Facebook 优化过的 HDFS[2]、阿里的 TFS[3] 以及开源社区的明星项目 CephFS[4]、GlusterFS[5]. 这些分布式文件系统在实际使用过程中在拓展性、存储规模方面都表现得十分出色. 但是, 在面向用户提供数据服务时相比于普通的文件系统缺少了灵活性, 普通的文件系统可以通过直接搭建 FTP 服务来为用户提供数据服务, 方便用户上传下载数据, 而分布式文件系统由于其自身的局限性, 无法直接提供类似的服务. 因此, 就分布式文件系统如何面向用户提供高效的数据服务, 本文提出基于 FTP 协议的解决方案——SPDScheme.

分布式文件系统向外提供数据服务接口, 但是这些接口对于用户并不友好. 需要由研发人员将这些接口进行封装, 使之成为方便用户使用的方式. 在工业领域, 目前常用的方案是将分布式文件系统挂载到普通文件系统上, 通过在普通文件系统搭建 FTP 服务端来向外提供服务, 这种方案的优势在于快速搭建, 但是劣势也十分明显, 无法对分布式文件系统的特点以及用户服务做出有针对性的调整. SPDScheme 主要设计了一个基于 FTP 协议的独立服务 SPDServer, 作用于用户和分布式文件系统之间, 根据分布式文件系统的高 IO、高吞吐量等特点进行优化, 同时根据一些用户需求进行针对性的调整. 另外通过使用一些开源技术保证服务的高可用、高并发以及可拓展性. 因此, 本文提出的 SPDScheme 具有如下几方面的技术优势.

1) 利用生成器、协程、多线程、缓冲区等技术, 针对分布式文件系统的特性设计优化数据传输的模型和算法, 极大的提高了数据服务的传输效率.

2) SPDServer 独立于分布式文件系统, 与分布式文件系统耦合程度很低, 服务挂掉不会对分布式文件系统产生任何的影响. 由于利用 Keepalived[6] 和 LVS[7] 实现高可用, 一台机器的服务出现问题, 也不会对用户服务产生任何影响.

3) 利用 LVS 对请求做负载均衡, 实现用户请求的高并发, 同时也使服务具备了非常好的拓展性, 可以随着用户请求的增加, 相应的增加服务节点, 提高服务的响应速度以及传输速度.

4) FTP 协议的认证方式是在服务启动时将明文密码配置到服务中, 使用过程非常不灵活, 而本文提出的 SPDServer 通过连接 MySQL 数据库, 将用户登录信息和权限信息保存到数据库中, 实现动态认证. 同时利用编码探测技术, 分析请求中内容的编码格式, 做到多种编码格式的兼容, 以及针对用户体验做出的其它一些调整, 对用户十分友好.

2 SPDScheme 总体架构

SPDScheme 是一个面向用户, 针对分布式文件系统的数据服务解决方案, 整体为 C/S 架构, 其中核心服务 SPDServer 作用于用户和分布式文件系统之间, 是一个相对独立的服务, 旨在为用户提供简单、方便、快捷、安全的数据服务, 主要由用户认证、文件操作、模式选择、编码兼容和日志收集等模块构成; 另外通过 Keepalived+LVS 实现服务的高可用和高并发. 方案架构如图 1 所示.

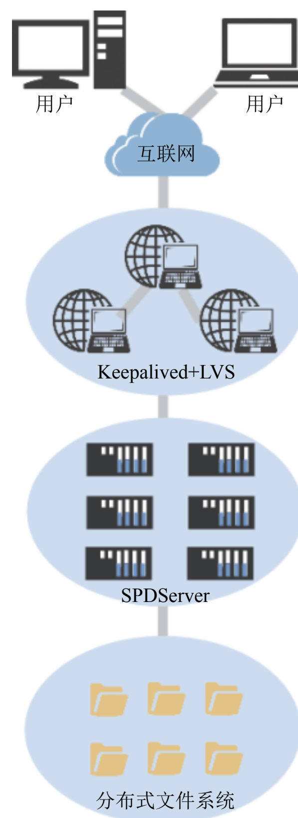


图 1 方案总体架构

2.1 SPDServer 底层框架

FTP 协议建立在客户端-服务器端模型体系结构之上[8], 在客户端和服务器端之间使用单独的控制和数据连接. 使用 FTP 协议进行数据传输, 需要有 FTP 客户端软件和 FTP 服务端的软件配合使用. 优秀的客户端软件有很多, 例如 Flashfxp[9]、Filezilla[10]、Xftp[11] 等, 用户可以选择其中任意一款作为客户端使用. FTP 服

