Ubiquitous, Flexible and Distributed Computing

Jun Zhang and Chris Phillips
Electronic Engineering Department, Queen Mary, University of London
327 Mile End Road, London, UK. E1 4NS
{jun.zhang, chris.phillips}@elec.qmul.ac.uk

***Abstract*-Motivated by the observation that there exists a lot of cheap and networked computing power and the inflexibility of Transmission Control Protocol (TCP) in the context of mobility, this research proposes a new system enabling applications to roam seamlessly while they are actively processing executable code and/or communicating with remote entities. By using this architecture flaws/limitations of existing networking protocols can be mitigated and it becomes possible to exploit this spare computer power. By solving those problems, a number of benefits can be achieved in the fields of application migration and distributed computing.**

I. INTRODUCTION

The Internet protocol suite provides a means by which many applications can communicate via a shared heterogeneous communication infrastructure. In the Internet protocol suite, TCP [1] is one of the main protocols at the Transport layer, which was designed to provide a reliable end-to-end byte stream in any kind of network. In the Internet, over 90% [2] of all transport layer data employs this protocol. Using TCP, two application entities, usually located on separate computers, can establish a connection and exchange data with each other.

The connection endpoints at the server and the client, known as sockets, are identified by TCP port numbers and Internet Protocol (IP) [3] addresses. These are fixed after the connection is established. This manner of working brings a lot of benefits. For example, the destination address is implicit with functions such as read() and write(), but it also brings some limitations. One of the most important limitations is service availability and performance under adverse conditions such as network congestion or denial of service attacks.

At the same time, users are usually concerned with the quality of service they receive rather than the exact identity of the server and a lot of cheap and networked computing power exists in the Internet. Therefore, server replication has been raised as a solution. In this approach, several servers cooperate together to provide the same service to the users and users are allowed to use any one of these servers. Server clusters have already been proposed [4][5]. Server clusters consist of heterogeneous computers that perform like a single system. However, simple replication of a server cannot solve the problem of providing continued service when the underlying resources may be transient in nature.

Later, some proposed solutions to provide connection migration [6][7][8] to address the problem of continued service are identified. Nevertheless, these solutions still have some problems; they cannot provide connection migration between different platforms and they cannot provide proactive and intelligent connection migration.

In this paper, we propose a generic and intelligent migration scheme for operation across multiple platforms. This paper addresses these problems by using a new architecture, which can enable services, applications or resources to roam seamlessly whilst they are active and communicating with remote entities.

II. RELATED WORK

Prior to our research, some relevant work has already been done. In this section, we examine briefly what they tried to achieve, and how they achieved it. In addition, the benefits and limitations of their approach(es), and points of novelty not addressed in research to-date will also be discussed.

The first relevant research is the work of Disco lab [9][10][11]. They focus on providing a framework for migrating servers whilst keeping alive the TCP connection. They propose a model called “Cooperative Service Model”. In this model, they consider a “pool”, which comprises some distributed and similar servers. These servers cooperate to sustain a service by migrating client connections within the pool. In order to cooperate successfully, they designed a new transport layer protocol called “Migratory TCP (M-TCP)” to support efficient migration of live connections. This work provides more reliable services and improved resilience to network congestion as they can dynamically migrate connections, though the migration is restricted to within the server pool and no server load-balancing scheme has been presented.

The scheme put forward by the Migrate Internet Mobility Project (MIT) has a similar focus to Disco lab. However, they achieve the goal through another approach. In their scheme, the migration is requested through a TCP option field. A list of available servers and a certificate for each server is passed from the server to the client when the connection is first established. Later, a migration request can be sent from the client to one of the available servers for connection migration. This work provides a unified framework to support Internet mobility, but load balancing cannot be achieved as the decision of connecting to which server is made by client without a knowledge of the network conditions.

