)
  - src/ (source code of Python API)
    - GstoreConnector.py (the package which you need to import when you use the Python API)
  - example/ (small example program to show the basic idea of using the Python API)
    - PythonAPIExample.py

C++ API

Interface To use the C++ API, please place the phrase `#include "GstoreConnector.h"` in your cpp code. Functions in GstoreConnector.h should be called like below:

```
// initialize the Gstore server's IP address and port.
GstoreConnector gc("127.0.0.1", 3305);
// build a new database by a RDF file.
// note that the relative path is related to gserver.
gc.build("LUBM10", "example/LUBM_10.n3");
// then you can execute SPARQL query on this database.
std::string sparql = "select ?x where \
{\
?x    <rdf:type>    <ub:UndergraduateStudent>. \
?y    <ub:name> <Course1>. \
?x    <ub:takesCourse>  ?y. \
?z    <ub:teacherOf>    ?y. \
?z    <ub:name> <FullProfessor1>. \
?z    <ub:worksFor>    ?w. \
?w    <ub:name>    <Department0>. \
}";
std::string answer = gc.query(sparql);
// unload this database.
gc.unload("LUBM10");
// also, you can load some exist database directly and then query.
gc.load("LUBM10");
// query a SPARQL in current database
answer = gc.query(sparql);
```

The original declaration of these functions are as below:

```
GstoreConnector();
GstoreConnector(string _ip, unsigned short _port);
```

```
GstoreConnector(unsigned short _port);
bool load(string _db_name);
bool unload(string _db_name);
bool build(string _db_name, string _rdf_file_path);
string query(string _sparql);
```

Notice:

1. When using GstoreConnector(), the default value for ip and port is 127.0.0.1 and 3305, respectively.
2. When using build(), the rdf_file_path(the second parameter) should be related to the position where gserver lies in.
3. Please remember to unload the database you have loaded, otherwise things may go wrong.(the errors may not be reported!)

Compile You are advised to see gStore/api/socket/cpp/example/Makefile for instructions on how to compile your code with the C++ API. Generally, what you must do is compile your own code to object with header in the C++ API, and link the object with static lib in the C++ API.

Let us assume that your source code is placed in test.cpp, whose position is ${GSTORE}/gStore/.(if using devGstore as name instead of gStore, then the path is ${GSTORE}/devGstore/ directory first:

Use `g++ -c -I${GSTORE}/gStore/api/socket/cpp/src/ test.cpp -o test.o` to compile your test.cpp into test.o, relative API header is placed in api/socket/cpp/src/.

Use `g++ -o test test.o -L${GSTORE}/gStore/api/socket/cpp/lib/ -lgstoreconnector` to link your test.o with the libgstoreconnector.a(a static lib) in api/socket/cpp/lib/.

Then you can type `./test` to execute your own program, which uses our C++ API. It is also advised for you to place relative compile commands in a Makefile, as well as other commands if you like.

Java API

Interface To use the Java API, please place the phrase `import jgsc.GstoreConnector;` in your java code. Functions in GstoreConnector.java should be called like below:

```
// initialize the Gstore server's IP address and port.
GstoreConnector gc = new GstoreConnector("127.0.0.1", 3305);
// build a new database by a RDF file.
// note that the relative path is related to gserver.
gc.build("LUBM10", "example/LUBM_10.n3");
// then you can execute SPARQL query on this database.
String sparql = "select ?x where " + "{" +
"?x    <rdf:type>    <ub:UndergraduateStudent>. " +
"?y    <ub:name> <Course1>. " +
"?x    <ub:takesCourse>  ?y. " +
"?z    <ub:teacherOf>    ?y. " +
"?z    <ub:name> <FullProfessor1>. " +
"?z    <ub:worksFor>    ?w. " +
"?w    <ub:name>    <Department0>. " +
"}";
String answer = gc.query(sparql);
//unload this database.
gc.unload("LUBM10");
//also, you can load some exist database directly and then query.
gc.load("LUBM10");// query a SPARQL in current database
answer = gc.query(sparql);
```

The original declaration of these functions are as below:

```
GstoreConnector();
GstoreConnector(string _ip, unsigned short _port);
GstoreConnector(unsigned short _port);
bool load(string _db_name);
bool unload(string _db_name);
bool build(string _db_name, string _rdf_file_path);
string query(string _sparql);
```

Notice:

1. When using GstoreConnector(), the default value for ip and port is 127.0.0.1 and 3305, respectively.
2. When using build(), the rdf_file_path(the second parameter) should be related to the position where gserver lies in.
3. Please remember to unload the database you have loaded, otherwise things may go wrong.(the errors may not be reported!)

Compile You are advised to see gStore/api/socket/java/example/Makefile for instructions on how to compile your code with the Java API. Generally, what you must do is compile your own code to object with jar file in the Java API.

Let us assume that your source code is placed in test.java, whose position is ${GSTORE}/gStore/.(if using devGstore as name instead of gStore, then the path is ${GSTORE}/devGstore/ directory first:

> Use `javac -cp ${GSTORE}/gStore/api/socket/java/lib/GstoreJavaAPI.jar test.java` to compile your test.java into test.class with the GstoreJavaAPI.jar(a jar package used in Java) in api/socket/-java/lib/.

Then you can type `java -cp ${GSTORE}/gStore/api/socket/java/lib/GstoreJavaAPI.jar:. test` to execute your own program(notice that the ":" in command cannot

be neglected), which uses our Java API. It is also advised for you to place relative compile commands in a Makefile, as well as other commands if you like.

