2004－OS－96　(7)
2004／6／17

Federated Computing within an Untrustable Infrastructure

Emil Meng,[†1] Hirotake ABE,[†2] Kazuhiko KATO,[†2,†3]
Mizuki OKA,[†4] Richard POTTER[†2] and Péter SURÁNYI[†3]

High availability is desired by many parties ranging from large-scale businesses running mission-critical servers to individuals hosting vanity web-sites. However, in order to achieve such high availability, one would need to have access to a great deal of resources, such as a computing cluster environment. Such solutions are usually prohibitively expensive for most parties, but if the services could be replicated in a peer-to-peer manner, it would dramatically bring down the cost. Since the advent of peer-to-peer computing became popularized, systems have been primarily created to ensure that access to data remains highly available. We take this previous idea and refine it to meet our goal of making services available in a secure manner.

1. Introduction

Harnessing the power of a massive number of computers orchestrated to attain a goal has been accomplished through many different groups. The Search for Extraterrestrial Intelligence (SETI)[1)] has exhibited a phenomenal success in garnering users to donate their processing power during their idle cycles and have in turn progressed science. Napster[2)] and more recently Kazaa[3)] have also brought millions users together to share and trade data. Google[4)] also has massive computing farms in order to carry out the demand of millions of searches per day.

The infrastructure of federated computing has historically been created and maintained within a trusted cluster. Most corporations use distributed computing solutions in order to achieve high availability and these solutions usually cost on the order of millions of dollars, if not more. For the average computer-user, the cost of creating such an infrastructure is a major prohibiting factor and thus we have set out to create a system that would guarantee high-availability in a peer-to-peer backbone.

Peer-to-peer networks have been insecure by nature due to the fact that any peer joining the network has the possibility of being an adversary. In order to stay chaos within an untrusted infrastructure, other means need to be in place in order to validate the information being passed.

Throughout this paper, we will present a general discussion about bringing federated computing into a peer-to-peer environment as well as propose a simple model that exemplifies our ideas. In section 2, we will explain what motivated us to conceive such an idea as well as related works that shaped our methods. Afterwards, we present our approach to solve the problem of building a federated computing group given an unstrustable infrastructure in section 3. In section 4, we discuss a variety of different attacks that may be launched against our system and hypothesize how they will fare through them. Section 5 will present our weaknesses in our system and we will close in section 6.

2. Motivation and Related Works

High availability is always nice to have but isn't necessarily financially practical for the vast majority of users. Our motivation comes from taking services, such as apache, and ensuring that they maintain high availability at a low cost. Using a peer-to-peer infrastructure can achieve such a goal and could possibly lead to other beneficial effects, which includes a lower utilization of global bandwidth as well as a faster retrieval time.

At this time we would like to define what we mean whenever we mention federated systems. Computers within federated systems work together to accomplish a certain task, but instead of being dedicated to that task, they remain autonomous and are under direct control of their

†1 University of Colorado, College of Engineering and Applied Sciences – Department of Computer Science
†2 CREST, Japan Science and Technology Agency
†3 University of Tsukuba, Graduate School of Systems and Information Engineering
†4 University of Tsukuba, Master's Program in Science and Engineering

owner.

Federated systems within a trusted infrastructure have existed for a long time, and there has also been advances in federated systems within untrusted networks. SETI[1)] in particular, has taken an untrusted infrastructure and created one of the world's most successful, and possibly the fastest, computing farms at a fraction of the cost of a traditional supercomputer. However, SETI's approach uses a master/slave relationship where the users grab data sets from a central server. This central server is where a single point-of-failure could occur, and thus does not satisfy our requirement for being highly-available.

The High Availability Linux Project[5)] also sets out to bring reasonably affordable high availability computing to both users and corporations alike. They achieve this goal by creating a trustworthy infrastructure that the administrator protects and maintains. While this is a feasible solution to most users wishing to have security by maintaining a reliable mainframe, a single point-of-failure still exists from an outsider view. These clusters are usually created behind a single internet connection, and if that connection were to cease functioning properly, any services run within that cluster would remain dead to the outside world.

Most existing peer-to-peer systems' purposes focus on making data available to its community. Some systems make the data available based on the popularity, while others wish to make the data longevous, such as Free-Haven[6)] or the Eternity Service[7)]. Our proposal departs from these traditional methods, and instead of making data available, we make services available. There are obviously many issues that need to considered, as a redirection of this magnitude requires critical analysis.

3. Our Approach

In this section we will flesh out what needs to be done in order for our peer-to-peer backbone to successfully support and maintain itself. We rely on virtual machines to provide service as well as peers to host them. On top of that we have a central authority to provide knowledge to both interested peers and clients.

