P2P Shared-Caching Model: Using P2P to Improve Client-Server Application Performance

Luong Quy Tho
SILILAB
thoqbk@gmail.com

Ha Quoc Trung
SoICT-HUST
trunghq@soict.hut.edu.vn

ABSTRACT
Client-server application model has several drawbacks such as server bottleneck and weak scalability. Peer-to-Peer (P2P) model resolves these problems by distributing tasks on the nodes participating in the system. P2P application development and protocol designing are much more difficult than client-server model. This paper proposes an approach to take the advantages of both models: the scalability of the P2P model and the simplicity of the client-server model. This paper presents a hybrid P2P and client-server model to achieve both goals based on caching mechanism which allows using cache content not only on a single client, but for all clients in the system. The proposed model has been applied to implement a Web application.

Categories and Subject Descriptors
C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design – *network topology, distributed networks.*

D.1.3 [**Programming Techniques**]: Concurrent Programming – *distributed programming, parallel programming.*

General Terms
Distributed Computing

Keywords
Distributed system, P2P and Client-Server Architecture

1. INTRODUCTION
Most of recent applications and protocols are developed based on the client-server model, because of its simplicity. Processing task in the client-server model is concentrated on the server, so the server will quickly become bottleneck for processing power and communication bandwidth. Extended client-server models are proposed to improve the performance of the model. Load balancing and replication techniques allow distributing load on number of servers [1]. Caching is a kind of asymmetric replication of the server's data on client side. Caching is used exclusively in the proxy model, where the proxy is delegated from many clients to access server's data, and can provide information from proxy cache when possible. These approaches still use the client server model, so they cannot resolve the server's bottleneck problem.

Peer-to-peer (P2P) model resolves client-server's model drawbacks by distributing the computational workload on peers (i.e. the host – members of the system). Using this model, the interaction between components in the system is changed conceptually, so these protocols and applications design are more complicated. Instead of designing a protocol between client and server, now a complicated interaction between peers must be considered.

In fact, P2P model resolves the problem of communication and computational performance of the hosts, but it doesn't concern the interaction between peers. Consequently, it is possible to mix P2P model and client server model so that the client server model is for interaction between user agents, while the P2P model is used for communication between hosts. The system then consists of the client-server component and the P2P subsystem. The remaining problems to be resolved are: (i) interaction between the client-server and P2P subsystem; (ii) adapting the system with data and application specific requirement.

Several models are used to resolve this problem: Local proxy model [9], native P2P application and common cache model.

In this paper, we present our proposal for using shared-caching model in distributed application development. The solution is implemented, experimented and adapted with raw data and file sharing application.

The paper is organized as the follows. In Section 2, we introduce our shared-caching model. The details of the application that we implemented to test the model are presented in Section 3. In Section 4, we describe the results and our future research directions.

2. SHARED-CACHING MODEL
In this section, firstly, we introduce two abstract models to improve client-server application: caching model and local proxy model. Basing on these models, we introduce our proposal model, the shared-caching model.

2.1. Caching model and local proxy model
Caching in client-server model is a kind of replication, improving the system performance by make data access local for the client. The mechanism of caching is described in Figure 1 and Figure 2. The client first tries to access the data from local cache. If the data is not available, then the client requests server to send the data. When receiving data from the server, the client stores it in local cache for later access. Several mechanisms are used to make the cache content consistent with the server content: polling, invalidation and leased invalidation.

SoICT '13, December 05 - 06 2013, Danang, Viet Nam

ACM 978-1-4503-2454-0/13/12…$15.00.
http://dx.doi.org/10.1145/2542050.2542090

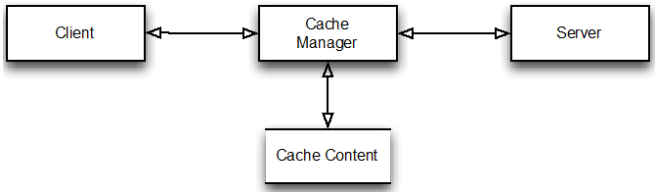

FIGURE 1: THE CACHING MODEL

The performance of caching mechanism depends on the cost of getting data from the server, the updating pattern of the data item and the cache invalidation rules. Issue that caching model cannot resolve are bottleneck and weak scalability. Because caching model only improves performance of application but does not change the structure of distributed application, it means that model of application is still client-server model.