务端工具同样层出不穷, 例如 Vsftpd[12]、Pyftpdlib[13]、Proftpd[14]、Twisted[15] 等, 这些服务端软件都有着非常好的性能表现, 但是不能直接使用在分布式文件系统上, 原因在于这些服务端软件都只能对接操作系统下的文件系统. 若要使得 FTP 服务端能够对接到分布式文件系统, 可以在现有的服务端软件之上进行传输算法、传输模型以及部分功能的重构, 会减少大量没有必要的重复研发, 因此 SPDServer 底层框架的选择非常关键. 通过对几款常用服务端软件测试对比如图 2.

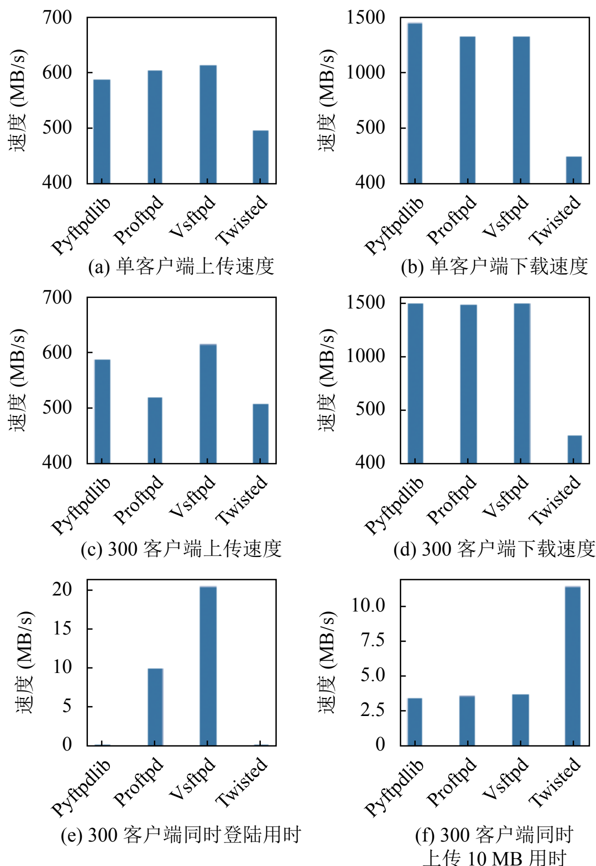


图 2　多种 FTP 软件性能比较

从测试数据中可以清楚地了解到, Pyftpdlib 在上传、下载速度以及多客户端的连接和断开所用时间上都表现十分优异, 因此选取 Pyftpdlib 软件作为 SPDServer 的底层框架.

2.2　高可用、高并发方案设计

SPDScheme 使用 Keepalived 实现服务的高可用, 使用 LVS 进行负载均衡, 实现服务的高并发, 并保证服务可以横向拓展, 在用户请求增加时, 可以通过增加服务节点的数量来增大并发量. Keepalived 的优势在于通过简单的配置就可以实现高可用, 使用灵活. LVS 的优势在于极强的负载均衡性能, 只分发请求, 不会有流量经过的特点, 保证了均衡器的 IO 性能不会受到大规模流量的影响. 同时由于其工作在网络的传输层, 上层应用协议支持广泛, FTP 协议可以很好的在 LVS 上层工作.

3　SPDServer 详细设计

在 Pyftpdlib 底层框架的基础上, SPDServer 利用缓冲区[16]、生成器[17]、协程[18]、多线程[19] 等技术, 针对分布式文件系统的特点进行了数据传输算法和传输模型的优化. 在用户认证、文件操作、模式选择、编码兼容和日志收集方面设计了全新的管理策略.

