gStore 系统使用手册

由 gStore 团队编写 [1]

2017 年 8 月 6 日

[1] 邮箱列表在第 13 章中给出。

目　录

1 前言 **1**

2 开始 **2**

2.1 第 00 章: 快速导览 2

2.1.1 开始使用 2

2.1.2 高级帮助 2

2.1.3 其他事项 3

2.2 第 01 章：系统要求 4

2.3 第 02 章：基本介绍 6

2.3.1 什么是 gStore 6

2.3.2 为什么选择 gStore 6

2.3.3 开源与授权 6

2.4 第 03 章：安装指南 7

2.5 第 04 章：如何使用 8

0. gconsole 8

1. gbuild 10

2. gquery 10

3. ghttp 12

4. gserver 13

5. gclient 14

6. 测试工具 15

7. gadd 16

8. gsub 16

9. gmonitor 16

10. gshow 17

3 高级 **18**

3.1 第 05 章：socket API 说明 18

3.1.1 简单样例 18

3.1.2 API 结构 18

3.1.3 C++ API . . . 20
接口 . . . 20
编译 . . . 21
3.1.4 Java API . . . 21
接口 . . . 21
编译 . . . 22
3.1.5 PHP API . . . 23
接口 . . . 23
运行 . . . 24
3.1.6 Python API . . . 24
接口 . . . 24
运行 . . . 26
3.2 第 06 章：HTTP API 说明 . . . 27
3.2.1 简单样例 . . . 27
3.2.2 API 结构 . . . 27
3.2.3 C++ API . . . 28
接口 . . . 28
3.2.4 Java API . . . 29
接口 . . . 29
3.3 第 07 章：web 应用 . . . 31
3.3.1 用例 . . . 31
3.4 第 08 章：项目结构 . . . 35
核心源代码如下列出： . . . 35
解析部分如下列出： . . . 37
程序如下列出： . . . 37
接口部分如下列出： . . . 38
更多细节 . . . 38
其他 . . . 39
3.5 第 09 章：出版物 . . . 40
和 gStore 相关的出版物在此列出： . . . 40
3.6 第 10 章：限制 . . . 41

3.7 第 11 章：FAQ 42
使用更新版本的 gStore 系统查询原始数据库时，为什么会出错？ 42
我试着写类似 Main/gconsole.cpp 的基于 gStore 的程序时，为什么会出错？ 42
我使用 Java API 时，为什么 gStore 报告 “garbage collection failed” 错误？ 42
我在 ArchLinux 中编译代码时，为什么报告 “no -ltermcap” 错误？ 42
为什么 gStore 报告错误称不支持一些 RDF 数据集的格式？ 42
我在 GitHub 上阅读的时候，为什么有一些文件打不开？ . . 42
为什么使用 gStore 时有时候会出现奇怪的字符？ 42
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）？ 42
在 gclient 控制台中，生成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，加载原来载入的数据库，但没有任何查询的输出（原始输出不为空？ 43
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！ 43
当我编译并运行 API 样例时，报告 “unable to connect to server” 错误？ 43
当我使用 Java API 写程序的时候，报告 “not found main class” 错误？ 43
3.8 第 12 章：技巧 44

4 其他 **45**

4.1 第 13 章：贡献者 45
人员 45
学生 45
毕业生 45

4.2 第 14 章：更新日志 46
4.2.1 2017 年 1 月 10 日 46
4.2.2 2016 年 9 月 15 日 46
4.2.3 2016 年 6 月 20 日 46
4.2.4 2016 年 4 月 1 日 46
4.2.5 2015 年 11 月 6 日 47
4.2.6 2015 年 10 月 20 日 47
4.2.7 2015 年 9 月 25 日 47
4.2.8 2015 年 2 月 2 日 47
4.2.9 2014 年 12 月 11 日 48
4.2.10 2014 年 11 月 20 日 48
4.3 第 15 章：测试结果 49
4.3.1 准备工作 49
4.3.2 结果 49
4.4 第 16 章：将来计划 53
4.4.1 提升内核 53
4.4.2 优化接口 53
4.4.3 意见收集箱 53
4.5 第 17 章：致谢列表 54
4.6 第 18 章：法律问题 55