Another approach to improve the system performance and resolve the disadvantages of client-server model is: using P2P. The notable application model using P2P to upgrade client-server application is: Local Proxy model. The idea is making data available not by client-server request-response mechanism, but by P2P data distribution model. When the information is not available on P2P subsystem, the client will get it from the server by request response mechanism. The P2P subsystem plays the role of cache memory, which is cheaper than data from the server.

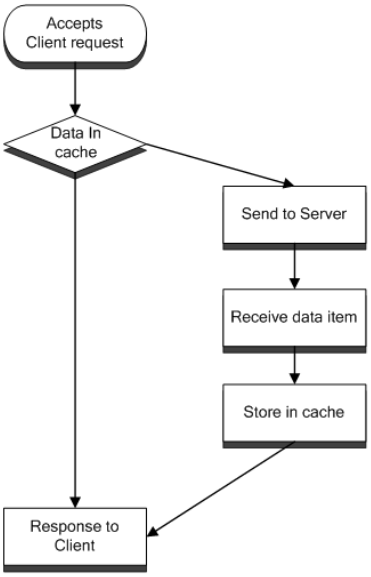

FIGURE 2: CACHING MECHANISM

Figure 3 describes this model. In this model, client does not interact directly with server but interact with module Local Proxy. Local Proxy can connect directly to server and use module P2P Agent to connect to P2P Subsystem. Algorithm to fetch data in this model is described in Figure 4.

The advantage of the model is resolving the weak scalability of client-server model by using P2P Subsystem. Especially, the high-level business of application is not affected because the model only modifies the communication interface of Client-Server model from Client-Server to Client-Local Proxy.

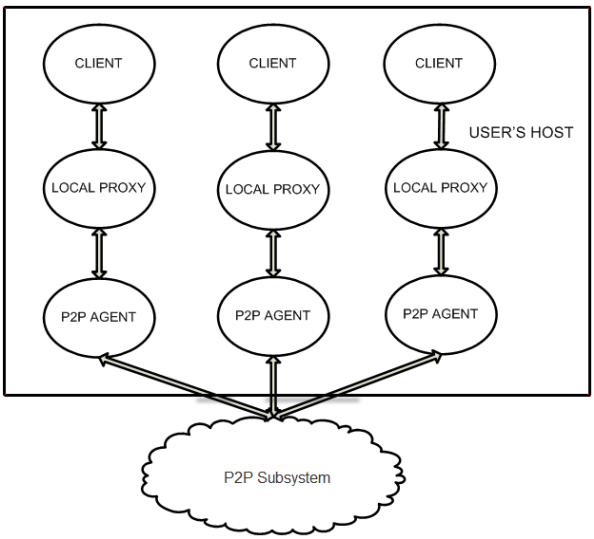

FIGURE 3: LOCAL PROXY MODEL[9]

The disadvantage of this method is the delay of getting data item from P2P subsystem. The data item can be retrieved from P2P system as a whole, so the data item may visit more intermediate peer before arriving at the destination. Anyway, the problem can be resolved by using small data items, but data item partition depends strongly on the nature of the application. In addition, before delivering the data item to client, the cache manager has to execute several activities: retrieving data from server, storing it on the local cache and put it into the P2P subsystem. The latter can be re-scheduled to be executed later, that it doesn't affect the performance of the system.

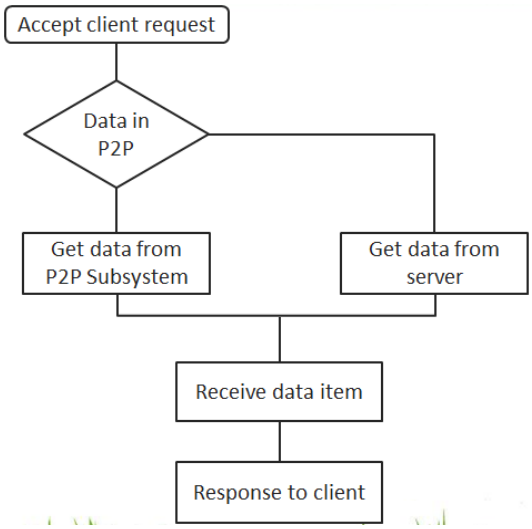

FIGURE 4: ACCESS DATA IN LOCAL PROXY MODEL

Based on the analysis of caching model and local proxy model, we propose shared caching model, a model allowing clients to share the cache content each other using P2P subsystem, thus improves the overall system's performance.