In another relevant study, Zap[12][13], researchers have developed a novel system for transparent migration of legacy and networked applications. They designed a *pod* abstraction, which provides a collection of processes with a host-independent virtualised view of the operating system. This decouples processes located within the pods from dependencies on the host operating system and other processes. By integrating Zap virtualisation with a checkpoint-restart mechanism, Zap can migrate a pod of processes as a unit among machines running independent operating systems without leaving behind any residual state after the migration. Though transparent application migration without modification to the operating system kernel or application can be achieved, no control scheme has been presented so that the migration still lacks intelligence.

In vOS [14][15][16], they designed a system that consists of a group of computers. This system decouples the application process from its physical environment by using “virtual” technology, which means that the application uses virtual files, virtual network connections and other virtual resources. This “virtualisation” is achieved by using API interception, which means intercepting calls made from the application to the underlying runtime system and reinterpreting the call. Similar to the work of Zap, transparent application migration without modification to the operating system kernel or application can also be achieved, but the migration is restricted within a group of machines and there is a lack of an intelligent control plane.

III. PROPOSED ARCHITECTURE

In our research, a new system architecture is proposed as shown in Figure 1.

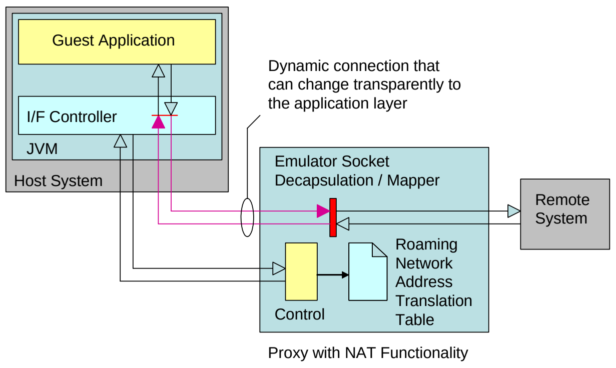
Guest Application
I/F Controller
JVM
Host System
Dynamic connection that can change transparently to the application layer
Emulator Socket Decapsulation / Mapper
Remote System
Roaming Network Address Translation Table
Control
Proxy with NAT Functionality

Fig. 1 Proposed System Architecture

This architecture consists of three main entities: the Host System, the Proxy and the Remote System. In this figure, a client application and its associated operating system are running on a transient Host System and this client application communicates with a remote endpoint through an Interface Controller and the Proxy. Descriptions of components and their functions are given below.

A. *Host System*

The Host System is composed of three components: the Guest Application, the Interface Controller and the Java Virtual Machine that encapsulates them.

The Guest Application is a Java based application running within the JVM and its activities are monitored and controlled by the Interface Controller.

The Interface Controller is Java software also running within the JVM. It is a bridge between the Guest Application and the underlying Host System and between the Host System and the Proxy. Therefore, it plays an important role having a number of key functions. Firstly, it intercepts all the system calls sent from the Guest Application and redirects the system calls according to requirements. For example, if the Guest Application would like to access the hard disk located on a remote computer, the Interface Controller will redirect the system calls to the appropriate remote computer. Secondly, it monitors the execution performance of the JVM on the Host System. This information will be sent to the Proxy to determine if/when to migrate the Guest Application, details of which will be discussed later. Thirdly, it is responsible for setting up the control connection between itself and the Proxy. This connection is used for activities such as host discovery, monitoring and migration coordination.. The Interface Controller also tunnels the data packets sent from the Guest Application. The destination IP address in the header of the data packets sent from the Guest Application is the IP address of the Remote System. But as the packets should get the destination via the Proxy, the data packets are encapsulated and sent to the Proxy first. Finally it performs check-pointing and freezing the execution of the Guest Application, including the information held in any dynamic memory such as the state of variables. When the Proxy decides to migrate the Guest Application to a new Host System, the image of the Guest Application is frozen and transferred to the new host. Besides the Guest Application, related data stored in the memory should be also transferred to the new host as the migration may be carried out when the application is active.

The last component in the Host System is the JVM. This is the basic environment that the Guest Application and the Interface Controller are running on. In fact, this basic environment does not have to be JVM though the Java programming language can be characterized as being a simple, object oriented, distributed, multithreaded, architecture with good portability and security features via the JVM[17][18].

