1. 代码结构

1）DMFS

从编译得到的结果来看，dmfs是主程序。



从编译文件（build/ link.txt）来看，链接中用到的文件有

|  |  |
| --- | --- |
| 文件 | 说明 |
| src/fs/dmfs.cpp |  |
| src/fs/filesystem.cpp |  |
| src/fs/storage.cpp |  |
| src/fs/lock.cpp |  |
| src/fs/bitmap.cpp |  |
| src/fs/TxManager.cpp |  |
| src/fs/hashtable.cpp |  |
| src/fs/sha256.c |  |
| src/net/Configuration.cpp |  |
| src/net/RPCServer.cpp |  |
| src/net/RdmaSocket.cpp |  |
| src/net/mempool.cpp |  |
| src/net/RPCClient.cpp |  |
| src/tools/debug.cpp |  |

从这里看，dmfs.cpp是入口。

由于文件不太多，可以选择按文件来写简要描述。

TxManager-Transaction Manager

1. dmfs.cpp

就两个函数，一个函数用来做正常退出时的提示。主函数无限循环。

主函数里在接收到一个字符后输出一个12位的地址。

2）nrfsCli和nrfsTest

nrfsCli可以作为DMFS的客户端。

3）libnrfs.so

Libnrfs封装了对dmfs调用。

依赖的源文件：

/src/net/Configuration.cpp

/src/net/RPCServer.cpp

/src/net/RdmaSocket.cpp

/src/net/mempool.cpp

/src/net/RPCClient.cpp

/src/client/nrfs.cpp

/src/client/JavaApiImpl.cpp

/src/client/libnrfs.cpp

/src/tools/debug.cpp

/src/fs/storage.cpp

/src/fs/filesystem.cpp

/src/fs/lock.cpp

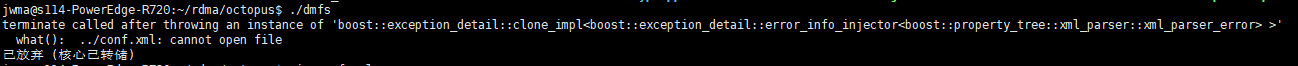
/src/fs/bitmap.cpp

/src/fs/TxManager.cpp

/src/fs/dmfs.cpp

/src/fs/hashtable.cpp

2. 运行流程



由此可以看到首先读取配置文件conf.xml。Configuration类里面读取配置文件。

但配置文件里面仅仅描述了节点的信息。

<?xml version="1.0" encoding="utf-8"?>

<address>

<node>

<id>1</id>

<ip>192.168.0.114</ip>

</node>

</address>

有了配置文件之后就是共享内存错误。



shmat是共享内存函数。

在mempool里类初始化时被调用。

在110上会出现shmget error和shmat error，但114上只有shmat error。

shmid = shmget(SHARE\_MEMORY\_KEY, DMFSTotalSize + LOCALLOGSIZE + DISTRIBUTEDLOGSIZE, IPC\_CREAT);

if (shmid == -1) {

Debug::notifyError("shmget error");

}

shmptr = shmat(shmid, 0, 0);

if (shmptr == (void \*)(-1)) {

Debug::notifyError("shmat error");

}

通过打印errno，发现110上错误原因均是

errno22 :    Invalid argument

先解决110的问题：

#define EINVAL 22 /\* Invalid argument \*/

可能原因：

参数size小于SHMMIN或大于SHMMAX

其中，SHMMAX和SHMMIN均为在内核中设置的参数。

*more /proc/sys/kernel/shmmax*

为268435456，即256MB，而程序里是2GB，显然不符合要求。

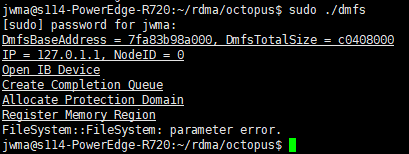
为了避免多个共享内存段，我们可以修改shmmax内核参数，使SGA存在于一个共享内存段中。  
通过修改/proc/sys/kernel/shmmax参数可以达到此目的。

|  |
| --- |
| [root@neirong root]# echo 1073741824 > /proc/sys/kernel/shmmax  [root@neirong root]# more /proc/sys/kernel/shmmax  1073741824 |

这里设为1G。  
对于shmmax文件的修改，系统重新启动后会复位。可以通过修改 /etc/sysctl.conf 使更改永久化。  
在该文件内添加以下一行  
这个更改在系统重新启动后生效  
**kernel.shmmax = 1073741824**