PHP API

Interface To use the PHP API, please place the phrase `include('GstoreConnector,php');` in your php code. Functions in GstoreConnector.php should be called like below:

```
// initialize the Gstore server's IP address and port.
$gc = new Connector("127.0.0.1", 3305);
// build a new database by a RDF file.
// note that the relative path is related to gserver.
$gc->build("LUBM10", "example/LUBM_10.n3");
// then you can execute SPARQL query on this database.
$sparql = "select ?x where " + "{" +
"?x    <rdf:type>    <ub:UndergraduateStudent>. " +
"?y    <ub:name> <Course1>. " +
"?x    <ub:takesCourse>  ?y. " +
"?z    <ub:teacherOf>    ?y. " +
"?z    <ub:name> <FullProfessor1>. " +
"?z    <ub:worksFor>    ?w. " +
"?w    <ub:name>    <Department0>. " +
"}";
$answer = gc->query($sparql);
//unload this database.
$gc->unload("LUBM10");
//also, you can load some exist database directly and then query.
$gc->load("LUBM10");// query a SPARQL in current database
$answer = gc->query(sparql);
```

The original declaration of these functions are as below:

```
class Connector {
public function __construct($host, $port);
public function send($data);
public function recv();
public function build($db_name, $rdf_file_path);
public function load($db_name);
public function unload($db_name);
public function query($sparql);
public function __destruct();
}
```

Notice:

1. When using Connector(), the default value for ip and port is 127.0.0.1 and 3305, respectively.
2. When using build(), the rdf_file_path(the second parameter) should be related to the position where gserver lies in.
3. Please remember to unload the database you have loaded, otherwise things may go wrong.(the errors may not be reported!)

Run You can see gStore/api/socket/php/PHPAPIExample for instructions on how to use PHP API. PHP script doesn't need compiling. You can run PHP file directly or use it in your web project.

Python API

Interface To use the Python API, please place the phrase `from GstoreConnector import GstoreConnector` in your python code. Functions in GstoreConnector.py should be called like below:

```
// initialize the Gstore server's IP address and port.
```

```
gc = GstoreConnector('127.0.0.1', 3305)
// build a new database by a RDF file.
// note that the relative path is related to gserver.
gc.build('LUBM10', 'data/LUBM_10.n3')
// then you can execute SPARQL query on this database.
$sparql = "select ?x where " + "{" +
"?x    <rdf:type>    <ub:UndergraduateStudent>. " +
"?y    <ub:name> <Course1>. " +
"?x    <ub:takesCourse>  ?y. " +
"?z    <ub:teacherOf>    ?y. " +
"?z    <ub:name> <FullProfessor1>. " +

"?z    <ub:worksFor>    ?w. " +
"?w    <ub:name>    <Department0>. " +
"}";
answer = gc.query(sparql)
//unload this database.
gc.unload('LUBM10')
//also, you can load some exist database directly and then query.
gc.load('LUBM10')// query a SPARQL in current database
answer = gc.query(sparql)
```

The original declaration of these functions are as below:

```
class GstoreConnector {
def _connect(self)
def _disconnect(self)
def _send(self, msg):
def _recv(self)
def _pack(self, msg):
def _communicate(f):
def __init__(self, ip='127.0.0.1', port=3305):
```

```
@_communicate
     def test(self)
@_communicate
def load(self, db_name)
@_communicate
def unload(self, db_name)
@_communicate
def build(self, db_name, rdf_file_path)
@_communicate
def drop(self, db_name)
@_communicate
def stop(self)
@_communicate
def query(self, sparql)
@_communicate
def show(self, _type=False)
}
```

Notice:

1. When using GstoreConnector(), the default value for ip and port is 127.0.0.1 and 3305, respectively.
2. When using build(), the rdf_file_path(the second parameter) should be related to the position where gserver lies in.
3. Please remember to unload the database you have loaded, otherwise things may go wrong.(the errors may not be reported!)

Run You are advised to see gStore/api/socket/python/example/PythonAPIExample for examples on how to use python API. Python file doesn't need compiling, and you can run it directly.

Chapter 06: HTTP API Explanation

This chapter provides API for ghttp. Compared with socket API, HTTP API is more stable and more standard, and can maintain connection. Socket API can not guaratee correct transmission, so the network transmission is faster.

Easy Examples

We provide JAVA and C++ API for ghttp now. Please see `api/http/cpp` and `api/http/java`. To use these examples, please make sure that executables have already been generated.

Next, **start up ghttp service by using `./ghttp` command.** It is ok if you know a running usable ghttp server and try to connect to it. (you don't need to change anything if using examples, just by default) Then, for Java and C++ code, you need to compile the example codes in the directory gStore/api/http/. We provide a utility to do this, and you just need to type `make APIexample` in the root directory of gStore. Or you can compile the codes by yourself, in this case please go to gStore/api/http/cpp/ and gStore/api/http/java/, respectively.

Finally, go to the example directory and run the corresponding executables. All these four executables will connect to a specified ghttp server and do some load or query operations. Be sure that you see the query results in the terminal where you run the examples, otherwise please go to Frequently Asked Questions for help or report it to us.(the report approach is described in README)

You are advised to read the example code carefully, as well as the corresponding Makefile. This will help you to understand the API, specially if you want to write your own programs based on the API interface.

