P2P VOLUNTEERS FOR RELIABLE SERVER FARMS

Ying Wang and Partha Dasgupta
Dept. of Computer Science and Engineering
Arizona State University, Tempe, AZ
{yingw, partha}@asu.edu

Abstract

In recent years, there are more and more critical services being developed on the Internet with high requirements on availability and reliability. Server farms are one of the key approaches to improving the availability and reliability. However, due to the high maintenance cost and the "site disaster" problem of server farms, we are looking for cheaper infrastructure that is logically and geographical distributed to provide highly reliable services in spite of attacks and server failures.

In this paper we provide highly reliable, attack resilient servers by using P2P farms of volunteers. The volunteer group can be dynamically constructed, is scalable and is resistant to service failures resulting from the loss of a few peers. In our approach, the system will first form a self-organizing group of peers who are interested in a service. Then statistical and operational data, service response delay, client request pattern, and so on are collected. Finally, the system applies this information to select new servers, and integrate into the server farm till the target level of service is met.

Keywords
Distributed software systems and applications, Operating systems, Distributed computing, Fault tolerance

1 Introduction

The Internet is a shared resource, a cooperative network built out of millions of hosts all over the world [1]. The Internet is increasingly being used as access medium for a variety of critical services, for which very high requirements are set on availability and reliability [2]. Examples range from e-commerce sites, to per-pay media servers, to information servers in distributed mission critical systems, such as air-traffic control systems [2].

The obvious solution to the above problem is to make the service replicated. Suppose "acme.com" wants to build a replicated server. They can purchase large server farms at a few geographical locations. However, the network diversity (i.e. how dispersed they are topologically) is most likely to be small. In addition thousands of locations of server farms cannot be economically handled by Acme.com. Thus the high distribution requirements make it infeasible for one company to achieve, because it is very expensive to manage or purchase the distributed service centers.

The other approach that we are researching is for acme.com to "outsource" the service system to any volunteer (possibly paid volunteer) to provide the service. The volunteers are paid small amounts of money and can be home computer users or almost anyone. The volunteers are expected to provide their own system administration functions and are not under the control of acme.com. All they do is download some server scripts and data from acme.com and join the acme.com server system.

The resulting system is not a P2P system, but a client server system, constructed out of P2P based parts. The server, instead of being a set of dedicated, clustered machines, is a set of geographically dispersed volunteer hosts, coordinated through a P2P like collaboration framework. The distributed service network has a dynamically configurable structure using an overlay network that is discussed in detail in this paper.

The volunteer system is structured as a P2P system, i.e. the groups of volunteers self-configure into peer network, with certain structure and collaboratively provide the service. Each peer is not expected to be highly reliable, attack proof or even trustworthy. A set of protocols monitor the peers and steer the clients to the peers based on their reliability. The payment to the volunteers is based on how much "help" they provide in delivering the acme.com service. (Of course, the same peers could also be carriers of other service providers).

Thus, we argue, this approach has a potential of much further distributed reach and location diversity than a acme.com purchased server farm. In addition, the volunteer approach will be cheaper and have significantly lower management costs. A lot of the service costs will be paid out of "idle cycles" similar to the way grid computing is done. Also the client-server model embedded in the server farms can help us yield better performance, availability and scalability than P2P massively decentralized approaches as shown in [3].

In the following section, we outline the work conducted by the research community on P2P systems, application layer multicast, and an overlay peer utility service. We describe the P2P volunteer network design in section 3. In section 4, we analyze the results from our message-level P2P service simulator. We discuss future work and conclusions in section 5.

2 Related Work

2.1 P2P Systems

P2P systems can be differentiated by the degree to which the overlay networks contain some structure or are created ad-hoc [2][4]. The structure means the way in which the content of the network is located with respect to the network topology. *Unstructured:* The main characteristic of unstructured P2P systems (such as Gnutella [5]) is that the placement of data is completely unrelated to the overlay topology. *Structured:* Structured systems were proposed (such as CAN [7], Tapestry [8]) in which the overlay network topology is tightly controlled and files (or pointers to them) are placed at precisely specified locations. Another approach is the P2P communities [8][9] which turns groups of ad-hoc P2P systems into logical communities.