3.1　传输算法、传输模型的优化

对于评价数据服务的性能, 上传文件和下载文件速度是一项重要指标. 上传文件时数据流的传输过程是客户端将文件分片发送到 SPDServer, SPDserver 收到分片的数据再发送到分布式文件系统, 下载文件的过程与之相反. 但是这个过程中, 存在着容易被忽视的问题, 客户端的读写能力和吞吐量相比于分布式文件系统要小的多, 如果 SPDServer 只是简单的接收数据片再发送到分布式文件系统, 就会导致部分资源闲置以及没有必要的网络开销. 而 SPDServer 能够进行高速的数据传输, 原因在于使用基于生成器的协议技术、缓冲区技术以及多线程技术用来解决上述存在的问题. 同时, 整个数据流的传输过程, 可以抽象成为生产者-消费者模型, 可以把 SPDServer 抽象成为中间存放商品的仓库, 使用生产者-消费者模型的思想对其进行优化.

利用生成器特性加快传输速度. 利用生成器, 本质是一种协程的思想. 若不使用生成器, 由于文件数据是分片读取或者写入, 每次读取或者写入都要读取该文件对应的元数据信息, 但多次查询元数据信息, 极大的耗费了传输时间. 利用生成器, 就可以保存查询到的元数据在生成器上下文当中, 在上传文件时, 通过文件操作可以返回一个生成器句柄, 每次向生成器写入数据即可; 在下载文件时, 同样返回一个生成器句柄, 每次从生成器读出数据即可.

SPDServer 使用生成器避免了大量的元数据查询, 在处理每个上传下载任务后可以快速恢复到服务主程序, 极大地提高了传输速度. 具体算法步骤如算法 1.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

算法 1. 基于协程思想的上传算法

```
1. Def Generator:
2.     Metadata ← getMetadata()
3.     While DATA_PIECE:
4.       DATA_PIECE ← yield
5.       Write(Metadata, DATAPIECE)
6. Def Upload:
7.     Generator ← getGenerator()
8.     While GET_DATA_PIECE(): // 接收数据片
9.         Generator.send(DATA_PIECE)
10.    Generator.close()
```

在协程思想的基础上利用缓冲区加快传输速度. 文件是分片进行传输的, 每一片的大小会有一个具体的值. 在上传文件时, 如果数据片的大小被设置的很小, 数据就会被分为更多的片, 在向存储后端发送数据时, 每次写入都会有时间开销, 次数越多, 耗费的时间也会越多. 如果数据片的大小被设置的很大, 单次写入的数据越大, 一个数据片耗费的时间也会越多, 次数虽然变小了, 但是总的时间开销依然很大. 于是数据片过大和过小都会造成时间上不必要的耗费, 因此这里需要一个经验值让传输速度达到最佳状态, SPDServer 采用这样的设计: 在内存中开辟一片缓冲区, 按数据片的顺序暂存在缓冲区, 当缓冲区中数据的大小等于这个经验值时, 就可以将缓冲区中的内容一起通过生成器发送到分布式文件系统中.

以字符串拼接的方案作为对比, 可以更好地理解缓冲区的作用. 对分片的数据进行拼接, 需要多次的赋值操作, 本质是对内存的反复申请释放, 产生大量的时间开销. 而对于缓冲区来说, 只需要每次按顺序将数据片放入缓冲区, 这样可以极大地节省反复申请内存的时间. 具体算法步骤如算法 2.

算法 2. 基于缓冲区的上传算法

```
1. Def createBuffer:
2.     Buffer ← SYSTEM_CALL()
3.       return Buffer
4. Def Upload:
5.     Generator ← getGenerator()
6.     Buffer ← createBuffer()
7.     While GET_DATA_PIECE(): // 接收数据片
8.       if sizeof(Buffer) < BUFFER_SIZE:
9.           Write(Buffer, DATA_PIECE)
10.        else:
11.            Generator.send(Buffer)
12.            Empty(Buffer)
13.    Generator.send(Buffer)
14.    Generator.close()
15.    Buffer.close()
```

在协程和缓冲区技术的基础上, 再使用多线程技术加速传输效率. 多线程和多进程是指对任务进行异步处理, 可以加快任务的处理效率. 任务从使用资源情况看可以分为两种, 一种是 I/O 密集型任务, 即占用大量的 I/O 资源, 另一种是计算密集型任务, 即占用大量的 CPU 资源. 多进程的优点在于每个进程相互独立, 但是占用内存多, 创建、销毁、切换进程效率低, 比较适合大量计算的计算密集型任务, 不用多次切换或创建销毁. 而多线程占用内存少, 切换简单. 适合需要快速切换、创建销毁的 I/O 密集型任务, 对于基于 FTP 协议的数据服务来说, 并没有复杂的计算过程, 有的只是密集的请求和读写过程, 是 I/O 密集型任务. 于是 SPDServer 采用多线程的方式处理各种客户端请求.

