P4P Network Communication Components Based on Half-Sync/Half-Async and Pipe/Filter Patterns

[1,2] Cheng Wang
[1]College of Computer Science and Technology, HuaQiao University, Xiamen 361021, China
[2] Xi'an Jiaotong University, Xi'an 710049, China
wangcheng@hqu.edu.cn

Zhicong Liang
boxlzc@gmail.com

***Abstract*—This paper describes P4P(Proactive network Provider Participation for peer-to-peer) network server based on the Half-Sync/Half-Async and Pipe/Filter design patterns, which implements the requirements of the P4P system. The P4P network server applies the Half-Async layer to listen to the specified network port and establishes network connections asynchronously; makes use of message queue layer to buffer established network connections; applies the Pipe/Filter pattern into the Half-Sync layer and takes the Half-Sync layer to receive data and send data concurrently. Thanks to these patterns and the design, it gains various levels of concurrency and flexibility.**

I. INTRODUCTION

With the rapid development of P2P(peer-to-peer) networks, some new modules and protocols are used to construct P2P systems, some people propose new architectures based on P2P called P4P to provide more effective cooperative traffic control between applications and network providers. As peer-to-peer (P2P) emerges as a major paradigm for scalable network application design, it also exposes significant new challenges to achieve efficient and fair utilization of Internet network resources[1]. To improve the feasibility, concurrency and effectiveness of distributed systems, more and more new architectures and modules will be proposed. P4P systems developed from and based on P2P systems. Consequently, P4P systems are becoming an important application of distributed software systems, and the researches on it are being of great significance. On account of the development of information technology, computers, mobile phones and various media terminals will continue to emerge, how to make these different terminals interact with each other is difficult and filled with challenges. How to resolve this difficulty depends on the development of the network communication components in these systems. As the diversity of operating systems and communication platforms, communications software developers often have to face so many problems as the performance of communication, the management of code, platform coverage, and so on. P2P applications may face various of requirements and challenges, there are many related researches, such as [2], [3], [4], [5], [6] and [7] try to conquer these challenges, in which some effective methods and systems are proposed to conquer these challenges. There are many design patterns used for different application fields, including classic design patterns[8] and distributed applications related design patterns[9][10][11][12][13], which help us conquer the design challenges. Some are used to create a specific types of software, such as the pattern languages for networked and concurrent computing[10] and enterprise application architectures[14]. The Half-Sync and Half-Async is one of these key patterns[8] in the domain of communication software. There are many common concurrency models for network server as follows:

A. Thread-Per-Connection Model.

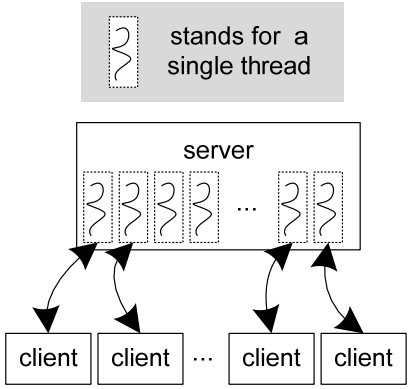

Figure 1. thread-per-connection model.

As is shown in figure 1, in this model, a new thread will be created and associated with the new connection request when a new connection request arrives. The new created thread is responsible for establishing connections, receiving network data, handling network data, sending result to clients, error handlings and shutdown the connection. This model takes the following disadvantages:

(a.) does not separate the business logic and network logic.
(b.) a huge number of threads will be created when a lot of networks connections arrive simultaneously, which costs much system resources and memory, the system may crash in the worst case.

doi:10.4304/jsw.9.11.2830-2836

(c.) developers are front with challenges from code maintain and business changes.
(d.) if all the threads were created in one process, the whole process will crash when any one of these threads crashes.

B. Process-Per-Connection Model.

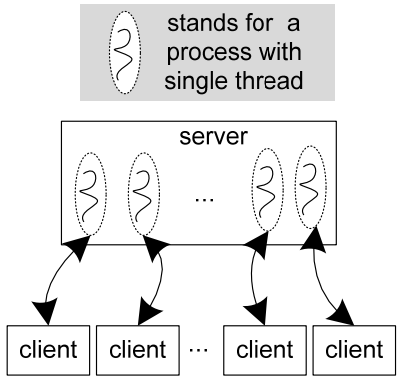

Figure 2. process-per-connection model.