2.2. Shared caching model

In shared caching model, P2P and cache component are added to traditional client-server application. With these components, we aim to resolve the server bottleneck and bandwidth problems of client-server model. Figure 5 describes the components of shared caching model.

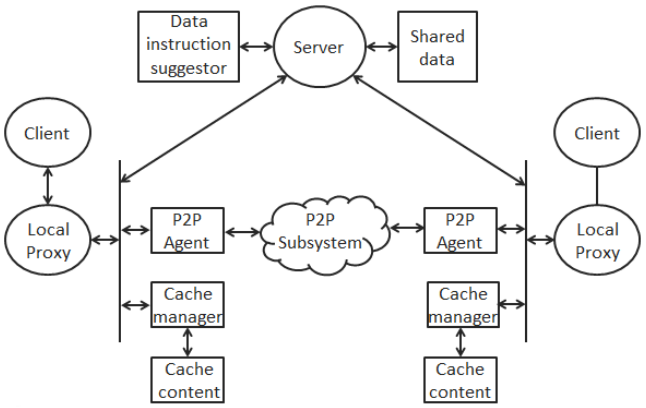

FIGURE 5: SHARED-CACHING MODEL

In client server model, server stores all application data and distributes to clients depending on their requests and application business. Channel for transferring these data is client-server connection. In our model, we add to client a cache component, these component implements two functions: (i) cache all data that is transferred to it. (ii) Expose a service to all nodes to answers questions about its current cached data.

Cached data is managed by an implementation of "page replacement algorithm" [5][6]. This algorithm allows client managing cached data effectively by two main tasks: (i) limiting the size of all cached data (ii) set high/low caching priority for data that having high/low accessing frequency.

The second component that we add in this model is P2P component, which allows a client creating a P2P connection to another one and exchanging data. With this function, cached data in each client not only be used in the current client but also be shared with others. The algorithm that client uses to access data in this model is: firstly it checks data in local cache, if the needed data is found, it responses to client, otherwise, client will send a request to server via client-server connection to ask about the instruction to get this data. Server will ask other clients for this data, evaluate and suggest the best way to get it for the client. In the following section, we describe about module "data instruction suggestor", this module is main difference between shared-caching model and P2P model.

In P2P model, the data lookup service is usually implemented by using distributed hash table (DHT). It is similar to a hash table; (key, value) pairs are stored in a DHT and any participating node can retrieve the value associated with a given key. Having many algorithms that implement DHT, i.e. Chord, Taspestry, CAN (content addressable network) [7], but all of them are complicated and data will be distributed among peers in the system that make the high data availability is not guaranteed when some peers go offline.

In shared-caching model, we provide a centralized implementation of the lookup service named "data instruction suggestor". This module is placed on server so it can easily to (i) collect information about cached data in one client via client-server connection (ii) collect reports about P2P connections from clients. Depending on this information it can suggest for one client the effective way to retrieve a specific data. This suggestion is called "data instruction"

3. EXPERIMENTAL IMPLEMENTATION AND EVALUATION

To evaluate the effectiveness of our model, we developed a file sharing application that can run in three models: client-server model, client-server with cache model and shared-caching model. After that we measure and compare the speed in each model with the same data set.

3.1. Quicknode introduction

Quicknode is a simple file sharing application that aims to allow user downloading files in server fast and easily. Quicknode is a web application; it includes a file server serving files to sharing between clients. In client, Quicknode allows user uploads, downloads file, manages personal directory and shares a personal file with others.

But with our experimental purpose (measure and compare the speed of downloading file between three models) we only develop the application in the limit with these functions: (i) allow a client downloading all files in server. (ii) Log information when download a file. To keep the application simple, instead of graphic user interface, we use console interface. The following are the list of commands that Quicknode supports:

Command	Example	Description
open $node_id	open qns	Open connection to other node (client or server)
download $file_id	download 31	Download file having file id equals to $file_id
empty $file_id	empty 31	Empty cache of file having file id equals to $file_id
logs		List all existed logs
log $log_id	log 2	View detail of log having log id equals to $log_id
file $file_id	file 31	View report (common file information, cache status) of file having file id equals to $file_id.
clear		Clear screen

3.2. Technologies used in Quicknode

Having many technologies to develop P2P application: openp2p (C++), libnice (C), libswift (C++), JXSE (Java)[11], WebRTC (Javascript, HTML5) [4]… After researching we chose WebRTC to implement P2P component for Quicknode because it is simple to develop and deploy(Javascript is simple language and web application is easy to deploy: without installing and not require any special system environment). In server we chose Nodejs[2] platform, a software platform that is a packed compilation of Google V8 Javascript engine [10], to write server application with Javascript.