API Structure

The HTTP API of gStore is placed in api/http/ directory in the root directory of gStore, whose contents are listed below:

- gStore/api/http/
  - cpp/ C++ API
    * client.cpp source code of C++ API
    * client.h
    * example.cpp (example program to show the basic idea of using the C++ API)
    * Makefile compile
  - java/ Java API
    * src/ (source code of Java API, used to build the lib/GstoreJavaAPI.jar)
      · jgsc/GstoreConnector.java (the package which you need to import when you use the Java API)
      · Makefile (compile and build lib)
    * lib/
      · .gitignore
      · GstoreJavaAPI.jar (only exist after compiled, you need to include this JAR in your class path)
    * example/ (small example program to show the basic idea of using the Java API)
      · JavaAPIExample.cpp
      · Makefile

C++ API

Interface To use the C++ API, please place the phrase `#include "Client.h"` in your cpp code. Functions in Client.h should be called like below:

```
CHttpClient hc;
string res;
int ret;
// build a new database by a RDF file.
ret = hc.Get("127.0.0.1:9000/build/lumb/data/LUBM_10.n3", res);
cout<<res<<endl;
// load databse
ret = hc.Get("127.0.0.1:9000/load/lumb", res);
cout<<res<<endl;
// then you can execute SPARQL query on this database.
ret = hc.Get("127.0.0.1:9000/query/data/ex0.sql", res);
cout<<res<<endl;
// output information of current database
ret = hc.Get("127.0.0.1:9000/monitor", res);
cout<<res<<endl;
// unload this databse
ret = hc.Get("127.0.0.1:9000/unload", res);
cout<<res<<endl;
```

The original declaration of these functions are as below:

```
CHttpClient();
int Post(const std::string & strUrl, const std::string & strPost, std::string & strR
int Get(const std::string & strUrl, std::string & strResponse);
int Posts(const std::string & strUrl, const std::string & strPost, std::string & str
int Gets(const std::string & strUrl, std::string & strResponse, const char * pCaPath
```

Java API

Interface To use the Java API, please place the phrase `import jgsc.GstoreConnector;` in your java code. Functions in GstoreConnector.java should be called like below:

```
// initialize the Gstore server's IP address and port.
GstoreConnector gc = new GstoreConnector("127.0.0.1", 9000);
// build a new database by a RDF file.
// note that the relative path is related to gserver.
gc.build("LUBM10", "example/LUBM_10.n3");
gc.load("LUBM10");
// then you can execute SPARQL query on this database.
String sparql = "select ?x where " + "{" +
"?x    <rdf:type>    <ub:UndergraduateStudent>. " +
"?y    <ub:name> <Course1>. " +
"?x    <ub:takesCourse>  ?y. " +
"?z    <ub:teacherOf>    ?y. " +
"?z    <ub:name> <FullProfessor1>. " +
"?z    <ub:worksFor>    ?w. " +
"?w    <ub:name>    <Department0>. " +
"}";
String answer = gc.query(sparql);
//unload this database.
gc.unload("LUBM10");
//also, you can load some exist database directly and then query.
gc.load("LUBM10");// query a SPARQL in current database
answer = gc.query(sparql);
gc.unload("LUBM10");
```

The original declaration of these functions are as below:

```
GstoreConnector();
GstoreConnector(int _port);
```

```
GstoreConnector(String _ip, int _port);
boolean load(String _db_name);
boolean unload(String _db_name);
boolean build(String _db_name, String _rdf_file_path);
boolean drop(String _db_name);
String query(String _sparql);
String show();
String show(boolean _type);
```

Chapter 07: Use gStore in Web

This Chapter provides a specific example on how to use our API in a web project.

Example

Now you have the basic idea on how to use our APIs to connect gStore. Yet you might be still a little confused. Here we provide a simple demo to show you what to do explicitly.

Let's say, you need to use gStore in a web project. PHP is a popular general-purpose scripting language that is especially suited to web development. So, using our PHP API can meet your requirements. Here is what we implement: http://59.108.48.18/Gstore/form.php.

First, get your web server ready so it can run PHP files. We won't give detailed instructions on this step here. You can easily google it according to your web server(for example, Apache or Nginx, etc.)

Next, go to your web document root(usually in /var/www/html or apache/htdocs, you can check it in config file), and create a folder named "Gstore". Then copy the GstoreConnector.php file into it. Create a "PHPAPI.php" file. Edit it like below:

```
<?php

include( 'GstoreConnector.php');
$host = '127.0.0.1';
$port = 3305;
$dbname = $_POST["databasename"];
$sparql = $_POST["sparql"];
$format = $_POST["format"];
$load = new Connector($host, $port);
$load->load($dbname);
```

```
$query = new Connector($host, $port);
$result = $query->query($sparql);
switch ($format) {
     case 1:
        $array = explode("<", $result);
      $html = '<html><table class="sparql" border="1"><tr><th>' .
                 $array[0] . "</th></tr>";
        for ($i = 1; $i < count($array); $i++) {
            $href = str_replace(">", "", $array[$i]);
            $html.= '<tr><td><a href="' . $href . '">' .
                    $href . '</a></td></tr>';
        }
        $html.= '</table></html>';
        echo $html;
        exit;

     case 2:
        $filename = 'result.txt';
        header("Content-Type: application/octet-stream");
        header('Content-Disposition: attachment;
                filename="' . $filename . '"');
        echo $result;
        exit;

     case 3:
        $filename = 'result.csv';
        header("Content-Type: application/octet-stream");
        header('Content-Disposition: attachment;
                filename="' . $filename . '"');
        $array = explode("<", $result);
        echo $array[0];
```

```
        for ($i = 1; $i < count($array); $i++) {
            $href = str_replace(">", "", $array[$i]);
            echo $href;
         }
        exit;
}
?>
```

This PHP file get three parametres from a website, including databasename, sparql and output format. Then it use our PHP API to connect gStore and run the query. Finally, the "switch" part gives the output.

