m\_mapKeyword: KeyID, SourceID, CKeyEntry

CKeyEntry:keyID,sourceID,LifeTime,AICH id,CKadTag

CKadTag:m\_name,m\_type

m\_mapSources: KeyID,SourceID,CEntry

CEntry: LiftTime ,CKadTag,KeyID,SourceID

搜索过程中使用到的Map

typedef std::map<CUInt128, CSearch\*> SearchMap;

SearchMap m\_mapSearches

///snow:在StartSearch、PrepareFindKeywords、PrepareLookup中赋值

typedef std::map<CUInt128, CContact\*> ContactMap;

ContactMap m\_mapPossible;

ContactMap m\_mapTried;

ContactMap m\_mapBest;

ContactMap m\_mapInUse;

std::map<Kademlia::CUInt128, bool> m\_mapResponded;

ContactList m\_listDelete;

问题1、文件名的关键字是怎么提取出来的？

2、KeywordID与SourceID是怎样分工的？ Sharefile中是否有相关信息？

3、Index中存储的关键字Map是怎样的？见PPT

4、Search的过程是？SendFindValue与StorePacket的分工

Load\_index.dat的作用

变量：CSearch::m\_uTotolLoad 、CSearch::m\_uTotalLoadResponses

m\_mapLoads

当我们有源文件需要发布时，发出KADEMILA2\_PUBLISH\_SOURCE\_REQ信息包

对方接收到该信息包，调用Process\_KADEMLIA2\_PUBLISH\_SOURCE\_REQ过程进行处理，过程再调用pIndexed->AddSources()函数进行源的添加，在添加时通过与本机存储的Indexed数据进行比对，计算出uLoad值（新的源文件时uLoad=1，其它情况uLoad=EnrtySize\*100/1000，最大值100）

if(!m\_mapSources.Lookup(CCKey(uKeyID.GetData()), pCurrSrcHash))

{

uLoad = 1;

}

else ///snow:已存在同ID的Source条目

{

uint32 uSize = pCurrSrcHash->ptrlistSource.GetSize();

for(POSITION pos1 = pCurrSrcHash->ptrlistSource.GetHeadPosition(); pos1 != NULL; )

{

Source\* pCurrSource = pCurrSrcHash->ptrlistSource.GetNext(pos1);

if( pCurrSource->ptrlEntryList.GetSize() ) ///snow:ptrlEntryList中的Entry条目不为

{

CEntry\* pCurrEntry = pCurrSource->ptrlEntryList.GetHead();

ASSERT(pCurrEntry!=NULL);

if( pCurrEntry->m\_uIP == pEntry->m\_uIP && ( pCurrEntry->m\_uTCPPort == pEntry->m\_uTCPPort || pCurrEntry->m\_uUDPPort == pEntry->m\_uUDPPort )) ///snow:IP、Port或UDPPort相一致

{

delete pCurrSource->ptrlEntryList.RemoveHead();

pCurrSource->ptrlEntryList.AddHead(pEntry);

uLoad = (uint8)((uSize\*100)/KADEMLIAMAXSOUCEPERFILE); ///snow:每文件最多个源，uLoad=该文件源数\*0.1

return true;

}

}

}

if( uSize > KADEMLIAMAXSOUCEPERFILE )

{

Source\* pCurrSource = pCurrSrcHash->ptrlistSource.RemoveTail();

delete pCurrSource->ptrlEntryList.RemoveTail();

pCurrSource->uSourceID.SetValue(uSourceID);

pCurrSource->ptrlEntryList.AddHead(pEntry);

pCurrSrcHash->ptrlistSource.AddHead(pCurrSource);

uLoad = 100; ///snow:每文件最多个源，该文件源数>1000，uLoad=100（最大值）

return true;

}

else

{

Source\* pCurrSource = new Source;

pCurrSource->uSourceID.SetValue(uSourceID);

pCurrSource->ptrlEntryList.AddHead(pEntry);

pCurrSrcHash->ptrlistSource.AddHead(pCurrSource);

m\_uTotalIndexSource++;

uLoad = (uint8)((uSize\*100)/KADEMLIAMAXSOUCEPERFILE);

return true;

}

}

并发回KADEMLIA2\_PUBLISH\_RES 信息包

SendPacket( &fileIO2, KADEMLIA2\_PUBLISH\_RES, uIP, uUDPPort, senderUDPKey, NULL);

本机在接收到该信息包时，读出uLoad值，并调用CSearchManager:: ProcessPublishResult(uFile, uLoad, true);传递uLoad值给ProcessPublishResult()进行处理。

void CSearchManager::ProcessPublishResult(const CUInt128 &uTarget, const uint8 uLoad, const bool bLoadResponse)

{

// We tried to publish some info and got a result.

