# FTP Upload Based on Multithreaded and Broken Transfer Resume Technology*

**Yimin Wu, Xin Meng**
School of Computer Science & Engineering, South China University of Technology Guangzhou, Guangdong Province, China
csymwu@scut.edu.cn, mengxin.x@qq.com

**Abstract** - FTP has been widely used on the Internet, the current application and research mainly focus on the FTP download technology, but upload technology research is rarely involved, especially for single file multithreaded upload technology. Now days, users often need to upload some large files to the server, however, traditional way has a low speed, and would be interrupted more easily, leading to retransmit frequently, which has a bad influence on efficiency. This paper proposes a method, using multithreaded and broken transfer resume technology, to solve this problem, and we also have implemented the FTP client software based on this technology in Linux, and give the performance test data and analysis; these can fully proved that the technology will significantly improve the efficiency of uploading large files.

**Index Terms** - FTP, multithreading, broken transfer resume, uploading

## 1. Introduction

FTP is one of a widely used service in network. Most files uploaded and download are completed by FTP [1]. It implements a reliable and effective data transmission. At the same time, as long as both system support FTP protocol, it can conveniently transmit files between different types of system and users won't be influenced by the differences of the hosts which have different kinds of file system. Therefore, the FTP has been widely used on the internet service.

FTP server can provide the bidirectional services for upload and download files, the current application and research mainly focus on the FTP download technology [2], but upload technology research is rarely involved, especially the multi-threaded uploading research for single file is less. In modern internet, it often needs to transmit large files which have several GB, such as multimedia file sharing, etc. However, duo to the huge file size, transmission process takes a longer time, and would be interrupted more easily, leading to retransmitting frequently, which has a bad influence on efficiency. Another example: when we use a super computer center for high-performance computing, it needs to upload a lot of calculation data through VPN network, if we use traditional way, we would get a low efficiency. Thus, the technology research of large file uploaded has a great significance to promote the network application.

Traditional FTP uses a single thread to upload files, it would be not conducive to preempt in network bandwidth, and leads to upload speed slowly, once the unexpected interrupt happens, it must upload the whole data again, and this would greatly affect efficiency. This paper proposes a method, using multithreaded and broken transfer resume uploading, to solve this problem, which can significantly improve the efficiency of uploading large files.

## 2. Related Work

FTP needs to build two connections: control connection and data connection. The control connection is always waiting for the communication between client and server; it will transfer the command from client to server, and back the server response information to client [3], default uses 21 as the port number. The data connection is used to transmit the data flow between client and server, in active mode, the FTP server default uses 20 as the port number.

FTP has two operating modes: Standard (PORT, active way) and Passive (PASV, Passive way). At the Standard modes, client and server build a control connection at the server's TCP port 21 firstly, when the client needs to transmit data to server, it will send "PORT" command, which includes the port number used by the client to receive and send data. The server through its own TCP port 20 actively connects to the port, designated by the client, to achieve data transmission. At the Passive modes, the process of building control connection is basically the same as Standard modes, but the data connection is different from each other. The client sends "PASV" command, which is used to request the server open a temporary port (the port number is usually between 1023 and 65535), when the client get the server's response information, which includes the temporary port number, it will build a data connection on the specified port to achieve data transmission.

The following picture is a model for the FTP service (See Figure1.).

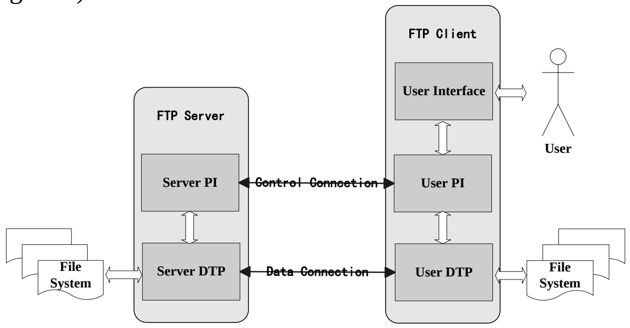


Fig1. A model for the FTP service

* This work is supported by the Project of Machinery and Equipment Innovation Design Demonstration Application Based on High Performance Computing Service #2013Y2-00037.

