基于对象存储的 rsync 实现

DEMO地址: https://github.com/kaiakz/rsync2os/

项目概述

本项目使用了rsync协议的子集, 在兼容原有的rsync的前提下, 实现了一套面向对象存储并且兼容传统文件系统的rsync客户端rsync2os. 本项目已经有了初步的实现, 并在github开源.

项目背景

Rsync sender的工作流程:

1. 遍历指定目录下所有文件, 收集元数据并生成文件列表, 然后把文件列表发送给reciever
2. 接收到reciever的请求(文件的index以及block checksum)
3. 根据block checksum向receiver发送数据

Rsync receiver的工作流程:

1. 获取到文件列表后, 遍历所有相关文件, 并读取文件元数据还有内容, 据此生成block checksum
2. 接受sender的数据, 并创建临时文件.每一段数据的开头都有一个标识, 如果这个标识大于0, 该数据内容会直接写入临时文件内, 如果小于0, 则去旧文件复制内容写入临时文件. 等于0则一个文件接收完毕.

这个同步过程涉及了大量的文件读写, 尤其是block checksum阶段, rsync需要逐段读取文件内容计算弱哈希以及强哈希, 更需要花费不少的CPU资源. rsync的这些特性需要对文件的随机读写, 这也限制了它只能在传统的文件存储上面工作.

项目实现

精简rsync协议

在rsync中, receiver会根据对比本地文件和从服务端获取的文件列表, 不同的处理情况如下:

(1) 如果文件不存在, 就会跳过checksum, 构造特定的数据包(长度为20bytes): 文件列表index + 0 block count + 0 block length + 0 checksum length + 0 block remainder + 空block, 这样可以向sender请求该完整的文件.

(2) 如果文件存在, 且文件大小和修改时间都与文件列表一致, 则跳过, 不需要发任何数据.

(3) 如果文件存在, 但文件大小或修改时间不一致, 则开始根据文件内容计算checksum, 以获取部分更新.

这样可以可以做到更小粒度的更新, 只更新修改过的部分, 避免完整文件的传输, 节约了带宽. 但这也对资源有了更高的要求.

考虑到这一点, 我对rsync协议进行了精简, 去掉了block checksum的部分, 对于上述的情况(3), 一律请求下载全文件, 同情况(1), 传输的粒度是一整个文件.

这个rsync协议子集兼容原有的协议, 并且避免了block checksum对CPU以及硬盘资源的占用. 获取到整个文件以后, 我们可以提交给不同的存储后端, 文件储存或对象存储.

文件列表缓存机制

避免了block checksum, 现在我们可以进一步优化性能. rsync的receiver收到文件列表之后将会对目录进行一下遍历, 读取文件的元数据, 进行对比.

受到rsync-huai以及HERMES项目的启发. rsync的客户端也可以通过加入文件列表缓存机制, 来避免上述的遍历过程.

文件列表是扁平的, 信息如下(来自于真实的数据, 不包括GID,UID和MDEV数据):

Path SRPMS/Packages/l/libcdaudio-0.99.12p2-17.el7.src.rpm

Size 362993

MTIME 1391577061

Mode 33204

Path SRPMS/Packages/l/libmacaroons-0.3.0-1.el7.src.rpm

Size 52528

MTIME 1529207910

Mode 33204

Path SRPMS/Packages/l/lugaru-1.2-2.el7.src.rpm

Size 34534518

MTIME 1489067575

Mode 33204

把文件列表转换为一个哈希表, 使用完整的Path+文件名作为key, 其余信息存储为value.

我们总是缓存上一次服务器发送的文件列表.因为当本地文件同步完成后, 这时候文件列表和本地的文件基本是一致的, 下一次向服务器请求更新时, 我们只需要对比这次获取的文件列表以及上次的文件列表, 得出文件的差异, 结合上面叙述的rsync协议子集, 可以做

到从远端下载本地缺失的文件, 删除远端没有的文件, 下载修改过的文件并且替换旧文件.

