gStore 系统使用手册

v0.7.0

由 gStore 团队编写 [1]

2018 年 11 月 5 日

[1] 邮箱列表在第 13 章中给出。

目　录

1 前言

RDF（*R*esource *D*escription *F*ramework，资源描述框架）是由 W3C 提出的一组标记语言的技术规范，用来表现万维网上各类资源的信息并发展语义网络。在 RDF 模型中，每个网络对象都由一个唯一命名的资源来表示，用一个 URI（*U*niform *R*esource *I*dentifier ，统一资源标识符）来标识。RDF 也利用 URI 去命名资源的属性和资源间的关系，以及关系的两端（通常被称为“三元组”）。因此，一个 RDF 数据集可以由一个有向、有标签的图来表示，其中资源是顶点，三元组是标签为属性或关系的边。更多的细节请参阅 RDF 介绍。

为了检索并操控一个 RDF 图，W3C 提供了一种结构化的查询语言，SPARQL (*S*imple *P*rotocol *A*nd *R*DF *Q*uery *L*anguage ，简单协议和 RDF 查询语言)。SPARQL 能够依据连接或分离关系，查询指定图模式和可选图模式。SPARQL 同时支持聚集函数、子查询、否定查询、根据表达式创造值、可扩展的值检验、根据源 RDF 的限制性查询。与 RDF 图类似，SPARQL 查询可以表示为有若干变量的查询图。这样一来，回答一个 SPARQL 查询就等价于在一个 RDF 图中找到一个匹配查询的子图。通过 SPARQL 介绍了解有关 SPARQL 的更多信息。

虽然有一些 RDF 数据管理系统（例如 Jena 、Virtuoso 、Sesame ）在关系系统中储存 RDF 数据，但现有的系统几乎都没有开发符合 SPARQL 语义的图模式。**在这里我们完善了基于图的 RDF 三元组存储，称为 gStore ，是北京大学、滑铁卢大学、香港科技大学的联合研究项目。中国北京大学计算机科学与技术研究所的数据管理实验室对该系统进行开发和维护。**对于 gStore 的详细描述可以在【出版物】一章我们的论文 [Zou et al., VLDB 11] 和 [Zou et al., VLDB Journal 14] 中找到。这份帮助文档包括系统安装、使用、API 、用例和 FAQ 。gStore 是 GitHub 上遵循 BSD 协议的开源项目。你可以使用 gStore 、报告问题、提出建议，或加入我们使 gStore 变得更好。你也可以在尊重我们的工作的前提下基于 gStore 开发各种应用。

请确保在使用 gStore 之前已经阅读了【法律问题】一章。

2 开始

2.1 第 00 章: 快速导览

Gstore 系统（也称作 gStore ）是一个用于管理大型图结构数据的图数据库引擎，是一个针对 Linux 操作系统的开源软件。整个项目用 C++ 编写，使用了一些库，例如 readline 、antlr 等等。

2.1.1 开始使用

从源码编译 本系统接口对用户非常友好，你可以在几分钟内学会使用。请在【系统要求】一章中检查你想要运行这一系统的平台。在确认后获取项目的源码。有以下几种方法：

- （建议）在终端输入 `git clone https://github.com/pkumod/gStore.git` 或使用 git GUI 获得
- 在这个库中下载 zip 文件并进行解压
- 使用你的 github 账号 Fork 这个库

之后你需要对这个项目进行编译，如果是第一次使用，你需要输入 `make pre` 准备 `ANTLR` 库和语法解析程序。以后就不需要再输入这一命令了。在 gStore 根目录下输入 `make` ，所有可执行程序都将自动生成。（如果想要更快地编辑，输入 `make -j4` ，使用多少线程取决于机器。）要检查程序的正确性，请输入 `make test` 命令。

我们建议你下载源码，这样就可以在 gStore 根目录下输入 `git pull` 来获取更新，或者输入 `git log` 检查当前版本、查看提交日志。如果你想使用其他分支下的代码，例如“dev”分支：

- 克隆 master 分支，并在终端输入 `git checkout dev`
- 直接克隆 dev 分支，输入 `git clone -b dev`

通过 Docker 部署 你也可以通过 Docker 部署 gStore 。我们提供了 Docker 文件和镜像。请参见 Docker 部署。

运行 要运行 gStore，请输入 `bin/gbuild database_name dataset_path` 生成一个你自己命名的数据库。你可以用 `bin/gquery database_name` 这一命令查询一个已存在的数据库。此外，`bin/ghttp` 是一个非常好的工具，可用通过 HTTP 协议连接数据库。

请注意，所有的命令都应该在 gStore 根目录下输入。

你可以在本文档的【如何使用】一章中找到详细描述。

2.1.2 高级帮助

如果你希望理解 gStore 系统的细节，或是尝试一些高级操作（例如，使用 API 、服务器/客户端），请参阅以下章节。

- 【基本介绍】：介绍 gStore 的原理和特征
- 【安装指南】：安装系统的指令
- 【如何使用】：使用 gStore 系统的详细指导
- 【HTTP API 说明】：基于 gStore HTTP API 开发应用
- 【socket API 说明】：基于 gStore socket API 开发应用
- 【项目结构】：展现本项目的结构和流程
- 【出版物】：与 gStore 相关的论文和出版物
- 【更新日志】：保存了系统更新的日志
- 【测试结果】：展现一系列的实验结果

2.1.3 其他事项

在【技巧】一章中，我们撰写了一系列短文，解决使用 gStore 来实现应用时可能出现的问题。

如果不需要及时回复，你可以在这个库的 Issues 部分报告建议或错误。如果你急于联系我们处理你的报告,请发送电子邮件到 bookug@qq.com 或 gStoreDB@gmail.com 提交你的建议和错误报告。我们团队的完整列表在【贡献者】一章中给出。

使用现有的 gStore 系统有一些限制，你可以在【限制】一章中看到。

有时候你可能会发现一些奇怪的现象（但不是错误案例），或者很难理解/解决（不知道接下来怎么做），可以参阅【FAQ】。

图数据库引擎是一个新的领域，我们还在努力发展。我们接下来要做的事在【将来计划】一章中列出，我们希望越来越多的人可以支持甚至加入我们。你可以通过很多方法支持我们：

- watch/star 我们的项目
- fork 这个库，向我们提交 pull 请求
- 下载并使用这一系统，报告错误或建议
- ...

启发我们或对这个项目做出贡献的人会在【致谢列表】中列出。

2.2 第 01 章：系统要求

我们已经在 linux CentOs 6.2 x86_64 和 CentOS 6.6 x86_64 系统做了测试。GCC 版本应该为 4.47 或更高。

项目	要求
操作系统	Linux , 例如 CentOS , Ubuntu 等等
架构	x86_64
磁盘容量	取决于数据集大小
内存空间	取决于数据集大小
glibc	版本 >= 2.14
gcc	版本 >= 4.8
g++	版本 >= 4.8
make	需要安装
boost	版本 >= 1.54
readline	需要安装
readline-devel	需要安装
openjdk	使用 Java api 时需要
openjdk-devel	使用 Java api 时需要
requests	使用 Python http api 时需要
pthreads	使用 php http api 时需要
curl-devel	使用 php http api 时需要
node	使用 Nodejs api 时需要，版本 >= 10.9.0
realpath	使用 gconsole 时需要
ccache	可选，可以加速编译过程
libcurl-devel	需要安装

表 1: 软件要求

注意事项：

为了方便环境设置，gstore/scripts/setup 文件夹下为不同的 Linux 发行版本提供了脚本。请在 root （或 sudo ）权限下根据你的系统选择相应的安装脚本。（对于 CentOS 系统，你需要自己安装 boost-devel 。）

1. 一些包的名字可能在不同平台上有所不同，只需要安装你自己的操作平台所对应的包

2. 要安装 readline 和 readline-devel ，只需要在 Redhat/CentOS/Fedora 中输入 `dnf install readline-devel` ，或者在 Debian/Ubuntu 中输入 `apt-get install libreadline-dev` 。请在其他系统中使用对应的指令。如果你使用的是 ArchLinux ，只要输入 `pacman -S readline` 就可以安装 readline 和 readline-devel 。（其他包也一样）

3. 使用 gStore 不需要安装 realpath ，但如果你想要使用 gconsole ，请输入 `dnf install realpath` 或 `apt-get install realpath` 进行安装。

4. 我们的项目使用了正则表达式，由 GNU/Linux 默认提供。

5. gStore 使用了 ANTLR3.4 生成 SPARQL 查询的语法分析代码。你不需要安装相应的 antlr 库，因为我们已经将 libantlr3.4 融入系统中。

6. 当你在 gStore 项目的根目录下输入 `make` 时，Java api 也会编译。如果你的系统里没有 JDK ，你可以修改 makefile 。我们建议你在 Linux 系统中安装 openjdk-devel 。

7. 在 CentOS 系统上你需要添加 epel 源才能安装 ccache ，但在 Ubuntu 系统上可以直接用 `apt-get install ccache` 命令安装。如果你无法安装 ccache（或者不想安装），请修改 makefile 文件（只需要将 CC 变量改为 g++ 即可）。

8. 如果要使用 gStore 的 HTTP 服务，则必须安装 boost 开发库（比如 boost-devel ，包括用于开发的 boost 头文件），且版本不能低于 1.54 。注意检查 makefile 中 boost 库的安装位置。要使用 Python 接口，在 CentOS 中，你需要输入 `pip install requests` 来安装 requests 。要使用 php 接口，你需要使用如下命令安装 pthreads 和 curl ：

```
1- 安装 curl-devel
# yum install curl-devel

2- 下载 php
# wget -c http://www.php.net/distributions/php-5.4.
36.tar.gz

下载 pthreads
# wget -c http://pecl.php.net/get/pthreads-1.0.0.tgz

3- 解压
# tar zxvf php-5.4.36.tar.gz
# tar zxvf pthreads-1.0.0.tgz

4- 将 pthreads 移动到 php/ext 文件夹
# mv pthreads-1.0.0 php-5.4.36/ext/pthreads
```

5- 重新配置

```
# ./buildconf --force
# ./configure --help | grep pthreads
```

你应该能看到上述命令列出了 --enable-pthreads 。
如果没有，用这个命令清除生成文件：

```
# rm -rf aclocal.m4
# rm -rf autom4te.cache/
# ./buildconf --force
```

6- 在 php 文件夹中运行配置命令

```
# ./configure --enable-debug --enable-maintainer-zts
--enable-pthreads --prefix=/usr
--with-config-file-path=/etc --with-curl
```

7- 安装 php
运行 make clear 以确保没有其他生成文件干扰

```
# make clear
# make
# make install
```

8- 复制 PHP 的配置文件，将本地库添加到 include 路径

```
# cp php.ini-development /etc/php.ini
```

编辑 php.ini ， Include_path 如下设置：

```
Include_path = "/usr/local/lib/php"
```

9- 安装 node

```
# wget https://npm.taobao.org/mirrors/node/v10.9.0/node-v10.9.0.tar.gz
# tar -xvf node-v10.9.0.tar.gz
# cd node-v10.9.0
# ./configure
```

```
# make
# sudo make install
```

10- 检查模块

```
# php -m (check pthread loaded)
```

你应该看到上述命令列出了 pthread

11- 如果没有列出 pthread ，更新 php.ini

```
# echo "extension=pthreads.so" >> /etc/php.ini
```

9. 其他问题请参阅【FAQ】一章。

2.3 第 02 章：基本介绍

与 *Gstore* 系统相关的第一篇论文是 *gStore_VLDBJ*，你可以在【出版物】一章中找到相关出版物。

2.3.1 什么是 gStore

gStore 是一个基于图的 RDF 数据管理系统（也称为“三元组存储”），维持了原始 RDF 数据的图结构。它的数据模型是有标签的有向多边图，每个顶点对应一个主体或客体。

我们用查询图 Q 来表示给出的 SPARQL 。查询过程涉及查找在 RDF 图 G 中与 Q 匹配的子图，而不是在关系型数据库中将表连接到一起。gStore 包含一个 RDF 图的指针（称为 VS 树）来加快查询过程。VS 树是一个深度平衡树，使用了大量裁减算法加快子图匹配。

gStore 项目获得中国国家自然科学基金（NSFC ）、加拿大自然科学和工程研究委员会（NSERC ）和香港 RGC 支持。

2.3.2 为什么选择 gStore

在一系列测试后，我们进行了分析并将结果记录在【测试结果】一章中。gStore 在回答复杂查询时（例如，包含循环）比其他数据库系统运行更快。对于简单查询，gStore 和其他数据库系统都运行得很好。