多线程的使用, 相比于单个线程的处理能力, 性能提升明显; 相比于多进程, 在提高对客户端请求的处理能力的同时, 没有增加更多的时间开销和内存开销, 进一步提高了数据服务服务的响应速度. 具体算法步骤如算法 3.

算法 3. 基于多线程的上传算法

```
1. Def Work(Generator, Buffer):
2.     Generator.send(Buffer)
3.     return ok
4. Def Upload:
5.     Ex ← ThreadingPool(size)
6.     Generator ← getGenerator()
7.     Buffer ← createBuffer()
8.     While GET_DATA_PIECE(): // 接收数据片
9.         if sizeof(Buffer) < BUFFER_SIZE:
10.            Write(Buffer, DATA_PIECE)
11.        else:
12.            Ex.submit(Work, Generator, Buffer)
13.            Empty(Buffer)
14.    Ex.submit(Work, Generator, Buffer)
15.    WAIT_COMPLETED(Ex)
16.    Generator.close()
17.    Buffer.close()
18.    Ex.close()
```

3.2 认证与用户管理

Pyftpdlib 原本的认证模块是在启动服务时, 传入用户名、口令以及对应的操作权限, 在用户登录时进行对比认证, 但是服务一旦启动, 这些信息无法被修改,

© 中国科学院软件研究所 http://www.c-s-a.org.cn

若要修改认证信息, 必须重启服务[20]. 由于面向用户服务时, 用户端登录信息以及权限信息是动态变化的, 无法将所有用户的数据在 FTP 服务启动时读入. 线上的数据服务, 也不可能经常性的重新启动. 因此, 登录过程需要重新设计为动态认证, 即拿到用户的登录数据后需要和一个可以动态更新的用户数据表进行比对认证.

SPDServer 使用 MySQL 数据库存储用户信息, 同时基于该数据库向用户提供 Web 服务, 用户通过 Web 端可以在任意时间修改数据库中的信息, SPDServer 每次处理认证请求时都会检查该数据库中的认证和权限信息, 实现动态认证的效果. 另外, 数据库中指定了用户的访问路径, 做到用户与用户之间隔离, 底层存储互不影响. 为了区分用户的读写权限, SPDServer 采用一个用户名对应两个口令的方式, 两个口令分别对应只读权限和读写权限. 在认证通过之后返回权限信息, 用于检查用户的每个请求是否有足够的权限, 并予以正确响应.

3.3 文件操作管理

Pyftpdlib 原本的文件操作模块包括各种文件系统操作函数, 这些函数需要全部对接到分布式文件系统, 才能够进行文件传输, SPDServer 对这些文件操作进行了功能的重构, 最核心的部分如表 1.

表 1 功能重构的文件操作函数

函数	功能
Open	返回文件句柄
Rename	修改文件名
Delete	删除文件
List_dir	列出某路径下的文件夹和文件
Make_dir	创建文件夹
Rmdir	删除文件夹
Is_file	判断是否为文件
Is_dir	判断是否为文件夹

3.4 模式管理

FTP 协议中的服务模式分为主动模式和被动模式, 主动和被动都是针对服务端而言, FTP 协议在进行数据传输时, 客户端和服务端会建立两个连接, 一个是控制连接, 另一个是数据连接[21]. 无论是主动模式还是被动模式, 服务端建立控制连接所用的都是 21 端口, 但是在建立数据连接时, 服务端会根据请求模式的不同, 采取不同的连接方式. 主动模式下, 服务端使用 20 端口主动连接客户端的高端端口, 但是客户端多为内网用户, 在防火墙之后, 防火墙会屏蔽掉服务器发过来的主动连接请求. 被动模式下, 服务端会随机打开一个高端端口, 被动等待客户端发过来的连接请求, 但是 FTP 服务需要开放到公网, 服务器的端口为了避免受到攻击不能任意开放. 两种连接方式都有着不能忽略的弊端, SPDServer 采用了折中的办法, 即指定被动模式下用于建立连接的高端端口区间. 固定的区间, 降低了受到攻击的风险, 也避免了一部分客户端在防火墙之后不能建立连接, 保证了数据服务的稳定.