After that, we need a website to collect those imformation(databasename, sparql and output format). We create a html file and use a form to do it, just like below:

```
<form id="form_1145884" class="appnitro"  method="post" action="PHPAPI.php">
  <div class="form_description">
    <h2>Gstore SPARQL Query Editor</h2>
    <p></p>
  </div>
  <ul>
    <li id="li_1" >
      <label class="description" for="element_1">
        Database Name
      </label>
      <div>
     <input id="element_1" name="databasename" class="element text medium"
       type="text" maxlength="255" value="dbpedia_2014_reduce">
        </input>
      </div>
    </li>
```

```
<li id="li_3">
<label class="description" for="element_3">Query Text </label>
  <div>
 <textarea id="element_3" name="sparql" class="element textarea large">
      SELECT DISTINCT ?uri
      WHERE {
        ?uri <type> <Astronaut> .
      { ?uri <nationality> <Russia> . }
      UNION
      { ?uri <nationality> <Soviet_Union> . }
      }
  </textarea>
  </div>
</li>

<li id="li_5" >
  <label class="description" for="element_5">
    Results Format
  </label>
  <div>
 <select class="element select medium" id="element_5" name="format">
      <option value="1" selected="ture">HTML</option>
      <option value="2" >Text</option>
      <option value="3" >CSV</option>
    </select>
  </div>

<li class="buttons">
  <input type="hidden" name="form_id" value="1145884" />
  <input id="saveForm" class="button_text" type="submit"
   name="submit" value="Run Query" />
```

```
    </li>
  </ul>
</form>
```

As you can see in the code, we use a <input> element to get the database-name, and <texarea> for sparql, <select> for output format. <form> lable has an attribute "action" which specifies which file to execute. So, when you click the "submit" button, it will call PHPAPI.php file and post the values from the form.

Finally, don't forget to start gserver on your server.

Chapter 08: Project Structure

This chapter introduce the whole structure of the gStore system project.

The core source codes are listed below:

- Database/ (calling other core parts to deal with requests from interface part)
  - Database.cpp (achieve functions)
  - Database.h (class, members and functions definitions)
  - Join.cpp (join the node candidates to get results)
  - Join.h (class, members,, and functions definitions)
- KVstore/ (a key-value store to swap between memory and disk)
  - KVstore.cpp (interact with upper layers)
  - KVstore.h
  - heap/ (a heap of nodes whose content are in memory)
    - Heap.cpp
    - Heap.h
  - node/ (all kinds of nodes in B+-tree)
    - Node.cpp (the base class of IntlNode and LeafNode)
    - Node.h
    - IntlNode.cpp (internal nodes in B+-tree)
    - IntlNode.h
    - LeafNode.cpp (leaf nodes in B+-tree)
    - LeafNode.h
  - storage/ (swap contents between memory and disk)

    * file.h
    * Storage.cpp
    * Storage.h
  - tree/ (implement all tree operations and interfaces)
    * Tree.cpp
    * Tree.h

- Query/ (needed to answer SPARQL query)
  - BasicQuery.cpp (basic type of queries without aggregate operations)
  - BasicQuery.h
  - IDList.cpp (candidate list of a node/variable in query)
  - IDList.h
  - ResultSet.cpp (keep the result set corresponding to a query)
  - ResultSet.h
  - SPARQLquery.cpp (deal with a entire SPARQL query)
  - SPARQLquery.h
  - Varset.cpp
  - Varset.h
  - QueryTree.cpp
  - QueryTree.h
  - GeneralEvaluation.cpp
  - GeneralEvaluation.h
  - RegexExpression.h

- Signature/ (assign signatures for nodes and edges, but not for literals)
  - SigEntry.cpp

  - SigEntry.h
  - Signature.cpp
  - Signature.h

- VSTree/ (an tree index to prune more efficiently)
  - EntryBuffer.cpp
  - EntryBuffer.h
  - LRUCache.cpp
  - LRUCache.h
  - VNode.cpp
  - VNode.h
  - VSTree.cpp
  - VSTree.h

The parser part is listed below:

- Parser/
  - DBParser.cpp
  - DBParser.h
  - RDFParser.cpp
  - RDFParser.h
  - SparqlParser.c (auto-generated, subtle modified manually, compressed)
  - SparqlParser.h (auto-generated, subtle modified manually, compressed)
  - SparqlLexer.c (auto-generated, subtle modified manually, compressed)

  - SparqlLexer.h (auto-generated, subtle modified manually, compressed)
  - TurtleParser.cpp
  - TurtleParser.h
  - Type.h
  - QueryParser.cpp
  - QueryParser.h

The utilities are listed below:

- Util/
  - Util.cpp (headers, macros, typedefs, functions…)
  - Util.h
  - Bstr.cpp (represent strings of arbitrary length)
  - Bstr.h (class, members and functions definitions)
  - Stream.cpp (store and use temp results, which may be very large)
  - Stream.h
  - Triple.cpp (deal with triples, a triple can be divided as subject(entity), predicate(entity), object(entity or literal))
  - Triple.h
  - BloomFilter.cpp
  - BloomFilter.h