5 结语 **56**

参考文献 **57**

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库组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协议的一个开源项目。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基础上基于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。目前只提供了源代码，也就是说要使用我们的系统，你必须对源码进行编译。

2.1.1 开始使用

本系统接口对用户友好，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：

- 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
- 在你的终端输入 `git clone git@github.com:Caesar11/gStore.git` 或使用 git GUI 获得

之后你需要对这个项目进行编译，只要在 gStore 根目录下输入 `make`，所有可执行程序就可以运行了。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/gconsole` 是一个非常好的工具，提供了你使用 gStore 需要的所有操作。请注意，所有的命令都应该在 gStore 根目录下输入。你可以在本文档的第 *04* 章【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
- 【socket API 说明】：基于 gStore socket API 开发应用

- 【HTTP API 说明】：基于 gStore HTTP 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 来实现应用时出现的常见问题。

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，请通过电子邮件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 >= 4.4.7
g++	版本 >= 4.4.7
make	需要安装
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
realpath	使用 gconsole 时需要
ccache	可选，可以加速编译过程

表 1: 软件要求

注意事项：

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。要使用更强大的正则表达式库，你不需要安装 boost 和 boost-devel。
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ccahe（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. 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，有望在 2016 年年底发布。

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

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，你只需要用一个指令对它进行编译。请在gStore根目录下运行`make`来编译gStore代码，连接ANTLR库，并生成可执行的“gbuild”、“gquery”、“gserver”、“gclient”、“gconsole”。另外，gStore的api也在此时生成。

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，在尊重我们工作的基础上开发自己的项目，输入`make tarball`指令将所有有用的文件压缩成.tar.gz 文件，易于传输。

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。

2.5 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/gconsole。因为所有的可指行程序都在 bin/中，它们可以使用了一些文件，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. 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）

```
[bookug@localhost gStore]$ bin/gconsole
Gstore Console(gconsole), an interactive shell based utility to communicate with gStor
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
```

在 gStore 根目录输入 `bin/gconsole` 来使用控制台，你会发现 `gstore>` 提示，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` 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` or `? 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` and `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！

1. gbuild gbuild 用于由 RDF 三元格式文件生成一个新的数据库。

`bin/gbuild db_name rdf_triple_file_path`

例如，我们从 example 文件夹下的 LUBM_10.n3 生成数据库。

```
[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 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` 获得关于 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。

gStore2.0 目前只支持简单 *"select"* 查询（不针对谓词）

我们还是以 LUBM_10.n3 为例。

```
[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>
```

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。
- 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 运行 ghttp 之后就可以通过 HTTP 协议访问 gStore，使用 9000 端口进行通信。服务启动后，你可以通过在浏览器中访问特定 url 或在程序中使用 HTTP API 连接到 gStore。按 Ctrl-C 停止服务。（HTTP 服务器支持多连接）。

```
[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
```

URL 规则如下：

参数：operation, db_name, ds_path, format, sparql

注意：请先完成 URL 编码，再将其发送到数据库服务器。

操作类型：build, load, unload, query, monitor, show, checkpoint

- `db_name` 数据库名称，例如 lubm
- `format` html, json, txt, csv

- `sparql` select ?s where ?s ?p ?o .
- `ds_path` 例如/home/data/test.n3

示例如下：

- 从数据集建立一个数据库：
  http://localhost:9000/?operation=build&db_name=[db_name]&ds_path=[ds_path]
- 加载一个数据库：
  http://localhost:9000/?operation=load&db_name=[db_name]
- 在当前数据库进行查询：
  http://localhost:9000/?operation=query&format=[format]&sparql=[sparql]
- 卸载数据库：
  http://localhost:9000/?operation=unload&db_name=[db_name]
- 监控服务器：
  http://localhost:9000/?operation=monitor
- 展示数据库：
  http://localhost:9000/?operation=show
- 保存当前数据库：
  http://localhost:9000/?operation=checkpoint

4. gserver gserver 是一个后台程序。会在使用 gclient 或 API 连接 gStore 时运行。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定制端口。

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

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 三元组文件生成数据库
- `load db_name` 载入一个已存在的数据库
- `unload db_name` 卸载一个数据库，但不会从磁盘上删除它，你可以再次载入

- `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. 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内或LUBM等，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，你可以到测试报告或【FAQ】一章寻求帮助。

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\_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 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\_10.n3
...
argc: 3 DB_store:lubm  remove file: data/LUBM\_10.n3
...
remove rdf triples done.
removed triples num: 99550
```

