一. 思路分析：

 以新写一个文件为例，具体的io操作为：

FUSE\_GETATTR

FUSE\_LOOKUP

FUSE\_CREATE

FUSE\_FLUSH

FUSE\_GETXATTR

FUSE\_WRITE

FUSE\_WRITE

---重复write操作

...

FUSE\_FLUSH

FUSE\_RELEASE

从上面可以看出对文件属性相关操作均一次（默认在根目录，如果文件所在目录较深则查找会多次，所以建议客户端直接使用挂载目录最好），大量操作都是写io流程，gluster默认读写块大小是128kB，假设10M的文件就需要7+80次流程调用，现在把块大小改为1M，则只需要7+10次流程调用，理论时间缩短5倍，当然事实上应用时间远不足5倍，网络、磁盘、io缓存等对其均有影响，但对于大文件来说，读写速率确实会有增长

二．代码

1.gluster

xlators/mount/fuse/src/fuse-bridge.c

4020:fino.max\_readahead = 1 << 20;

4021:fino.max\_write = 1 << 20;

5710:gf\_asprintf(&mnt\_args, "%s%s%s%sallow\_other,max\_read=1048576",

glusterfsd/src/glusterfsd.c

1517:ctx->page\_size = 1024 \* GF\_UNIT\_KB;

libglusterfs/src/iobuf.c

27: struct iobuf\_init\_config gf\_iobuf\_init\_config[]: [1024,1024]

408: iobuf\_pool->default\_page\_size = 1024 \* GF\_UNIT\_KB;

xlators/performance/io-cache/src/io-cache.h

#define IOC\_PAGE\_SIZE (1024 \* 1024) /\* 128KB \*/

xlators/performance/write-behind/src/write-behind.c

#define WB\_AGGREGATE\_SIZE 1048576 /\* 128 KB \*/

Api/src/glfs.c

88: Glusterfs\_ctx\_default\_init():ctx->page\_size = 1024\*GF\_UNIT\_KB

2.fuse内核

../fs/fuse/fuse\_i.h  
#define FUSE\_MAX\_PAGES\_PER\_REQ 256

../fs/fuse/inode.c

fc->bdi.ra\_pages = 1024\*1024/ PAGE\_CACHE\_SIZE

fuse\_kern\_chan.c

#define MIN\_BUFSIZE 0x100000

三．测试结果

通过提升客户端处理io各层的缓存页来提升性能，具体代码修改见附件一

通过2\*(8:2) 分布式纠删来检测，使用两种测试验证性能的提升比率（截图见附件二）：

1. fio -filename=1;2 -direct=1 -iodepth=1 -thread -rw=write -ioengine=psync -bs=1M -size=50G -numjobs=4 -runtime=600 -group\_reporting -name=mytest

2. iozone -r 512k -s 50g -i 0 -t 4 (可以较好的压满文件读写性能)

使用一体机内置x86节点：32G内存 cpu：2核4线程, 千兆口

|  |  |  |
| --- | --- | --- |
|  | 顺序写iozone | 顺序写fio |
| 原gluster-client | 82MB/s | 44MB/s |
| 新gluster-client | 96MB/s | 56MB/s |
| 性能提升比率 | 17% | 27% |

请注意：新版本有对内核fuse模块修改重新加载，编译的版本对应内核为3.10.0，如果安装客户端的系统是高版本则版本包中fuse也需要升级