In order to improve the efficiency of the FTP transfer, we can use multiple processes or multi-threaded way to parallel transmission, and compared with process, thread has some relative advantage, such as less resource, more flexible, and so on. In the same process, all threads share the same global memory; it makes more easily share information [4].

## 3. Design

When users need to upload a file to FTP server, users can set the number of threads (In this paper, we use the capital letters "N" represent the number of threads), and each thread will build its own control connection and data connection, it means that one thread is corresponding to one client, and the file will be logically divided into N blocks, then each thread will get one block as its tasks. At the same time, we use another thread manage temporary configuration file, which is stored with the uploaded filed under the same path, and used to record the information that the broken transfer resume operation needs. The temporary configuration file will be deleted after the file transfer has been completed. When unexpected interrupt happens during the file transfer, the client can get the breakpoint information from the temporary configuration file, and uses multithreaded and broken transfer resume way to upload the rest of the file. To achieve the design idea, we need the steps as follows:

(1)Check whether there is a corresponding temporary configuration file, if not, then go to step (2); else go to step (3).

(2)Get the users' settings, which contain threads number N, uploaded file's information, file transmission format (binary format by default), and so on, and then according to the number of threads, divide uploaded file into N parts, this is logical divide, and just get each start and end offset. Finally, start another thread to create and manage the temporary configuration file, the thread will record (every second by default) each thread's uploading progress. Then, go to step (4).

(3)Get the threads number N, and each thread's uploading progress from the temporary configuration file, after that, redistribute the rest of tasks for each thread, and then, create another thread to manage the temporary configuration file; the thread will record (every second by default) each thread's uploading progress. Then, go to step (4).

(4)Create N threads at the same time, and create local sockets for each thread, and build N connections with the target FTP server.

(5)Each thread sends "USER xxx \r\n" and "PASS yyy \r\n" commands to the FTP server, in order to login the FTP server. The "xxx" is the user name, and the "yyy" is the password.

(6)After login successfully, each thread uses TYPE command to set a unified data format, "TYPE I \r\n" is the command for binary format, etc.

(7)Each thread sends "PASV \r\n" command, which requests the server-DTP to "listen" on a data port (which is not its default data port) [5], and each client can get the IP address of the FTP server and a port number from the response of the server, then using the IP address and the port, each thread can build a data connection with the FTP server.

(8)Each thread sends "RSET xxx \r\n" command to set up the start offset of the file transfer, and different threads have different start offsets.

(9)Each thread sends "APPE xxx \r\n" command to the FTP server for a file upload request, and the "xxx" is the file name used on the server. The operation requests that every thread has the same uploaded file name, and the FTP server must support broken transfer resume uploading.

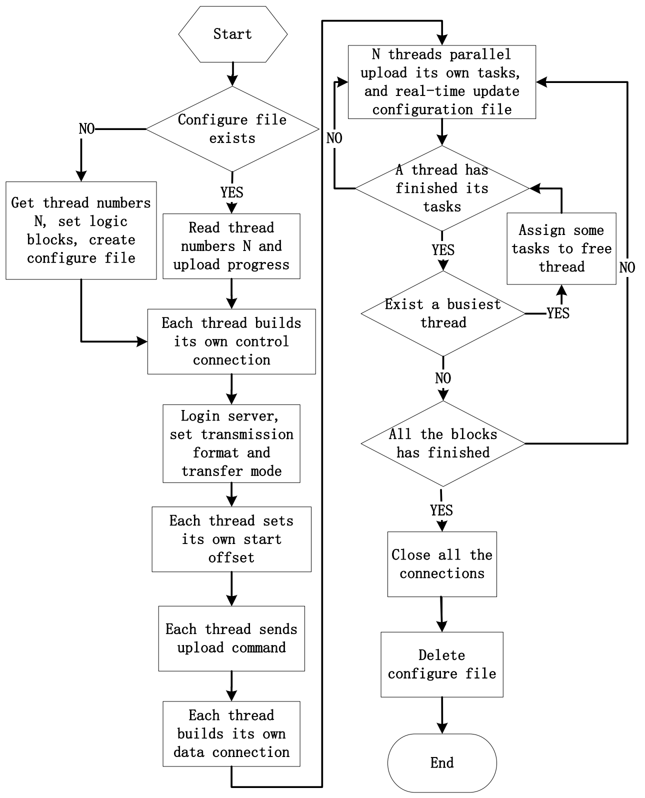
(10)Each thread builds a data connection with the FTP server, and is allocated a data block, then begins uploading its own tasks as the parallel way.