3.1 Overview

In this section, we outline what is necessary in order to create a highly-available federated system within a peer-to-peer backbone. First and foremost, we are going to need virtual machines to run the services that we wish to make highly available. Furthermore, we explain how any service can be run within the virtual machines within the peer-to-peer backbone. Since we have been endlessly stating how we are creating this system with a peer-to-peer infrastructure, unsurprisingly, our system needs peers to host these virtual machines. Along with that, there is a central authority that acts as a guide to give necessary information to all the peers, as well as the regular clients. The system also is very lax on security and it is up to the peers to make sure that everyone is playing by the rules.

3.2 Virtual Machines

In order for a peer to run a given service, it is not feasible to do so given a list of rules and/or specifications. Many services are very complex and thus a simple rule-set would not suffice in mimicking the original intent. For example, apache may have a configuration file, but if a page it serves requires some module that is non-existent on another peer's machine, there is no easy way to distribute that dependency. Thus instead of mimicking, we propose that an entire virtual machine be run. There a few different virtual machines that can be used for this purpose, but we will feature Scrapbook for User-Mode Linux (SBUML)[8)].

The reason we choose SBUML over other virtual machine solutions is that it falls in line with our design goal to make solution inexpensive. Another major factor in deciding to use SBUML is that it, as its name implies, can take snapshots of a machine's state. This allows that state to be resumed by anyone who received it.

3.3 Running Services

As mentioned before, a server only has to take a SBUML snapshot of the machine so they can distribute their saved state to the community. Usually these snapshots are very large (on the order of hundreds of megabytes), however, since SBUML also supports a *delta mode*, a taken snapshot that can be scaled back in size. This *delta mode* takes a common base, and any changes above the common base is recorded. Thus, if one takes a kernel as their common base, and only installs a simple service on top of it, a snapshot that would normally take hundreds of megabytes would then take only a fraction of that.

As our motivating example above, let us again take a complex and convoluted apache service setup. In this complex setup, there are

many different modules that are loaded and the server will also have a complex configuration file that is specialized for a single version of apache. With the use of SBUML, there is no need to worry about making sure that the environment is correct so that the service is acceptable because the entire state of the machine is passed. Since the apache service at this point is already in a running state, so there is also no hassle of booting the virtual machine and making you start the requested service. Thus, all that needs to be done in order to correctly run this complex service is to acquire the *delta mode* snapshot, and execute it.

Thus we suggest that a standard base kernel be created and distributed so all servers have a common base. This is done for a number of reasons. The first and most obvious one is that there is a lot of space saved by everyone using the same base kernel. Thus the smaller delta snapshots only needs to be distributed, making the system more efficient. The other reason for doing so is to make the system more secure. Peers can validate which kernel they're running by checksumming it against the known checksum provided by the central authority. Thus if only trusted kernels are allowed to be run, which we highly advise, there is a lower possibility of a server sending a malicious stated virtual machine into the community.

3.4 Peers

Peers build the backbone that allows systems like ours to operate. Unfortunately, in a poorly designed system, often a single or small group of malicious users can severely cripple the operation of the system. Thus in order to ensure order within the system, much like other peer-to-peer infrastructures, we will reward the users who contribute to the community. A peer cannot expect others to willing serve his content if he himself is unwilling to do the same for others.

Each peer will also create a paired cryptographic key so that he may cryptographically sign his virtual machine he wishes to insert into the network. The reason this needs to be done is for the end-user accessing the peer's content will be able to verify that it indeed came from that peer, and not from a malicious host masquerading to be an authoritative voice of the peer. This key, along with any commitments the peer has made in hosting others' services, will be reported back to the central authority.

Another important issue to depict here is that when multiple services run on a single system, there will probably be some resource conflicts as far as networking is concerned. Some services will be more popular than others, and if there are two instances of a service on the same physical machine, a conflict for the port will ensue. To resolve this conflict, we propose that all virtual machines create a unique virtual private network address, and have the physical network interface bind and map random ports to the specified port within the requesting virtual private network; i.e., if two apache servers are run, one would have be running on 10.0.0.1:80, and the other would be on 10.0.0.2:80. And the physical address xxx.xxx.xxx.xxx would have two random ports chosen to map to each of the two private addresses respectively. Below, figure 1 depicts what was described above.