另外，当今是大数据时代，出现了越来越多的结构化数据，原来的关系型数据库系统（或是基于关系表的数据库系统）不能高效地处理结构化数据。相反，gStore 可以利用图数据结构的特征并提升性能。

此外，gStore 是一个高扩展性项目。很多关于图数据库的新想法被提出，大多数都可以在 gStore 中使用。例如，我们组也在设计一个分布型 gstore 系统。

2.3.3 开源与授权

gStore 的源代码遵循 BSD 开源协议。你可以使用 gStore 、报告建议或问题，或者加入我们使 gStore 变得更好。在尊重我们的工作的前提下，你也可以基于 gStore 开发各种应用。

2.4 第 03 章：安装指南

2.4.1 从源码安装

用户应该详细阅读 init.conf 文件，并根据自己的实际情况修改它。（这个文件包含 gStore 系统的基本配置）

gStore 是一个绿色软件，你只需要用三个指令对它进行编译。请在 gStore 根目录下运行

```
$ sudo ./scripts/setup/setup_$(ARCH).sh
$ make pre
$ make
```

来准备依赖库，连接 ANTLR 库，编译 gStore 代码，并生成可执行的 “gbuild” 、“gquery” 、“ghttp” 、“gserver” 、“gclient” 。另外，gStore 的 api 也在此时生成。

安装脚本和依赖库的准备只需要执行一次，之后可以直接输入 `make` 编译代码。

（如果想要更快地编译，输入 `make -j4` ，使用多少线程取决于机器。）

如果输入了 `make dist` 命令，编译时需要再次执行 `make pre`。

如果你想使用 gStore 的 API 样例，请运行 `make APIexample` 编译 C++ API 和 Java API 的样例代码。关于 API 的更多细节，请参阅【API】一章。

使用 `make clean` 指令清除所有对象、可执行程序，使用 `make dist` 指令清除 gStore 根目录下的所有对象、可执行程序、库、数据集、数据库、调试日志和临时/文本文件。

你可以自由修改 gStore 的源代码，在尊重我们工作的前提下开发自己的项目，输入 `make tarball` 指令将所有有用的文件压缩成.tar.gz 文件以便于传输。

如果你想使用测试工具，输入 `make gtest` 编译 gtest 程序。你可以在【如何使用】一章中看到关于 gtest 程序的更多细节。

2.4.2 Docker 部署

简单来说，我们提供两种方式通过容器部署 gStore ：一种是通过项目根目录的 Dockerfile 文件自主构建，然后运行容器；另一种是直接下载已经自动构建完成的镜像，然后直接运行.

环境准备 关于安装使用 Docker ，官方针对常见 Linux 发行版文档已经写得很详细，在此直接给出参考地址：Docker 文档.

需要注意的是，Docker 版本过高可能导致一些问题，建议仔细阅读注意事项。当前测试环境版本是 Docker CE 17.06.1 。

通过 Dockerfile 构建镜像 假设已经拥有正常的 Docker 环境跟网络后，首先通过 `git clone` 下载项目，然后进入项目根目录，输入命令 `docker build -t gstore` ，即可开始构建，默认使用根目录的 Dockerfile ，Dockerfile 文件内有更详细的说明。

构建完成后，直接通过 `docker run -it gstore` 即可进入容器执行其他操作。

直接拉取镜像运行 无需下载项目或自己构建,直接输入 `docker pull suxunbin/auto_gstore:latest` 拉取已经在 docker hub 上自动构建完成的镜像。拉取完成后 `docker run -it suxunbin/auto_gstore:latest` 即可直接进入容器使用。

存在的问题 因为容器化伴随着一些不确定的影响，包括不限于网络，锁，缓存，权限等问题，调试起来很难定位，已知可能存在以下问题：(宿主机是 CentOS 7.4)

1. 在 Dockerfile 的构建过程中添加环境 `ENV CC="ccache g++"` 会导致编译错误，原因未知，且可能影响镜像缓存层，导致之后构建反复报错。
2. 通过 docker run 启动容器后，可能出现启动后即自动关闭的情况。

后续工作

性能测试 待后续补充，容器化跟原生运行 gStore，在不同文件数/网络等情况下，容器的性能损耗具体比重。

连接其他容器测试 待后续补充。

构建精简优化 因为 gcc:8 镜像占用了 1.7G ，自带很多不必要的东西（例如 go 的环境），希望后续除了在调低 gcc 版本之外，能对源镜像本身改进出精简版。

2.5 第04章：如何使用

gStore 目前包含五个可执行程序和其他文件。

gStore 的所有指令都应该在 gStore 根目录下使用，例如 bin/ghttp 。因为所有的可执行程序都在 bin/ 中，它们可能使用了一些文件，其路径在代码中声明，但不是绝对路径。我们之后会让使用者给出他们系统中安装/配置 gStore 的绝对路径，以确保所有的路径都是绝对的。然而，现在你必须这么做以避免错误。

0. 数据格式 RDF 数据应该以 N-Triple 格式给出（目前不支持 XML ），查询必须符合 SPARQL 1.1 语法。并非所有 SPARQL 1.1 中的语法都可以在 gStore 中解析，例如，gStore 不支持实体属性查询，数据和查询的实体、字面量或谓词中不允许出现制表符、'<' 和'>' 。

1. gbuild gbuild 用于由 RDF 三元格式文件 build 一个新的数据库。

```
bin/gbuild db_name rdf_triple_file_path
```

在上述命令的参数中,db_name 是你设置的数据库名称,rdf_triple_file_path 是数据文件所在的路径。例如，我们从 data/lubm 文件夹下的 lumb.nt 文件 build 数据库。

```
[bookug@localhost gStore]$ bin/gbuild lumb ./data/lubm/lubm.nt
gbuild...
argc: 3 DB_store:lubm      RDF_data: ./lubm.nt
begin encode RDF from : ./data/lubm/lubm.nt ...
```

2. gquery gquery 用包含 SPARQL 的文件查询一个已有的数据库（每个文件包含一条 SPARQL 查询）。

输入 `bin/gquery db_name query_file` 在名为 db_name 的数据库中用 query_file 中的语句执行 SPARQL 查询。

使用 `bin/gquery --help` 获取关于 gquery 用法的详细信息。

输入`bin/gquery db_name` 进入 gquery 控制台。程序会给出一个命令提示符（“gsql>”），你可以在此处输入命令。使用 `help` 查看所有指令的基本信息，`help command_t` 给出特定指令的详细信息。

输入 `quit` 离开 gquery 控制台。

对于 `sparql` 指令，输入包含单个 SPARQL 查询的文件路径。(支持将结果重新定向到文件。)

程序完成查询时，会再次显示命令提示符。

我们还是以 lubm.nt 为例。

```
[bookug@localhost gStore]$ bin/gquery lubm
gquery...
argc: 2 DB_store:lubm/
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/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>
```

注意：

- 如果没有答案，会输出 “[empty result]”，在所有结果后面会有一个空行。
- 我们使用了 readline 库，你可以用键盘上的方向键查看历史指令、移动或修改整个命令。
- 支持路径补全（不是内嵌命令补全）。

3. ghttp ghttp 以类似 HTTP 服务器的方式运行 gStore （你需要指定一个端口，并在你的环境中打开这个端口，建议使用 `iptables` 工具）。从浏览器访问特定的 url ，gStore 就能执行相应的操作。

输入 `bin/ghttp db_name serverPort` 或者 `bin/ghttp serverPort db_name` 在 serverPort 端口上启动服务，并 load 名为 db_name 的数据库。

注意：参数 serverPort 或 db_name 可以省略。如果省略了 serverPort，其值会被默认设置为 9000 。如果省略了 db_name，服务会以没有 load 数据库的形式启动。

操作: build, load, unload, query, monitor, show, checkpoint, checkall, user, drop

```
// build a new database by a RDF file.
gc.build("test", "data/lubm/lubm.nt", "root", "123456");

// drop a database already built but leave a backup.
gc.drop("test", "root", "123456");

// drop a database already built completely.
gc.drop_r("test", "root", "123456");

// load database
gc.load("test", "root", "123456");

// then you can execute SPARQL query on this database.
answer = gc.query("root", "123456", "test", sparql);

// output information of a database
cout << answer << std::endl;

// unload this database
```

```
gc.unload("lubm", "root", "123456");

// show all databases already built and if they are loaded
gc.show();

// show statistical information of a loaded database
gc.monitor("lubm");

//add a user(with username: Jack, password: 2)
answer = gc.user("add_user", "root", "123456", "Jack", "2");

//add privilege to user Jack(add_query, add_load, add_unload)
answer = gc.user("add_query", "root", "123456", "Jack", "lubm");

//delete privilege of a user Jack(delete_query, delete_load, delete_unload)
answer = gc.user("delete_query", "root", "123456", "Jack", "lubm");

//delete user(with username: Jack, password: 2)
answer = gc.user("delete_user", "root", "123456", "Jack", "2");
```

参数：

db_name ： 数据库名称，例如 lubm

format ： html, json, txt, csv

sparql ： select ?s where ?s ?p ?o .

ds_path in the server ： 例如 /home/data/test.n3

operation ： 操作类型，例如 load, unload, query ...

type: 针对 user 的操作类型,例如 add_user, delete_user, add_query, add_load...

username ： 执行操作的用户名

password ： 执行操作的用户密码

为了统一响应格式，同时让用户更有效地处理响应信息，我们把 ghttp.cpp 中的响应格式都改成了 JSON 。

用户发送的请求会收到 JSON 格式的响应，包括 StatusCode （一个代表响

应状态的数字），StatusMsg （一个描述服务器信息的字符串）和 ResponseBody（响应的主体部分，如果是一个 query 请求，主体部分就是查询结果，有时主体部分可以为空）。

我们在下表中定义了一些 StatusCode 和对应的信息。用户向 ghttp 发送请求后，他可以解析服务器返回的 JSON 获取 StatusCode ，StatusMsg 和 ResponseBody ，然后根据需求利用这些信息。如果你不明白某一 StatusCode 的具体含义，请查阅下表。

StatusCode	StatusMsg	Description
0	”success”	”operation success”
0	”file delete successfully”	
101	”could not open path”	”could not open file”
0	”import RDF file to database done”	
201	”database already built”	
202	”your db name to be built should not end with ”.db””	
203	”database not built yet”	
204	”import RDF file to database failed”	
0	”database loaded successfully”	
301	”database already load”	
302	”no load privilege, operation failed”	
303	”unable to load due to loss of lock”	
304	”database not load yet”	
305	”failed to load the database”	
401	”empty SPARQL”	
402	”update query returns true”	
403	”search query returns false”	
404	”no query privilege, operation failed”	
501	”unable to monitor due to loss of lock”	

StatusCode	StatusMsg	Description
0	”database unloaded”	
601	”no unload privilege, operation failed”	
602	”unable to unload due to loss of lock”	
0	”database saved successfully”	
801	”no database”	
802	”no users”	
0	”check identity successfully”	
901	”wrong username”	
902	”wrong password”	
903	”username not found”	
904	”exactly 1 argument is required”	
905	”exactly 2 argument is required”	
0	”operation on users succeeded”	”add, delete users or modify the privilege of a user, operation succeeded”
907	”username already existed, add user failed”	
908	”you cannot delete root, delete user failed”	
909	”username not exist, delete user failed”	
910	”you can’t add privilege to root user”	
911	”add privilege failed”	
912	”add query or load or unload privilege failed”	
913	”you can’t delete privilege of root user”	
914	”delete privilege failed”	

StatusCode	StatusMsg	Description
915	”not root user, no privilege to perform this operation”	
916	”username not exist, change password failed”	

表 2: StatusCode Definition

ghttp 支持并发只读查询，但当查询中包含更新操作时，整个数据库将被上锁。虽然我们在实验中可以运行 13000 个并发查询，但是在几十个内核线程的机器上，我们还是建议并发数低于 300 。想要使用 gStore 的并发特性，最好将系统设置中的'open files' 和'maximum processes' 设置为 65535 或更大值。setup 文件夹下有三个脚本，可以用来在不同的 Linux 发行版本中修改设置。