The interface part is listed below:

- Server/ (client and server mode to use gStore)
  - Client.cpp

  - Client.h
  - Operation.cpp
  - Operation.h
  - Server.cpp
  - Server.h
  - Socket.cpp
  - Socket.h

- Main/ (a series of applications/main-program to operate on gStore)
  - gbuild.cpp (import a RDF dataset)
  - gquery.cpp (query a database)
  - gserver.cpp (start up the gStore server)
  - gclient.cpp (connect to a gStore server and interact)

More details To acquire a deep understanding of gStore codes, please go to Code Detail. See use case to understand the design of use cases, and see OOA and OOD for OOA design and OOD design, respectively.

If you want to know the sequence of a running gStore, please view the list below:

- connect to server
- disconnect server
- load database
- unload database
- create database
- delete database

- connect to database
- disconnect database
- show databases
- SPARQL query
- import RDF dataset
- insert a triple
- delete a triple
- create account
- delete account
- modify account authority
- compulsively unload database
- see account authority

It is really not strange to see something different with the original design in the source code. And some designed functions may have not be achieved so far.

Others The api/ folder in gStore is used to store API program, libs and examples, please go to API for details. And test/ is used to store a series test programs or utilities, such as gtest, full_test and so on. Chapters related with test/ are How To Use and Test Result. This project need an ANTLR lib to parse the SPARQL query, whose code is placed in tools/(also archived here) and the compiled libantlr.a is placed in lib/ directory.

We place some datasets and queries in data/ directory as examples, and you can try them to see how gStore works. Related instructions are in How To Use. The docs/ directory contains all kinds of documents of gStore, including

a series of markdown files and two folders, pdf/ and jpg/. Files whose type is pdf are placed in pdf/ folder, while files with jpg type are placed in jpg/ folder.

You are advised to start from the README in the gStore root directory, and visit other chapters only when needed. At last, you will see all documents from link to link if you are really interested in gStore.

Chapter 09: Publications

Publications related with gStore are listed here:

- Lei Zou, M. Tamer Özsu,Lei Chen, Xuchuan Shen, Ruizhe Huang, Dongyan Zhao, gStore: A Graph-based SPARQL Query Engine, VLDB Journal , 23(4): 565-590, 2014.
- Lei Zou, Jinghui Mo, Lei Chen,M. Tamer Özsu, Dongyan Zhao, gStore: Answering SPARQL Queries Via Subgraph Matching, Proc. VLDB 4(8): 482-493, 2011.
- Xuchuan Shen, Lei Zou, M. Tamer Özsu, Lei Chen, Youhuan Li, Shuo Han, Dongyan Zhao, A Graph-based RDF Triple Store, ICDE 2015: 1508-1511.
- Peng Peng, Lei Zou, M. Tamer Özsu, Lei Chen, Dongyan Zhao: Processing SPARQL queries over distributed RDF graphs. VLDB Journal 25(2): 243-268 (2016).
- Dong Wang, Lei Zou, Yansong Feng, Xuchuan Shen, Jilei Tian, and Dongyan Zhao, S-store: An Engine for Large RDF Graph Integrating Spatial Information, in Proc. 18th International Conference on Database Systems for Advanced Applications (DASFAA), pages 31-47, 2013.
- Dong Wang, Lei Zou and Dongyan Zhao, gst-Store: An Engine for Large RDF Graph Integrating Spatiotemporal Information, in Proc. 17th International Conference on Extending Database Technology (EDBT), pages 652-655, 2014 (demo).
- Lei Zou, Yueguo Chen, A Survey of Large-Scale RDF Data Management, Comunications of CCCF Vol.8(11): 32-43, 2012 (Invited Paper, in Chinese).

Chapter 10: Limitations

1. Queries related with unbounded predicates are not supported.
2. This version only supports SPARQL select query.
3. Only support RDF file in N3 file format. More file formats will be supported in the next version.

Chapter 11: Frequently Asked Questions

When I use the newer gStore system to query the original database, why error?

The database produced by gStore contains several indexes, whose structures may have been changed in the new gStore version. So, please rebuild your dataset just in case.

Why error when I try to write programs based on gStore, just like the Main/gconsole.cpp?

You need to add these phrases at the beginning of your main program, otherwise gStore will not run correctly:
//NOTICE:this is needed to set several debug files
Util util;

Why does gStore report "garbage collection failed" error when I use the Java API?

You need to adjust the parameters of jvm, see url1 and url2 for details.

When I compile the code in ArchLinux, why the error that "no -ltermcap" is reported?

In ArchLinux, you only need to use `-lreadline` to link the readline library. Please remove the `-ltermcap` in the makefile which is located in the root of the gStore project if you would like to use ArchLinux.

Why does gStore report errors that the format of some RDF datasets are not supported?

gStore does not support all RDF formats currently, please see formats for details. However, it is quite easy for you to convey your RDF data format to the N3 file format that is used in gStore.

When I read on GitHub, why are some documents unable to be opened?

Codes, markdowns or other text files, and pictures can be read directly on GitHub. However, if you are using some light weight browsers like midori, for files in pdf type, please download them and read on your computer or other devices.

Why sometimes strange characters appear when I use gStore?

There are some documents's names are in Chinese, and you don't need to worry about it.