As is shown in figure 2, this model is similar with the thread-per-connection model, but processes instead of threads are created when network connection requests arrived. This model takes the following disadvantages:
(a.) does not separate the business logic and network logic either.
(b.) a huge number of processes will be created when a lot of networks connections arrive simultaneously, which costs a lot of CPU time, system resource and memory, the system may crash when system load becomes bigger and bigger.
(c.) developers are front with challenges from code maintain and business changes.
(d.) frequent IPC(Inter-Process Communication) will happen, which costs much system resource and leads to bad performance.

*C. Thread-pool Mode*l.

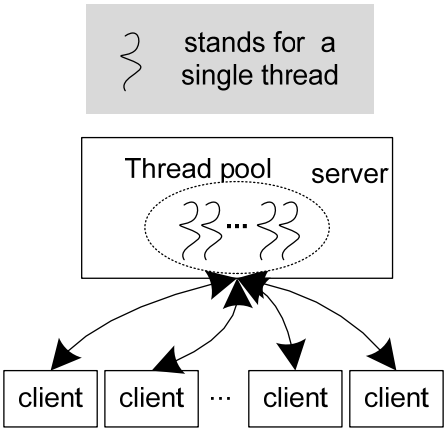

Figure 3. thread-pool model.

As is shown in figure 3, in this model, network server creates a fixed number of threads before connection requests arrive. The server selects an idle thread from thread-pool to provide services to clients when a new connection request arrives. In this model, the number of threads is fixed, one thread may provide services to different network connections during different time. This model costs fixed system resources, will not bring huge load to the system, but it is not able to handle all the network connections when the number of concurrent network connections is more than the number of threads in the thread pool, which makes some network connection requests blocked for a long time.

D. Leader/Follower Model[10].

The Leader/Followers architectural pattern provides an efficient concurrency model where multiple threads take turns to share a set of event sources in order to detect, demultiplex, dispatch, & process service requests that occur on the event sources[10]. There is no message queue in this model, it is not able to handle a lot of current network connections simultaneously.

E. Half-Sync/Half-async Model[10].

The Half-Sync/Half-Async architectural pattern decouples asynchronous and synchronous service processing in concurrent systems, to simplify programming without unduly reducing performance. The pattern introduces two intercommunicating layers, one for asynchronous and one for synchronous service processing[10]. This model includes three level layers, Half-Sync layer, message queue layer and Half-Async layer, which is shown in figure 4 as follows:

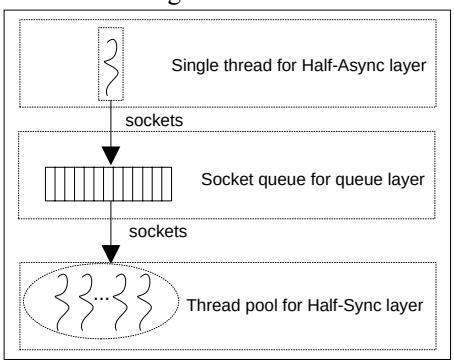

Figure 4. Half-Sync/Half-Async architectural pattern[10].

Each layer`s responsibilities are as follows:
(a.) Half-Async Layer. This layer is responsible for handling asynchronous network connection requests from clients and establishing connections. Establishing a connection is a quick operation, which will not block other tasks or reduce the performance of the system. The concurrency strategy in this layer is various, including single thread strategy, thread pool strategy and so on. We choose single thread strategy here.
(b.) Queue Layer. This layer is responsible for buffering requests and provides communication mechanism between Half-Async Layer and Half-Sync Layer. This layer separates and decouples Half-Async Layer and Half-Sync Layer, which makes the strategies of these three layers independent and flexible.
(c.) Half-Sync Layer. This layer is responsible for handling requests in the queue layer. Generally speaking, there is a thread pool in this layer to handle

requests concurrently.

This model takes the following benefits:

(a.) higher-level tasks are simplified[10].
(b.) business logic and network logic is separated, which makes the design more flexible.
(c.) synchronization policies in each layer are decoupled[10].
(d.) inter-layer communication is localized at a single point[10].
(e.) performance is improved on multi-processors[10].
(f.) code maintain and business changes are easy to handle.