2.2 Application Layer Multicast

In order to improve the system performance and efficiency, we use application layer multicast for the group communications of the server farms formed by peers. Typically, the goal of application layer multicast algorithms is to build distribution trees that minimize the additional routing overhead compared to native IP multicast. There are several approaches that are close to our proposal, including overcast, End System multicast, P2P systems, which we discuss below.

As one of the tree first solutions, Overcast [10] explicitly measures available bandwidth on an end-to-end path and builds a multicast tree that maximizes the available bandwidth from the source to the receivers. End system multicast [11][12] is one of the tree-over-mesh solutions. It builds a mesh structure across participating end-hosts and then constructs source-rooted trees by running a routing protocol. Some P2P projects (CAN [7] and Tapestry [8]) address the scalability issue by creating application layer overlays. The common idea of these approaches is to realize a scalable lookup service for objects where the responsibility for managing the object space is shared equally among a network of peer nodes.

2.3 Overlay Peer Utility Service

Opus [13] is a large-scale overlay utility service that provides a common platform and the necessary abstractions for simultaneously hosting multiple distributed applications. Opus consists of a number of cooperating components: service level agreements (SLAs) specify the amount each customer is willing to pay for a given level of network service. Opus uses these SLAs to perform resource allocation among competing applications during times of peak demand. The application's SLA, current demand level and node allocation determines the target per-application overlay topology. Different from Opus which is aimed to provide general utilities for all kinds of network services, and our approach is specifically for high reliable service and so we focus on solving the reliability problems of current Internet services.

3 Our Approach

3.1 Overview

In our approach, we build an infrastructure using P2P architecture that could dynamically construct and configure server farms to meet target levels of service based on the changing network characteristics and peers' statues. The basic approach is: first, form the peers interested in a specific service into a group in a self-organizing fashion. Second, collect statistical and operational data, such as service response delay, client request pattern, and so on. Then, apply the information to select new servers, and integrate them into the server farm, till target level of service is met. Last, a "frequent flyer" program is used to encourage the peers to contribute resources and get bonus from the contribution.

3.2 Architecture

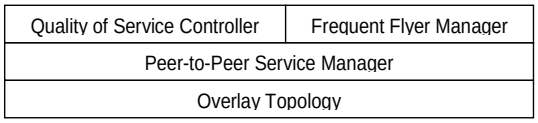

Figure 3-1. System Architecture

As shown in figure 3-1, the P2P services system is composed of three layers: The overlay topology layer is used to form and maintain the overlay topology of P2P service systems. The P2P service manager layer provides management functions of the P2P service. And the top layer includes two modules: The QoS controller adjusts the server farm's configuration to improve the quality of service and maintain the quality of service at the target level. The "frequent flyer" manager motivates peers to contribute resources by providing bonus to the contributors.

3.3 Overlay Topology

3.3.1 System Topology Overview

In our system, the overall topology of the system is a tree of trees. All the servers form the main tree with the service source as the root, i.e. each node in the main tree is a server. And each server is the root of a tree for clients, whose nodes are all clients expect the root node. As shown in figure 2, the clients of server A form one group, and the dash circle around A indicate the size of the client group, and the same for server B. Since the statistics data like request pattern is required to be monitored at the client level, also client needs service information to select appropriate servers, clients of one server form a multicast tree.

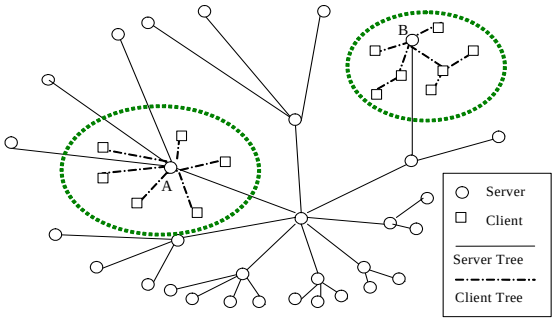

Figure 3-2. System Topology

Each tree in our P2P service system topology is a source specific overlay application multicast tree. We adapted the ideas from application layer multicast research, e.g. [10] on building a tree based on the network condition, including latency, and bandwidth for better system performance and scalability.