3.5 编码管理

Windows 系统默认编码为 GBK, 而 Pyftpdlib 底层框架默认编码为 UTF8, Windows 用户在访问服务时, 客户端发送请求为 GBK 编码格式, 而服务端处理请求时, 会使用默认的 UTF8 编码格式进行解码, 两者属于完全不同的编码类型, 于是产生了乱码问题. 如果只是将 SPDServer 的默认编码修改为 GBK 编码, 那么当 Linux 系统上的客户端访问时, 同样会出现乱码问题, 因为 Linux 默认编码是 UTF8. 因此, 选取一种编码格式作为默认编码, 并不能够解决多平台兼容问题. SPDServer 提出通过智能检测技术来处理编码问题的方法.

Chardet[22] 是一个非常优秀的字符编码识别方法库, SPDServer 使用 Chardet 工具对接收到的请求编码进行智能判断, 准确地获取接收到请求的编码类型, 有针对性的对接收到的请求进行解码. 同时, 在响应该请求时也使用相同的编码对客户端进行反馈. 因此, SPDServer 可以处理任意编码的请求, 有着十分强大的编码兼容能力, 能够避免使用不同操作系统导致的编码问题.

3.6 日志策略

日志是服务运行时不可缺少的部分, 好的日志策略可以保证系统的平稳运行, 在出现问题时, 迅速准确地定位到系统问题所在[23]. SPDServer 拥有单独的日志管理模块, 该管理模块支持两种输出, 一种是标准输出到屏幕, 另外一种是输出到文本; 线下测试使用输出到屏幕, 方便研发调试, 线上运行环境输出到文本, 方便大量存储、记录用户的行为并追踪定位问题. 另外, 线上环境的日志输出到文本, 以文本的大小为单位, 当文本的大小达到设定值时, 文本自动分割打包, 新产生的日志继续写入到空的文本中. 当文本的大小又达到设定值时, 继续自动分割打包, 依次类推. 这样可以留存长时间的日志记录, 避免了日志写入单一文件, 导致文件过大不好处理, 又可以保证日志的时限.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

4 SPDScheme 实际运行效果

一个方案的实际运行效果, 必须在实际项目中使用, 这样才能够全面、准确的了解. 中国科技云 iHarbor 存储系统是一个带有文件目录树的分布式对象存储系统, 本质上可以作为一个分布式文件系统使用. iHarbor 使用了本文提出的方案 SPDScheme. 本文对 iHarbor 存储系统的数据服务进行了兼容性测试、性能测试、稳定性测试以及并发拓展测试. 测试结果表明, 单个数据传输连接可以达到 200 MB/s 的传输速度, 在兼容性、稳定性、可拓展性等方面, 均表现优秀. 具体测试数据过程详见下文.

4.1 实验环境

测试所用的客户端与 iHarbor 数据服务在同一网段内, 均为万兆光纤网络, 可以避免网络带宽不足造成的瓶颈, 并测试出数据服务实际性能的上限. 存储设备方面, 客户端和 iHarbor 底层存储均使用普通机械硬盘. 另外, 在性能测试环节, SPDServer 部署在单个节点上, 测试数据基于单个 SPDServer. 测试使用的 SPDServer 配置均为 8 核 CPU, 8 GB 内存. 在 LVS 支持的范围内, 每增加一个新的 SPDServer 节点, 就会增加和测试 SPDServer 节点同样的处理能力, 损耗可忽略不计.

4.2 兼容性测试

表 2 是 FTP 客户端兼容性测试结果, 测试方法是在不同的操作系统平台上, 使用多种不同的客户端工具, 访问中国科技云 iHarbor 存储系统数据服务, 查看服务是否可以正常响应. 测试使用的客户端几乎包含了各大平台当前常用的客户端工具. 测试结果表明数据服务在客户端的兼容性方面考虑地比较全面, 可以支持主流的 FTP 客户端工具.

