University of Science and Technology of China

GBFS*图文件系统
可行性报告

丁峰　牛田　谢灵江　张立夫 †

2018 年 4 月 15 日

*Graph Based File System

†丁峰：PB16110386　牛田：PB15010419　谢灵江：PB16111096　张立夫：PB15020718

目录

1 理论依据

要创作一个图文件系统（GBFS），首先需要对文件之间的关系进行处理。显然，处理文件之间的关系依据是文件的内容以及它们的属性，例如考察哪些文件有相同的所有者，内容的相似性有多大。经过调研，发现若直接以文件为结点，并将有关联的文件相连来构造一个图，并在图上进行必要的操作的方案过于复杂，而且效率低下。因此，我们考虑使用评价较好的图形数据库 Neo4j 帮助我们完成这一工作。在这里，我们将文件看成结点，文件之间的关系看作边，由此可以构造出一个图。

图论与有向加权图：一个图由若干点和连接它们的边构成。有向加权图是由若干点和连接它们的有向加权边组成的。图文件系统所要实现的目标就是把一个个文件当作结点，把文件之间的亲疏关系用加权无向边表示。文件之间关系越紧密，那么这两个文件之间的边权越大。

我们可以通过比较文件属性来确定文件之间的关系，在图数据库中建立文件的图模型，并最终以图结构的形式显示出来。除此之外，还可以利用机器学习相关知识，用机器学习的方法预测用户的行为，根据用户打开一个文件前后打开的其他文件的统计结果，计算得到文件之间的关系，进而更新图数据库中文件之间的边权关系。

2 技术依据

2.1 实现概论

首先我们需要实现一个 FUSE，与内核中的 VFS 接口相配，然后 FUSE 需要将 Neo4j 封装，因为我们需要使用 Neo4j 数据库来查看、增加、删除文件之间的关系，并且将 FUSE 中与文件有关的操作，例如 read()，open() 与数据库中的操作结合起来，最终实现的目标是，能在 Neo4j 的浏览器中能清楚地看到文件之间的各种联系，而 FUSE 对文件进行的创建、更改、删除等操作也可以及时地反馈到数据库中。

2.2 Neo4j

Neo4j 是流行的图形数据库之一。Neo4j 是用 Java 语言编写的。在 Neo4j 中，使用的是具有标签属性的图模型（The Labeled Property Graph Model），它存储着四类事物：节点、关系、属性、标签。其中，节点是主要

的数据元素，节点之间通过关系相连。标签用于将节点分成组。关系可以具有一个或多个属性，而属性是基本值，就是一个字符串，属性可以被索引和约束，如下图：

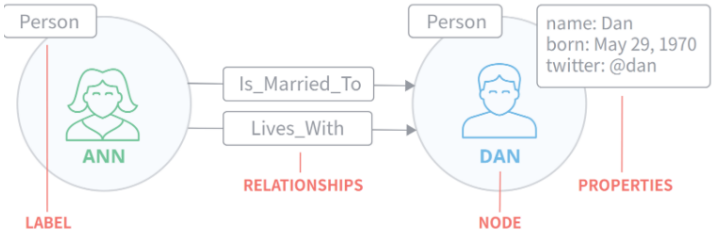

图 1: 节点关系属性

Neo4j 的浏览器是一个很棒的可视化工具，主要应用于程序和数据库开发。当用户需要可视化开发的结果时，只要给出链接分析 (link analysis) 或详细依赖关系 (detailed dependency)，就可以使用各种强大的图形可视化工具。效果如图：

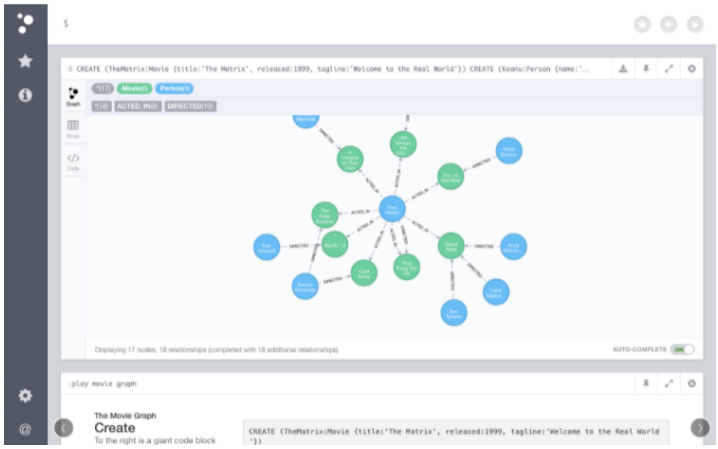

图 2: Neo4j GUI 界面

Neo4j 提供了包含很多图形算法的开放库，可以被 Cypher 写的程序调用，此库可以在 Neo4j Desktop 中安装。库中的高性能算法已经被优化过，有非常高的效率，这些算法在必要时可以帮助我们发掘隐藏在数据背后的

信息，还有助于提高用户的体验。例如，求最短路径算法或者评价访问路径的效果算法，这些算法通常用于查找文件。

Cypher 是一种图形查询语言，在语言中可以直接将节点，关系和属性描述为 ASCII 字符串，使查询、读取和识别更容易实现。在 Neo4j 图形平台上，我们将使用该语言将文件信息输入到 Neo4j 数据库中。当机器学习部分内容完成后，也可以将学习后得到的文件的边权关系放入数据库中。