3.3.2 Tree building

In our approach, we assume that each peer knows a list of other peers from bootstrap, and the peers in the list are called neighbors of the peer. We achieve the tree building in the following three steps. First, the source sends out tree-creating-message to all of its neighbors. Second, for each peer that receives the tree-creating-message will broadcast the tree-create-message with TTL (Time To Live) equals to two, and if it is interested in the service, it will send ion message to its parent. Third, for each peer receives joining request will follow a tree-joining algorithm. Basically, if the peer has the capability to accommodate one more children, it will accept, otherwise, it will compare its worst child with the requesting one, and accept the better one, reject the worse one. While after the first multicast tree of the service is setup, the tree will be split into multiple trees, and we address the process of split and organizing in section 3.4.2.

3.3.3 Maintenance

While in P2P network environment, the P2P overlay networks are dynamic, heterogeneous and unpredictable. To maintain the system topology, we use two heartbeat timers at each peer in the tree. One timer is used to send out heartbeat message periodically and the other is for detecting connections to its neighbors (parent and children), if it does not receive its parent's heartbeat on time, then it will start selecting new parent. If it does not receive its child's heartbeat, delete the child from its children list.

3.3.4 Optimization

In the process of optimization, the overlay topology mimics the real physical network connections so that the difference between them is minimized. The peers execute the optimization algorithm periodically to adjust their positions in the tree dynamically to achieve better system performance and scalability. The peer follows the optimization algorithm to find a better parent upon receiving the heartbeat message from its parent.

The basic idea of the optimization algorithm is that: first, if there is no penalty for the peer to get the message from its sibling, that's the physical path from the parent to the peer goes through the sibling, then we use the sibling as the peer's new parent, then we can reduce the link stress between the parent to the sibling, that means we saved network resources. Second, if it is possible for the peer to move closer to the source in the tree, that is if the capacity of its ancestor is not full, it's more efficient to add the peer under its ancestor than under its parent. Peers receive join requests from siblings or decanters will follow tree-joining algorithm.

3.4 P2P Service Manager

The P2P Service Manager is the layer used to setup, and maintains the P2P services. We address how it works on peers in different roles, general server, client, and root server.

3.4.1 Servers

In our approach, the servers refer to the peers who provide P2P service to other peers. A peer may be a server in one P2P service, and a client is another P2P service. In our overlay topology, the server belongs to the tree for servers of the specific service, and the server has a group of clients reporting aggregation data to it.

- Data Propagation

Servers provide data propagation functions for system management. The data propagation includes the data multicasting from the server to its clients and also the data propagation from the server to all the other servers.

- Data Collection

Every server periodically collects data from its clients for decision-making usage. Also the server records its own data and aggregates the data from its decanters servers in server farm, and then report to its parent.

- Failure Recovery

When a server is online again, it first contacts its previous parent in tree for servers, and joins the tree. Then it will download the new updated service from its parent if there is any. Followed, the server will update all the service related information, like the updated server list.

- Network partition

Network partition happens only when the root server is offline, since the other type of partitions can be easily fixed by rejoining from the root. When the root server is offline, a new root server will be elected though root server election algorithm, and the elected root server will

perform the responsibilities of the root server, until the original root server is back.

- Service Quitting

When a server is going to be offline permanently or is willing to stop providing the service, it will quit the service and propagate its leaving to all the other servers and all the clients. On the other hand, if there is no or only a very small number of service request for that server for a period of time, the server will stop its service to improve overall system efficiency, since the cost for its existence including control overhead, communication cost on server updates is larger than the benefit it can provide.

3.4.2 Clients

In our overlay topology, the client belongs to one server and the clients could switch to other servers at any time.