(11)If there is some thread has finished its tasks, it will actively share some remaining tasks from the busiest thread.

(12)If unexpected interrupts happens again, repeat the above steps, until the whole file has been uploaded, then close all the data connections and control connections.

(13)Delete the temporary file, thus a file upload process is successful.

The flow chart for multithreaded and broken transfer resume uploading by this method is as follows (See Figure 2.):



Start
Configure file exists
NO
YES
Get thread numbers N, set logic blocks, create configure file
Read thread numbers N and upload progress
Each thread builds its own control connection
Login server, set transmission format and transfer mode
Each thread sets its own start offset
Each thread sends upload command
Each thread builds its own data connection
N threads parallel upload its own tasks, and real-time update configuration file
A thread has finished its tasks
NO
YES
Exist a busiest thread
YES
Assign some tasks to free thread
NO
All the blocks has finished
NO
YES
Close all the connections
Delete configure file
End

Fig2. The flow chart for multithreaded and broken transfer resume uploading

## 4. Technical Essentials

According to the proposed method in this paper, in order to implement multithreaded and broken transfer resume uploading, there are some main technical points as follows:

(1) During the multithreaded uploading operation, FTP client builds multiple connections with FTP server by multithreading. According to the number of threads N set by users, the uploaded file will be logically divided into N blocks, assigned to different threads to upload. In the process of file divided, try to make sure each block size is basically uniform, and each block size is integer times to the send or receive buffer size.

(2) During the broken transfer resume uploading operation, get the breakpoint information from the temporary configure file firstly, and set the number of threads and the offset for each thread, then use the way as essential (1) to upload the left data.

(3) During the operations of essential (1) or (2), FTP client will create another thread to manage the temporary configure file, which includes the information of uploaded file's full path on the FTP server, file size, block numbers, each block's start and end offset, each block's upload progress, whether the block is finished, and file transformat.

(4) Redistribution task (RDT). When a thread has finished its own block, manager thread will actively assign some uploading tasks to it form the busiest thread, which still has the most uploading tasks left at the moment. If the idle thread is named ***A***, the busiest thread is named ***B***, $T_L$ represents the average time-consuming for building an FTP connection, $C_B$ represents the remaining task left by B, and $s_B$ represents the average uploading speed of the ***B***, the RDT algorithm will be described as follows:

① Query all the threads to know whether a free thread, as ***A***, exists. If not, then wait for 10 seconds, and query again; otherwise, turn to the next step.

② Find out the busiest thread as ***B***.

③ Determine whether the expressions $C_B / s_B \geq 4T_L$ are correct. If it's not true, the algorithm is ended successfully; otherwise, go to the next step.

④ Divide $C_B$ into two subtasks equally, and redistribute the first task to the thread ***B***, and the left one to the thread ***A***, then back to the step ①.

Using the RDT algorithm, we can ensure that all threads are active, and efficiently complete the whole uploading tasks.

(5) After the operations of essential (1) or (2) is finished successfully, FTP client will delete the temporary configure file automatically to save disk space.