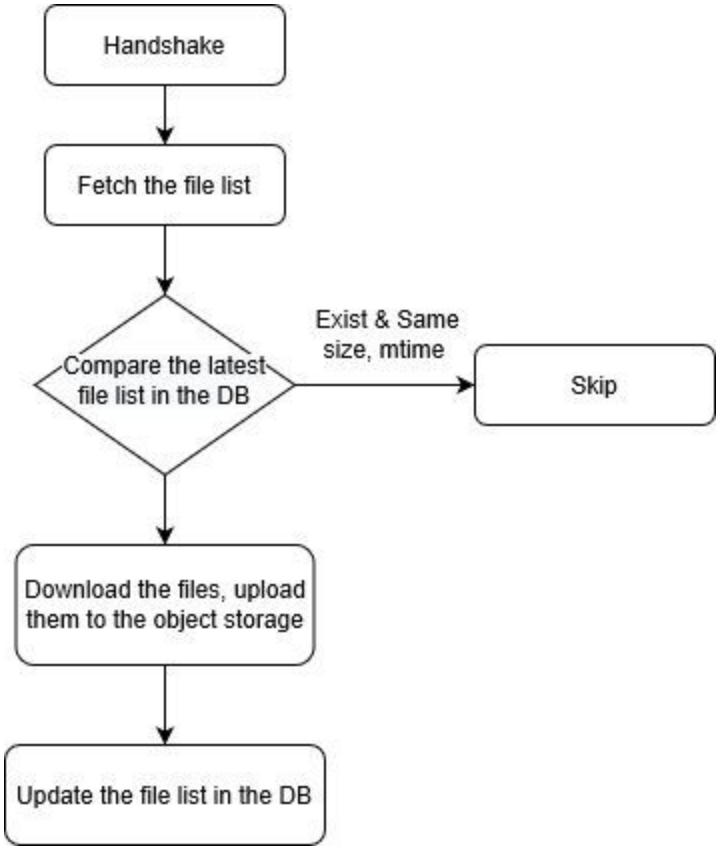
(注: 前提是一致性, 上述情况只适用于存储只有rsync2os这一个写者的情况, 当不得已需要修改文件时, 需要同时删除缓存的相关项目, 这里可以提供一个删除功能. 当一致性无法保证时, rsync2os会重新执行全量下载, 确保本地文件与缓存一致)

这些缓存数据将会保存到数据库中, rsync2os运行时会读取到内存中.

PS: 进一步来说, 在文件列表的缓存机制下, 我们也可以作为一个server向下游提供服务. 类似于rsync-huai, 但也同时精简了block checksum, 假装每一个block都不一样, 强制推送数据. 该功能暂时不在本项目的考虑内.

Rsync2os实现方案

rsync2os使用go语言开发.



A) 支持rsync://网络协议

格式为rsync://host[:port]/module[/path], 必须的参数是host以及module, path可选, 而port默认为873. host的IP地址可以通过查询DNS获得.

B) 与rsync server建立连接, 发送握手数据

rsync2os像正常的rsync客户端一样, 建立TCP连接后首先向服务器发送协议版本, 以及请求的module.

C) 发送argument列表

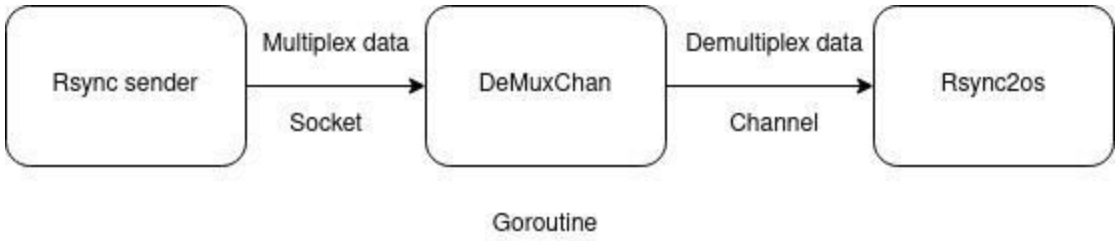
rsync2os不需要GID UID等数据, 所以固定发送argument为—server--sender-l-p-r-t..

D) 发送exclude文件列表

rsync2os在发送完argument后发送exclude文件列表. 其实这个exclude过程也可以在本地获取到文件列表后模拟.

E) De-Multiplex

从这里开始所有来自于sender的数据都会经过特定的Multiplex方式加上数据头(TAG), 一般TAG的数字为7(MSG_DATA + MSG_BASE). 不管是rsync还是openrsync在这方面的实现都过于复杂, 因为它们的读写都需要根据上下文并且记录状态. rsync2os重新实现了De-Multiplex, 借助Go语言的协程DeMuxChan自动处理Multiplex包, 把数据重新整理为流, 通过channel提供给上层.