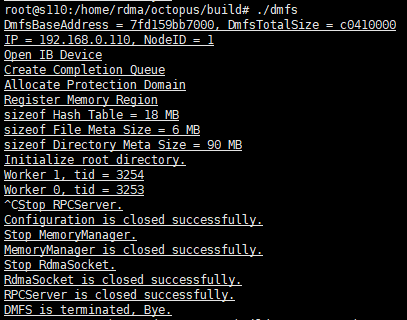
修改共享 内存最大值后，可以运行。

114上shmat是没有权限errno13 :    Permission denied，采用root权限运行后，



从图中看是配置文件错误，修改配置文件，即可。

经过上述修改后，系统可以运行。



然后，有一个客户端nrfsCli, nrfsCli也从conf.xml里读取配置文件信息，并连接服务器。

还有一个fuse客户端fusenrfs,可以通过./fusenrfs -f -o direct\_io /mnt/dmfs 来挂载使用。

二者应该是都调用了库libnrfs.so，来连接服务器的，这样可以共用一套代码。另外，

libjninrfs.so是为JAVA版本准备的，如HDFS、MapReduce。

2）分布式部署

Octopus支持分布式部署，应为分布式内存文件系统（dmfs）

在conf.xml里配置多个节点。

<?xml version="1.0" encoding="utf-8"?>

<address>

<node>

<id>1</id>

<ip>192.168.0.110</ip>

</node>

<node>

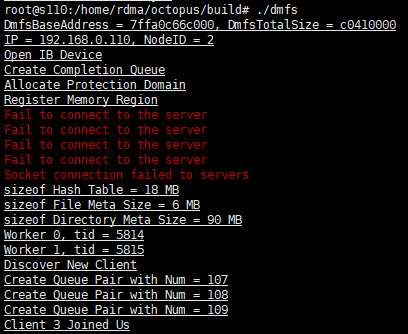
<id>2</id>

<ip>192.168.0.113</ip>

</node>

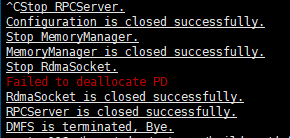
</address>

好像按nodeID有先后顺序。即113先启动的话会提示无法连接到server。



（这是113是第一个节点的情况）

结束的时候不分先后。但有时候会遇到无法释放申请的资源的问题。（这个问题在有客户端连接，而客户端异常退出，包括ctrl+c杀掉，而不是正常的quit命令退出的时候会遇到。）



猜测（更根据运行情况，没看代码）

nodeID小的会做主服务器，另外的会去连接它。

在两个节点启动的时候，使用本机做客户端，还是使用另外的机器做客户端。

不管哪种情况，现阶段在客户端都无法达到和单机上的一样的效果。运行ls就卡死。

此外，通过客户端和服务器在不同的节点（通过ib连接）时，会出现客户端无法响应的情况。而且，服务器端也无法识别客户端的加入（在服务器端的输出界面上没有显示）。

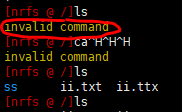
在单机上可以识别客户端的加入。(在110上不行，在113上可以)

问题1：遇到很奇怪的问题，同样的配置（两节点，113主，110从），在113上先后起两个客户端，结果不同。

第一次，ls可以，但touch file的时候卡死，等一段时间（约10秒）之后，关闭客户端（Ctrl+C），

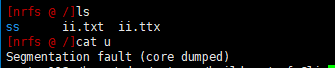
然后重启客户端，运行ls、mkdir等都可以。

在113客户端上出现奇怪一幕，ls是非法命令。



暂时未复现。

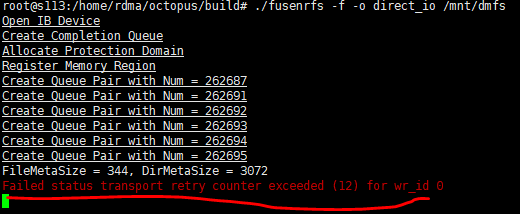
问题2：在读取不存在的文件时，会遇到错误。



问题3：

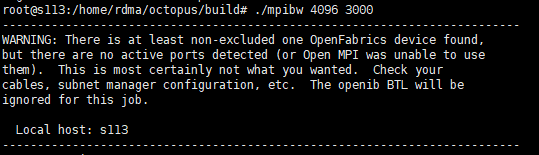
在双节点可以运行后，在同一机器（113）上使用客户端和fuse方访问。客户端出错后，出现卡死，fuse也无法访问，反之亦然。

问题4：使用vim file读取已存在且有内容的文件file时，遇到错误，这是fusenrfs的守护进程出现的消息。



应该是遇到了不支持的操作。

问题5：在运行测试程序mpibw和mpitest的时候，会出现错误



“There is at least one OpenFabrics device found but there are no active ports detected (or Open MPI was unable to use them). This is most certainly not what you wanted. Check your cables, subnet manager configuration, etc. The openib BTL will be ignored for this job.”