B. *Proxy*

The Proxy is another important entity in the proposed system architecture. It is a bridge between the Host System and outside world. If the Host System uses TCP to communicate with the remote entities, the communication pathway between them actually comprises multiple connections to support TCP spoofing at the Proxy. Besides data connection(s), there is a control connection between the Host System and the Proxy.

The Proxy has two elements: a Roaming Network Address Translation Table and a Control Plane. The Proxy has two main functions with these two elements: Firstly, it provides connection management services, including Network Address Translation (NAT), to ensure that the communication path can be set up and maintained between the Guest Application and a remote entity. Secondly, it provides a higher-level control and coordination function. This function includes monitoring the

Guest Application performance, configuration management, and also determining when and where to move the Guest Application to. These two main functions can be divided into several sub-functions:

As shown in figure 1, the communication pathway between the Guest Application and the remote entities is composed of multiple connections. The first connection is between the Host System and the Proxy. The Proxy is a point of presence for the Guest Application and sets up another connection with the Remote System, assuming the Guest Application is acting as a client. In order to ensure that retransmission timeout occurrences are limited and the TCP connection(s) are not dropped during migrating the image of the Guest Application, TCP spoofing is used to splice separate connections from the Guest Application in the Host System to the Proxy and from the Proxy to the Remote System. The remote entities do not connect to the Guest Application directly as the remote entities do not know the IP address of the Host System which the Guest Application is running on.

If the Guest Application is a server and a remote entity would like to connect to the Guest Application, the connection request will firstly sent to the Proxy. Then the server will redirect this request to the Guest Application as if it is the remote entity. Therefore, the IP address of the Proxy must be public so that the remote entities can know this IP address via manual configuration or lookup via a DNS server.

The Proxy maintains the Roaming Network Address Translation Table that lists the available Host Systems (Roaming Network Address Translation Table in the proposed system architecture) and information related to these Host Systems. The information should include: the IP address and the Interface Controller port number(s) of the Host System, transmission time delay between the Host System and the Proxy and system information of these hosts (Such as CPU speed, CPU usage, total physical memory, available physical memory, network link speed and network utilization).

When a Host System machine is ready to accept the Guest Application, it sends a message to tell the Proxy that it is available and ready to provide service. After the Proxy receives this message, it updates the Roaming Network Address Translation Table so that this machine is listed in the table. Later, the Proxy will send a message to the Host System if the Proxy decides to migrate the guest system to this Host System.

The Proxy updates the Roaming Network Address Translation Table regularly in order to make sure information of the available Host Systems is up to date. This communication is accomplished by a message sent by each hosts at regular intervals. When the Proxy receives latest information from the hosts, it updates the Roaming Network Address Translation Table. If the Proxy cannot get the updated information from a certain server for a period of time, the Proxy will delete this host from the Table.

For a certain host, it may not be available any longer. To inform the Proxy that it will not be available anymore, it sends a message. When the Proxy receives this message, the host will be purged from the Table.

The Proxy monitors the performance of Host Systems listed in the table. That is for deciding whether to migrate the guest system and where to migrate. This monitoring is based on the information sent by each Host System and a performance threshold. If the performance of the Host System falls below this performance threshold for a certain period of time, the Proxy will consider migrating the guest system to a better host.

The Proxy decides the Guest Application migration. Whether the Guest Application needs migration is based on the performance of the Host System and where to migrate the application is based on the information in the table and artificial intelligence technologies.

C. Remote System

The Remote System can either be a traditional communication endpoint or have the same structure as the Host System, containing an Interface Controller. Besides, it can be a server or a client. If the Remote System is a client, it always considers the Proxy as the server and just knows the IP address of the server. The connection request to the server will be always sent to the Proxy and then redirected to a suitable server by the Proxy.

IV. SYSTEM OPERATIONS SPECIFICATION

After an overview of the whole system architecture and a description of each system component, some system operations will be discussed in this section.