In centos7, if the watdiv.db(a generated database after gbuild) is copied or compressed/uncompressed, the size of watdiv.db will be different(generally increasing) if using `du -h` command to check?

It's the change of B+-trees' size in watdiv/kv_store/ that causes the change of the whole database's size. The reason is that in storage/Storage.cpp, many operations use fseek to move file pointer. As everyone knows, file is organized in blocks, and if we request for new block, file pointer may be moved beyond the end of this file(file operations are all achieved by C in gStore, no errors are reported), then contents will be written in the new position!

In **Advanced Programming In The Unix Environment**, "file hole" is used to describe this phenomenon. "file hole" will be filled with 0, and it's also one part of the file. You can use `ls -l` to see the size of file(computing the size of holes), while `du -h` command shows the size of blocks that directory/file occupies in system. Generally, the output of `du -h` is large than that of `ls -l`, but if "file hole" exists, the opposite is the case because the size of holes are neglected.

The actual size of files containing holes are fixed, while in some operation systems, holes will be transformed to contents(also 0) when copied. Operation `mv` will not affect the size if not across different devices.(only need to adjust the file tree index) However, `cp` and all kinds of compress methods need to

scan the file and transfer data.(there are two ways to achieve `cp` command, neglect holes or not, while the output size of `ls -l` not varies)

It is valid to use "file hole" in C, and this is not an error, which means you can go on using gStore. We achieve a small program to describe the "file holes", you can download and try it yourself.

In gclient console, a database is built, queried, and then I quit the console. Next time I enter the console, load the originally imported database, but no output for any queries(originally the output is not empty)?

You need to unload the using database before quiting the gclient console, otherwise errors come.

If query results contain null value, how can I use the full_test utility? Tab separated method will cause problem here because null value cannot be checked!

You may use other programming language(for example, Python) to deal with the null value cases. For example, you can change null value in output to special character like ',', later you can use the full_test utility.

When I compile and run the API examples, it reports the "unable to connect to server" error?

Please use `./gserver` command to start up a gStore server first, and notice that the server ip and port must be matched.

When I use the Java API to write my own program, it reports "not found main class" error?

Please ensure that you include the position of your own program in class path of java. The whole command should be something like `java -cp /home/bookug/project/devGstore/api/java/lib/GstoreJavaAPI.jar:. JavaAPIExample`, and the ":." in this command cannot be neglected.

Chapter 12: Recipe Book

This chapter introduces some useful tricks if you are using gStore to implement applications.

no tips available now

Part III

Others

Chapter 13: Contributors

Please contact with Lei Zou(zoulei@pku.edu.cn), Li Zeng(zengli-bookug@pku.edu.cn), Jiaqi Chen(chenjiaqi93@pku.edu.cn) and Peng Peng(pku09pp@pku.edu.cn) if you have suggestions or comments about gStore or you need help when using gStore.

Faculty

- Lei Zou (Peking University) Project Leader
- M. Tamer Özsu (University of Waterloo)
- Lei Chen (Hong Kong University of Science and Technology)
- Dongyan Zhao (Peking Univeristy)
- Zhiyuan Deng (Wuhan University)

Students

Li Zeng and Jiaqi Chen are responsible for the gStore system optimization. Peng Peng is responsible for the distributed version of gStore, which is expected to be released before October.

- Peng Peng (Peking University) (PhD student)
- Youhuan Li (Peking University) (PhD student)
- Shuo Han (Peking University) (PhD student)

- Li Zeng (Peking University) (Master student)
- Jiaqi Chen (Peking University) (Master student)

Alumni

- Xuchuan Shen (Peking University) (Master's student, graduated)
- Dong Wang (Peking University) (PhD student, graduated)
- Ruizhe Huang (Peking University) (Undergraudate intern, graduated)
- Jinhui Mo (Peking University) (Master's, graduated)

Chapter 14: Updated Logs

Jan 10, 2017

preFilter() function in Join module is optimazed using the pre2num structure, as well as the choose_next_node() function. A global string buffer is added to lower the cost of getFinalResult(), and the time of answering queries is reduced greatly.

In addition, we assign buffers of different size for all B+ trees.(some of them are more important and more frequently used) WangLibo merges several B+ trees into one, and the num of all B+ trees are reduced to 9 from 17. This strategy not only reduces the space cost, but also reduces the memory cost, meanwhile speeding up the build process and query process.

What is more, ChenJiaqi has done a lot of work to optimaze the SPARQL query. For example, some unconnected SPARQL query graphs are dealed specially.

Sep 15, 2016

ZengLi splits the KVstore into 3 parts according to the types of key and value, i.e. int2string, string2int and string2string. In addition, updates are supported now. You can insert, delete or modify some triples in the gStore database. In fact, only insert() and remove() are implemented, while the modify() are supported by removing first and insert again.

Jun 20, 2016

ZengLi has enabled the gStore to answer queries with predicate variables. In addition, the structures of many queries have been studied to speed up the query processing. ChenJiaqi rewrites the sparql query plan to acquire a more efficient one, which brings many benefits to us.

Apr 01, 2016

The structure of this project has changed a lot now. A new join method has been achieved and we use it to replace the old one. The test result shows that speed is improved and the memory cost is lower. We also do some change to Parser/Sparql*, which are all generated by ANTLR. They must be modified because the code is in C, which brings several multiple definition problems, and its size is too large.