解决办法：以管理员身份执行命令行:

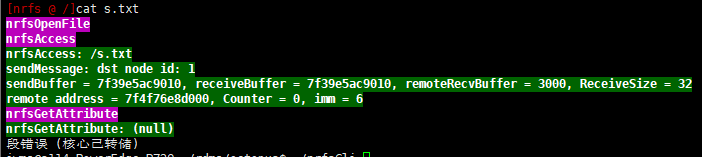
/etc/init.d/openibd restart

/etc/init.d/opensmd restart

然后，再用 ibstat 检查端口的状态，端口进入 " active" 状态，恢复正常了。

再尝试运行并行程序，先前出现的“警告信息”消失了。

问题6： 客户端执行cat file（如果文件不存在），则直接报内存错误



但是服务器端不受影响，可以继续正常执行新的客户端连接与请求。

现象1：

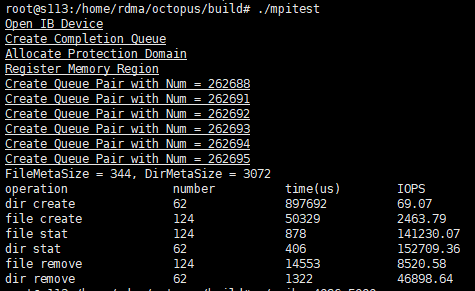
在客户端执行不同的操作，反应的服务器不一样，读取不同的文件，一个服务器有debug输出，另一个无。

正常现象，因为文件的分布按hash来的，分布在不同的服务器上。

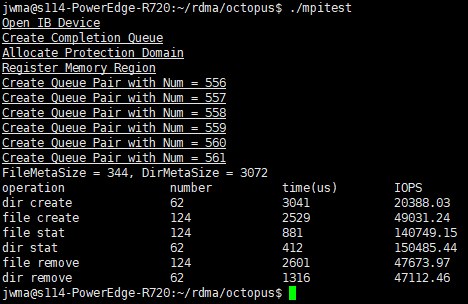
4）运行与测试结果

在客户端可以进行各种操作。

使用mpitest（测试元数据操作的），得到的测试结果。采用客户端和其中一台服务器共节点的模式

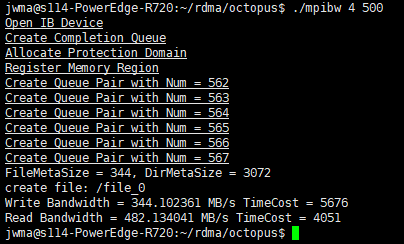


采用两服务器，一客户端的模式，客户端在单独的节点。

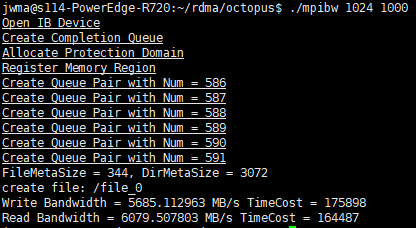


对于mpibw，两服务器，一客户端的模式，客户端在单独的节点测试结果。

其中mpibw需要两个参数，一个是块大小（以k为单位，因为代码里有乘1024），另一个数操作（读写）次数。

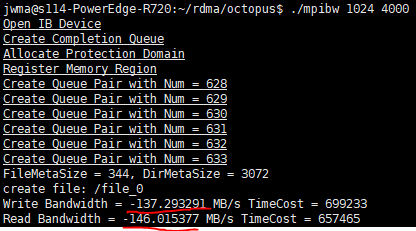


改变block\_size



问题1：

在进行读写带宽测试的时候，发现，在测试次数大于2000的时候，结果完全不正常。



这是由于存储时间的值采用int类型，会发生溢出。导致了错误的结果。

允许采用debug模式，输出的信息更多了，但不是都有用。

2）

3. 代码细节

1）单节点上空间分配

#define CLIENT\_MESSAGE\_SIZE 4096

#define MAX\_CLIENT\_NUMBER 1024

#define SERVER\_MASSAGE\_SIZE CLIENT\_MESSAGE\_SIZE

#define SERVER\_MASSAGE\_NUM 8

#define METADATA\_SIZE (1024 \* 1024 \* 1024)