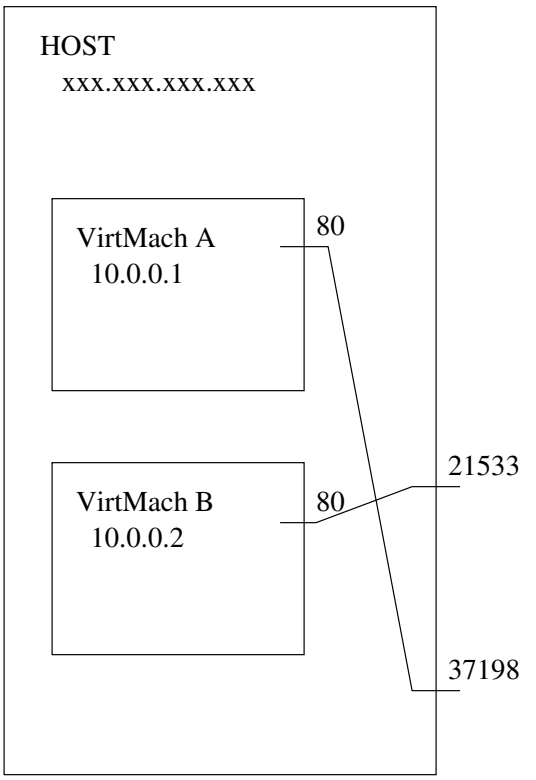

Fig. 1 A simple example of remapping a real ip/port address to a virtual machine's virtual ip/port address.

3.5 Central Authority

The central authority in our peer-to-peer system acts as an informant and maintainer of our system. It has multiple jobs including: the hosting of all public keys for all peers, remembering the commitments made by peers to serve

each other's content, and informing clients on how to access their desired service.

First and foremost, all public keys generated by peers in the system must report their identity to the central authority if he wishes to be known within the system. This is so that there is an authoritative voice describing who can answer queries about the identity of the peers within the system. This is needed because an adversary might claim to offer someone's service, but in turn offers something completely different. The only way to guard against this is to make sure that the received information is signed by the originating host.

Secondly, the central authority needs to be aware of peers as they commit to host each other's services for two reasons. As peers commit to each other, more pathways are created to get to a single service. If these pathways become lost, unknown, or unusable, it is effectively useless. Thus the first job of the central authority is to 'remember' these pathways so it can teach others how to access services. Another reason to do this is for accountability. If a peer promises to host a service, and for some reason no longer keeps his promise, that peer should be punished, though he won't be for the time being.

Lastly, the central authority needs to be able to tell clients how to get to their desired host. If their request is for something that isn't within its database, the request is then relinquished to DNS. However, if it can find a match (hostname and port pair), then it will return a list of known serving peers and the port to which each peer has designated to run the requested service.

3.6 Clients

Clients are the users of the system who wish to access the services hosted by the peers. Please note that there is a distinction between clients and peers. Peers are the workhorses in the system that host the services while the clients only find the peers via the central authority. When receiving a list from the central authority, the client then caches that information for a fixed period, and each subsequent request will be done in a round-robin order until all nodes have been visited, keeping track of the latency and bandwidth of each server. From that information, the client should be able to choose the best node with the best latency/bandwidth pair. The user will then use the 'closest' node and thus reduce the overall traffic in the global network.

3.7 Self-policing

Since the data being served by peer is generally dynamic in nature, it is impossible for clients to validate the information coming from the peer by either checksums or by signatures. The motivation for an adversary to become a peer within the system and change the contents of the virtual machine when hosting someone else's service is rather compelling, and can be done easily without regular clients being able to differentiate between a tampered service and an untampered service. Thus we assign the responsibility of checking for compliance to the peer that 'owns' the virtual service, as he is the only one who can manually validate the dynamic contents, and has the most vested interest in ensuring that the peers who are serving his data comply with the rules of the system.

Peers may receive a list from the central authority detailing which peers are serving their virtual machine. If the virtual machine's owner finds that a peer is misbehaving, he may report this information to the central server. The central server in turn removes the peer from the authorized list of peers serving that particular virtual machine. The reason we don't remove the peer entirely from the system is because that would give incentive for adversaries to falsely complain to the central authority. And since we only ban that particular virtual machine from the misbehaving peer, there is no incentive for the peer to lie because if he did so, he would only be hurting himself.

4. Attacks on our System

In this section, we will describe different attacks that may be carried out against our system. For each attack, we will describe how it can be performed as well as the effect that it may have on our system.

Physical Destruction of Peers: The physical destruction of a single peer is relatively meaningless to the entire system. Since the design of the peer-to-peer system is for high-availability, it would make sense that if a peer, or even a large portion of peers, becomes unaccessible or destroyed, the system will continue to work. Thus, the system should also be safe to almost any act of God: earthquakes, fires, small meteor strikes.

Denial of Service Attacks Against Peers: Much like the physical destruction of peers, a denial of service will do little to affect the high-availability characteristic of the sys-

tem. Needless to say, there is no way for the victimized peer to receive contact information from the central authority, but his virtual machine will still be accessible by other peers that are serving it.