如果通过 ghttp 发送了包含更新操作的查询，在关闭数据库服务器前，必须在 ghttp 控制台中输入并完成 `checkpoint` 命令。否则，更新可能不能同步到磁盘，会在 ghttp 服务器关闭时丢失。

注意：你无法通过 `Ctrl+C` 停止 ghttp ，这会导致数据库的变更丢失。要停止服务，请输入 `bin/shutdown serverPort` 。

4. gserver　这一功能暂时停止维护。

gserver 是一个后台程序。在使用 gclient 或 socket API 连接 gStore 之前，应该启动 gserver 。它通过套接字与客户端通信。

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

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

你也可以为监听分配一个指定端口。

```
[bookug@localhost gStore]$ bin/gserver -s -p 3307
Server started at port 3307.
```

注意：gserver 不支持多线程。如果你同时在多个终端启动 gclient ，gserver 会崩溃。

5. gclient　这一功能暂时停止维护。

gclient 是用于发送命令和接收反馈的客户端。

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

gsql>quit
```

你也可以分配 gserver 的 ip 和端口。

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

我们现在可以使用以下命令：

- `help` 显示所有指令的信息
- `import db_name rdf_triple_file_name` 从一个 RDF 三元组文件 build 数据库
- `load db_name` load 一个已存在的数据库
- `unload db_name` unload 一个数据库，但不会从磁盘上删除它，你可以再次 load
- `sparql "query_string"` 用一个 SPARQL 查询字符串（在“”内）查询当前数据库
- `show` 显示当前数据库的名称

注意：

- 在 gclient 控制台最多只能载入一个数据库
- 你可以在指令的不同部分之间加上 ‘ ’ 或 ‘\t’，但不要使用 ‘;’ 之类的字符
- 在指令前不能有空格或制表符

6. gconsole gconsole 是 gStore 的主要控制台，与其他函数和一些系统指令整合对 gStore 进行操作。提供了完整的命令名称、命令行编辑特征、可以获取历史命令。尝试 gconsole 将是一次奇妙之旅！（空格或制表符可以在开头或结尾使用，不需要输入任何特殊字符作为分隔符）

在 gStore 根目录下输入 `bin/gconsole` 来使用控制台，你会看到 `gstore>` 提示符，意味着你处于本机模式并可以输入本机命令。控制台还有另一种模式，称为远程模式。在本机模式下输入 `connect` 进入远程模式，输入 `disconnect` 退回到本机模式。（控制台连接到 gStore 服务器，其 ip 为'127.0.0.1'，端口号为 3305，你可以输入 `connect gStore_server_ip gStore_server_port` 来指定它们。）

你可以在本机模式或远程模式中用 `help` 或 `?` 查看帮助信息，你也可以输入 `help command_name` 或 `? command_name` 查看某一指令的信息。请注意，本机模式和远程模式的指令有一些区别。例如，`ls` ，`cd` 和 `pwd` 这样的系统指令在本机模式中提供，但不在远程模式中提供。也请注意，帮助页中的一些指令还没有完全实现，将来我们可能会改变控制台的一些函数。

我们已经完成的工作足以让你便捷地使用 gStore ，尽情享受吧！

7. gadd gadd 向数据库中插入一个文件中的三元组。

用法：`bin/gadd db_name rdf_triple_file_path`

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

8. gsub gsub 从数据库中删除某一文件中的三元组。

用法：`bin/gsub db_name rdf_triple_file_path`

```
[bookug@localhost gStore]$ bin/gsub lubm ./data/lubm/lubm.nt
...
argc: 3 DB_store:lubm  remove file: ./data/lubm/lubm.nt
...
remove rdf triples done.
```

```
removed triples num: 99550
```

9. gmonitor 启动 ghttp 后，输入 `bin/gmonitor ip port` 查看服务器信息。

```
[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 启动 ghttp 后，输入 `bin/gshow ip port` 查看当前加载的数据库。

```
[bookug@localhost gStore]$ bin/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
```

11. shutdown 启动 ghttp 之后，你可以输入 `bin/shutdown port` 来停止服务。

12. ginit 如果想要恢复 ghttp 服务器的初始配置，输入 `bin/ginit` 来重建 system.db 。

13. 测试工具 test/ 文件夹下有一系列测试程序，我们会介绍两个比较有用的：gtest.cpp 和 full_test.sh **gtest 用多个数据集和查询测试 gStore** 。

要使用 gtest ，请先输入 `make gtest` 编译 gtest 程序。gtest 程序为数据集生产结构日志。请在工作目录下输入 `./gtest --help` 获取更多信息。

如果需要请改变 test/gtest.cpp 中的路径。

你应该如下设置数据集和查询：

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

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

请注意，DIR 是你要用于 gtest 的所有数据集的根目录，WatDiv 和 lubm 一样，是数据集类。在 WatDiv 内，请将所有的数据集（用.nt 命名）放在 database/ 文件夹下，并将所有查询（和数据集对应，用.sql 命名）放在 query 文件夹下。

之后你可以用指定的参数运行 gtest 程序，输出会被分类并储存到 gStore 根目录下的三个日志内：load.log/ （数据库加载时间和大小），time.log/ （查询时间）和 result.log/ （所有查询结果，不是整个结束字符串，而是记录选定的两个数据库系统是否匹配的信息。）

程序产生的所有日志都以 TSV 格式储存（用 '\t' 分隔），你可以直接将它们加载入 Calc/Excel/Gnumeric 。请注意，时间单位是 ms ，空间单位是 kb 。

full_test.sh 用多个数据集和查询比较 gStore 和其他数据库系统的性能。

要使用 full_test.sh ，请下载你想要比较的数据库系统，并在这一脚本中准确设置数据库系统和数据集的位置。命名策略和日志策略应该与 gtest 的要求一致。

在这一脚本中仅测试比较了 gStore 和 Jena ，如果你愿意花时间阅读这一脚本，很容易添加其他数据库系统。如果遇到问题，你可以参考 【测试报告】或【FAQ】一章寻求帮助。

3 高级

3.1 第 05 章：HTTP API 说明

我们提供了 HTTP API （建议使用）和 socket API ，分别对应 ghttp 和 gserver 。

本章节提供了使用 http 服务的 API ，在服务器端运行 ghttp 时使用。和 socket API 相比，HTTP API 更稳定且能保持连接，也更规范，而 socket API 不保证传输正确，所以网络传输会更快。

3.1.1 简单样例

我们目前提供了 JAVA 、C++ 、Python 、PHP 和 Nodejs 的 HTTP API 。请参考 `api/http` 的样例代码。要使用这些样例，请确保已经生成了可执行程序。

接下来，**用 `./ghttp` 指令启动 gStore 服务器。**如果你知道一个正在运行的可用的 gStore 服务器，你可以尝试连接它，请注意**服务器 ip 、服务器和客户端的端口号必须匹配。**（样例使用默认设置，不需要更改。）之后，对于 Java 和 C++ ，你需要在 gStore/api/ 目录下编译样例代码。

最后，打开样例目录并运行相应的可执行程序。这些程序发出 HTTP 请求连接到指定的 gStore 服务器并做一些加载或查询操作。请确保你在运行样例的终端还者浏览器中看到了查询结果，如果没有，请参阅【FAQ】一章或向我们报告。（【README】中描述了报告方法。）

我们建议你仔细阅读样例代码和相应的 Makefile 。这会帮助你理解 API ，特别是如果你想基于 API 接口写自己的程序。

3.1.2 API 结构

gStore 的 HTTP API 在 gStore 根目录的 api/http/ 目录下，内容如下：

- gStore/api/http/
  - cpp/ （C++ API）
    - client.cpp （C++ API 源代码）
    - client.h
    - example (样例程序，展示使用 C++ API 的基本思路)
      - example.cpp
      - Benchmark.cpp

        - Benchmark1.cpp
        - CppAPIExample.cpp
        - Makefile
    * Makefile （编译并生成 lib ）
- java/ (Java API)
    * client.java
    * lib/
    * src/
        - jgsc/GstoreConnector.java
        - Makefile
    * example/ （样例程序，展示使用 Java API 的基本思路）
        - Benchmark.java
        - Benchmark1.java
        - JavaAPIExample.java
        - Makefile
- python/ (Python API)
    * example/
        - Benchmark.py
        - PyAPIExample.py
    * src/
        - GstoreConnector.py
- php/ (Php API)
    * example/
        - Benchmark.php phpAPIExample.php
    * src/
        - GstoreConnector.php
- nodejs/ (the Nodejs API)
    * example.js
    * index.js
    * package.json
    * README.md

3.1.3 C++ API

接口 要使用 C++ HTTP API ,请在你的 cpp 代码中加入 `#include "client.h"` 。client.h 中的函数可以如下调用：

```
// 初始化
GstoreConnector gc("172.31.222.93", 9016);

// 从一个 RDF 文件新建数据库
// 注意，采用以 gserver 为基础的相对路径
gc.build("test", "data/lubm/LUBM_10.n3", "root", "123456");

// load 刚才 build 好的数据库
gc.load("test", "root", "123456");

// 之后就可以在这个数据库上执行 SPARQL 查询
std::string answer = gc.query("root", "123456", "test", sparql);
std::cout << answer << std::endl;

// 执行一个 SPARQL 查询，并将结果保存到 ans.txt
gc.query("root", "123456", "test", sparql, "ans.txt");

// unload 这一数据库
gc.unload("test", "root", "123456");

// 你也可以直接 load 已存在的数据库并进行查询
gc.load("lubm", "root", "123456");
answer = gc.query("root", "123456", "lubm", sparql);

std::cout << answer << std::endl;
gc.unload("lubm", "root", "123456");
```

原始的函数声明如下：

```
GstoreConnector();

bool build(std::string _db_name, std::string _rdf_file_path, std::string username, st
```

```
bool load(std::string _db_name, std::string username, std::string password);

bool unload(std::string _db_name, std::string username, std::string password);

void query(std::string username, std::string password, std::string db_name, std::stri

std::string query(std::string username, std::string password, std::string db_name, st
```

3.1.4 Java API

接口 要使用 Java API，请参照 gStore/api/http/java/src/jgsc/GstoreConnector.java 。函数应该如下调用：

```
// 初始化
GstoreConnector gc = new GstoreConnector("172.31.222.78", 3305);

// build 数据库
gc.build("LUBM10", "data/lubm/lubm.nt", "root", "123456");

// load 刚才 build 的数据库
gc.load("LUBM10", "root", "123456");

// 你也可以直接 load 已存在的数据库并进行查询
gc.load("LUBM10", "root", "123456");

// 执行 SPARQL 查询
answer = gc.query("root", "123456", "LUBM10", sparql);
System.out.println(answer);

// 执行 SPARQL 查询并将结果保存到 ans.txt
gc.query("root", "123456", "LUBM10", sparql, "ans.txt");

// unload 数据库
gc.unload("LUBM10", "root", "123456");
```

这些函数的原始声明如下：

```
public class GstoreConnector();

public boolean build(String _db_name, String _rdf_file_path, String _username, String

public boolean load(String _db_name, String _username, String _password);

public boolean unload(String _db_name,String _username, String _password);

public String query(String _username, String _password, String _db_name, String _spar

public void query(String _username, String _password, String _db_name, String _sparql
```

3.1.5 Python API

接口 要使用 Python API，请参照 gStore/api/http/python/src/GstoreConnector.py 。函数应该如下调用：

```
# 给定 IP 和端口号, 启动 gc
gc =  GstoreConnector.GstoreConnector("172.31.222.78", 3305)

# 从一个 RDF 图 build 数据库
ret = gc.build("test", "data/lubm/lubm.nt", username, password)

# load 数据库
ret = gc.load("test", username, password)

# 查询
print (gc.query(username, password, "test", sparql))

# 查询并将结果保存至文件
gc.fquery(username, password, "test", sparql, filename)

# unload 数据库
ret = gc.unload("test", username, password)
```

函数的原始声明如下：

```
public class GstoreConnector()
```

```
def build(self, db_name, rdf_file_path, username, password):

def load(self, db_name, username, password):

def unload(self, db_name, username, password):

def query(self, username, password, db_name, sparql):

def fquery(self, username, password, db_name, sparql, filename):
```