## 5. Performance Analyses

The TCP protocol provides reliable ordered communication, thus applications built on top of it, can make use of these features [2]. According to the design idea, we have used C++ to realize FTP uploading tool in Linux, and have successfully tested on vsftpd and wu-ftpd servers. The results proved that the design idea is full of feasibility.

In further tests, we use two computers (hardware configurations: 2.80GHz Intel（R）Core(TM) i5-2003 CPU, 4.00GB RAM, Gigabit-NIC, Ubuntu 12.04 OS), one is used wu-ftpd 2.6.2 as FTP server, the other is used our tool as FTP client. They connect with each other through our school's Gigabit LAN. We make the clients, which have the single-thread and the multi-thread, upload a same file with 250M size at the same time, in order to get the time that different thread numbers used, and get the average result of ten times test (See Table 1).

TABLE 1 Transmission speed-up ratio

| Thread Numbers | 1 | 2 | 4 | 6 | 8 | 10 |
|---|---|---|---|---|---|---|
| Transmission Time (s) | 16070 | 9071 | 4411 | 3200 | 2610 | 2262 |
| Speed-up Ratio | 1 | 1.77 | 3.64 | 5.02 | 6.16 | 7.10 |

From the table 1, we can easily get that: the transmission speed-up ratio is non-linear; it is affected by multithreaded connections time-consuming, system scheduling time-consuming, network bandwidth throttling, and so on. We also can get that transmission speed-up ratio will increase with the increase of number of threads, but the increase rate will gradually decrease, and the speed-up ratio will reach a peak eventually, then the speed-up ratio will drop with the thread numbers increasing. Figure 3 is a line chart corresponding with table 1.

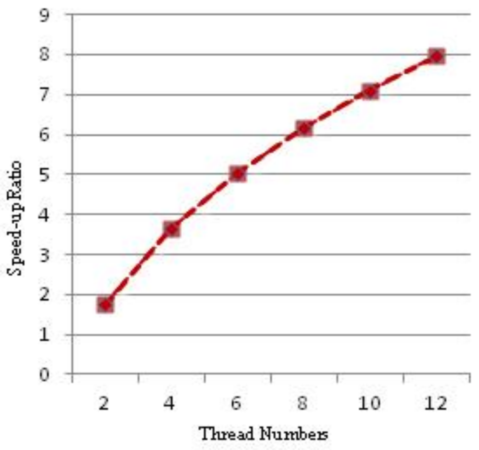


Fig3. The line chart for transmission speed-up ratio

## 6. Conclusions

This paper proposes the technology, which is suitable for large file upload base on FTP. Using multi-threaded uploading can effectively shorten the time cost on uploading large files, and using broken transfer resume method can avoid uploading repeated data. These will reduce the amount of data transmission, and can be more significant to improve the efficiency of uploading large files. This technology can play a significant advantage in some application, such as network bandwidth is limited, but needing to upload large files frequently. This technology also has a strong universality, because it doesn't need to modify the FTP protocol. Future research should focus on improving the safety of the file upload.

## References

[1] Liu Xia, Feng Chao-sheng, Yuan Ding, Wang Can. Design of Secure FTP System. IEEE 2010

[2] Jameela Al-Jaroodi, Nader Mohamed. DDFTP: Dual-Direction FTP. IEEE/ACM International Symposium on Cluster, Cloud and Gird Computing 2011

[3] W. Richard Stevens. TCP/IP illustrated, Volume 1: The Protocols. Beijing: China Machine Press 2005, pp.316-331

[4] W.Richard Stevens, Bill Fenner, Anderw M.Rudoff. UNIX Network Programming Volume 1: The Sockets Networking API, Third Edition. Beijing: POSTS & TELECOM PRESS 2010, pp.675-699

[5] RFC-959: J.Posted, J.Reynolds. FILE TRANSFER PROTOCOL (FTP). 1985