Each model owns its advantages and disadvantages, after comparing all the above models, we choose Half-Sync/Half-Async to build the P4P systems. With these advantages of Half-Sync/Half-Async pattern, it is easy to build a flexible network communication server with high performance and various levels of concurrency.

II. Requirements In The P4p System

The requirements of network communication components in the P4P system are as follows:

(1.) sending and receiving data.
(2.) good expansibility, concurrency and flexibility.
(3.) the server owns the capabilities to handle thousands of network connections concurrently.
(4.) implementing dynamic and exchangeable data handling flow to provide sufficient flexible protocol parsing strategies to fit for various requirements and businesses in the P4P system.

We should design the p4p systems flexible enough with high performance and various level of concurrency to fit all the requirements above, so it is very important to choose appropriate patterns to design the architecture.

III. Architecture And Design

When designing the network communication components of the P4P system, concurrency is a very important factor. How to maximize the number of concurrent connections is a problem front with developers.

As we stated above, Half-Sync/Half-Async model takes several advantages to bring good concurrency and performance. But this pattern is not flexible enough in front of the requirements of the P4P systems, especially for the Half-Sync layer. Because the data handling business in the P4P system is very complex and various, there are many different protocols need to parse. How do you implement a sequence of transformation modules so that you can combine and reuse them independently[15]? Fortunately, Pipe/Filter[16] pattern is designed to resolve this problem, which is shown in figure 5. This pattern requires the following[15]:

(1.) The output of the data source must be compatible with the input of filter 1.
(2.) The output of filter 1 must be compatible with the input of filter 2.
(3.) The output of filter 2 must be compatible with the input of the sink data.

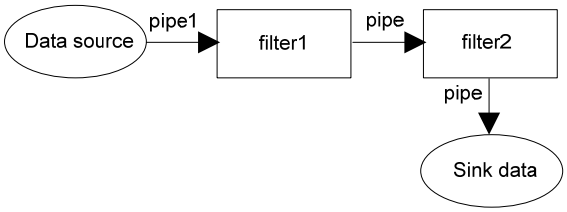

Figure 5. Pipe/Filter Pattern.

We use the Half-Sync/Half-Async and Pipe/Filter patterns to design and implement these network communication components in the P4P system.

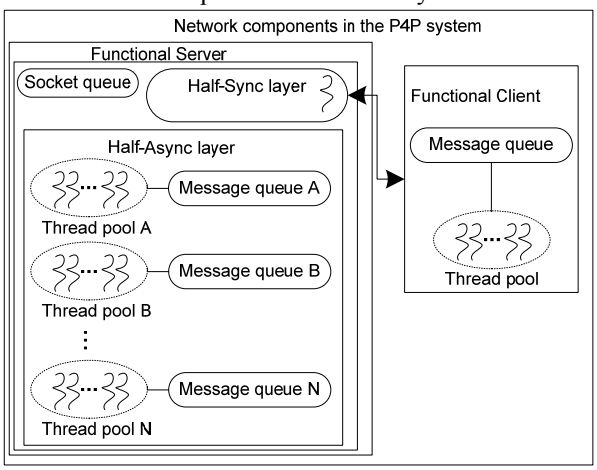

Figure 6. Architecture of network communication components.

As shown in figure 6, the network communication components in the P4P system includes two parts: one is Functional Server, the other is Functional Client. Functional Server provides functions such as monitoring the specified port, maintenance of passive connections, receiving, buffering, handling and sending data, while Functional Client provides functions such as initiating active connections, maintenance of active connections, receiving and sending data. Functional Server use a single thread to listen to the specified network socket and put the sockets into socket queue; creates several thread pools and corresponding message queues to buffer data. The Pipes and Filters architectural pattern divides the task of a system into several sequential processing steps[16]. In the P4P system, each sequential processing step is a data handling module which contains a thread pool and a message queue. Functional Client creates a thread pool to handle messages and creates a message queue to buffer data.

Data handling work flow

This section describes how to design the data handling work flow framework of the Functional server. This framework implements the Pipes and Filters pattern[16].