There is a bug in the original Stream module, which brings some control characters to the output, such as ^C, ^V and so on. We have fixed it now and enabled the Stream to sort the output strings(both internal and external). In addition, SPARQL queries which are not BGP(Basic Graph Pattern) are also supported now, using the naive method.

A powerful interactive console, which is named `gconsole` now, is achieved to bring convenience to users. What is more, we use valgrind tools to test our system, and deal with several memory leaks.

The docs and API have also changed, but this is of little importance.

Nov 06, 2015

We merge several classes(like Bstr) and adjust the project structure, as well as the debug system.

In addition, most warnings are removed, except for warnings in Parser module, which is due to the use of ANTLR.

What is more, we change RangeValue module to Stream, and add Stream for ResultSet. We also better the gquery console, so now you can redirect query results to a specified file in the gsql console.

Unable to add Stream for IDlist due to complex operations, but this is not necessary. Realpath is used to supported soft links in the gquery console, but it not works in Gstore.(though works if not in Gstore)

Oct 20, 2015

We add a gtest tool for utility, you can use it to query several datasets with their own queries.

In addition, gquery console is improved. Readline lib is used for input instead of fgets, and the gquery console can support commands history, modifying command and commands completion now.

What is more, we found and fix a bug in Database/(a pointer for debugging log is not set to NULL after fclose operation, so if you close one database and open another, the system will fail entirely because the system think that the debugging log is still open)

Sep 25, 2015

We implement the version of B+Tree, and replace the old one.

After testing on DBpedia, LUBM, and WatDiv benchmark, we conclude that the new BTree performs more efficient than the old version. For the same triple file, the new version spends shorter time on executing gload command.

Besides, the new version can handle the long literal objects efficiently, while triples whose object's length exceeds 4096 bytes result in frequent inefficient split operations on the old version BTree.

Feb 2, 2015

We modify the RDF parser and SPARQL parser.

Under the new RDF parser, we also redesign the encode strategy, which reduces RDF file scanning times.

Now we can parse the standard SPARQL v1.1 grammar correctly, and can support basic graph pattern(BGP) SPARQL queries written by this standard grammar.

Dec 11, 2014

We add API for C/CPP and JAVA.

Nov 20, 2014

We share our gStore2.0 code as an open-source project under BSD license on github.

Chapter 15: Test Result

Preparation

We have compared the performance of gStore with several other database systems, such as Jena, Sesame, Virtuoso and so on. Contents to be compared are the time to build database, the size of the built database, the time to answer single SPARQL query and the matching case of single query's results. In addition, if the memory cost is very large(>20G), we will record the memory cost when running these database systems.(not accurate, just for your reference)

To ensure all database systems can run correctly on all datasets and queries, the format of datasets must be supported by all database systems and the queries should not contain update operations, aggregate operations and operations related with uncertain predicates. Notice that when measuring the time to answer queries, the time of loading database index should not be included. To ensure this principle, we load the database index first for some database systems, and warm up several times for others.

Datasets used here are WatDiv, Lubm, Bsbm and DBpedia. Some of them are provided by websites, and others are generated by algorithms. Queries are generated by algorithms or written by us. Table 2 summarizes the statistics of these datasets.

The experiment environment is a CentOS server, whose memory size is 82G and disk size is 7T. We use full_test to do this test.

Result

The performance of different database management systems is shown in Figures 1, 2, 3 and 4.

Notice that Sesame and Virtuoso are unable to operate on DBpedia 2014

and WatDiv 300M, because the size is too large. In addition, we do not use Sesame and Virtuoso to test on the LUBM 5000 due to format questions. Generally speaking, Virtuoso is not scalable, and Sesame is so weak.

This program produces many logs placed in result.log/, load.log/ and time.log/. You can see that all results of all queries are matched by viewing files in result.log/, and the time cost and space cost of gStore to build database are larger than others by viewing files in load.log/. More precisely, there is an order of magnitude difference between gStore and others in the time/space cost of building database.

Through analysing time.log/, we can find that gStore behave better than others on very complicated queries(many variables, circles, etc). For other simple queries, there is not much difference between the time of these database systems.

Generally speaking, the memory cost of gStore when answering queries is higher than others. More complicated the query is and more large the dataset is, more apparent the phenomenon is.

You can find more detailed information in original test report. Notice that some questions in the test report have already be solved now. The latest test report is formal experiment.

Dataset	Number of Triples	RDF N3 File Size(B)	Number of Entities
WatDiv 300M	329,539,576	47,670,221,085	15,636,385
LUBM 5000	66718642	8134671485	16437950
DBpedia 2014	170784508	23844158944	7123915
Bsbm 10000	34872182	912646084	526590