A. Data Connection Establishment and Maintenance

The communication between the Guest Application and the remote entities can be divided into two main types, UDP and TCP. In the case of UDP, no connection is established between the Guest Application and the remote entity. What the Guest Application and the Remote System should do is just to send messages to each other. If the message is sent from the Guest Application, the message will firstly intercepted by the Interface Controller. Then the Interface Controller redirects the message to the Proxy. Later, the Proxy redirects the message again to the Remote System. The message sent from the Remote System goes the reverse way. In the case of TCP, the process is more complex.

For TCP connection initiated by the Guest Application, the general process is as follows:

1. The Guest Application tries to connect to the Remote System by sending a connection request packet.
2. The Interface Controller intercepts the connection request packet sent from the Guest Application.
   The Interface Controller places the packets within the payload of a further IP datagram whose destination port conforms to the security constraints imposed on the JVM and the destination IP address being the public address of the Proxy.
3. The Proxy receives these data packets and decapsulates them to liberate the inner IP data packets.

4. The Proxy redirects the data packets to the Remote System.
5. If the remote entity accepts this connection request, it will reply to the Proxy and sets up a connection with the Proxy.
6. Once the connection between the Proxy and the remote entity is established, the Proxy replies to the Interface Controller and set up another connection with the Interface Controller.
7. Once the connection between the Interface Controller and the Proxy is established, the Interface Controller set up another connection with the Guest Application.

Then a communication pathway between the Guest Application and the remote entity is completed.

B. Resources Discovery

In order to utilise transient Host Systems' resources efficiently, the system architecture should have a mechanism to discover available resources in the Internet. In our design, the available resources should actively inform the Proxy that it is available. The general process of resources discovery is as follows:

1. After the resource (e.g. a computer) starts-up, a JVM instance containing an Interface Controller application, which located on this resource, runs automatically.
2. It sends connection request to the Proxy to set up a control connection with the Proxy.
3. The Proxy accepts this connection request and replies to the Interface Controller. As a result, a control connection between the Interface Controller and the Proxy is set up.
4. After the control connection is set up, the Proxy adds a new record to the Roaming Network Address Translation Table to record the information of this new Host System.
5. The Interface Controller gathers the Host System's information and sends it to the Proxy. This information includes: CPU speed, CPU usage, total physical memory, available physical memory and network link speed.
6. The Proxy receives this information and updates the record in the table.
7. The Interface Controller sends updated information to the Proxy at regular intervals.

If the transient Host System is no longer available, the record should be deleted from the table. This can be achieved in two ways. The first is that the Interface Controller sends a message to the Proxy to inform that it will not be available anymore. After receiving this message, the Proxy deletes the record of that Host System from the table. The second way is that the Proxy has not received any monitoring message from the Interface Controller after a certain period of time, and then the Proxy considers that the Host System is not available anymore so it deletes the record of that Host System from the table.

C. Performance Monitoring and Migration Determination

One goal of this system architecture is to provide migration capabilities to the running Guest Applications. This migration is firstly determined by the Proxy based on the performance of the Host System that the Guest Application is running on. As discussion in previous sections, the Proxy gets this information from the Interface Controller on the Host System. For a certain period of time, the Interface Controller on the Host System gathers the updated information of the system and sends to the Proxy. After receiving the updated information, the Proxy updates the Roaming Network Address Translation Table.

After updating the information, the Proxy checks the performance of the Host System. The performance can be assessed by a calculation model. In this model, the utility of some important system components will be calculated. If the average utility of these system components is below a certain level, the Proxy will consider of migrating the application.