In the front of the requirements from current P4P applications, there are more and more data handling requirements appears, the general static data handling work flow is not flexible and robust enough. For example, a static and complex network data handling flow is shown in figure 7, if we wanted to add or delete some data

handling module, we have to stop the system firstly, rewrite the whole data handling work flow or do much changes and rebuild the source code statically and then re-launch the system, which will cost us much time and make system unstable.

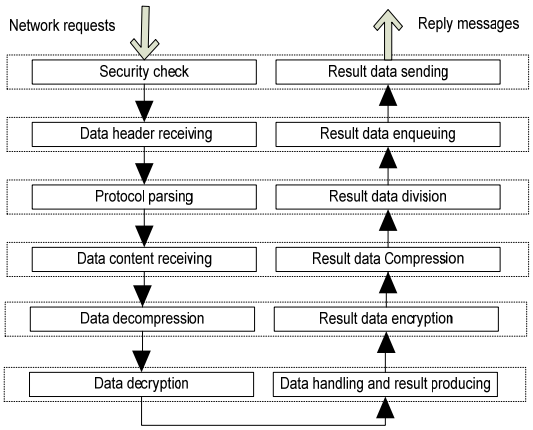

Figure 7. A complex network data handling flow.

So with the help of the Pipes and Filters pattern, we build a more flexible and dynamic data handling work flow model. In this model, we use a list to link all the data handling modules sequentially, when a new module is needed to add into the work flow, just inserts the new one into the list. Each module owns its private threads pool to handle data, owns its message queue to buffer data, which makes the whole system more flexible and gains good performance. The new data handling work flow is shown in figure 8.

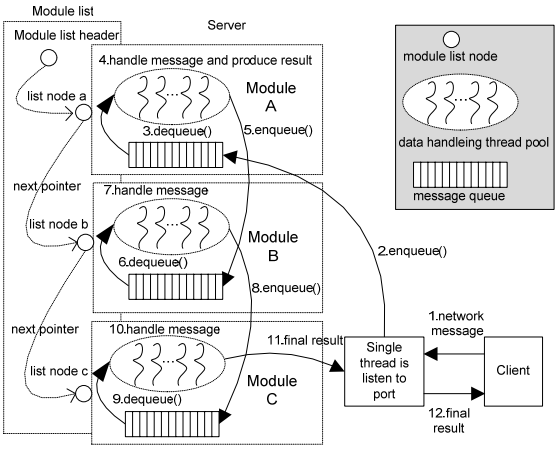

Figure 8. data handling work flow.

The whole procedure of data handling work flow is as follows:

(1.) The single thread listens to the specified network port, and establishes connections when connection request arrive.
(2.) The single thread which listens to the port puts network connections into the queue of module A.
(3.) Module A gets an item from its own queue.
(4.) Thread pool of Module A handles the item and gets the result.
(5.) According to the module list, module A finds out its next module which is module B and puts the result into the queue of module B.
(6.) (7.)(8.)(9.) Module B and the following modules will do the similar procedure with module A until the handled result arrives to the tail module.
(10.)(11.)(12.) The tail module of the module list handles the result passed from its previous module, produces the final result and sends it to the client.

Thanks to the design, each module owns its private thread pool, the data handlings are concurrent and independent in both each thread pool and each module. Each data handling module is a list node, when some new module is needed to add into the data handle flow during the system is running, it is just needed to insert this new module into the list, which is shown in figure 9 as follows. A lock is hold to protect the list during the inserting. Deleting a module is in the backward procedure. Because adding or deleting a module is not a frequent operation and is always done during wee hours, the cost of holding lock is acceptable.

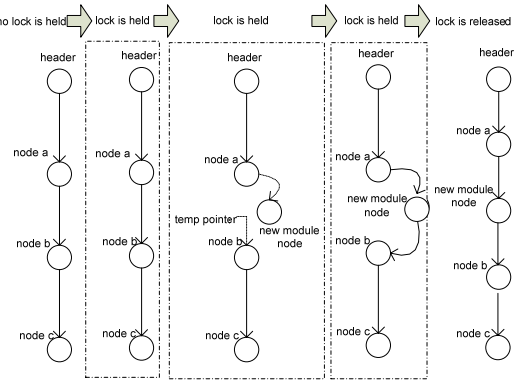

Figure 9. add a new module into data handling flow.