F) 接收解析文件列表

rsync2os开始接收文件列表, 文件列表是根据我们发送的aurgment决定的, 解析方式也因此不同.完成后根据Path对文件列表进行字典序排序. 根据文件mode, 选择Regular Files, 与上次的文件列表进行对比, 上游的新文件或者修改过的文件放入下载队列, 上游不存在的文件则进入删除队列.

G) 请求文件

遍历下载队列, 向上游发送特定的数据包(长度为20bytes): 文件列表index + 0 block count + 0 block length + 0 checksum length + 0 block remainder + 空block. 发送结束请求-1, 然后开始下载.

H) 接收文件

在rsync中, sender对于在发送每一个文件之前, 都会先发送该文件的index, 紧接着的是连续不定长的数据, 格式是(token + 数据体), 因为是请求整个文件, 所以rsync2os只需要把数据缓存在内存中, 当token为0时这个文件传输结束, rsync2os可以选择把文件写入文件存储或者对象存储中.

假如一个文件过大, 对于一个支持分段上传的对象存储系统, rsync2os可以一边下载文件一边进行文件分段上传.

I) 对象存储后端

rsync2os理论上也能提供传统文件存储的支持, 但这里主要集中讨论对象存储作为后端的情况. rsync2os计划使用storage作为对象存储访问中间件, 从而提供对多种对象存储的支持.

rsync2os使用自建的minio对象存储测试, 为每一个module创建同名bucket. 出于一致性的考虑, rsync2os使用唯一一个具有读写权限的帐号管理对象存储的文件.

J) 文件列表缓存

Rsync2os每次同步完成时会把远端最新的文件列表缓存到数据库之中, 这里使用的是boltdb, 每个bucket对应每个rsync的module(也可以说是minio的bucket), 一个key-value对应一个文件, key是每一个文件的完整的路径名(从module开始), value则包括该文件的修改时间, 大小以及mode(protobuf序列化).

这个数据库可以反映对象存储的文件情况.

获取到最新的文件列表后, 通过与上次同步成功的文件列表进行比较, 得出下载队列(不存在于本地的新文件或者修改过的文件)以及删除队列(服务器不存在的文件).

下载队列由一串文件的index组成, rsync2os根据下载队列伪造下载请求, 下载完毕后根据文件列表信息保存到minio上. 同时更新数据库的数据.

删除队列由一串对象存储的key(也是数据库的key)组成, 下载队列清空后开始删除对象存储以及数据库的对应项目. 类似于rsync的—delete-after, 为了保证原子性, 删除队列可以保存进数据库中, rsync2os每次同步前以及同步后检查删除队列, 进行文件删除以清空队列. 删除顺序为对象存储的文件, 数据库文件列表的项, 最后才是删除队列. 在进行下一步操作时务必检查前一步是否成功(对象存储还存在该文件? 数据库还存在该项?)

数据库的文件需要进行定时备份, 以及理想情况下只有rsync2os一个写者. 保证数据库的k-v与对象存储中的文件一一对应.

K) 测试

主要包括功能还有错误处理两方面:

测试增量或全量同步不同情况的目录, 包括处理大文件以及软链接.

同步过程中, 假如rsync2os意外中断, 一般是在两种情况: 文件未下载完成, 文件下载完成未修改缓存. 因为缓存是放在数据库中的, 所以原子性可以有保证. 所以这两种情况都会导致文件被重新下载然后再更新缓存.

测试rsync2os在不同阶段出现异常时是否能够完整记录, 以及正确处理(模拟原版rsync), 目的是保证数据库以及对象存储上的文件一致性.

L) 原版兼容性

rsync2os现阶段目标为将文件正确地同步至对象存储中, 鉴于对象存储以及运行机制, 只会兼容一部分的rsync选项, 不会也没必要兼容所有的rsync选项. rsync2os无法像原版rsync一样在本地不同文件夹间进行同步, 现阶段设计只考虑作为receiver客户端的情况, 不实现sender客户端.

rsync2os目前的工作形式类似于—delete-after, 实际上是不支持—delete和—delete-before的.

可选的参数为, -g, -D, -o(拉取GID, UID, MDEV参数).

项目计划表

7.1 ~ 8.14	初步实现上述功能.
8.15 ~ 9.15	项目测试, 性能优化, 修改Bugs.
9.16 ~ 9.30	编写文档以及完善代码注释.