9. gmonitor 启动 ghttp 后，进入 gStore/bin/目录下，输入 `./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，进入 gStore/bin 目录下，输入 `./gshow ip port` 查看当前加载的数据库。

```
[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
```

3 高级

3.1 第05章：socket API说明

本章节将引导你用我们的socket API连接gStore,在服务器端运行gserver时使用。另外，我们还提供ghttp对应的API，请见【HTTP API说明】。

3.1.1 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++来说，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接口写自己的程序。

3.1.2 API 结构

gStore的API在gStore根目录的api/目录下，内容如下：

- gStore/api/
  - 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/ （Python 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

3.1.3 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文件新建一个数据库
// 注意，文件地址是相对gserver的地址
gc.build("LUBM10", "example/LUBM_10.n3");
// 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);
// 卸载这一数据库
gc.unload("LUBM10");
// 你也可以直接加载已存在的数据库然后进行查询
gc.load("LUBM10");
// 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。

3.1.4 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文件新建一个数据库
// 注意，文件地址是相对gserver的地址
gc.build("LUBM10", "example/LUBM_10.n3");
// 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);
// 卸载这一数据库
gc.unload("LUBM10");
// 你也可以直接加载已存在的数据库然后进行查询
gc.load("LUBM10");// 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 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/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。

3.1.5 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("LUBM10", "example/LUBM_10.n3");
// 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);
// 卸载这一数据库
$gc->unload("LUBM10");
// 你也可以直接加载已存在的数据库然后进行查询
$gc->load("LUBM10");// 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。

3.1.6 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文件新建一个数据库
// 注意，文件地址是相对gserver的地址
gc.build('LUBM10', 'data/LUBM_10.n3')
// 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)
// 卸载这一数据库
gc.unload('LUBM10')
// 你也可以直接加载已存在的数据库然后进行查询
gc.load('LUBM10')// 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。

3.2 第06章：HTTP API说明

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。

3.2.1 简单样例

我们目前提供了JAVA和C++的HTTP API。请参考`api/http/cpp`和`api/http/java`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`./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++来说，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/http/cpp和gStore/api/http/java。

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。

3.2.2 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.cpp (样例程序，展示使用C++ API的基本思路)
    - Makefile （编译并生成lib）
  - 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

3.2.3 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：

```
CHttpClient hc;
string res; //用于接收结果
int ret;
// 由一个RDF文件新建一个数据库
ret = hc.Get("127.0.0.1:9000/build/lumb/data/LUBM_10.n3", res);
cout<<res<<endl;
// 加载数据库
ret = hc.Get("127.0.0.1:9000/load/lumb", res);
cout<<res<<endl;
// 然后你可以在这一数据库上执行SPARQL查询
ret = hc.Get("127.0.0.1:9000/query/data/ex0.sql", res);
cout<<res<<endl;
//输出当前加载的数据库信息
ret = hc.Get("127.0.0.1:9000/monitor", res);
cout<<res<<endl;
//卸载数据库
ret = hc.Get("127.0.0.1:9000/unload", res);
cout<<res<<endl;
```