At the same time, to make this system more flexible, each module owns a state to indicate its current state. Each module is in one state of four states: idle, active, running and inactive. The translation among these states is shown in figure 10.

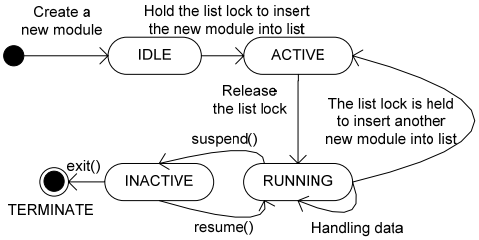

Figure 10. State translations of data handling modules.

Application Level Binary Exponent Backoff Strategy

As we known, network crowd may happen during the network is busy, which will cost much resources and time. Binary exponent back-off algorithm[17][18] is applied into Ethernet (802.3)[19] to reduce network crowd. Similarly, in network communication system, clients always send connection requests to the server simultaneously, but the handling abilities of server is limited, so it is needed to apply some strategy to coordinate the clients. The

application level binary exponent back-off strategy is applied into the functional clients to decrease network crowd in the P4P system. The pseudo code of application level binary exponent back-off pseudo algorithm is shown in figure 11:

```
int backoff_reconnect(int NumberOfReconnect, int initialDelayTime,
string serverAddress) {
  int i, sock;
 ClientConnector connector;
 ClientBuffer buffer;
  int sock=socket(AF_INET, SOCK_STREAM,0);
 ClientTim timeout(initialDelayTime );
for(i = 0; i < NumberOfReconnect; i++) {
     if(i != 0) sleep(timeout);
          if (connector.connect (sock, serverAddress, &timeout) == -1)
 timeout *= 2; /* exponential backoff */
else{
connector.send(sock, buffer);
break;
     }
     }
    return (i == NumberOfReconnect)? -1 : 0;
}
```

Figure 11. Exponent backoff algorithm in clients.

The description of the application level exponential back-off algorithm is as follows:

(1.) Define the basic timeout time, in this communication system the basic timeout is defined by parameter initialDelayTime.

(2.) Define a parameter named NumberOfReconnect, which stands for the times of network reconnections.

(3.) New time out is equals to two multiplied by old timeout, the initial value of time out equals to initialDelayTime.

(4.) If connection still failed after NumberOfReconnect times, reports the error to high level applications of this communication system.

(5.) If connection was established, send messages to network.

(6.) After applying Half/Sync-Half/Async and Pipe/Filter patterns into the system, it fit for all the requirements, which not only gains high performance but also obtains enough flexible.

IV. The Run-Time Procedure Of Network Communication Components

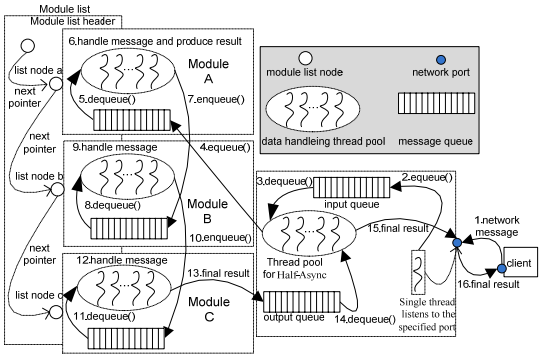

Figure 12. The run-time procedure.

Figure 12 shows the run-time process of network communication component in P4P system. Functional server runs a single thread to listen to the network socket, while each data handling module owns a private thread pool to handle data. The number of threads in each thread pool is specified by parameters to satisfy different performance of different machines. In order to communicate among threads, there is a message queue in each module. The run-time procedure of network communication component in P4P systems are as follows:

(1.) Clients initiate connection requests to the server and then send messages to the server. Both UDP[20] connections and TCP[21] connections are supported in the P4P systems to provide different services to various of businesses and requirements.

(2.) The single thread which listens to the network port detects the connection requests, establishes the connections and put the sockets into the input queue.