- Joining

There is only one server of a specific service at the every beginning; all the clients joins same server. Later, the system will add servers for the service gradually, and each time a new server is added, the clients compare the distance (RTT) to new server with the old server, and choose the closer one to join. Anytime when the server is offline or overloaded, the clients of it will switch to the next closest server among the available servers.

- Data Gathering

According to the service requirement, the client is asked to collect some service related data. The most important data includes the request pattern for the service, and the evaluation of the servers, etc.

- Server Selection

Since there may be thousands of servers for one specific P2P service, how to select the appropriate one is an important issue for the client. In our approach, each client maintains a list of servers, and their performance records. Before each service call, the client will sort the servers according to their performance, and select the best one, and after each service call, the client will evaluate the server performance, and put it in the records.

- Server Promotion

All the clients of a P2P service are the potential servers of that service. There are two possible ways to promote clients to servers. First, the client can volunteer to provide service. Second, the client is selected by the system to be the best new server and the client agrees to become a server.

- Service Volunteer

A client of a P2P service can express its willing to become a server to its current server, and the server will transmit the service to it upon receiving the request.

- Placement of New Server

In order to improve the scalability and efficiency, we follow the following steps to choose a new server. First, the server chooses the worst average metrics from its client children list based on the data collected. And this selected client also chooses the worst average metrics from itself and its children list, and so on so far, the client received the worst quality of service could be identified. And that client will be the new server.

- Server Transmit

A client becomes a server when another server transmits the service to it. After the transmission, the server becomes the parent of the new server in the main tree. The new server propagates the adding of new server to all the servers and clients. Clients may switch to the new server according to the distance.

3.4.3 Root Servers

The root servers refers to the peers that are willing to provide itself created service to the other peers and propagate the service to other peers to share its workload. The root server is the root of the tree for server of the specified service, and it plays the most important role in the P2P server.

- Service Creation

When a new service is setup on a peer in the network, the peer could choose to create a P2P service, and then the peer becomes the root server of the P2P service. To do that, the peer needs to call the underlying overlay topology layer to build a multicast tree with itself as the root.

- Service Update

Each time there is an update to the service, the root server propagate the new service through the tree for service. Every time the server is recovered from failure, it will ask it parent for new update if there is any.

3.5 “Frequent Flyer” Program Manager

In our P2P service system, we design the “frequent flyer” program to motivate the peers to contribute resource. As in the airline company, the customer will get bonus by accumulating fight miles with this airline. Similar in our “frequent flyer” program, the peers contribute to the system will get its bonus, money from the root service provider. The more it contributes, the more it could get.

In our approach, the total available time of the P2P server is used as the metrics for rewarding. The root server maintains a list of all the servers, for each server, the total available time and its bonus is recorded. The root server periodically accesses the servers in the list, and computes their available time and bonus. For each month or year, the root server gives out the bonus, money, to the servers according to the list and then reset the list.

3.6 Quality of Service Controller

The QoS controller provides functions to achieve more specific requirements of the services. In our approach, we use feedback to control the QoS. The QoS requirement is set up when the service is created, and data will be collected for decision making, after the action is taken, goes back to the data collection to make the second decision, and so on so far, QoS is improved step by step, and maintained at that level.

Depends on the requirement item, the corresponding metrics will be collected through client's data gathering. After the data collection, the servers compare the their statistics metrics with the requirement, if the statistics data is worse, then they start adding new servers as their children following placement of new server and service transmit. If the statistics data is already better than the requirement, then do nothing.

4 Simulation

In this section we present results from our message level simulator. Our simulator is written in C#, and can simulate the entire P2P service system with thousands of nodes. Through our simulation experiments, we used Georgia Tech [14] random graph generators to generate topologies of the P2P network.

We designed and implemented experiments under three kinds of situations. *Targeted attack:* a number of servers in the server farms are attacked. *Instable environment:* all the peers go on and off randomly. *Low capability server:* the servers can only response to a limited number of requests at a time.

4.1 Targeted Attack

We simulate targeted attack by turned off a number of random picked servers periodically, and the servers recover from the attack at a random time in a range.