Other technologies in Quicknode are listed below:

Technology	Description
SocketIO[3]	Javascript library to create connection between client(browser) and server(Nodejs, depend on browser, it will create a corresponding connection: websocket, flash-socket, ajax long polling)
2Q algorithm [6]	In page replacement algorithm, 2Q algorithm belongs to LRU (Least Recently Used)

	algorithm group. We implement this algorithm to manage cached data in client.
HTML5 filesystem	In browser, using to store cached raw data
HTML5 localstorage	In browser, using to persist data as (key; value) pair

3.3. Quicknode protocols and design

In this section, we describe data models and mechanism to download file as multi-parts. We also introduce the algorithm that used in Quicknode to download one file from server and/or clients.

Algorithm 1: client downloads file from server and other clients

Input: *fileId:* id of the downloaded file, *environmentParams:* client's information, *webSocket:* connection to server
callback: this function will be invoked when receiving all data

```
1:  buffer ← new DataBuffer()
2:  missingRanges ← ["0-"] /*Contain list of ranges that must
    download */
    // Callback function when receiving data
3:  function onReceivedData(ranges, data)
4:    missingRanges ← missingRanges – ranges
5:    buffer.save (ranges,data )
6:    if missingRanges = ∅ then
7:      callback (fileId, buffer)
8:    end if
9:  end function
    // Main business logic
10: cachedFile ← getCachedData ( fileId )
11: onReceivedData(cachedFile.ranges, cachedFile.data)
12: if missingRanges <> ∅ then
13:   begin
14:     request ← { fileId, missingRanges,
15:                 environmentParams}
16:     instructions ← webSocket.requestInstructions(request)
17:     for instruction in instructions
18:       begin
19:         if instruction.type = "webrtc_p2p" then
20:           connection ← openWebRTCConnection
21:                 (instruction.node_id)
22:         else if instruction.type = "xhr" then
23:           connection ← openXHRConnection
24:                 (instruction.node_id)
25:         end if
26:         connection.download( fileId, instruction.ranges,
27:                 onReceivedData)
28:       end for
29:   end
30: end if
```

Two most important data models in Quicknode are line and range. Line is minimum data unit that transferred over the system, all lines in Quicknode have a fixed size and equal to each other. Each line has a unique index to distinguish it from other ones in the same file. The index of a line starts from zero and identified by position of its first byte in the file. Range is set of continuous lines that represent by index of lines in order and with minimized format. Examples: (1) set of lines 1, 2, 3 is represented in range is: {"1-3"} (2) all lines of a not empty file: {"0-"} (3) empty file: {}.

To download multi-parts of a specific file, client sends pair of {file_id, array of ranges} to target node. For example: to download lines 1,2,3,8,9,10 of file having id = 0, client sends the request {file_id: 1, ranges:["1-3", "8-10"]} to target.

Algorithm 1 describes the way to download a file in Quicknode when using shared-caching model. If Quicknode runs in mode client-server with cache, line 16 will be modified to:

16: *instructions* ← [{type:"xhr", ranges: *missingRanges*}]

If Quicknode runs in mode client-server without cache, line 16 will be modified like case "client-server with cache" and function getCachedData will always return empty cached file data.

3.4. Test and evaluation

To test Quicknode, we used three machines: one for server (Windows 2003, Intel Xeon 2.40GHz, 512MB of RAM) and two others (Windows 7, Core i3 2.40GHz, 4GB of RAM) run Quicknode client applications in Chrome(Version 31.0.1610.3) browser. Two client machines are on the same LAN, client machines connect with server machine via WAN.

We run the test in set of files(11 files) having size: 10.39 KB; 50.04 KB; 100.54 KB; 192 KB; 503.54 KB; 994.27 KB; 1.89 MB; 3.99 MB; 10.45 MB; 19.28 MB; 48.11 MB. With each file we run 12 tests and measure the average speeds of fetching file from sources: Cache, P2P, Client-Server. Configurations of 12 tests are:

	Ratio(%) of file data fetching via P2P and Client-Server +P2P					
	0	20	40	60	80	100
1.Cache off						
2.Cache on, 50%						

After that we calculate the general average speeds of fetching data from each source equals average value of above average speed (in corresponding source)

The general average speeds of fetching data via corresponding sources are: from other client via P2P: 721 KB/s; from server: 353 KB/s and from local cache: 6.08 MB/s.