(3.) Thread pool for Half-Async gets these sockets items from the input queue, and calls select function to wait for network messages` coming. This function allows the process to instruct the kernel to wait for any one of multiple events to occur and to wake up the process only when one or more of these events occurs or when a specified amount of time has passed[22]. When messages are coming, the thread pool receives messages, handles these messages, and produces the results.

(4.) Thread pool for Half-Async puts the results into the message queue of the first data handling module of the module list, which is Module A in figure 12.

(5.) The thread pool of Module A gets messages from its message queue.

(6.)(7.) The thread pool of Module A handles messages and put the results into the message queue of Module A`s next Module, which is Module B in figure 12.

(8.)(9.)(10.)(11.)(12.) Module B and the following modules will handle the messages from their previous modules similar with Module A does until meets the final module.

(13.) The final module handles the messages, produces and puts the final results into the output queue of the thread pool for Half-Async.

(14.)(15.)(16.) The thread pool for Half-Async gets the final results from its output queue and sends them to clients by corresponding sockets.

From the run-time view of the P4P system, we got a clear understanding about how the system works and how these components coordinate with each other.

V. Performance Testing

We did the testing with the following hardware devices and configurations: Intel i7-4700HQ/4G platform with Ubuntu Server 12.04 and network with the speed of 1000Mps.

Testing in figure 13 shows high performance and advantages of network server based on Pipe/Filter and Half-Sync/Half-Async patterns compared with network server based on thread-per-connection and network server based on thread pool. With the number of network connections increasing, the performance of this system

keeps at high level relatively.

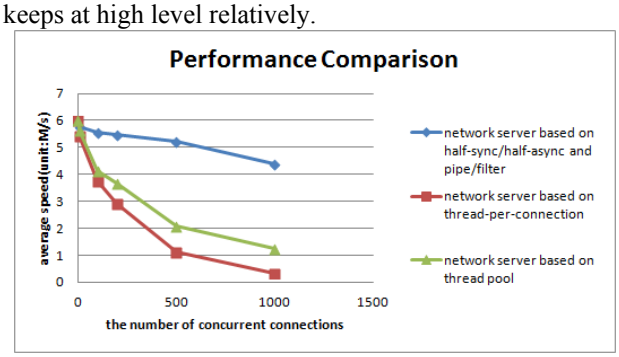

Figure 13. Network performance comparison.

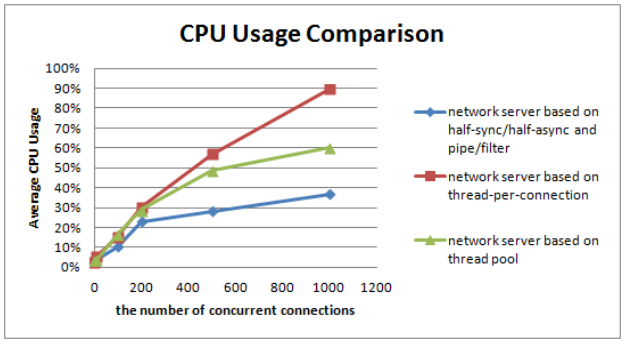

Figure 14. CPU Usage comparison.

As is shown in figure 14, with the increasing of the number of concurrent connections, the average cpu usage of network server based on half-sync/half-async and pipe/filter shows a smooth curve upward relatively, which means costing lower system resource and gains higher performance compared to others.

According to these above testing results, our system gains various levels of concurrency, which better meets the requirements and needs of the P4P system.

FUTURE WORK

Our future work will focus on the following aspects:

(1) The optimization of p4p protocols analysis. Because there are all kinds of application level protocols to support in this system, we need to optimize the protocol analysis components.

(2) The optimization of data handling components and flows. As we known, the data handling will cost much time and system resource, so it is necessary to improve the performance of data handling flow and components.

(3) Try to apply other network communication related design patterns into this system, which could improve the flexibility, extension further and make this system more maintainable.

(4) Do more stability related testing. Because the system is complex and will cover both Windows and Linux operating systems, it is necessary and important to do stability related testing.