3.1.6 Php API

接口 要使用 Php API，请参照 gStore/api/http/php/src/GstoreConnector.php 。函数应该如下调用：

```
// 启动 gc
$gc = new GstoreConnector("172.31.222.78", 3305);

// build 数据库
$ret = $gc->build("test", "data/lubm/lubm.nt", $username, $password);
echo $ret . PHP_EOL;

// load rdf
$ret = $gc->load("test", $username, $password);
echo $ret . PHP_EOL;

// 查询
echo $gc->query($username, $password, "test", $sparql) . PHP_EOL;

// fquery-- 查询并将结果保存到文件
$gc->fquery($username, $password, "test", $sparql, $filename);

// unload rdf
$ret = $gc->unload("test", $username, $password);
echo $ret . PHP_EOL;
```

函数的原始声明如下：

```
class GstoreConnector

function build($db_name, $rdf_file_path, $username, $password)

function load($db_name, $username, $password)

function unload($db_name, $username, $password)

function query($username, $password, $db_name, $sparql)

function fquery($username, $password, $db_name, $sparql, $filename)
```

3.1.7 Nodejs API

接口 使用 Nodejs API 之前，在 nodejs 文件夹下输入 `npm install request` 和 `npm install request-promise` 来安装需要的模块。

要使用 Nodejs API，请参照 gStore/api/http/nodejs/index.js 。函数应该如下调用：

```
# start a gc with given IP and Port
gc =  new GStoreClient(username, password, "http://127.0.0.1:9000")

# build database with a RDF graph
ret = gc.build("test", "data/lubm/lubm.nt")

# load the database
ret = gc.load("test")

# query
print (gc.query("test", sparql))

# unload the database
ret = gc.unload("test")
```

这些函数的原始声明如下：

```
class GStoreClient

async build(db = '', dataPath = '')

async load(db = '')

async unload(db = '')

async query(db = '', sparql = '')
```

3.2 第06章：socket API说明

这一 API 暂时停止维护。

3.2.1 简单样例

我们目前提供了 JAVA ，C++ ，PHP 和 Python 的 gStore API 。请参考 `api/socket/cpp/example` ，`api/socket/java/example` , `api/socket/php` 和 api/socket/php/example 的样例代码。要使用 Java 和 C++ 的样例，请确保已经生成了可执行程序。如果没有生成，只需要在 gStore 根目录下输入 `make APIexample` 来编译代码和 API 。

接下来，**用 `./gserver` 指令启动 gStore 服务器。**如果你知道一个正在运行的可用的 gStore 服务器，你可以尝试连接它，请注意**服务器 ip 、服务器和客户端的端口号必须匹配。**（样例使用默认设置，不需要更改。）之后，对于 Java 和 C++ ，你需要在 gStore/api/ 目录下编译样例代码。我们提供了一个程序，只需要在 gStore 根目录下输入 `make APIexample` 。或者你可以自己编译代码，在本例中，请分别打开 gStore/api/socket/cpp/example/ 和 gStore/api/socket/java/example/ 。

最后，打开样例目录并运行相应的可执行程序。对 C++ 而言，用 `./example` 指令运行。对 Java 而言，用 `make run` 指令或 `java -cp ../lib/GstoreJavaAPI.jar:. JavaAPIExample` 运行。PHP 和 Python 文件不需要编译，可以直接执行。这些程序都会连接到指定的 gStore 服务器并做一些加载或查询操作。请确保你在运行样例的终端看到了查询结果，如果没有，请参阅【FAQ】一章或向我们报告。（【README】中描述了报告方法。）

我们建议你仔细阅读样例代码和相应的 Makefile 。这会帮助你理解 API ，特别是如果你想基于 API 写自己的程序。

3.2.2 API 结构

gStore 的 socket API 在 gStore 根目录的 api/socket/ 目录下，内容如下：

- gStore/api/socket/
  - cpp/ （C++ API）
    - src/ （C++ API 的源代码，用于生成 lib/libgstoreconnector.a ）
      - GstoreConnector.cpp （与 gStore 服务器交互的接口）
      - GstoreConnector.h

        - Makefile （编译并生成lib）
    * lib/ （静态库所在）
        - .gitignore
        - libgstoreconnector.a (只在编译后存在，使用 C++ API 时需要连接这个库)
    * example/ （样例程序，展示使用 C++ API 的基本思路）
        - CppAPIExample.cpp
        - Makefile
- java/ （Java API ）
    * src/ （Java API 的源代码，用于生成 lib/GstoreJavaAPI.jar ）
        - jgsc/GstoreConnector.java （使用 Java API 时需要导入的包）
        - Makefile （编译并生成库）
    * lib/
        - .gitignore
        - GstoreJavaAPI.jar （只在编译后存在，你需要在类目录中包括这一 JAR ）
    * example/ （样例程序，展示使用 Java API 的基本思路）
        - JavaAPIExample.cpp
        - Makefile
- php/ (PHP API)
    * GstoreConnector.php （PHP API 的源码，在使用 PHP API 时，你需要 include 这一文件）
    * PHPAPIExample.php （样例程序，展示使用 PHP API 的基本思路）
- python/ (Python API)
    * src/ （P ython API 源码）
        - GstoreConnector.py （使用 Python API 时需要导入的包）
    * example/ （样例程序，展示使用 Python API 的基本思路）
        - PythonAPIExample.py

3.2.3 C++ API

接口 要使用 C++ API ,请在你的 cpp 代码中加入 `#include "GstoreConnector.h"` 。GstoreConnector.h 中的函数可以如下调用：

```
// 初始化 Gstore 服务器的 IP 地址和端口
GstoreConnector gc("127.0.0.1", 3305);
// 由一个 RDF 文件 build 一个数据库
// 注意，文件地址是相对 gserver 的地址
gc.build("lubm", "../data/lubm/lubm.nt");
// 然后你可以在这一数据库上执行 SPARQL 查询
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 这一数据库
gc.unload("lubm");
// 你也可以直接 load 已存在的数据库然后进行查询
gc.load("lubm");
// 在当前数据库查询 SPARQL
answer = gc.query(sparql);
```

原始的函数声明如下：

```
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);
```

注意：

1. 在使用 GstoreConnector() 时，ip 和端口的默认值分别是 127.0.0.1 和 3305 。

2. 在使用 build() 时，rdf_file_path （第二个参数）应该和 gserver 的位置相关。

3. 请记得卸载你导入的数据库，否则可能会出错。（错误可能不被报告!）

编译 我们建议你在 gStore/api/socket/cpp/example/Makefile 中查看如何用 C++ API 编译你的代码。通常来说，你必须要将代码编译为包含了 C++ API 头的目标文件，并将目标文件连接到 C++ API 中的静态库。

我们假设你的源代码在 test.cpp 中，位置为 ${GSTORE}/gStore/ 。（如果名字是 devGstores 而不是 gStore ，那么路径为 ${GSTORE}/devGstore/ ）

用 `g++ -c -I${GSTORE}/gStore/api/socket/cpp/src/ test.cpp -o test.o` 将你的 test.cpp 编译成 test.o ，相关的 API 头在 api/socket/cpp/src/ 中。

用 `g++ -o test test.o -L${GSTORE}/gStore/api/socket/cpp/lib/ -lgstoreconnector` 将 test.o 连接到 api/socket/cpp/lib/ 中的 libgstoreconnector.a (静态库)。

接下来，你可以输入 `./test` 执行使用了 C++ API 的程序。我们还建议你将相关的编译命令和其他你需要的命令放在 Makefile 中。

3.2.4 Java API

接口 要使用 Java API，请在 java 代码中加入 `import jgsc.GstoreConnector;` 。GstoreConnector.java 中的函数应该如下调用：

```
// 初始化 Gstore 服务器的 IP 地址和端口
GstoreConnector gc = new GstoreConnector("127.0.0.1", 3305);
// 由一个 RDF 文件 build 一个数据库
// 注意，文件地址是相对 gserver 的地址
gc.build("lubm", "../data/lubm/lubm.nt");
// 然后你可以在这一数据库上执行 SPARQL 查询.
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 这一数据库
gc.unload("lubm");
// 你也可以直接 load 已存在的数据库然后进行查询
gc.load("lumb");// 在当前数据库查询SPARQL
answer = gc.query(sparql);
```

这些函数的原始声明如下：

```
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);
```

注意：

1. 在使用 GstoreConnector() 时，ip 和端口的默认值分别是 127.0.0.1 和 3305 。
2. 在使用 build() 时，rdf_file_path （第二个参数）应该和是对于 gserver 的相对路径。
3. 请记得 unload 你导入的数据库，否则可能会出错。（错误可能不被报告!）

编译 我们建议你在 gStore/api/socket/java/example/Makefile 中查看如何用 Java API 编译你的代码。通常来说，你必须要将代码编译为包含了 Java API 中 jar 文件的目标文件。

我们假设你的源代码在 test.java 中，位置为 ${GSTORE}/gStore/ 。（如果名字是 devGstores 而不是 gStore ，那么路径为 ${GSTORE}/devGstore/ ）

用 `javac -cp ${GSTORE}/gStore/api/socket/java/lib/GstoreJavaAPI.jar test.java` 将 test.java 编译为使用了 api/socket/java/lib/ 中 GstoreJavaAPI.jar （Java 中使用的 jar 包）的 test.class

接下来，你可以输入 java -cp ${GSTORE}/gStore/api/socket/java/lib/GstoreJavaAPI.jar:. test 执行使用了 Java API 的程序（注意，命令中的“:.”不能省略）。我们还建议你将相关的编译命令和其他你需要的命令放在 Makefile 中。

3.2.5 PHP API

接口 要使用 PHP API，请在你的 PHP 代码中加入 include('GstoreConnector,php'); 。GstoreConnector.php 中的函数应该如下调用：

```
// 初始化 Gstore 服务器的 IP 地址和端口
$gc = new Connector("127.0.0.1", 3305);
// 由一个 RDF 文件新建一个数据库
// 注意，文件地址是相对 gserver 的地址
$gc->build("lubm", "../data/lubm/lubm.nt");
// 然后你可以在这一数据库上执行 SPARQL 查询
$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 这一数据库
$gc->unload("lubm");
// 你也可以直接 load 已存在的数据库然后进行查询
$gc->load("lubm");// 在当前数据库查询SPARQL
$answer = gc->query(sparql);
```

这些函数的原始声明如下：

```
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();
}
```

注意：

1. 在使用 GstoreConnector() 时，ip 和端口的默认值分别是 127.0.0.1 和 3305 。
2. 在使用 build() 时，rdf_file_path （第二个参数）应该和 gserver 的位置相关。
3. 请记得卸载你导入的数据库，否则可能会出错。（错误可能不被报告!）

运行 gStore/api/socket/php/PHPAPIExample 展示了如何使用 PHP API。PHP 脚本不需要编译，你可以直接运行，或将其用在你的网页中。

3.2.6 Python API

接口 要使用 Python API ，请在代码中加入 `from GstoreConnector import GstoreConnector` 。GstoreConnector.py 中的函数应该如下调用：

```
// 初始化 Gstore 服务器的 IP 地址和端口
gc = GstoreConnector('127.0.0.1', 3305)
// 由一个 RDF 文件 build 一个数据库
// 注意，文件地址是相对 gserver 的地址
gc.build('lubm', '../data/lubm/lubm.nt')
// 然后你可以在这一数据库上执行 SPARQL 查询
$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 这一数据库
gc.unload('lubm')
// 你也可以直接 load 已存在的数据库然后进行查询
gc.load('lubm')// 在当前数据库查询sparql
answer = gc.query(sparql)
```

函数的原始声明如下：

```
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)
}
```

注意：

1. 在使用 GstoreConnector() 时，ip 和端口的默认值分别是 127.0.0.1 和 3305 。
2. 在使用 build() 时，rdf_file_path （第二个参数）应该和 gserver 的位置相关。
3. 请记得卸载你导入的数据库，否则可能会出错。（错误可能不被报告！）

运行 gStore/api/socket/python/example/PythonAPIExample 展示了如何使用 python API。Python 文件不需要编译，可以直接运行。

3.3 第07章：web 应用

本章提供了 API 的具体用例。

3.3.1 用例

现在你对我们的 API 已经有了基本的了解，但你可能仍然会有点迷惑。这里我们提供一个简单的样例来告诉你应该做什么。

比如说，你需要在一个 web 项目中使用 gStore 。PHP 是一种广泛使用的脚本语言，适用于 web 开发。所以，使用我们的 PHP API 可以满足你的要求。这是我们完成的一个网页：demo。

