TinyDFS: a tiny distributed file system

# 项目简介

TinyDFS是殷鹏程作为分布式系统大作业，所开发的一款简单的Key/Value式分布式文件系统，整个项目基于C#+Windows Communication Foundation，实现了一个分布式文件系统所必须的上传、下载、冗余保存文件的功能。

**开发环境：**Visual Studio 2013

**开发语言：**C# 4.0

**项目主页：**<https://github.com/handesy/TinyDFS/>

# 系统架构

## 系统构成

TinyDFS由三大部分构成：客户端（**TinyDFS.Client**），命名解析主节点（**TinyDFS.NameNode**），文件块服务器（**TinyDFS.FileServer**）。不同的组件之前通过WCF，基于标准的HTTP协议进行通信。因此可以分散到不同的物理主机上执行。

### 客户端

客户端是一个简单的控制台程序。可以进行上传文件、下载文件、删除文件、列出文件列表等命令。它通过标准的HTTP协议与NameNode和块服务器通信，完成相应的操作。

### 命名解析主节点

NameNode用于进行整个分布式文件系统的名字解析。它维护着文件名与文件的所有块之间的映射信息。

TinyDFS中，文件存储的最小单位是块（**Chunk**）。对于大小超过64MB的文件，其被分割为若干大小在64MB左右的块，分散地存储在不同的块服务器中。每个块由一个128Bit的GUID来唯一标识。NameNode维护了每个文件名与其对应的块的映射。

TinyDFS中没有目录的概念，文件按名（**Key**）存取，名是一个任意长度的非空字符串，当然也可以将一个文件命名为类似于“filedirA/filedirB/filename”这样的形式。

### 块服务器

块服务器负责具体的文件块的存放。块在块服务器上以对应的GUID命名。块服务器为客户端请求块下载块内容提供了标准的WCF接口。

### 文件冗余存储

为了保证整个分布式文件系统的健壮性，每个文件块在TinyDFS上实行冗余存储的策略。**如果块服务器i上存放了块i1，那么定会存在另一台块服务器j，保存了i1的一份副本**，这样，即使块服务器i宕机，客户端仍旧可以从j处获得相应的块，从而获得完整的文件。

## 文件操作流程

TinyDFS客户端可以向整个分布式文件系统执行文件上传、下载、删除与列出所有文件等命令。

### 文件上传

Client

NameNode

**GetUploadInfo**

**ChunkingInfo**

FileServer1

FileServer2

FileServer3

**Upload Chunks**

当客户端执行文件上传命令时，其通过WCF远程调用NameNode的GetUploadInfo()方法。NameNode根据文件的长度，计算文件应该被划分为的块数，并指定每个块的GUID以及其对应的存放位置（块服务器ID）。

客户端在获得了文件分块与存储位置的信息后，将文件切割为对应的文件块，然后，直接同每个块服务器通信，远程调用块服务器的UploadChunk(Stream stream)方法将每个块上传到块服务器上。当所有的块被上传完成后，文件上传操作方结束。

### 文件下载

Client

NameNode

**GetDownLoadInfo**

**ChunkingInfo**

FileServer1

FileServer2

FileServer3

**Download Chunks**

**By GUID**

当客户端执行文件下载命令时，其通过WCF远程调用NameNode的GetDownLoadInfo(fileName)方法。得到文件的块分布信息，之后，客户端同每一个块所对应的块服务器通信，远程调用相应块服务器的GetChunkByGUID(string guid)方法，下载得到该块。

当所有的块下载完毕之后，客户端将所有块按照顺序拼接，得到原始的文件。

### 文件删除

Client

NameNode

**Delete File**

**Success**

FileServer1

FileServer2

FileServer3

**Detete Chunks**

**By GUID**

当客户端执行文件删除命令时，其通过WCF远程调用NameNode的DeleteFile(fileName)方法，NameNode按照其所维护的文件名与文件块的映射，定位每个文件块对应的文件块服务器，再调用相应块服务器的DeleteChunk方法删除相应的块。最后返回给客户端成功信息。