First, we compare of the actual data collected by the simulator and the statistics data collected by the system, and their relation to the number of servers. The results (Fig. 4-1) show that with the increasing of the number of servers, both the actual data and statistics data are getting closer to the system QoS requirement and the number of servers increases exponentially until the target level of service achieved, and then keeps around the ideal number of servers.

Next, we compare the system performance in networks of difference sizes. Figure 4-2 shows that the larger size the network is, the more time it takes to stabilize the number of servers required for the system. The reason is that it takes time for the system to collect data and add servers to adapt to the changes of the system. Since in this simulation, we simulated targeted attacks with 10% peers

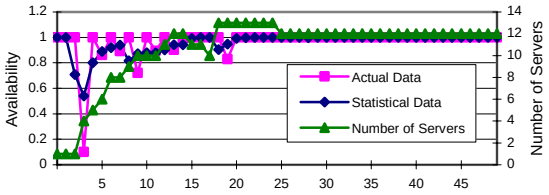

Figure 4-1. Actual Data vs Statistical Data in targeted attacks

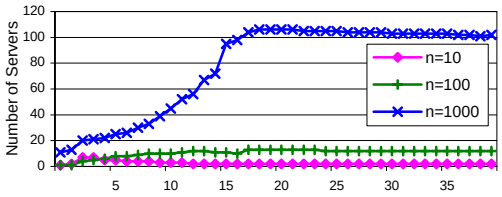

Figure 4-2. Num of Server vs Network Size in targeted attacks

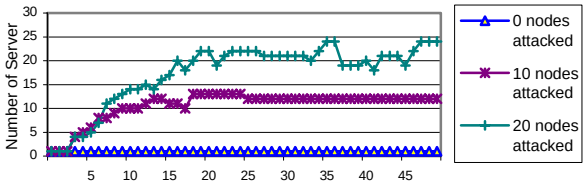

Figure 4-3. Attack Severity vs Number of Servers in targeted attacks

been attacked, the larger size the network, the more number of servers been attacked and the more number of servers needed to add, and the more time it takes to reach to requirement.

Last, we compared the impact of the attacks in difference severity. The results (Fig. 4-3) show that the more number of servers been attacked, the more time the system needs to reach the ideal number of servers, the more number of servers needed in the server farm and the larger range of the number of servers could wave around the ideal number of server. The reason is similar as we described in previous experiment. Basically, the more number of servers to be added, the longer time it takes to achieve the system requirement.

4.2 Instable Environment

We simulate the instable environment by turned on and off all of peers periodically with configured peer on and off interval ranges. We compared the impact of the on interval/off interval rate on system performance. The results (Fig. 4-4) show that with the increasing of the rate, i.e. decreasing the off interval while keeping the on

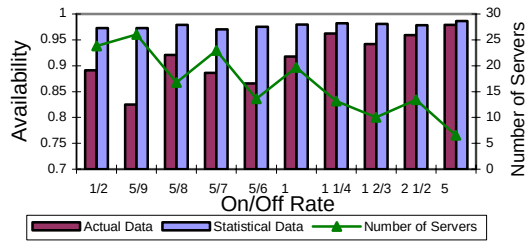

Figure 4-4 System Performance in Instable Environment

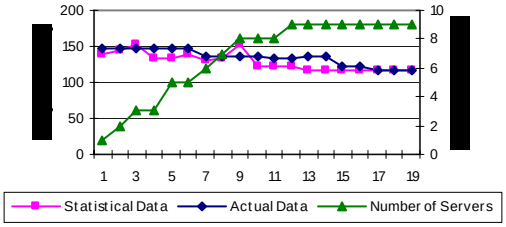

Figure 4-5. Response Delay vs Number of Servers in stable environment

interval the same, the number of servers required decreased and the system performance increased.

4.3 Stable Environment

We assume in the stable environment, there is no attacks and turn downs on any peers in the network. First, we show that the system could be used to improve the service response delay. The results (Fig. 4-5) show that the number of servers increase linearly, and with the increasing of the number of servers, the system statistical data and actual data collected by the simulator decrease gradually until they reached the requirement. Second, we did our simulation in different sizes of networks. We set the response delay as the server response time plus 10% of the average RTT between two peers. The results show that the larger the network, the less percentage of servers needed to reach the requirement. Also, the larger the network, the more rounds needed to reach the requirement.