#define LOCALLOGSIZE (40 \* 1024 \* 1024)

#define DISTRIBUTEDLOGSIZE (1024 \* 1024)

从代码来看，总共约3.1G映射到共享内存空间

这是代码里mempool的

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

+-------+-------+-----+-------+-------------+-------------+----------+------------+---------+

| Cli\_1 | Cli\_2 | ... | Cli\_N | SERVER\_SEND | SERVER\_RECV | MetaData | Data\_block | LogFile |

+-------+-------+-----+-------+-------------+-------------+----------+------------+---------+

/ \

--------------------------/ \---------------------------

/ \

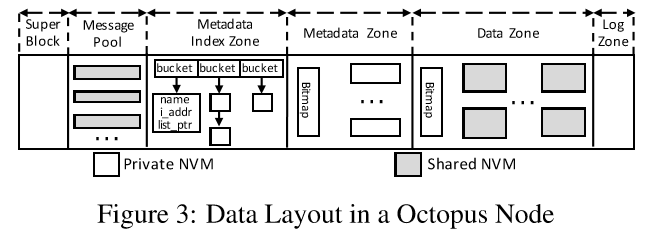
+-----------------------+-----------------------+-----+-----------------------+

| Ser\_1 (1, 2, ... 8) | Ser\_2 (1, 2, ... 8) | ... | Ser\_M (1, 2, ... 8) |

+-----------------------+-----------------------+-----+-----------------------+

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

这是论文里的内存布局



代码里的内存布局：



2）多CQ与多QP

/\* Server interconnect, use cq at 0. \*/

server互联的时候也也是使用cq[0]，即收发共用一个完成队列。

每个客户端和服务器之间的连接是两个QP。

3）~~分读写线程，处理不同的请求。~~

这一点在第一次看的时候看错了，worker线程主要用于多线程读写数据。

Queue<TransferTask \*> queue[WORKER\_NUMBER];/\* Used for Data transfer. \*/

uint16\_t TransferSignal; /\* Used to notify compeletion of data transfer. \*/

thread worker[WORKER\_NUMBER];

4）OutboundHamal与InboundHamal

这两个函数以前没有遇到过。

OutboundHamal里面有一个 内存拷贝 的操作。

二者是做搬运数据搬运的，其中Outbound与inbound对应不同RDMA操作。

数据拷贝对大文件和小文件采取不同的操作，大文件多线程。

InboundHamal从SendPoolAddr拷贝到bufferSend

OutboundHamal从bufferSend拷贝到SendPoolAddr

SendPoolAddr在RDMA read和write的时候做源地址，在bufferReceive + TotalSizeSend做目的地址(收取的消息存放处)

5）DataTransferWorker

在worker线程中执行，worker线程在RdmaSocket初始化时就被创建，一直运行。轮询task队列，并执行。

6）Octopus在设计时仍然使用了内存拷贝，但是其将所申请的约3GB内存直接用到了RDMA

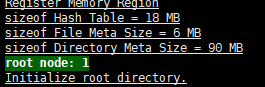
的MR区域，这应该是NIC可以访问的，无需拷贝，如果需要连接起来多个分散的内存区，可以使用sgelist，散射表。

7）共享内存可以让不同进程访问同一内存区，在客户端和服务器在同一节点的时候有很大性能优势，避免拷贝。

8）mknod操作的时候如果定位到file在remote node，就直接返回false。

9）rename一个文件会造成与两个节点通信。因为文件名改了之后hash值变了，映射到的服务器改了。这是正常现象，但是这种设计的性能开销不可忽略。

10）一些数据



11）除DataTransferWorker线程外，还有cq\_worker线程，在RPCServer初始化的时候被创建，用于轮询CQ队列。Worker数量与cqsize相当。

12）lock

锁与元数据相关，使用元数据存放基址来初始化。

LockService::LockService(uint64\_t \_MetaDataBaseAddress)

: MetaDataBaseAddress(\_MetaDataBaseAddress){}

需要对元数据上锁。

元数据地址的MetaDataBaseAddress + Address的第一个字（64位）作为锁，0表示无锁，一个非零64位数表示上锁。

uint64\_t key = (uint64\_t)NodeID;

key = key << 16;

key += ID;

key = key << 32;

该值指出了NodeID，ID根据读写有不同的值。