FIGURE 7: COMPARE SPEED BETWEEN P2P AND CLIENT-SERVER

In this experiment, we also test the dependency of time spent and downloading speed when changing the ratio of file data that is fetched via P2P in two cases: (1) cache is off (2) file was cached 50%.

From this graph (Figure 8) we see that, when more data is fetched from P2P, the average speed will increase very much (from 353 to 721 KB/s). In case the file was cached 50%, the time spent to get file reduces 2 times because 50% data is fetched from

local storage which has extreme fast speed in comparison with P2P and Client-Server.

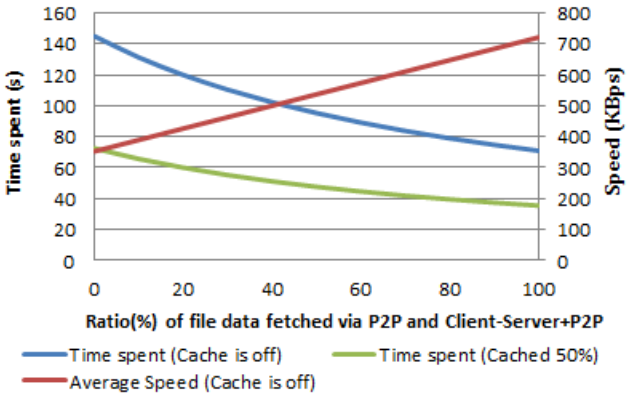

FIGURE 8: GRAPH OF TIME SPENT AND SPEED WHEN DOWNLOADING A FILE HAVING SIZE 48.11MB

With these data, we conclude that the performance of P2P and cache component in Quicknode is good when clients are on the same LAN and our proposal model can be used to upgrade many kind of client-server application to resolve the server bottleneck and bandwidth problem. Anyway, additional exchanges make the improvement not so significant in the case of small files. In the future, we will deploy the clients on WAN to have much more data about this model.

4. CONCLUSION

In this paper, we present a shared caching model for client-server application. One step far from caching mechanism, this solution allows clients to use caches of other clients as a secondary cache. The advantages of the solution are:

- Application of P2P model to improve Client-Server Application performance.
- No modification is needed at a Server side, while only cache management model in client side has to be modified.
- Keep the simplicity of client-server model

Application and experimental implementation are completed and evaluated. The result of the experiment shows that:

- Performance of P2P and cache component in client server application is good.
- The shared caching model is an effective approach to improve performance of client server application

In the future, we intend to focus our research on module "data instruction suggestor" to make it smarter. By using a smart algorithm like heuristic to calculate suggestion with many input parameters, i.e. performance, speed, current state and even history of client… the server will make the better suggestion which will improve overall system performance.

5. ACKNOWLEDGMENTS

The authors would like to acknowledge ministerial project NB2012-01-30, Hanoi University Of Science And Technology, ACM SoICT 2013 symposium for supporting us to finish this work.

6. REFERENCES

[1] Hiroshi YAMAMOTO, Daisuke MARUTA and Yuji OIE: Replication Methods for Load Balancing on Distributed Storages in P2P Networks. *IEICE Transactions 89-D(1): 171-180(2006)*

[2] Nodejs. Available at http://nodejs.org/

[3] SocketIO. Available at http://socket.io/

[4] WebRTC. Available at http://webrtc.org

[5] Page Replacement Algorithms. http://en.wikipedia.org/wiki/Page_replacement_algorithm

[6] Theodore Johnson and Dennis Shasha: 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm http://www.vldb.org/conf/1994/P439.PDF

[7] Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan: Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking, vol. 11, no. 1, pp. 17–32, 2003*

[8] Ha Quoc Trung (2012): New approach to develop the messenger application: from client server design to p2p implementation. *ACSIT-2012.*

[9] Nguyen Quang Thu: Study of distributed replication: design a messenger application using local proxy model. Master's thesis, Hanoi University of Science and Technology, 4 2013.

[10] Jesse Burstyn, Katricia Barleta, Lukas Berk, P Bennett Cole and Tom Franzon: Conceptual Architecture of Google Chrome. http://archrometects.files.wordpress.com/2009/10/assignment-01-conceptual-architecture-of-google-chrome-archrometects.pdf

[11] JXSE. Available at https://jxse.kenai.com/