Table 2: Datasets

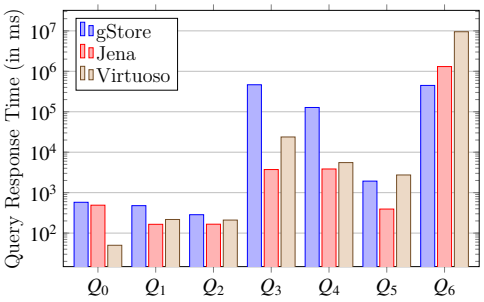


Figure 1: Query Performance over DBpedia 2014

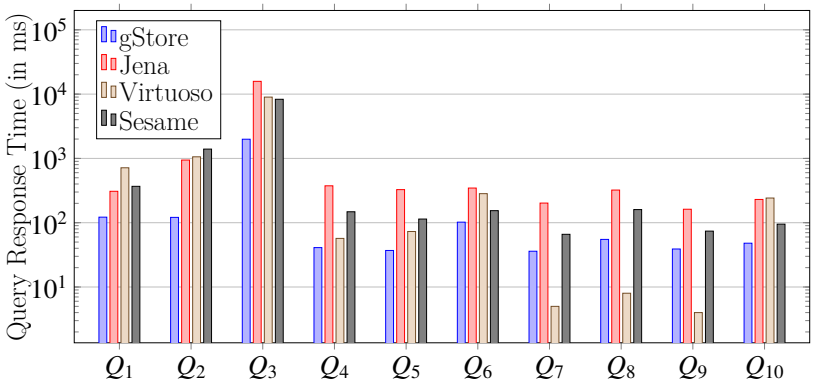
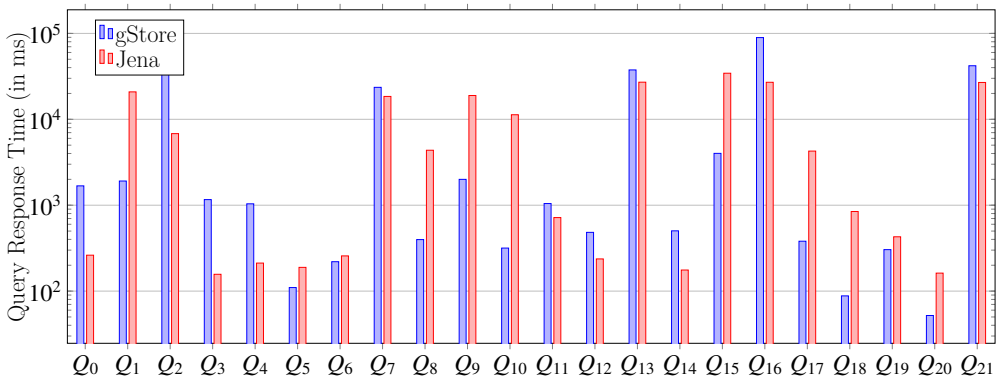
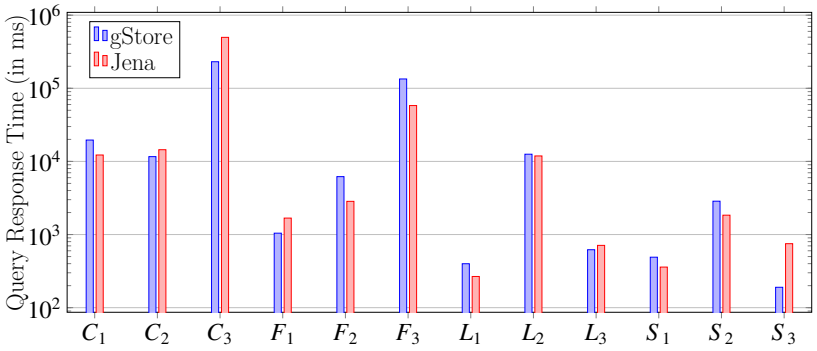


Figure 2: Query Performance over Bsbm 10000



(a) LUBM 5000

Figure 3: Query Performance over LUBM



(a) WatDiv 300M

Figure 4: Query Performance over WatDiv

Chapter 16: Future Plan

Improve The Core

- optimize the join operation of node candidates. multiple methods should be achieved, and design a score module to select a best one
- add numeric value query function. need to answer numeric range query efficiently and space consume cannot be too large
- add a control module to heuristically select an kind of index for a SPARQL query to filter(not always vstree)
- typedef all frequently used types, to avoid inconsistence and high modify cost

Better The Interface

- build a console named gconsole, which provides all operations supported by gStore.(parser and auto-complete is required)
- write web interface for gStore, and a web page to operate on it, just like virtuoso

Idea Collection Box

- to support soft links in console: realpath not work…(redefined in ANTLR?)
- store command history for consoles
- warnings remain in using Parser/(antlr)!(modify sparql.g 1.1 and regenerate). change name to avoid redefine problem, or go to use executable to parse

- build compress module(such as key-value module and stream module), but the latter just needs one-pass read/write, which may causes the compress method to be used both in disk and memory. all operations of string in memory can be changed to operations after compress: provide compress/archive interface, compare function. there are many compress algorithms to be chosen, then how to choose? what about utf-8 encoding problem? this method can lower the consume of memory and disk, but consumes more CPU. However, the time is decided by isomorphism. Simple compress is not good, but too complicated method will consume too much time, how to balance? (merge the continuous same characters, Huffman tree)

- mmap to speedup KVstore?

- the strategy for Stream:is 85% valid? consider sampling, analyse the size of result set and decide strategy? how to support order by: sort in memory if not put in file; otherwise, partial sort in memory, then put into file, then proceed external sorting

Chapter 17: Thanks List

This chapter lists people who inspire us or contribute to this project.

GitHub user zhangxiaoyang
https://github.com/zhangxiaoyang
1. add python api
2. fix logger message

Chapter 18: Legal Issues

What's more, you need to include the label "powered by gStore", as well as the logo of gStore, in your software product which is using gStore.

We would be very grateful if you are willing to tell us about your name, institution, purpose and email. Such information can be sent to us by emailing to gStoreDB@gmail.com, and we promise not to reveal privacy.

End

Thank you for reading this document. If any question or advice, or you have interests in this project, please don't hesitate to get in touch with us.