VI. CONCLUSIONS

In this paper, we compared several network server models and stated the disadvantages and advantages of these models firstly. And then, we proposed the architecture of the P4P network communication components based on Half-Sync/Half-Async and Pipe/Filter Patterns. Finally, we give the testing results and related analysis about these models. The network communication components in this system take the Half-Sync/Half-Async pattern framework as its core framework, applies the Pipe/Filter pattern framework as its auxiliary framework and organizes all the data handling modules into a list to add and delete modules dynamically. Thanks to these frameworks and designs, it gains various levels of concurrency and flexibility which fits for the requirements of the P4P system.

ACKNOWLEDGEMENTS

This work was financially supported by National Natural Science Foundation of China (Grant No. 51305142) and introduction of talents Huaqiao University Scientific Research Projects (Project No. 12BS217).

REFERENCES

[1] Haiyong Xie,Y. Richard Yang, Arvind Krishnamurthy,Yanbin Liu,Avi Silberschatz. P4P: Provider Portal for Applications.SIGCOMM'08, August 17–22, 2008, Seattle, Washington, USA.

[2] Adeela Bashiry, Sajjad A. Madaniy, Jawad Haider Kazmiy, Kalim Qureshi. Task Partitioning and Load Balancing Strategy for Matrix Applications on Distributed System, JOURNAL OF COMPUTERS, VOL. 8, NO. 3, MARCH 2013.

[3] Jinghua Wu,Yun Xu. A Decision Support System for Borrower's Loan in P2P Lending, JOURNAL OF COMPUTERS, VOL. 6, NO. 6, JUNE 2011.

[4] Zhengzhen Zhou, Yonglong Luo, Liangmin Guo, Meijing Ji. A Trust Evaluation Model based on Fuzzy Theory in P2P Networks, JOURNAL OF COMPUTERS, VOL. 6, NO. 8, AUGUST 2011.

[5] Choffnes, D. and F. Bustamante, "Taming the Torrent: A practical approach to reducing cross-ISP traffic in P2P systems", Proceedings of ACM SIGCOMM, August 2008.

[6] R. Bindal, P. Cao, W. Chan, J. Medval, G. Suwala, T. Bates and A. Zhang, "Improving Traffic Locality in BitTorrent via Biased Neighbor Selection". In IEEE International Conference on Distributed Computing System (ICDCS 2006).

[7] K. Shanahan and M. Freedman, "Locality Prediction for Oblivious Clients". International workshop on Peer-To-Peer Systems (IPTPS 2005).

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1995.

[9] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. Pattern-Oriented Software Architecture, A System of Patterns, Volume 1. Wiley and Sons, New York, 1996.

[10] Douglas C. Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2. Wiley & Sons, New York, 2000.

[11] Michael Kircher, Prashant Jain. Pattern-Oriented Software Architecture: Patterns for Resource Management, Volume 3. Wiley in 2004.
[12] Frank Buschmann, Kevlin Henney, Douglas C. Schmidt. Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing, Volume 4.Wiley & Sons in 2007.
[13] Frank Buschmann, Kevlin Henney, Douglas C. Schmidt. Pattern-Oriented Software Architecture: On Patterns and Pattern Languages, Volume 5. Wiley & Sons in 2007.
[14] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, and R. Stafford, Patterns of Enterprise Application Architecture. Reading, Massachusetts: Addison-Wesley, 2002.
[15] http://msdn.microsoft.com/en-us/library/ff647419.aspx
[16] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. Pattern-Oriented Software Architecture, A System of Patterns, Volume 1. Wiley and Sons, New York, 1996.
[17] Larry L. Peterson, Bruce S. Davie, Computer Networks, Edition 4: A Systems Approach, pp.116-123, 2007, Elsevier, Inc.
[18] Kevin R. Fall, W. Richard Stevens. TCP/IP Illustrated, Volume 1: The Protocols, pp.114-116, 2012, Addison Wesley.
[19] "IEEE Standard 802.3-2008". IEEE. Retrieved 22 September 2010.
[20] RFC768; Postel, Jon. User Datagram Protocol, IETF, August 1980.
[21] RFC793; Postel, Jon. Transmission Control Protocol, IETF, September 1981.
[22] W. Richard Stevens, UNIX Network Programming Volume 1, Third Edition: The Sockets Networking API. pp.209-121, 2003, Addison Wesley.