首先，准备好你的 web 服务器，使其能够运行 PHP 文件。对于这一步我们不做细节介绍，根据你的 web 服务器（例如 Apache ，或者 Nginx 等等），你可以轻易地搜索到相关操作。

接下来，进入你的 web 文件根目录（通常在 /var/www/html 或者 apache/htdocs ，可以在配置文件中查看），创建一个文件夹并命名为“Gstore”。然后把 GstoreConnector.php 文件拷贝到这个文件夹下。创建一个 PHP 文件，命名为“PHPAPI.php”。如下编辑：

```
<?php
//require "../src/GstoreConnector.php";
include( 'gStoreConnector.php');

$username = "root";
$password = "123456";
$gc = new GstoreConnector("127.0.0.1", 9001);

$dbname = $_POST["databasename"];
$query = $_POST["query"];
$format = $_POST["format"];
$result = $gc->query($username, $password, $dbname, $query);
$js = json_decode($result);
switch ($format) {
case 1:
$vars = $js->head->vars;
$rows = $js->results->bindings;
```

```
$html = '<html><table class="sparql" border="1"><tr>';
for ($i = 0; $i < count($vars); $i++) {
$html.= '<th>' . $vars[$i] . '</th>';
}
$html.= '</tr>';
for ($i = 0; $i < count($rows); $i++) {
$html.= '<tr>';
$row = $rows[$i];
for ($j = 0; $j < count($vars); $j++)
$html.= '<td>' . htmlspecialchars($row->
$vars[$j]->value) . '</td>';
$html.= '</tr>';
}
$html.= '</table></html>';
echo $html;
exit;

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

$rows = $js->results->bindings;

for ($i = 0; $i < count($rows); $i++)
echo $rows[$i] . "\n";
exit;

case 3:
$filename = 'sparql.csv';
header("Content-Type: application/octet-stream");
header('Content-Disposition: attachment;
filename="' . $filename . '"');

$vars = $js->head->vars;
```

```
echo implode(",",$vars);
echo "\n";
$rows = $js->results->bindings;
for($i = 0; $i < count($rows); $i++) {
$row = $rows[$i];
for($j = 0; $j < count($vars); $j++) {
echo ($row->$vars[$j]->value);
echo ",";
}
echo "\n";
}

exit;
}
?>
```

这一 PHP 文件从一个网页中获取三个参数：数据库名、sparql 和输出格式。然后它用我们的 PHP API 连接到 gStore，执行查询。最后，代码中的“switch”部分按照要求的格式给出结果。

之后，我们需要写一个可以搜集上述参数（数据库名、sparql 和输出格式）的网页。我们创建一个 html 文件，使用表单来完成这一步骤。如下所示：

```
<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>
```

```
17
18      <li id="li_3">
19        <label class="description" for="element_3">Query Text </label>
20        <div>
21          <textarea id="element_3" name="sparql" class="element textarea
            large">
22           SELECT DISTINCT ?uri
23           WHERE {
24             ?uri <type> <Astronaut> .
25           { ?uri <nationality> <Russia> . }
26           UNION
27           { ?uri <nationality> <Soviet_Union> . }
28           }
29        </textarea>
30        </div>
31      </li>
32
33      <li id="li_5" >
34        <label class="description" for="element_5">
35          Results Format
36        </label>
37        <div>
38          <select class="element select medium" id="element_5" name="
            format">
39           <option value="1" selected="ture">HTML</option>
40           <option value="2" >Text</option>
41           <option value="3" >CSV</option>
42          </select>
43        </div>
44
45      <li class="buttons">
46        <input type="hidden" name="form_id" value="1145884" />
47        <input id="saveForm" class="button_text" type="submit"
48         name="submit" value="Run Query" />
49      </li>
50    </ul>
51  </form>
```

你可以在代码中看到，我们用 <input> 元素得到数据库名，<texarea> 得到 sparql ，<select> 得到输出格式。<form> 标签有一个属性“action”，指明了要执行哪个文件（在本例中是“PHPAPI.php”）。因此，当你在网页中点击“submit”时，就会执行 PHPAPI.php 并且传递表单中收集到的参数。

最后，不要忘了在你的服务器上启动 ghttp 。

3.4 第 08 章：项目结构

itextbf(本章介绍了 gStore 系统项目的整体结构。)

核心源代码如下列出：

- Database/ （调用其他核心部分，处理接口部分的请求）
  - Database.cpp （实现函数）
  - Database.h （类、成员和函数定义）
  - Join.cpp （连接候选结点得到结果）
  - Join.h （类、成员和函数定义）
  - Strategy.cpp
  - Strategy.h
- KVstore/ （键-值存储，在内存和磁盘间交换）
  - KVstore.cpp (interact with upper layers)
  - KVstore.h
  - ISArray/
    - ISArray.cpp
    - ISArray.h
    - ISBlockManager.cpp
    - ISBlockManager.h
    - ISEntry.cpp
    - ISEntry.h
  - ISTree/
    - ISTree.cpp
    - ISTree.h
    - heap/
      - ISHeap.cpp
      - ISHeap.h
    - node/
      - ISIntlNode.cpp

        - ISIntlNode.h
        - ISLeafNode.cpp
        - ISLeafNode.h
        - ISNode.cpp
        - ISNode.h
      * storage/
        - ISStorage.cpp
        - ISStorage.h
  - IVArray/
  - IVTree/
  - SITree/

- Query/ （回答 SPARQL 查询时需要）
  - BasicQuery.cpp （不含聚集操作的基本查询类型）
  - BasicQuery.h
  - IDList.cpp （查询结点/变量的候选列表）
  - IDList.h
  - ResultFilter.cpp
  - ResultFilter.h
  - ResultSet.cpp （储存对应查询的结果集）
  - ResultSet.h
  - SPARQLquery.cpp （处理整个 SPARQL 查询）
  - SPARQLquery.h
  - Varset.cpp
  - Varset.h
  - QueryCache.cpp
  - QueryCache.h
  - QueryTree.cpp
  - QueryTree.h
  - GeneralEvaluation.cpp

  - GeneralEvaluation.h
  - TempResult.cpp
  - TempResult.h
  - RegexExpression.h

- Signature/ （为结点和边分配签名，但不为文字分配）
  - SigEntry.cpp
  - SigEntry.h
  - Signature.cpp
  - Signature.h

- VSTree/ （高效修剪的树索引）
  - EntryBuffer.cpp
  - EntryBuffer.h
  - LRUCache.cpp
  - LRUCache.h
  - VNode.cpp
  - VNode.h
  - VSTree.cpp
  - VSTree.h

解析部分如下列出：

- Parser/
  - DBParser.cpp
  - DBParser.h
  - RDFParser.cpp
  - RDFParser.h
  - SparqlParser.c （自动生成，手动细微修改，压缩）
  - SparqlParser.h （自动生成，手动细微修改，压缩）

  - SparqlLexer.c （自动生成，手动细微修改，压缩）SparqlLexer.c (自动生成，手动细微修改，压缩)
  - SparqlLexer.h （自动生成，手动细微修改，压缩）
  - TurtleParser.cpp
  - TurtleParser.h
  - Type.h
  - QueryParser.cpp
  - QueryParser.h

程序如下列出：

- Util/
  - Util.cpp （头，宏，定义类型，函数…）
  - Util.h
  - Bstr.cpp （展现任意长的字符串）
  - Bstr.h （类、成员和函数定义）
  - Stream.cpp （储存并使用临时结果，可能非常大）
  - Stream.h
  - Triple.cpp（处理三元组，一个三元组可以分为主体（实体）、谓词（实体）和客体（实体或字面量））
  - Triple.h
  - BloomFilter.cpp
  - BloomFilter.h
  - ClassForVListCache.h
  - VList.cpp
  - VList.h

接口部分如下列出：

- Server/ （使用 gStore 的客户端和服务器模式）
  - Client.cpp

  - Client.h
  - Operation.cpp
  - Operation.h
  - Server.cpp
  - Server.h
  - Socket.cpp
  - Socket.h
  - client_http.hpp
  - server_http.hpp

- web/ （操作 gStore 的一系列应用/主程序）
  - ...

更多细节 如果你在源代码中看到和原始设计的不同的东西，这并不奇怪。一些设计出的函数可能目前还没有实现。

其他 gStore 中的 api/ 文件夹用于存储 API 程序、库和样例，请参在 【http API】 一章中获取更多信息。test/ 文件夹用于存储一系列测试程序，例如 gtest，full_test 等等。和 test/ 有关的章节是【如何使用】和【测试结果】。本项目需要 ANTLR 库解析 SPARQL 查询，其代码在 tools/ 中（也在这里实现），编译的 libantlr.a 在 lib/ 目录下。

我们在 data/ 目录下放置了一些数据集和查询作为样例，你可以尝试它们，看看 gStore 怎样工作。相关说明在【如何使用】一章中。docs/ 目录包含 gStore 的各类文档，包括一系列 markdown 文件，还有 pdf/ 和 jpg/ 两个文件夹。pdf 文件储存在 pdf/ 文件夹下，jpg 文件在 jpg/ 文件夹下。