原始的函数声明如下：

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

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", 9000);
// 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("LUBM10", "data/LUBM_10.n3");
// 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);
// 卸载这一数据库
gc.unload("LUBM10");
// 你也可以直接加载已存在的数据库然后进行查询
gc.load("LUBM10");// 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(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);
```

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：。

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

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;
}
?>
```

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>

  <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>
```

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gserver。

3.4 第 08 章：项目结构

(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）
- KVstore/ （键-值存储，在内存和磁盘间交换）
  - KVstore.cpp （和上层交互）
  - KVstore.h
  - heap/ （结点堆，内容在内存中）
    - Heap.cpp
    - Heap.h
  - node/ （B+-树中的各种结点）
    - Node.cpp （IntlNode 和 LeafNode 的基类）
    - Node.h
    - IntlNode.cpp （B+-树的内部结点）
    - IntlNode.h
    - LeafNode.cpp （B+-树的叶子结点）
    - LeafNode.h
  - storage/ （在内存和磁盘间交换内容）
    - file.h
    - Storage.cpp
    - Storage.h
  - tree/ （实现所有的树操作和接口）
    - Tree.cpp
    - Tree.h

- 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
  - 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
  - QueryTree.cpp
  - QueryTree.h
  - GeneralEvaluation.cpp
  - GeneralEvaluation.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 (auto-generated, subtle modified manually, compressed)
  - 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（实体或文字））or literal))
  - Triple.h
  - BloomFilter.cpp
  - BloomFilter.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
- Main/ （操作 gStore 的一系列应用/主程序）
  - gbuild.cpp （导入一个 RDF 数据集）