# 实验演示

## 实验环境

在一台主机上启动一个NameNode，4个块服务器，当然，各个块服务器可以分布到不同的主机上，因为整个系统的节点间依靠标准的Http协议来进行通讯。

NameNode Service：<http://localhost:8080/NameNodeService>

FileServer1 Service：<http://localhost:7575/FileService>

FileServer2 Service：<http://localhost:7576/FileService>

FileServer3 Service：<http://localhost:7577/FileService>

FileServer4 Service：<http://localhost:7578/FileService>

当各个组件启动后，可以看到，NameNode跟各个块服务器间维持正常的心跳通信。

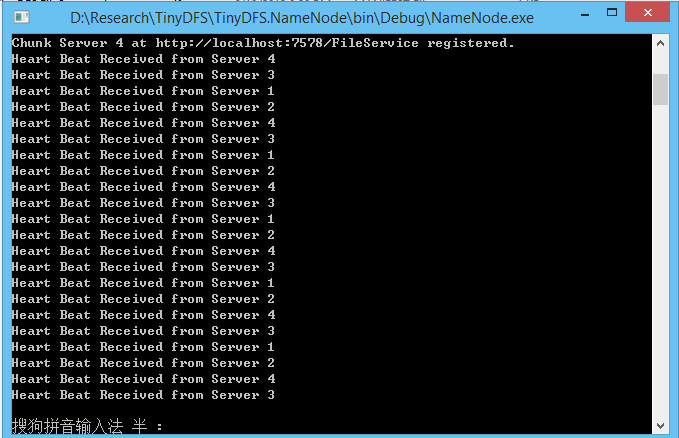


Figure 1NameNode启动界面

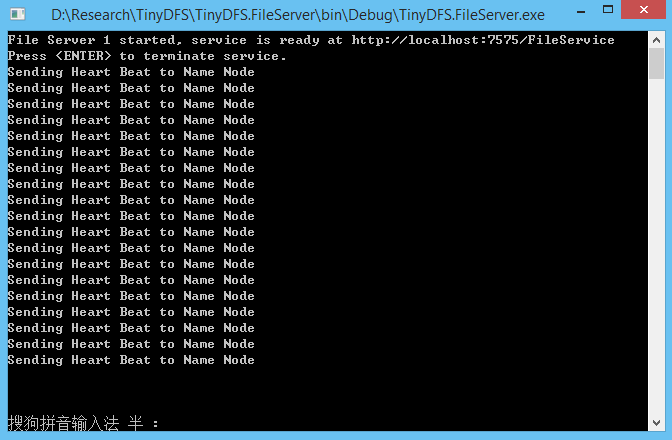


Figure 2一个块服务器的启动界面

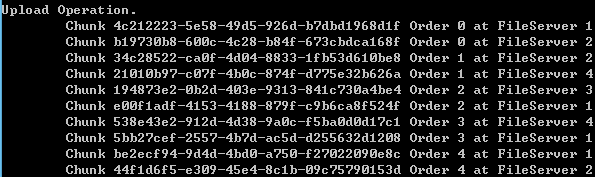
## 上传演示

下面以一个350MB大小的文件为例，演示TinyDFS的文件上传功能。

上传命令：TinyDFS.Client.exe upload {FileName}

演示命令：TinyDFS.Client.exe upload D:\Research\KB\kb.entity

上传完毕后，可见NameNode窗口显示了如下的块分布信息：



这说明，文件被分为了5份，每一份有两个块拷贝。

## 获取文件列表演示

命令：TinyDFS.Client.exe list

演示结果：



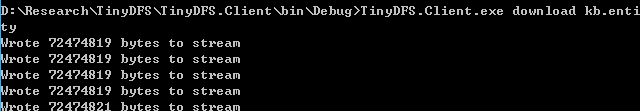
可见，刚才上传的文件现在已经保存到了TinyDFS上了。

## 下载文件演示

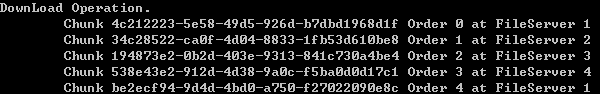
命令：TinyDFS.Client.exe download {FileName}

演示命令：TinyDFS.Client.exe download kb.entity

执行命令后，可以看到，客户端显示了5个块的下载情况。



而在NameNode的控制台里，则显示了NameNode返回给客户端的这5个块的GUID，顺序以及所在的块服务器等信息。



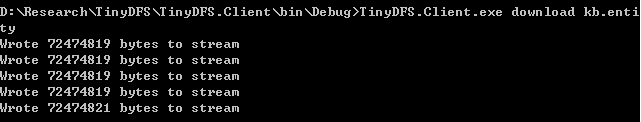
## 有宕机情况下的下载文件演示

为了测试TinyDFS的鲁棒性，模拟块服务器宕机，以检测客户端是否能够正确通过备份的块来得到完整的文件。

继续上面的测试，停止块服务器4的进程。一会儿，NameNode由于无法收到FileServer4的心跳包，显示Server4已宕机。

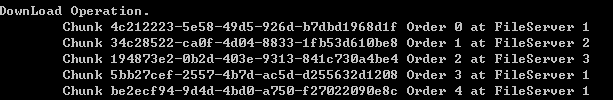


这时，再次执行上节中的下载命令。



文件正常下载。

查看NameNode为本次下载请求所返回的文件块分布信息：



可以发现，NameNode并没有使用保存在已经宕机的Server4上的块3（Chunk 3），而是使用了在Server1上的该块的备份，因此，这时候客户端仍旧能正常获取到整个文件。

### 文件删除演示

命令：TinyDFS.Client.exe delete {FileName}

演示命令：TinyDFS.Client.exe delete kb.entity

删除完毕后，接着执行list命令，程序列出空列表，整个过程如下图所示，说明文件已经在分布式文件系统上被删除。