Besides the performance of the Host System, other factors need to be taken into account when deciding whether to migrate or not. The first factor is whether there is any other resource available and second factor is that whether those resource(s) are better than those currently offered. The general process of migration determination is as follows:

1. The Proxy checks the Roaming Network Address Translation Table. If there is any other resource available, the Proxy begins to compare the old resource (the Host System which the Guest Application is running on) and the new resource(s).
2. The comparison is also based on a calculation model, like benchmark test of computer hardware. If the performance of the candidate Host System is better than the current Host System, the Proxy will consider migration the Guest Application. In addition to this calculation result, there are still some issues that should be addressed before making migration decision. These issues include risk quantification and risk hedging. Risk arises when the Proxy makes a migration decision. For example, a Host System may be very powerful but it may not be available for a long time. If the Guest Application is running a task which should be completed within a limited time and the Proxy considers of migrating it to the powerful Host System, the Proxy should also consider the risk whether the powerful Host System can complete the task before it is not available.
3. The consideration can be based on "machine learning". By using knowledge like statistics, belief networks and decision trees, the Proxy quantifies the risk of migration. If the Proxy still decides to migrate the task to a new Host System, the Proxy can hedge the risk by preparing a backup plan if there is any other Host System available.

D. Process of Migration

After the Proxy makes the migration decision, the process of Guest Application migration needs to be carried out. The general process of migration is as follows:

1. The Proxy sends a migration request message to the Interface Controller on the current Host System through the control connection.
2. After the Interface Controller receives this message, it sends an ACK message back to the Proxy.
3. The Interface Controller of the current Host System redirects a migration request message to the Interface Controller of the candidate Host System.
4. After the Interface Controller on the candidate Host System receives this message, it sends an ACK message back to the Interface Controller of the current Host System.
5. After the Interface Controller on the current Host System receives this ACK message, it sends a connection request to the candidate Host System to set up a TCP connection.
6. After the connection is set up, the Interface Controller freezes the Guest Application.
7. The Interface Controller transfers the image of the Guest Application and information held in the memory to the new Host System. During the period of freezing and transferring, if the Guest Application has connection with the remote entity and the remote entity has sent data packets to the Guest Application, the Interface Controller will store these packets in a buffer and redirect them to the new Host System.
8. After transferring the image and information held in the memory of the old Host System, the Guest Application resumes its execution. The Interface Controller on the new Host System sends a inform message to the Proxy to tell the Proxy that the migration completed.

V. EXAMPLE SCENARIOS

Some use-case scenarios have been created in order to assess the performance of the proposed architecture under different circumstances. One scenario is roaming guest application. The processes of resource discovery, monitoring the performance of host system, migration determination and guest system migration have been discussed in the previous section. A scenario which extends this basic example is considered next.

Consider a “virtual” computer application that comprises a number of distributed components, such as a storage unit and a CPU engine. Each of these Guest Application elements is housed in a different host together with the Interface Controller. The CPU engine communicates with the component resources via the Proxy. However, when the CPU engine roams to a new location the pathways to the resources may become rather long, leading to unacceptable latencies. Under these circumstances, the Proxy locates and negotiates with another Proxy that is closer to the terminal’s new location. Assuming the new Proxy is willing, the terminal “reconnected” to the new, closer Proxy. Some time later the component resources are moved one-by-one to hosts that are closer to the new Proxy and the latencies are reduced, accordingly. This approach also allows different transient resources in the Internet to be used flexibly based on their instantaneous availability.

A further scenario is the use of proactive and reactive protection schemes against distributed “denial of service” attacks using a roaming server mechanism. Proactive protection will either require the server application to roam at approximately regular intervals, based on guest system availability and so forth. Alternatively, security measures located within the Interface Controller and/or the Proxy, such as intrusion detection or profiling could indicate impending trouble and so trigger the server to move.

Yet another example scenario showing the flexibility of the architecture is the case of transparent parallelism. In this scenario an application provides service to remote clients carrying out some computing task. All client applications connect to one server. This may lead to overload of the first server. Later, based on increasing client demand the Proxy decides to lighten the load of the first server and provide better service to the clients. It decides to clone the server application onto other server(s) and some client applications can then be redirected to the new server(s). Therefore, the load of each server can be maintained at an acceptable level and the service to the clients can be more reliable.