我们建议你从 gStore 根目录下的【README】开始，然后只在需要的时候浏览其他章节。如果你真的对 gStore 感兴趣，最后，你会在链接中看到所有文件。

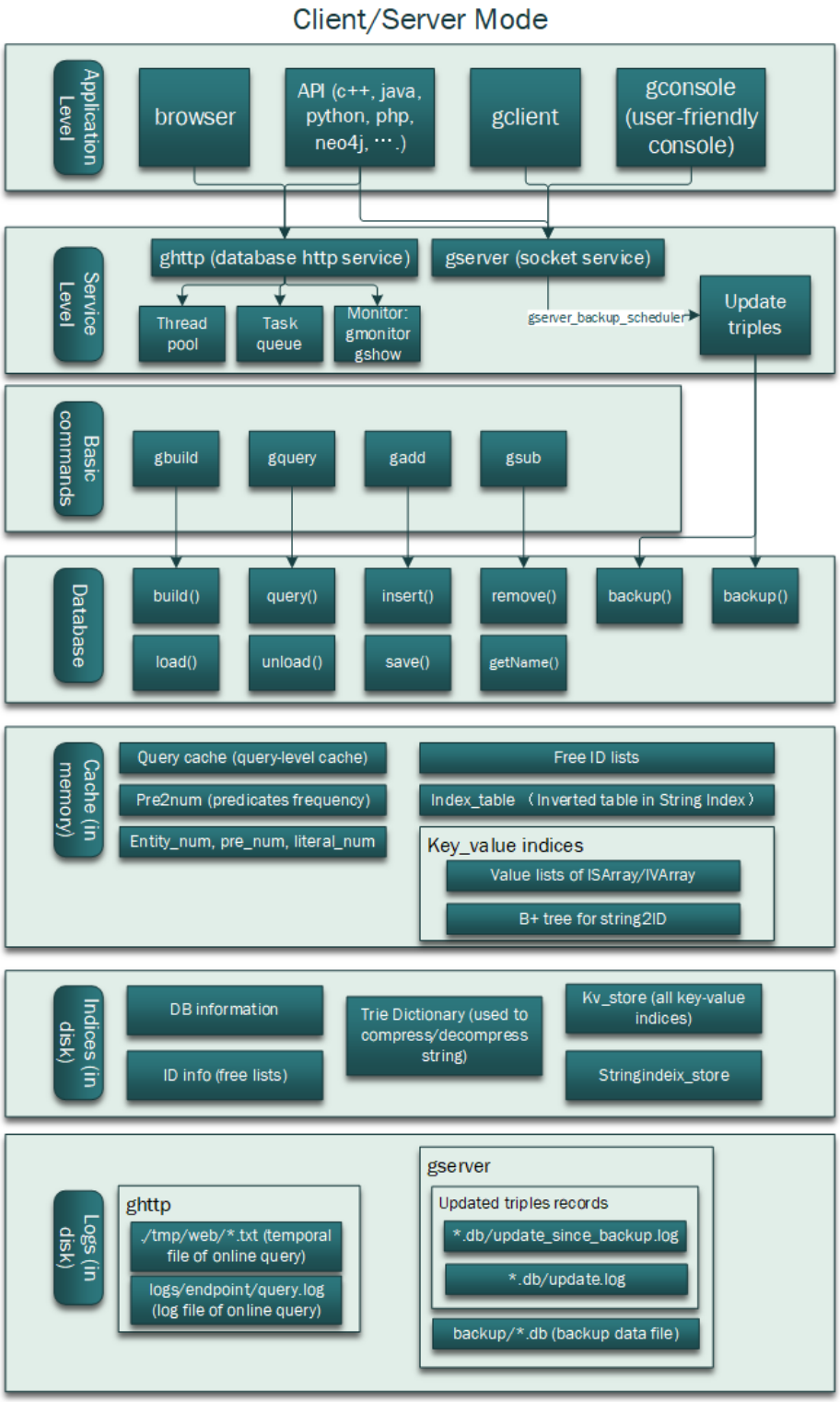


图 1: framework

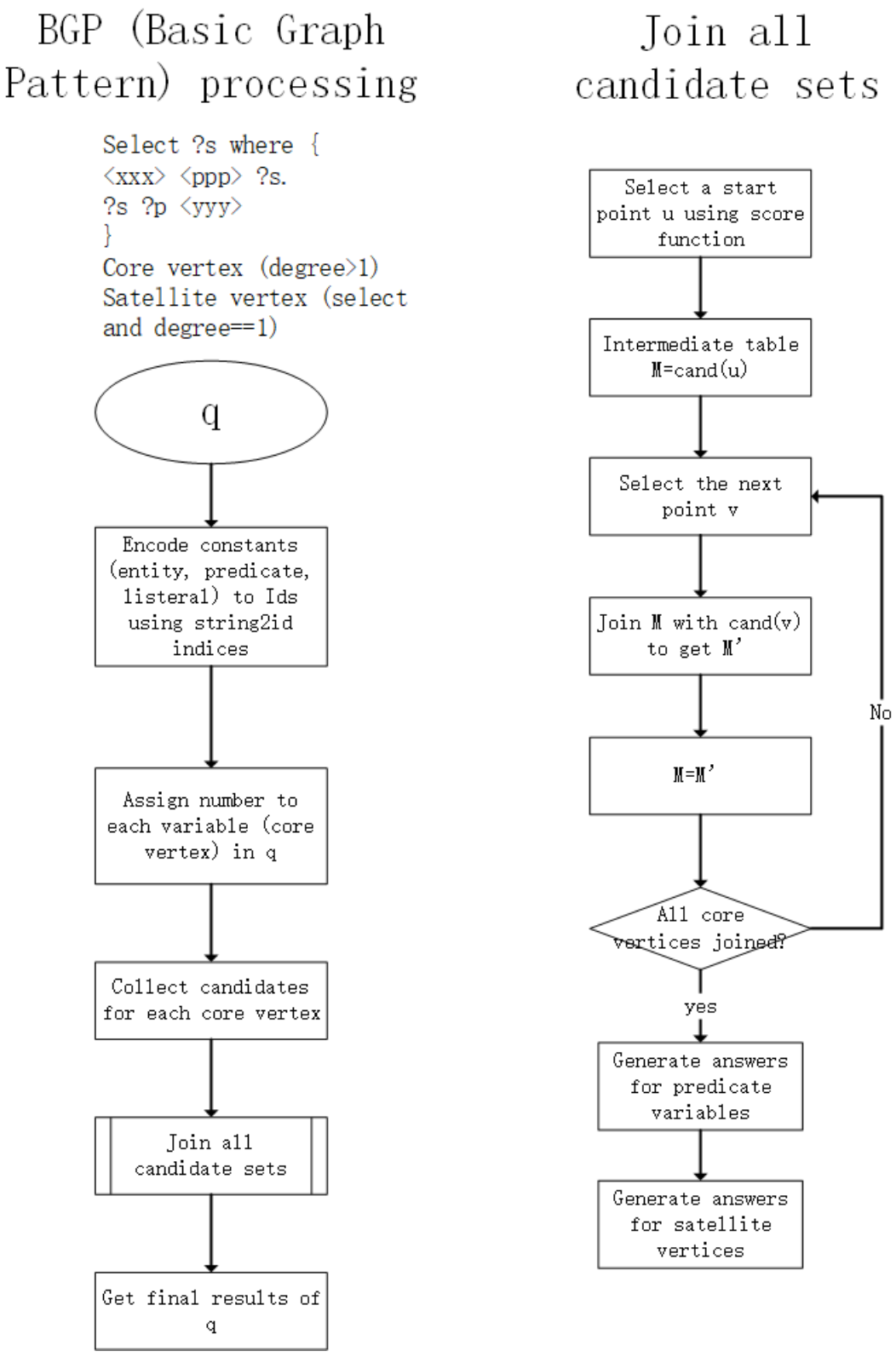


图 2: Basic Graph Pattern Processing

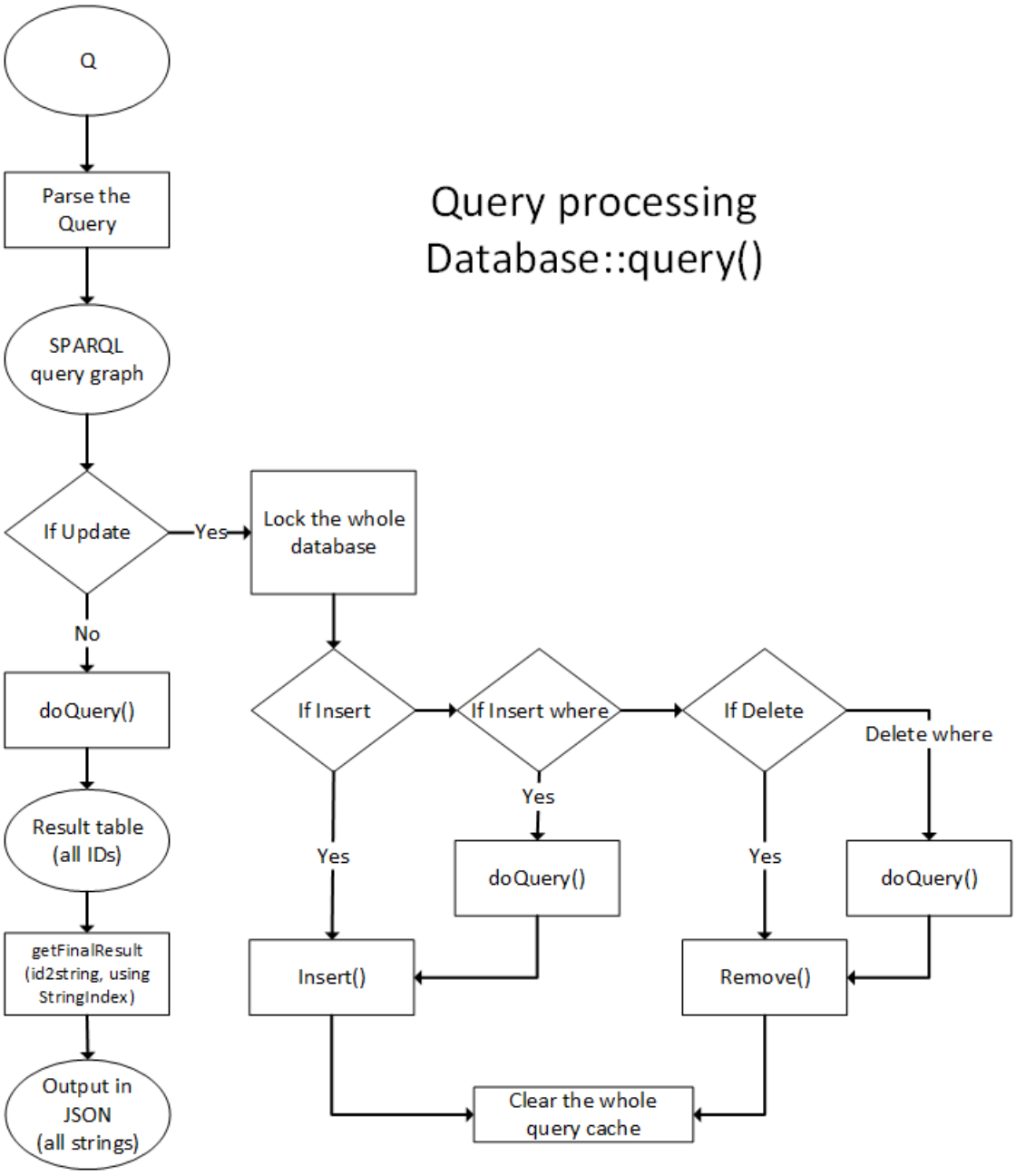


图 3: Query Processing

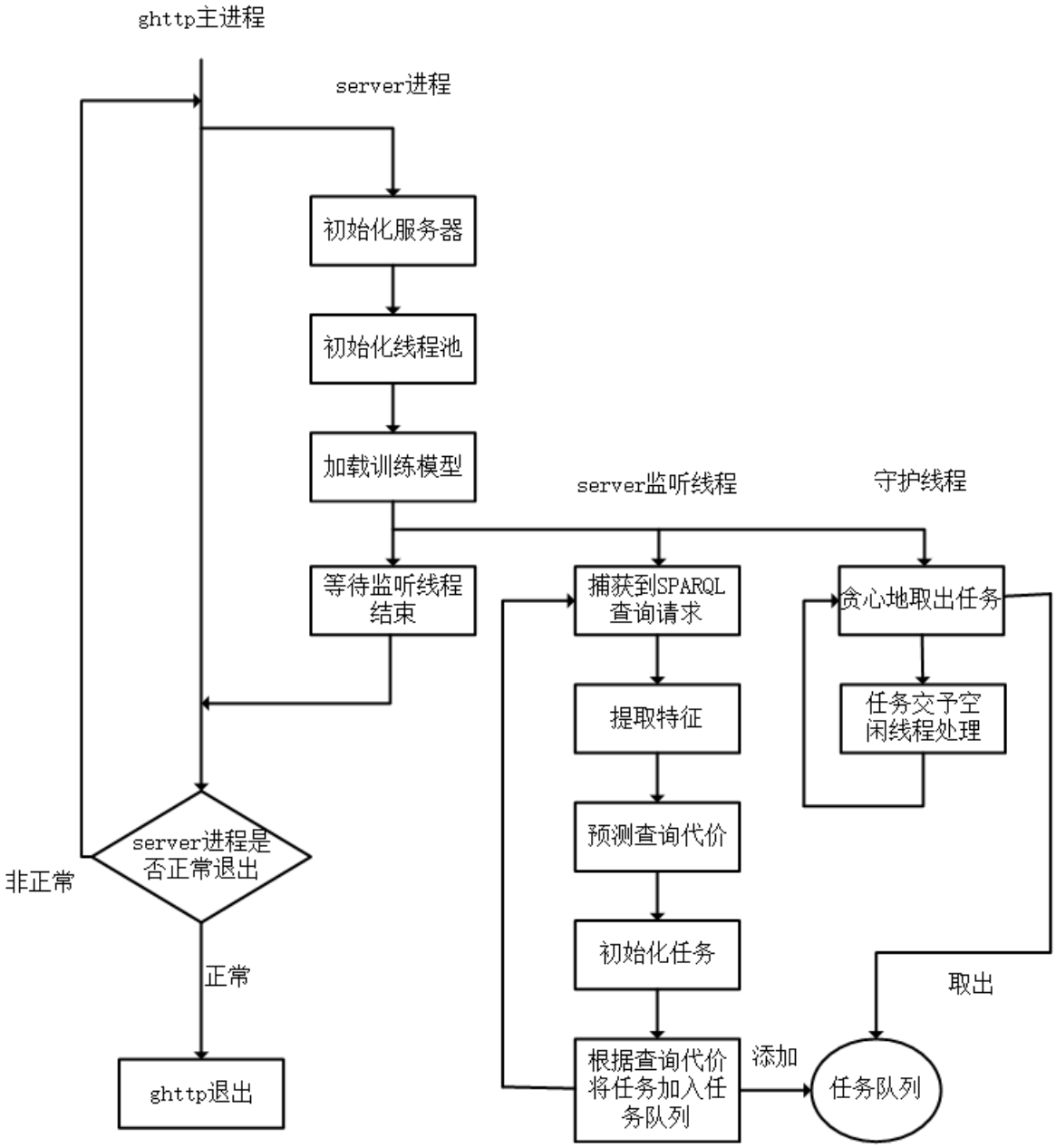


图 4: ghttp thread

3.5 第09章：出版物

和 **gStore** 相关的出版物在此列出：

- 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).