Network size	Response delay requirement	Num of Servers	Stabilized response delay	Rounds
10	102	3 (30%)	102	6
100	120	9 (9%)	117	13
1000	300	8 (0.8%)	292	84

Table 4-1. System Performance vs Network Size

Last, we did our simulation with different requirement in a network of 100 peers. The results show that the higher the requirement, the more number of servers needed to reach the requirement, the more rounds needed to reach the requirement.

Response delay requirement	Num. of Servers	Stabilized response delay	Rounds
150	1	147	1
140	3	135	4
130	5	123	7
120	9	117	13
115	11	114	19

Table 4-2. System Performance vs QoS Requirement

5 Conclusion

In this paper, we addressed the problem of providing P2P service for high reliability. We use the P2P architecture for the attack resilient services, and we argue that this structure has advantages over the replicated approach used conventionally.

Through various simulations using our message-level simulator, we show that our infrastructure can provide high reliable service in various environments. The P2P service is resilient to targeted attacks, random attacks, and it also shows good performance in instable environment where peers are turned on and off frequently. Also, in the normal stable environment, the P2P service could provide high quality of service by automatically configuring the server farms.

We believe that our approach is a novel way of providing high reliable service on internet, especially in service-critical environments.

References

[1] Andy Oram, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies* (O' Reilly, March 2001)

[2] G. Shenoy, S. K. Satapati, R. Bettati, Hydranet-Ft: Network Support for Dependable Services, Department of Computer Science, Texas A&M University.

[3] A. Vahdat, J.Chase, R. Braynard, D. Kostic, P.Reynolds and A. Rodriguez, Self-Organizing Subsers: From Each According to His Abilities to Each According to His Needs, Duke University.

[4] Karl Aberer, Manfred Hauswirth, An Overview on Peer-to-Peer Information Systems, Swiss Federal Institute of Technology (EPFL), Switzerland, 2002

[5] Q.Lv, S.Ratnasamy, and S.Shenker, Can heterogeneity make gnutella scalable? , *Proc. 1st International Workshop on Peer-to-Peer Systems*, Cambridge, MA, March 2002

[6] S. Ratnasamy, P. Francix, M. Handley, R. Karp, and S. Shenker, A scalable content-addressable network, Proc. ACM Sigcomm, Aug 2001.

[7] B.Y. Zhao, J. Kubiatowicz and A. Jeseph, Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing, University of California, Berkeley, CA, Apr. 2001

[8] M. Khambatti, K. Ryu and P. Dasgupta, Peer-to-Peer Communities: Formation and Discovery. *In 14th IASTED International Conference on Parallel and Distributed Computing Systems*, Cambridge, MA, November 2002.

[9] M. Khambatti, K. Ryu and P. Dasgupta, Structuring Peer-to-Peer Networks using Interest-Based Communities? *International Workshop on Databases Information Systems and Peer-to-Peer Computing*, Berlin, Germany, Sep 2003.

[10] J. Jannotti, D.K. Gifford, K. L. Jognson, M. F. Kaashoek and J. w. O'Toole Jr, Overcast: Reliable Multicasting with an Overlay Network, *Operating System Design and Implementation*, Oct. 2000

[11] Yang-hua Chu, Sanjay G. Rao, Srinivasan Seshan and Hui Zhang, Enabling Conferencing Application on the Internet using an Overlay Multicast Architecture, *Proc. ACM SIGCOMM* 2001.

[12] Yang-hua Chu, Sanjay G. Rao, and Hui Zhang, A Case for End System Multicast, Proc. ACM SIGMETRICS, 2000.

[13] R. Braynard, D. Kostic, A. Rodriguez, J. Chase and A. Vahdat, Opus: an Overlay Peer Utility Service, *Proc. 5th International Conference on Open Architectures and Network Programming*, June 2002.

[14] Modeling Topology of Large Internetworks http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.htm