VI. CONCLUSIONS

In this paper a novel system architecture is proposed for enabling applications to roam seamlessly while they are actively processing executable code and/or communicating with remote entities. The important components are introduced, namely the Interface Controller within the JVM and the Proxy. The architecture enables the decoupling of running applications from the underlying environment, facilitating both robustness and opportunistic processing on transitory resources, as well as permitting the ability to create multiple clones of an application for performance gains. The architecture can also be used to allow complex services to be delivered to a terminal with limited local processing capability even whilst it is “on the move” using the intelligent control plane to manage the migration mechanism as needed.

REFERENCES

[1] RFC 793, Transmission Control Protocol, http://www.ietf.org/rfc/rfc0793.txt

[2] ntop, http://status2.ira.cnr.it:3000/ipProtoDistrib.html

[3] RFC 791, Internet Protocol, http://www.ietf.org/rfc/rfc0791.txt

[4] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer, Paul Gauthier, Cluster-Based Scalable Network Services. In Proceedings of SOSP ’97, October 1997. http://www.cs.berkeley.edu/~brewer/cs262b/TACC.pdf#search=%22cluster-based%20scalable%20network%20services%22

[5] Trevor Schroeder, Steve Goddard, and Byrav Ramamurthy, Scalable Web Server Clustering Technologies, Network, IEEE, Volume 14, Issue 3, May-June 2000 Page(s):38-45

[6] Alex C. Snoeren, David G. Andersen, and Hari Balakrishnan, Fine-Grained Failover Using Connection Migration. In Proceedings of 3rd USENIX Symp. On Internet Technologies and Systems (USITS), Mar. 2001.

[7] Chu-Sing Yang and Mon-Yen Luo, Realizing Fault Resilience in Web Server Cluster. SuperComputing ACM/IEEE 2000 Conference.

[8] Florin Sultan, Kiran Srinivasan, Deepa Lyer, Liviu Lftode, Migratory TCP: Highly Available Internet Services Using Connection Migration, Rutger University Technical Report DCS-TR-462, December 2001.

[9] Florin Sultan, Kiran Srinivasan, Deepa Iyer, and Liviu Iftode, “MigratoryTCP: Connection Migration for Service Continuity in the Internet”, http://discolab.rutgers.edu/mtcp/icdcs02.ps

[10] Florin Sultan, Kiran Srinivasan, Deepa Iyer, and Liviu Iftode, “Migratory TCP: Highly Available Internet Services Using Connection Migration”, http://discolab.rutgers.edu/mtcp/dcs-tr-462.ps

[11] Florin Sultan, Kiran Srinivasan, Deepa Iyer, and Liviu Iftode, “Transport Layer Support for Highly-Available Network Services”, http://discolab.rutgers.edu/mtcp/hotos01.ps

[12] Steven Osman, Dinesh Subhraveti, Gong Su, and Jason Nieh, “The Design and Implementation of Zap: A System for Migrating Computing Environment”,
http://www.ncl.cs.columbia.edu/publications/osdi2002_**zap**.pdf

[13] Shaya Potter, Jason Nieh, Dinesh Subhraveti, “Secure Isolation and Migration of Untrusted Legacy Application”, http://www.ncl.cs.columbia.edu/publications/cucs-005-04.pdf

[14] Tom Boyd and Partha Dasgupta, “Process Migration: A Generalized Approach Using a Virtualizing Operating System”, Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on 2-5 July 2002 Page(s):385 – 392.

[15] Ravikanth Nasika and Partha Dasgupta, “Transparent Migration of Distributed Communicating Processes”, http://cactus.eas.asu.edu/Partha/Papers-PDF/1900-2001/PDCS-ISCA2000.pdf

[16] Tom Boyd, Partha Dasgupta, “Injecting Distributed Capabilities into Legacy Applications Through Cloning and Virtualization”, http://www.dvo.ru/bbc/pdpta/vol3/p251.pdf

[17] About the Java Technology, http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html

[18] James Gosling, Henry McGilton, A White Paper: The Java Language Environment, http://java.sun.com/docs/white/langenv/index.html