3.6 第 10 章：限制

1. 查询处理中的过滤效果还不是很好（少于 25% ）
2. 只支持 N-Triples 格式的 RDF 文件。

3.7 第 11 章：FAQ

使用更新版本的 gStore 系统查询原始数据库时，为什么会出错？

gStore 生产的数据库包含一些索引，其结构可能在新的 gStore 版本中发生了改变。所以，以防万一，请重新生成数据集。

我试着写类似 Main/gconsole.cpp 的基于 gStore 的程序时，为什么会出错？

你需要在你的主程序开头加入这些语句，否则 gStore 无法正确运行：

```
//NOTICE:this is needed to ensure the file path is the work path
chdir(dirname(argv[0]));
//NOTICE:this is needed to set several debug files
Util util;
```

我使用 Java API 时，为什么 gStore 报告 “garbage collection failed” 错误？

你需要调整 jvm 参数，参见 url1 和 url2 获取更多细节。

我在 ArchLinux 中编译代码时，为什么报告 “no-ltermcap” 错误？

在 ArchLinux 下，你只需要用 `-lreadline` 连接 readline 库。如果你要使用 ArchLinux ，请移除 gStore 根目录下 makefile 中的 `-ltermcap` 。

为什么 gStore 报告错误称不支持一些 RDF 数据集的格式？

gStore 现在不支持所有的 RDF 格式，请参阅格式获取细节。

我在 GitHub 上阅读的时候，为什么有一些文件打不开？

代码、markdown、其他文本文件和图片可以直接在 GitHub 上阅读。如果你使用的是轻量级浏览器，例如 midori ，请将 pdf 文件下载后在电脑或其他设备上阅读。

为什么使用 gStore 时有时候会出现奇怪的字符？

一些文件的名称是中文，你不需要担心这个问题。

在 centos7 系统中，如果复制或压缩/解压 watdiv.db （gbuild 生成的一个数据库），用 `du -h` 命令进行检查，watdiv.db 的大小会改变（通常会变得更大）？

是 watdiv/kv_store/ 中 B+- 树大小的改变导致整个数据库大小的改变。原因是，在 storage/Storage.cpp 中，很多操作用 fseek 移动文件指针。大家都知道，文件是以块的形式组织的，如果我们请求新的块，文件指针可能移动到当前文件外（gStore 中的文件操作都用 C 实现，没有报告错误），然后内容将写入新的位置！

在 **Unix 环境下的高级编程**中，“文件洞”描述了这一现象。“文件洞”被 0 填充，也是文件的一部分。你可以用 `ls -l` 查看文件的大小（计算了洞的大小），`du -h` 命令显示目录 / 文件在系统中占用的块的大小。通常来说，`du -h` 的输出会比 `du -h` 更大，但如果“文件洞”存在，就会出现相反的结果，因为洞的大小被忽略了。

包含洞的文件的实际大小被修正，在一些操作系统中，拷贝时洞会被转变为内容（也是 0 ）。如果不是在不同的设备间，操作 `mv` 不会影响大小（只需要调整文件树索引）。然而，`cp` 和各类压缩方法需要扫描文件并传输数据（考虑到是否忽略洞，有两种方法实现 `cp` 命令，但 `ls -l` 输出的大小不变）。

在 C 中使用“文件洞”是有效的，这不是一个错误，你可以继续使用 gStore 。我们实现了一个小程序描述“文件洞“，你可以下载并尝试。

在 gclient 控制台中，build 并查询了一个数据库，然后我退出了控制台。下次我进入控制台时，load 原来载入的数据库，但没有任何查询的输出（原始输出不为空？

在退出 gclient 控制台之前，你需要 unload 数据库，否则会出现错误。

如果查询结果包括 null 值，我要怎么使用 full_test 程序？用制表符分隔的方法会造成问题，因为不能检测到 null 值！

你可使用其他编程语言（例如，Python ）处理这种问题。例如，你可以在输出中将 null 值变为 ‘,’ 之类的特殊字符，然后你就可以使用 full_test 了。

当我编译并运行 API 样例时，报告 “unable to connect to server” 错误？

请先用 `./gserver` 命令启动 gStore 服务器，请注意服务器 ip 和端口号必须匹配。

当我使用 Java API 写程序的时候，报告 “not found main class” 错误？

请确保你在 java 的类路径中包含了你的程序的位置。完整的命令应该和 `java -cp /home/bookug/project/devGstore/api/java/lib/GstoreJavaAPI.jar:. JavaAPIExample` 类似，命令中的 “:.” 不能省略。

3.8 第 12 章：技巧

本章节介绍在使用 gStore 实现应用时的一些实用技巧。

3.8.1 配置

如果你想将 gStore 作为一个 SPARQL 终端使用，最好在 Util.h 中设置 SPARQL_ENDPOINT 参数，Util.h 中的所有 DEBUG 超级参数都不应该使用。另外，如果你不将 gStore 作为终端使用，但你需要更新数据库，应该在 Util.h 中设置 ONLY_READ 参数。

3.8.2 备份

作为 HTTP 服务器运行时，gStore 提供了备份功能，你可以在 Util 中修改备份的时间间隔。如果磁盘存在问题，备份功能将无法使用。因此，如果对安全性的需求较高，最好使用多台机器或者云做另外的备份。

3.8.3 查询

作为 HTTP 服务器运行时，gStore 为查询设定了时间限制（1 小时）。你可以在 Util 中修改这一参数，我们的建议下限是 1 分钟。

3.8.4 KVstore

KVstore 的效率会对整个系统产生重要影响，你可以根据需求和内存修改 KVstore.h 中的相关参数。

3.8.5 字符串缓存

在查询过程中，运行到 getFinalResult() 函数时，为了加速从磁盘读取字符串，gStore 为实体和字面量提供了字符串缓存。你可以根据机器的内存，在 Database.h 的 setStringBuffer() 中设定这一参数。

3.8.6 HTTP API

使用 HTTP Java API 时，需要注意效率问题（由于 jvm 的垃圾回收过程）。在使用 HTTP 协议提供的 REST API 之前，我们强烈建议仔细阅读 api/http/java 中的代码。

4 其他

4.1 第 13 章：贡献者

如果你对 gStore 有什么建议或意见，或者使用 gStore 时需要帮助，请与邹磊（zoulei@pku.edu.cn ）、曾立（zengli-bookug@pku.edu.cn ）、陈佳棋（chenjiaqi93@pku.edu.cn ）和彭鹏（pku09pp@pku.edu.cn ）联系。

4.1.1 人员

- 邹磊（北京大学）项目领导
- M. Tamer Özsu （滑铁卢大学）
- 陈雷（香港科技大学）
- 赵东岩（北京大学）
- 邓智源（武汉大学）

4.1.2 学生

曾立和陈佳棋负责 *gStore* 系统优化，彭鹏负责 *gStore* 的分布式版本。

- 韩硕（北京大学）（博士研究生）
- 曾立（北京大学）（博士研究生）
- 陈佳棋（北京大学）（硕士研究生）
- 胡琳（北京大学）（博士研究生）
- 苏勋斌（北京大学）（硕士研究生）
- 李荆（北京大学）（硕士研究生）

4.1.3 毕业生

- 沈许川（北京大学）（硕士研究生，已毕业）
- 王栋（北京大学）（博士研究生，已毕业）
- 黄睿哲（北京大学）（本科实习生，已毕业）

- 莫景辉（北京大学）（硕士研究生，已毕业）
- 彭鹏（北京大学）（博士研究生，已毕业）
- 李友焕（北京大学）（博士研究生，已毕业）

4.2 第 14 章：更新日志

4.2.1 Apr 24, 2018

曾立在 ghttp 中实现了多线程来提升服务器的性能。

此外，苏勋斌增加了 openmap ，用于 sort 和 qsort 以挖掘潜在性能。然而这一优化的结果不是很好。因此，我们取消了 openmp 的代码，还是使用标准的 sort 和 qsort 函数。

李荆在 ghttp 中为多用户和多数据库提供了支持，优化了 web 服务器和 SPARQL 查询终端的功能。

秦宗岳设计了新的 key-value 索引，取代了原来的 B+ 树。细节上，我们认为原来的 ISTree 和 IVTree 效率不够，所以我们选择完善 array+hash 方法替代了 B+ 树。需要注意的是，这些索引现在不支持并行化，所以当并行执行多个查询时，我们必须加锁以确保索引顺序访问。另外，秦宗岳设计了一种新的方法压缩 RDF 数据集中的原始字符串。例如，可以用一些特殊字符来提取并压缩前缀。

胡琳修复了 preFilter 和 Join 函数中的 bug，对回答 SPARQL 查询的性能没有影响。

4.2.2 Oct 2, 2017

SPARQL 查询现在支持 Bind 和 GroupBy ，Join 模块大幅优化。

另外，为了节省内存、支持更大的数据集（例如 freebase ，约含 25 亿三元组），弃用了 VSTree 。实际上，我们已经在一个包含 35 亿三元组的微生物数据集上完成了实验，查询时间约在 10 秒量级。

此外，我们用 HTTP1.1 协议重新设计了数据库服务器，提供了 REST 接口和 Java API 示例。用户可以直接在浏览器中通过 URL 访问服务器，我们基于此搭建了几个 SPARQL 终端（包括 freebase ，dbpedia 和 openKG）。

上面没有提及的是数据库服务器的鲁棒性，支持重启功能和查询超时处理（默认为 1 小时）。备份功能也加入了服务器中，还有重做功能，可以完成被系统崩溃所打断的更新操作。

4.2.3 2017 年 1 月 10 日

Join 模块中的 preFilter() 函数通过 pre2num 结构进行了优化，choose_next_node() 函数也是。加入了一个全局字符串缓存来降低 getFinalResult() 的开销，查询的时间大大减少。

另外，我们为所有的 B+ 树分配了不同大小的缓存（有些更加重要，且更常

用）。王力博将几个 B+ 树合并为一个，B+ 树的总量从 17 减到了 9 。这一策略不仅减少了空间开销，还减少了内存开销，并且加快了 build 过程和查询过程。

陈佳琪优化了 SPARQL 查询。例如，一些未连接的 SPARQL 查询图可以特别处理。

4.2.4 2016 年 9 月 15 日

曾立将 KVstore 根据键值的类型分成三部分，即 int2string 、string2int 和 string2string 。另外，现在支持更新。你可以插入、删除或修改 gStore 数据库中的一些三元组。实际上，只有插入和删除已经实现，修改可以通过先删除再插入实现。

4.2.5 2016 年 6 月 20 日

gStore 可以执行包含谓词变量的查询了。另外，我们研究了很多查询的结构以加快查询过程。我们重写了 sparql 查询计划，目前这个更为有效。

4.2.6 2016 年 4 月 1 日

这一项目的结构现在已经改变了很多。我们实现了一个新的连接方法，并取代了旧方法。测试结果显示，速度有所提升、内存消耗更低。我们还对 Parser/Sparql* 做了一些改变，都由 ANTLR 生成。代码是用 C 实现的，因此必须做出一些修改，这带来了一些定义问题，还有就是它太大了。

原始的 Stream 模块中存在问题，会使结果中出现一些控制字符，例如 ^C , ^V 等等。我们现在修复了这一错误，使 Stream 能够对输出字符串进行排序（内部和外部都可以）。另外，使用本地方法，现在还支持非 BGP（Basic Graph Pattern ，基本图模式）的 SPARQL 查询。

我们实现了强大的交互式控制台，称为 `gconsole` ，方便了使用者。此外，我们用 valgrind 工具测试我们的结果，处理了一些内存泄露问题。

文档和 API 也做了更改，这一点比较不重要。

4.2.7 2015 年 11 月 6 日

我们合并了一些类（例如 Bstr ）并调整了项目结构和调试系统。

另外，我们移除了大部分警告，除了 Parser 模块下的警告，它们是由于使用 ANTLR 出现的。

此外，我们将 RangeValue 模块改为 Stream 模块，并为 ResultSet 添加了 Stream 。我们还优化了 gquery 控制台，现在你可以在 gsql 控制台将查询结果重新定向至指定的文件。

由于操作复杂，我们不能在 IDlist 中添加 Stream ，但这不是必需的。Realpath 被用于支持 gquery 控制台中的软件连接，但在 Gstore 中不起作用（如果不是 Gstore 将会起作用）。