3 FUSE (Filesystem in Userspace)

虚拟文件系统 VFS 作为内核子系统，为用户空间程序提供了文件和文件系统相关的接口，系统中的文件系统依赖 VFS 共存，而且也依靠 VFS 进行协同工作。VFS 的存在使得直接使用系统调用而无需考虑具体文件系统和实际物理介质，新的文件系统和新类型的存储介质在进入 linux 时程序无需重写甚至无须重新编译。

构建文件系统是一件复杂的工作，在 FUSE 出现之前，Linux 中的文件系统都实现在内核态，编写新的特定功能的文件系统十分繁琐，代码编写和调试都很不方便，就算是仅仅在传统文件系统中添加一个小小的功能，只是因为在内核中实现就要做很大的工作量。在 Linux2.6 之后的内核中增加了 FUSE 的支持之后，工作量就大大减少了。编写 FUSE 文件系统时，只需要内核加载了 FUZE 内核模块即可，无需重新编译内核。

FUSE（用户态文件系统）是一个实现在用户空间的文件系统框架，通过 FUSE 内核模块的支持，使用者只需要根据 fuse 提供的接口实现具体的文件操作就可以实现一个文件系统。

我们可以通过 FUSE 的组成来分析 FUSE 的工作原理——fuse 主要由三部分组成：FUSE 内核模块、用户空间库 libfuse 以及挂载工具 fusermount。

- fuse 内核模块：实现了和 VFS 的对接，实现了一个能被用户空间进程打开的设备，当 VFS 发来文件操作请求之后，将请求转化为特定格式，并通过设备传递给用户空间进程，用户空间进程在处理完请求后，将结果返回给 fuse 内核模块，内核模块再将其还原为 Linux kernel 需要的格式，并返回给 VFS；

- fuse 库 libfuse：负责和内核空间通信，接收来自/dev/fuse 的请求，并将其转化为一系列的函数调用，将结果写回到/dev/fuse；提供的函数可以对 fuse 文件系统进行挂载卸载、从 linux 内核读取请求以及发送

响应到内核。libfuse 提供了两个 API：一个“high-level”同步 API 和一个“low-level”异步 API 。这两种 API 都从内核接收请求传递到主程序（fuse_main 函数），主程序使用相应的回调函数进行处理。当使用 high-level API 时，回调函数使用文件名（file names）和路径（paths）工作，而不是索引节点 inodes，回调函数返回时也就是一个请求处理的完成。使用 low-level API 时，回调函数必须使用索引节点 inode 工作，响应发送必须显示的使用一套单独的 API 函数；

- 挂载工具：实现对用户态文件系统的挂载。

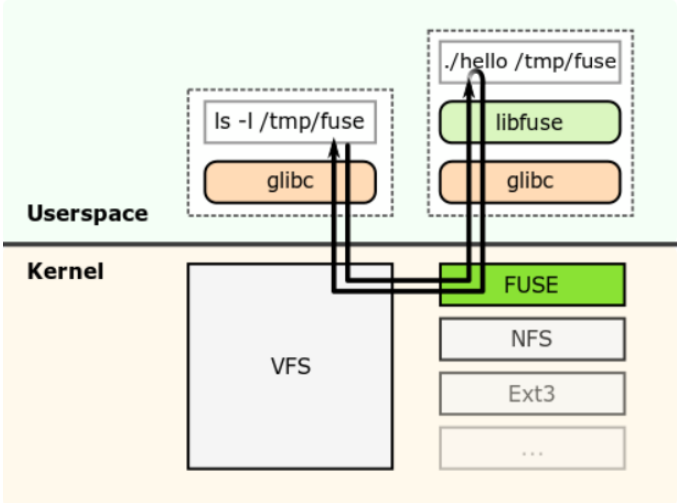

图 3: FUSE 工作流程图

在用户空间写文件系统代码的优势?

- 稳定性：文件系统文档接口
- 兼容性：在多个文件系统上运行成为可能

  for Linux/*BSD/OpenSolaris/MacOSX
- 所有用户空间的 API 都可以与文件系统的代码相容:

  Write a filesystem in Perl, Java, ···

  Debug filesystem using userspace tools

  Cache management: just let mmap() do it

- 系统更加安全

  Filesystem as unprivileged user process/daemon

- 系统更加稳定

  Crashing filesystems don't crash the kernel

  Hanging filesystem code can simply be killed

  Greedy file systems can be resource-controlled

在实现 FUSE 过程中，重点需要实现一些文件的基本操作函数，如 readdir、open、write、close 等操作，并要将 Neo4j 中的文件结点和边结合，即对权重较大的边，考虑对相关的文件进行预处理操作。比如，通过机器学习和文件相关性发现，用户在打开了文件 A 后，有很大的概率会修改文件 B，那么在用户打开了文件 A 后，文件系统根据数据库提供的信息可以对文件 B 进行预处理，加快用户对文件 B 的相关操作。

4 创新点

1 使用 FUSE，在用户态即可定制文件系统，无需进行复杂的内核操作。使用 FUSE 具有稳定性、兼容性的优势，还可以使系统更加安全。由于文件系统在用户态实现，因此具有跨平台的优势。

2 利用强大的关系数据库来处理文件之间复杂的联系，相比于其他的数据库文件系统，使用专门的关系数据库 neo4j 可以更好的管理文件之间的关联。

3 相比起以往更注重工作性能的文件系统和图数据库的利用来说，我们这次的工作是在结合两者进行用户交互体验的提升，而并非加速。对于很多用户来说，计算机工作速度的小幅改变或许微不足道，但是操作的便捷与流畅性是清晰可见的。

参考文献

[1] https://neo4j.com/

[2] http://www.oug.org/files/presentations/losug-fuse.pdf