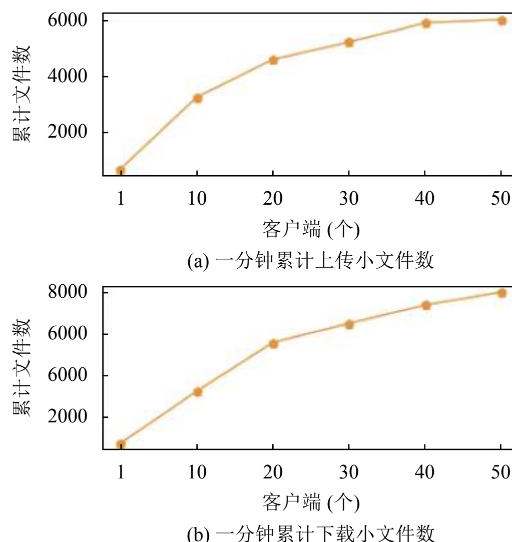
表 2 FTP 客户端兼容性测试

FTP客户端	测试结果
FlashFXP	正确响应
XFTP	正确响应
LFTP	正确响应
GFTP	正确响应
Filezilla	正确响应
Windows网络位置	正确响应

4.3 性能与稳定性测试

性能与稳定性的测试用例主要考虑到客户端个数、文件个数、单个文件大小等因素.

测试在不同并发客户端数下, 一小时可以上传或者下载的小文件数量, 用于测试的小文件的大小都在几个字节左右. 客户端的数量从 1 个到 50 个. 测得的结果如图 3, 横坐标表示并发客户端数, 纵坐标表示用时 1 分钟可以上传或者下载的小文件数目.



(a) 一分钟累计上传小文件数

(b) 一分钟累计下载小文件数

图 3 测试小文件上传下载情况

数据统计了多个客户端传输过程中的文件累计传输数量. 可以看到, 随着客户端并发数的增大, 并没有过多的影响到数据服务的性能. 同时, 没有出现明显的波动情况, 验证了数据服务的稳定性. 在单客户端下, 可以在 1 分钟内上传 600 个以上的小文件; 在 50 个客户端同时上传的情况下, 也能达到将近 6000 个小文件, 效率很高.

测试在不同并发客户端数下, 上传或者下载单个 10 GB 大文件用时. 测试结果如图 4, 横坐标表示并发客户端数, 纵坐标表示上传或者下载单个 10 GB 大文件的用时.

数据统计了在多个客户端传输大文件过程中的最长用时, 最短用时, 平均用时. 可以看到, 随着客户端的增多, 数据服务的效率稍有下降, 但是总的来看服务依旧是高效、稳定的.

4.4 高可用与并发量测试

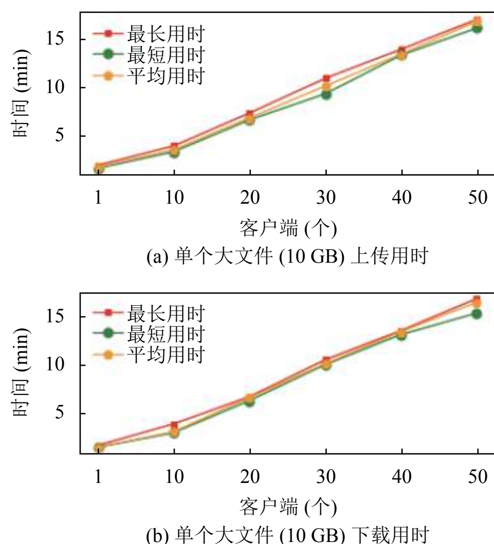
对 iHarbor 数据服务的高可用和并发测试情况如下. 在服务运行时, 人为停掉一个 LVS 节点和一个 SPDServer 服务, 数据服务不受任何影响.

用并发测试软件 JMeter 对启动单个 SPDServer 的数据服务进行并发测试. 在 1 s 内并发下载 100 KB 的文件, 可以保证 420 个连接的高效服务, 吞吐率保持在 36 req/s. 受硬件资源限制, 本次实验测试的最大规模为 10 个节点. 并且在 LVS 支持的范围内, 每增加一个新的 SPDServer 节点, 就会增加和测试 SPDServer 节

www.c-s-a.org.cn

© 中国科学院软件研究所 http://www.c-s-a.org.cn

点相同的处理能力, 损耗可忽略不计, 可以通过增加节点的方法, 增大并发量.



(a) 单个大文件 (10 GB) 上传用时

(b) 单个大文件 (10 GB) 下载用时

图 4 测试大文件上传下载情况

5 结束语

随着存储技术的不断发展, 越来越多的应用需要建立在方便、快捷的数据服务之上. 同时, 越来越多的需求也在被用户提出来, 这些需求对存储环境的拓展性、易用性和可靠性提出了更高要求. 本文提出的解决方案经过运行在真实系统上, 有效性、可靠性、高效性得到验证, 能够为用户提供优质的数据服务.