4.2.8 2015 年 10 月 20 日

我们新增了一个 gtest 工具，你可以使用它查询数据集。

另外，我们优化了 gquery 控制台。Readline 库被用于输入，而不是 fgets ，现在 gquery 控制台可以支持历史命令、修改命令和完成命令。

此外，我们发现并修复了 Database/ 中的一个错误（用于调试日志的指针在 fclose 操作后没的被设置为 NULL ，所以如果你关闭一个数据集再打开另一个数据集，系统会无法工作，因为系统认为调试日志还处于打开状态）。

4.2.9 2015 年 9 月 25 日

我们完成了 B+ 树的版本，取代了旧版本。

在测试了 DBpedia ，LUBM 和 WatDiv benchmark 后，我们得出结论，新的 B 树比旧版本更高效。对于相同的三元组文件，新版本在执行 gload 指令上花费的时间更少。

另外，新版本可以有效地处理长文本客体，三元组的客体长度超过 4096 字节在旧版本的 B 树上会导致频繁的无效分隔操作。

4.2.10 2015 年 2 月 2 日

我们修改了 RDF 解析和 SPARQL 解析。

在新的 RDF 解析中，我们重新设计了编码策略，减少了 RDF 文件的扫描次数。

现在我们可以正确解析标准 SPARQL v1.1 的语法，并可以支持用这一标准语法写成的基本图模式（BGP）SPARQL 查询。

4.2.11 2014 年 12 月 11 日

我们添加了 C/CPP 和 JAVA 的 API 。

4.2.12 2014年11月20日

我们将 gStore 作为一个遵循 BSD 协议的开源软件，在 github 上分享了 gStore2.0 的代码。

4.3 第 15 章：测试结果

4.3.1 准备工作

我们比较了 gStore 和其他几个数据库系统的性能，例如 Jena 、Sesame 、Virtuoso 等等。比较的内容是建立数据库的时间、建立的数据库大小、回答单个 SPARQL 查询的时间和单个查询结果的匹配。另外，如果内存开销很大（>20G），我们会在运行数据库系统时记录内存开销（不准确，仅用于参考）。

为了确保所有的数据库系统都能正确运行所有的数据集和查询，数据集的格式必须能由全部数据库系统支持，查询不应包括更新操作、聚集操作和与不确定谓词相关的操作。请注意，在测试回答查询所用的时间时，加载数据库的时间不计算在内。为了确保这一原则，我们先为一些数据库系统加载数据库索引，并为其他系统做准备。

这里使用的数据集是 WatDiv ，Lubm ，Bsbm 和 DBpedia 。一些由网站提供，另外的由算法生成。查询由算法生成或者是我们自己写的。表3总结了这些数据集的统计信息。

实现环境是 CentOS 服务器，内存大小为 82G ，硬盘大小为 7T ，我们使用 full_test 进行测试。

4.3.2 结果

不同数据库管理系统的性能在图5，6，7，8中显示。

注意，Sesame 和 Virtuoso 无法对 DBpedia 2014 和 WatDiv 300M 进行操作，因为数据集太大。另外，由于格式问题，我们不使用 Sesame 和 Virtuoso 测试 LUBN 5000。总的来说，Virtuoso 不可测量，Sesame 太弱。

这一程序产生的大量日志存放在 result.log/ ，load.log/ 和 time.log/ 中。看一下 result.log/ 中的文件，会发现所有的查询结果都是匹配的，load.log/ 中的文件显示，gStore 新建数据库的时间开销和空间开销大于其他系统。更准确地说，

表 3: 数据集

数据集	三元组数量	RDF N3 文件大小（B）	实体数量
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

在新建数据库时，gStore 和其他系统的时间/空间开销存在量级差。

通过分析 time.log/ ，我们会发现在复杂查询上（多变量、圈等等），gStore 比其他系统表现更好。对于其他简单查询，这些数据库系统所用的时间没有太大差异。

总的来说，回答查询时 gStore 的内存开销比其他系统更高。查询越复杂、数据集越大，这一现象越明显。

你可以在原始实验报告中找到更详细的信息。请注意，实验报告中的一些问题现在已经得到了解决。最新版的实验报告是正式实验。

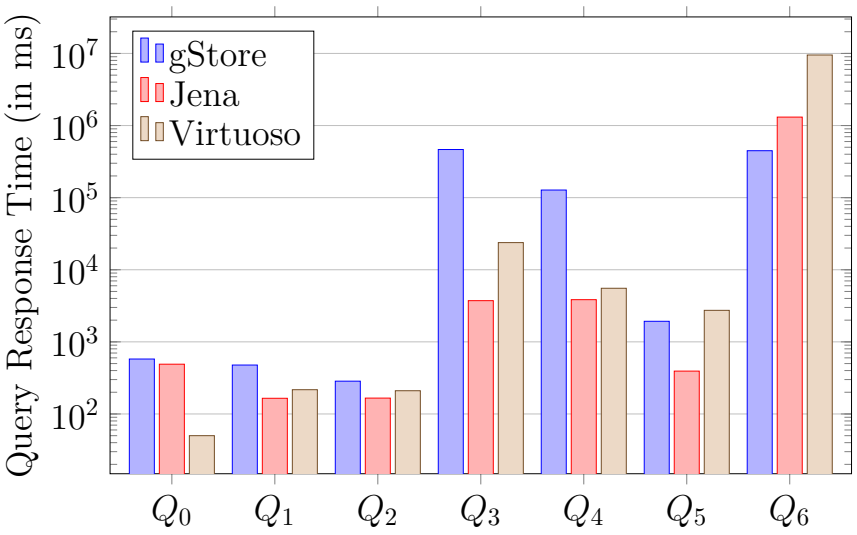


图 5: DBpedia 2014 查询性能

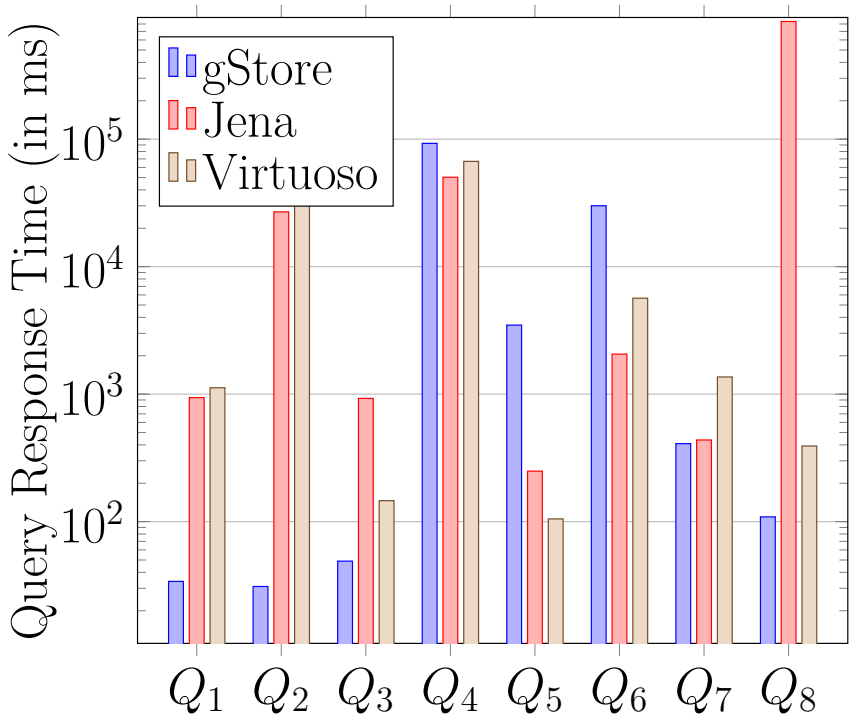
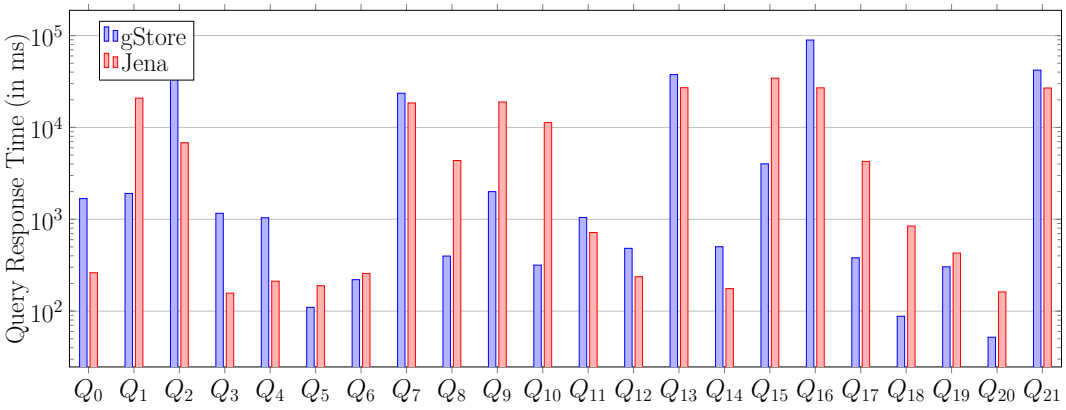
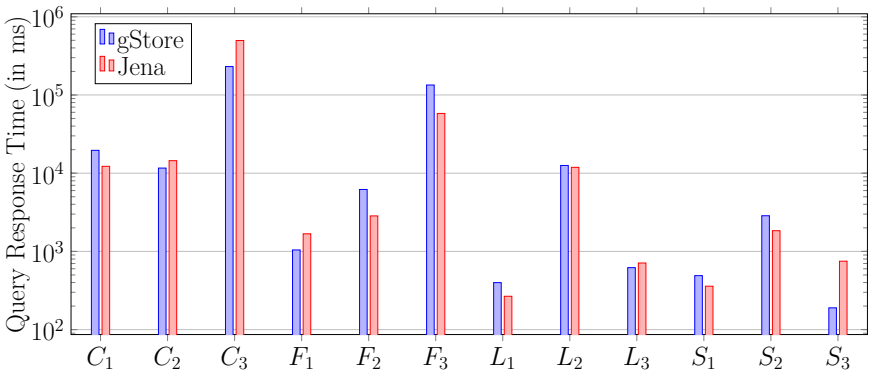


图 6: Bsbm 10000 查询性能



(a) LUBM 5000

图 7: LUBM 查询性能



(a) WatDiv 300M

图 8: WatDiv 性能

4.4 第 16 章：将来计划

4.4.1 提升内核

- 使用 GPU 或 FPGA 加速 SPARQL 的 join 过程
- 优化索引，支持并发读
- 添加数值查询函数。需要高效回答数值范围查询，空间消耗不能太大
- 定义所有常用的类型，避免不一致和高修改代价

4.4.2 优化接口

- ghttp 的可用性（错误代码、API 等等）
- 优化 socket 接口
- docker 设置

4.4.3 意见收集箱

- 使用 Parser/(antlr)!(modify sparql.g 1.1 and regenerate) 时还会有警告信息. 改变名称，避免重新定义问题，或者使用可执行程序解析
- 用 mmap 加速 KVstore ？
- Stream 的策略：85% 有效吗？考虑到抽样，分析结果集的大小再决定策略？如何支持：没有存入文件时在内存中排序；否则，在内存中部分排序，然后存入文存，再进行外部排序。

4.5 第 17 章：致谢列表

本章列出了启发我们或为项目做出贡献的人

4.5.1 zhangxiaoyang

- 添加了 python socket API (api/socket/python)

4.5.2 王定峰

fei123581321@qq.com

- 在 socket API 中增加对 python3 的支持 (api/socket/python3)

4.5.3 王力博

wlbqe@pku.edu.cn

- 压缩了 B+ 树的数量
- 为 gserver 和 ghttp 添加了备份功能
- 为 gserver 和 ghttp 添加了 REDO 功能（以防更新查询时的意外中断）
- 在服务器崩溃时自动重启
- 查询运行超过 1 小时，杀死进程

4.5.4 吕鑫

lvxin1204@163.com

- 提供 HTTP 服务器

4.5.5 邓智源

331563360@qq.com

- 提供可并行版本的 B+ 树

4.5.6 崔昊

prospace@bupt.edu.cn

- 提供查询层面的缓存

4.5.7 imbajin

- 为 docker 布署提供支持

4.6 第 18 章：法律问题

5 结语

感谢你阅读这一文档。如果有任何问题或意见，或者对这一项目有兴趣，请与我们联系。