CSearch \*pSearch = NULL;

SearchMap::const\_iterator itSearchMap = m\_mapSearches.find(uTarget);

if (itSearchMap != m\_mapSearches.end())

pSearch = itSearchMap->second;

// Result could be very late and store deleted, abort.

if (pSearch == NULL)

return;

switch(pSearch->GetSearchTypes())

{

case CSearch::STOREKEYWORD:

if( bLoadResponse )

pSearch->UpdateNodeLoad( uLoad ); ///snow:uLoad的值被添加到m\_uTotalLoad，m\_uTotalLoadResponses++

break;

case CSearch::STOREFILE:

case CSearch::STORENOTES:

break;

}

…

}

void CSearch::UpdateNodeLoad( uint8 uLoad )

{

// Since all nodes do not return a load value, keep track of total responses and total load.

m\_uTotalLoad += uLoad;

m\_uTotalLoadResponses++;

}

///snow:取加载的Load节点与有回应的Load节点的比值

uint32 CSearch::GetNodeLoad() const

{

// Node load is the average of all node load responses.

if( m\_uTotalLoadResponses == 0 )

{

return 0;

}

return m\_uTotalLoad/m\_uTotalLoadResponses;

}

在CSearch对象进行析构的时候，如果m\_uTotalLoad/m\_uTotalLoadResponses>20的时候，将该对象添加到m\_mapLoad

CSearch::~CSearch()

{

// Check if this search was contacting a overload node and adjust time of next time we use that node.

if(CKademlia::IsRunning() && GetNodeLoad() > 20)

{

switch(GetSearchTypes())

{

case CSearch::STOREKEYWORD:

Kademlia::CKademlia::GetIndexed()->AddLoad(GetTarget(), ((uint32)(DAY2S(7)\*((double)GetNodeLoad()/100.0))+(uint32)time(NULL)));

break;

}

}

CIndexed对象在析构的时候，将m\_mapLoads中的信息写入Load\_index.dat

CIndexed对象在初始化的时候，将Load\_index.dat读入m\_mapLoads

(CIndexed只有一个对象)

int CIndexed::CLoadDataThread::Run()

{

…

if (!m\_pOwner->m\_bAbortLoading)

{

///snow:加载load\_index.dat，示例：00 00 00 09 35 CA 58 01 00 00 00 17 7E D7 1A 51 A4 6C 77 CB 28 15 65 F9 B8 89 EA 2E 41 CB 58

CBufferedFileIO fileLoad;

if(fileLoad.Open(m\_sLoadFileName, CFile::modeRead | CFile::typeBinary | CFile::shareDenyWrite))

{

setvbuf(fileLoad.m\_pStream, NULL, \_IOFBF, 32768);

uint32 uVersion = fileLoad.ReadUInt32(); ///snow:前四个字节是版本号 01 00 00 00

if(uVersion<2) ///snow:版本号小于，只能是

{

/\*time\_t tSaveTime = \*/fileLoad.ReadUInt32(); ///snow:保存时间09 35 CA 58

uint32 uNumLoad = fileLoad.ReadUInt32(); ///snow:条目数 01 00 00 00

while(uNumLoad && !m\_pOwner->m\_bAbortLoading)

{

fileLoad.ReadUInt128(&uKeyID); ///snow:16个字节uKeyID:17 7E D7 1A 51 A4 6C 77 CB 28 15 65 F9 B8 89 EA

if(m\_pOwner->AddLoad(uKeyID, fileLoad.ReadUInt32(), true)) ///snow:4字节的加入时间 2E 41 CB 58

uTotalLoad++;

uNumLoad--;

}

}

fileLoad.Close();

}

else

DebugLogWarning(\_T("Unable to load Kad file: %s"), m\_sLoadFileName);

}

…

}