接下来将继续研究数据传输模型以及算法中的瓶颈, 以及 SPDServer 中各个模块的解耦, 如何根据用户需求在拓展功能时更加的方便灵活. 同时希望本文提出的解决方案 SPDScheme 可以应用到更多的存储系统架构中.

参考文献

1 Ghemawat S, Gobioff H, Leung ST. The google file system. Proceedings of the 19th ACM Symposium on Operating Systems Principles. Bolton Landing, NY, USA. 2003. 29–43.

2 Shafer J, Rixner S, Cox AL. The hadoop distributed filesystem: Balancing portability and performance. 2010 IEEE International Symposium on Performance Analysis of Systems & Software. White Plains, NY, USA. 2010. 122–133.

3 Alibaba Taobao. TFS. https://github.com/alibaba/tfs.

4 Weil SA, Brandt SA, Miller EL, *et al*. Ceph: A scalable, high-performance distributed file system. Proceedings of the 7th Symposium on Operating Systems Design and Implementation. Berkeley, CA, USA. 2006. 307–320.

5 何华. GlusterFS 的数据分布策略与性能优化研究 [硕士学位论文]. 长沙: 国防科学技术大学, 2013.

6 盛乐标, 周庆林. 虚拟机环境中 Keepalived 虚拟 IP 自动漂移的研究. 电子技术与软件工程, 2019, (5): 153–154.

7 Wei Q, Xu GL, Li YL. Research on cluster and load balance based on linux virtual server. International Conference on Information Computing and Applications. Tangshan, China. 2010. 169–176.

8 Gien M. A file transfer protocol (FTP). Computer Networks (1976), 1978, 2(4–5): 312–319.

9 苏悦. FlashFXP 使用有秘技. 电脑知识与技术, 2006, (7): 30–31.

10 Woodraska D, Sanford M, Xu DX. Security mutation testing of the FileZilla FTP server. Proceedings of the 2011 ACM Symposium on Applied Computing. Taichung, China. 2011. 1425–1430.

11 黄仕凰. 基于 FTP 协议的客户端软件开发. 科技信息, 2008, (21): 164–166. [doi: 10.3969/j.issn.1001-9960.2008.21.130]

12 刘陆民, 董园飞. 基于 vsftpd 虚拟用户设置的研究与实现. 无线互联科技, 2018, 15(5): 17–18. [doi: 10.3969/j.issn.1672-6944.2018.05.009]

13 Extremely fast and scalable Python FTP server library. https://github.com/giampaolo/pyftpdlib.

14 王钊, 唐志, 康征. 建立 proftp 服务器并实现用户访问目录控制和磁盘限额的探索与实践. 数字石油和化工, 2009, (10): 87–90.

15 Twisted Matrix Lab. https://twistedmatrix.com/trac/.

16 Li HL, Thng ILJ. Edge node buffer usage in optical burst switching networks. Photonic Network Communications, 2007, 13(1): 31–51.

17 Radošević D, Magdalenić I. Python implementation of source code generator based on dynamic frames. 2011 Proceedings of the 34th International Convention MIPRO. Opatija, Croatia. 2011. 969–974.

18 Spivey M. Faster coroutine pipelines. Proceedings of the ACM on Programming Languages, 2017, 1(ICFP): 5.

19 Roth A, Sohi GS. Speculative data-driven multithreading. Proceedings HPCA 7th International Symposium on High-Performance Computer Architecture. Monterrey, Mexico. 2001. 37–48.

20 张欣艳. 流模式下 FTP 文件传输效率分析及改进. 智能计算机与应用, 2019, 9(2): 126–129. [doi: 10.3969/j.issn.2095-2163.2019.02.029]

21 赵学作. 打造安全可控的 FTP 服务器. 网络安全和信息化, 2019, (7): 125–127.

22 Dan Blanchard. Chardet. https://github.com/chardet.

23 Sree TR, Bhanu SMS. Secure logging scheme for forensic analysis in cloud. Concurrency and Computation: Practice and Experience, 2019, 31(15): e5143.

© 中国科学院软件研究所 http://www.c-s-a.org.cn