  - gquery.cpp （查询一个数据库）
  - gserver.cpp （启动 gStore 服务器）
  - gclient.cpp （连接到 gStore 服务器并交互）

更多细节 获得对 gStore 代码的深层理解，参阅代码细节。参阅用例理解用例的设计，参阅OOA和OOD分别查看 OOA 设计和 OOD 设计。

如果你想了解运行 gStore 的流程，阅读下列内容：

- 连接到服务器
- 与服务器断开连接
- 加载数据库
- 卸载数据库
- 创建数据库
- 删除数据库
- 连接到数据库
- 从数据库断开连接

- 展示数据库
- SPARQL 查询
- 导入 RDF 数据集
- 插入一个三元组
- 删除一个三元组
- 创建账号
- 删除账号
- 修改账号权限
- 强制卸载数据库
- 查看账号权限

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【socket 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。

3.5 第09章：出版物

和gStore相关的出版物在此列出：

- Lei Zou, M. Tamer $\ddot{O}$zsu,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 $\ddot{O}$zsu, Dongyan Zhao, gStore: Answering SPARQL Queries Via Subgraph Matching, Proc. VLDB 4(8): 482-493, 2011.
- Xuchuan Shen, Lei Zou, M. Tamer $\ddot{O}$zsu, Lei Chen, Youhuan Li, Shuo Han, Dongyan Zhao, A Graph-based RDF Triple Store, ICDE 2015: 1508-1511.
- Peng Peng, Lei Zou, M. Tamer $\ddot{O}$zsu, 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. 对 SPARQL update 查询的支持还不是很好。
2. 只支持 N3 格式的 RDF 文件。下一版本会支持更多文件格式。

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 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。很容易将 RDF 数据格式转换为用于 gStore 的 N3 文件格式。

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，对于 pdf 文件请将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，生成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，加载原来载入的数据库，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，你需要卸载数据库，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。

目前没有可用的提示

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。

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）

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，有望在十月之前发布。

- 彭鹏（北京大学）（博士研究生）
- 李友焕（北京大学）（博士研究生）
- 韩硕（北京大学）（博士研究生）
- 曾立（北京大学）（硕士研究生）
- 陈佳棋（北京大学）（硕士研究生）

毕业生

- Xuchuan Shen （北京大学）（硕士研究生，已毕业）
- Dong Wang （北京大学）（博士研究生，已毕业）
- Ruizhe Huang （北京大学）（本科实习生，已毕业）