DOS against the Central Authority: A denial of service against the central server will not allow any authoritative changes to be made, nor will it allow any new clients/peers to get info about the current state of the system. While peers that have knowledge about the system before the attack can still act with cached information, new peers/clients will not be able to gather any information as to how to find their requested resource. This is a very serious problem and an aspect where a distributed central authority presents a solution.

Faulty Trusted Kernel: While we endorse the use of a kernel that is widely accepted as safe, there is always the possibility that there is a bug in that kernel that could allow crackers to take advantage of machines. For example there were the past mmap bugs as well as the close call where malicious code was successfully inserted into the official cvs repository of the Linux kernel. While revoking kernels is an option, it is a very costly one, though ultimately, possibly the only feasible one. Another possible solution is to install some sort of intrusion detection system and incorporate that into the base snapshot.

Highly-available Spam Haven: Since there are no restrictions on what can be sent on a virtual machine, spammers can create their own spam relay to propagate and distribute their solicitations through our system, making their service highly available. This is an obvious annoyance to everyone and unfortunately, there is no good solution. Of course, the possible maluse of our system does not pertain only to spam, but anything that one wishes to anonymously serve, such as warez distribution sites, etc.

5. Weaknesses in the Architecture

This section will discuss where weaknesses lie in our system.

The aim in our system is to provide high availability. Our system is designed to be able to continue to provide service when hosts go down, as well as when the network experiences difficulty. However, if the central authority ever becomes unavailable, our system fails to meet its designed purpose. We are currently working to see if it is possible to distribute the central authority without compromising security, but that remains outside the scope of this paper. Meanwhile, traditional high-available solutions for the central authority should suffice in guaranteeing that the system will not fail.

As virtual machines are under the complete discretion of the user who is running it, there are no checks or balances made against peers that accept foreign virtual machines. This allows the peer to browse through anything within the virtual machine at his liking. Information within a database could very well be sensitive, such as a listing of people's names, addresses, and credit card information. And since the dissemination of such sensitive information would not be acceptable, it would unfortunately make our solution unfeasible to those who need services that require privacy.

Bandwidth usage is also another vital point that hasn't been thoroughly addressed in this paper. The problem is that some services require or attract higher bandwidth usages than others. Naturally, peers will have different bandwidth allocations, and there is the notion of whether giving bandwidth-hungry services to those who can handle them is a fair practice or not. Rather, we could also make a rule to simply distribute the virtual machines randomly, and hope that the receiving peer can handle the bandwidth expected by the service. There are many different possible solutions to this problem and should be resolved in a future work.

Another problem is that there are many services that we expect to run that will require the virtual machine to change and thus affect the service it is originally providing (i.e., a local database commit). A change that is only apparent on a single peer that isn't distributed to all peers could very well throw the overall service into an incorrect state. Thus for services that depend on a global database that needs to be in sync, we delegate responsibility to more traditional solutions, such as a dedicated database server outside of our system. This unfortunately reintroduces a single point-of-failure.

Since our system takes absolutely no penalizing action against misbehaving peers, a peer may continue to disrupt the system without any consequences. While this is undesirable, if we do enact a retribution system, it will open the doors to other types of attacks that are more vicious and harder to control. So in order to

prevent other problems from occurring, we allow this one to be incorporated unchecked by the central system.

Throughout this paper, we assume that the central authority is an unbiased and trustable entity. If this assumption ever fails, the system is not guaranteed to work. If the central authority wishes to silence a particular peer, it may do so by not informing anyone else about the peer's listings, effectively making that peer unknown to the world.

6. Conclusion

Federated computing has been established in trusted computing infrastructures, but rarely seen in an untrusted infrastructure. In this paper, we have looked into the possibility of achieving a federated computing cluster via the use of a peer-to-peer backbone. While there are many challenges that still need to be resolved, we hope that we have whet the appetite of the reader as to the different possible benefits that such a system could bring.

References

1) of California: Berkeley, U.: The search for extraterrestrial intelligence (1999) `http://setiathome.ssl.berkeley.edu/`.
2) napster.com: napster.com (2003) `http://www.napster.com/`.
3) kazaa.com: kazaa.com (2003) `http://www.kazaa.com/us/index.htm`.
4) Barroso, L.A., Dean, J., Holzle, U.: Web search for a planet: The google cluster architecture. (2003)
5) Team, H.A.L.P.: High-availability linux project (2003) `http://linux-ha.org/`.
6) Dingledine, R., Freedman, M.J., Molnar, D.: The free haven project: Distributed anonymous storage service. (2000)
7) Anderson, R.J.: The eternity service. (1996)
8) Potter, R.: Scrapbook for user-mode linux (2003) `http://sbuml.sourceforge.net/`.