- Jinhui Mo （北京大学）（硕士研究生，已毕业）

4.2 第 14 章：更新日志

4.2.1 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。

4.2.2 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。

4.2.3 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。

4.2.4 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。

4.2.5 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）。

4.2.6 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）。

4.2.7 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。

4.2.8 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。

4.2.9 2014 年 12 月 11 日

我们添加了 C/CPP 和 JAVA 的 API。

4.2.10 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2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1，2，3，4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，在

表 2: 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

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。

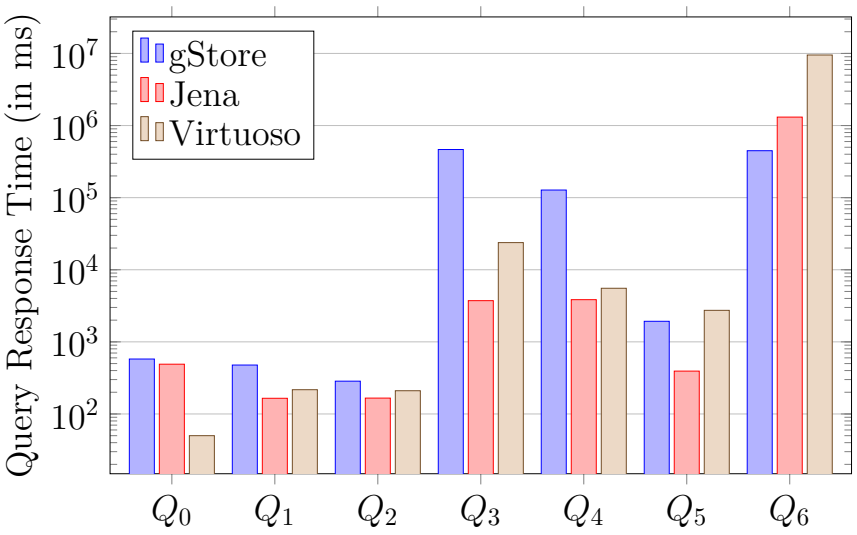


图 1: DBpedia 2014 查询性能

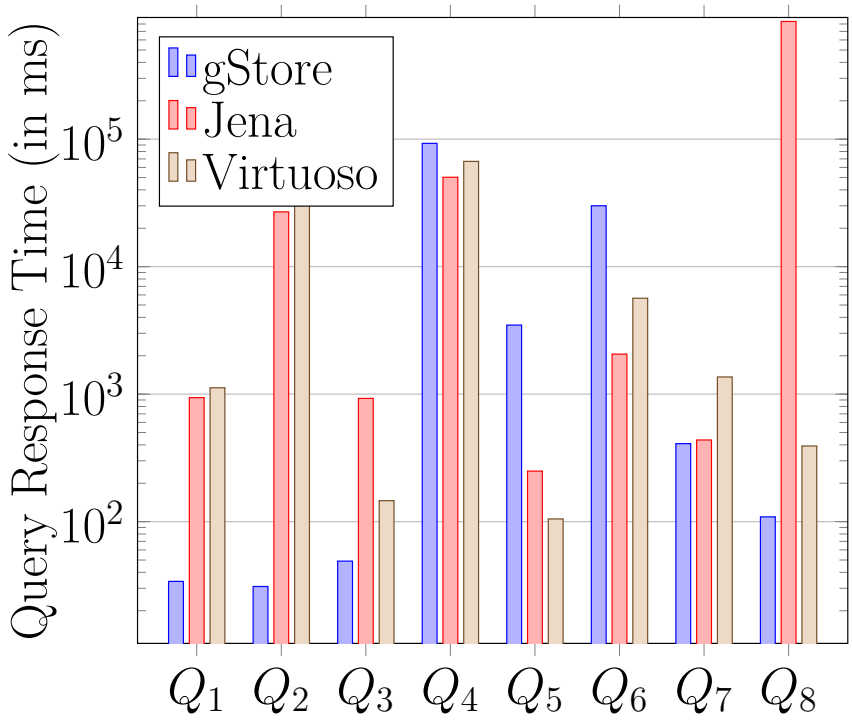
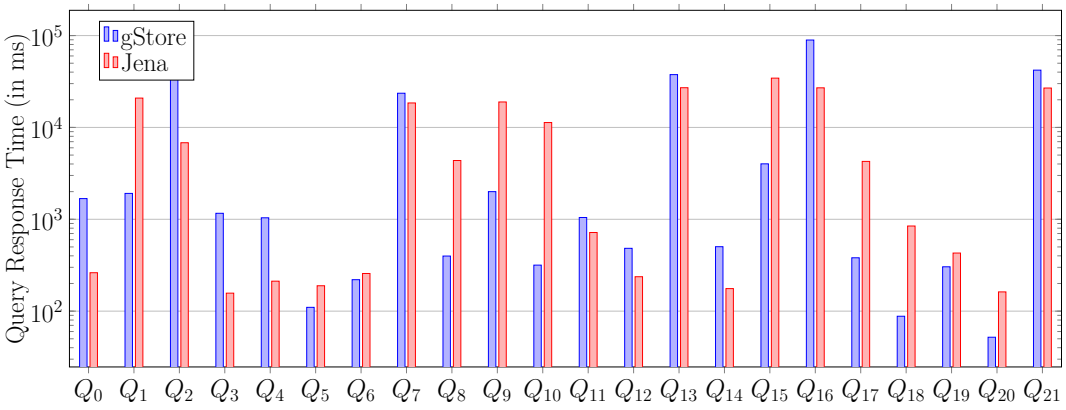
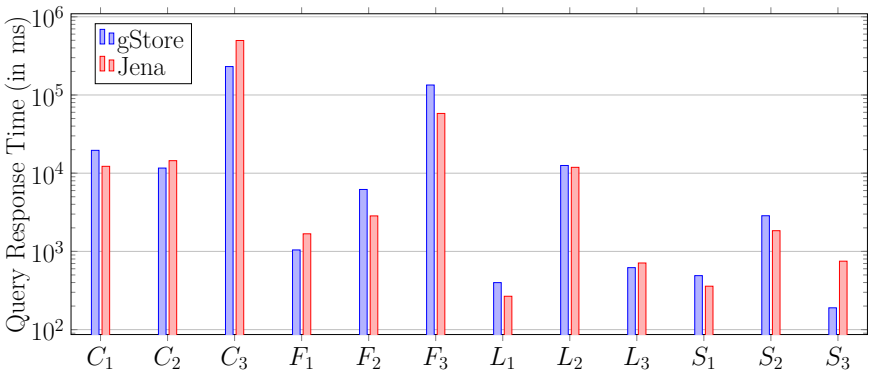


图 2: Bsbm 10000 查询性能



(a) LUBM 5000

图 3: LUBM 查询性能



(a) WatDiv 300M

图 4: WatDiv 性能

4.4 第 16 章：将来计划

4.4.1 提升内核

- 优化候选结点的连接操作。应该实现多种方法，并设计一个评分模块选择最好的方法
- 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
- 添加控制模块,启发式地为一条 SPARQL 查询选择一种索引（不总是 vstree）
- 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

- 建立一个称为 gconsole 的控制台，提供 gStore 支持的所有操作（需要解析器和自动完成）
- 写一个 gStore 的网络接口和操作用的网页，就像 virtuoso 一样

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
- 生成压缩模块（例如键-值模块和流模块），但后者只需要一次读/写，可能导致硬盘和内存中都使用压缩方法。所有对内存中字符串的操作都可以改成压缩后的操作：提供压缩/获取接口、比较函数。有很多压缩算法可供选择，那么如何选择？utf-8 编码的问题怎么处理？这一方法可以降低内存和硬盘开销，但会占用更多 CPU。然而，时间取决于同构。简单压缩不是很好，但过于复杂的压缩方法会花费太多时间，如何权衡？（合并连续的相同字符，哈夫曼树）
- 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

目前还没有人

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。

参考文献