Information Technology Journal 3 (2): 188-191, 2004
ISSN 1682-6027

Implementing Strong Code Mobility

Muhammad Kamran Naseem, Sohail Iqbal and Khalid Rashid
Department of Computer Science, International Islamic University, Islamabad, Pakistan

Abstract: This study presents the idea of implementing strong code mobility in terms of platform independence, microprocessor architecture reliance and resource management. Proposed system establishes a shared connection with the resources and its surrounding environment based on distributed structured XML-based knowledge. The resources managed by the process are shared between the nodes, so that the developer can program in a centralized setting. The goal is to present a solution for strong code mobility for commonly used platforms.

Key words: Strong code mobility, code mobility, code mobility implementation

INTRODUCTION

Code mobility is not a new concept. The research work on distributed operating systems has followed a more structured approach. In this research area, main problem is to support the migration of active processes and objects, along with their codes and data, at the operating system level over a network. In particular, process migration concerns the transfer of a process being run by an operating system, from machine where it is running to a different one[1]. Though Java Applets have gained great popularity under the definition of code mobility but they also provide weak code mobility, where only code is transferred, which yet has to start execution. There are no such common technologies which provide strong code mobility. In this study an implementation of strong code mobility which is platform independent but microprocessor architecture specific is presented. A technique for user level process migration between computers, for collecting the memory contents of a process on one computer in an information stream and for restoring the data content from the information stream to the memory space of a new process on a different computer are discussed. Unlike Java feature of object serialization[2], the data fetching and re-establishment method enables complicated data structures such as indirect memory references that is pointers, to be migrated properly between two computer nodes. Study is based on Intel 386 and onward microprocessor architecture.

Intel memory addressing technique: The microprocessor has a set of rules that apply to segments whenever memory is addressed. The rules, which apply in the real and protected mode, define the segment register and offset register combination[3]. For example, the code segment register is always used with the instruction pointer to address the next instruction in the program. This combination is CS: IP or CS: EIP, depending upon the microprocessor's mode of operation. The code segment register defines the start of the code segment and the instruction pointer locates the next instruction within the code segment.

Segment and offset addressing scheme allows relocation: The complicated scheme of segment plus offset which allows programs to be relocated in the memory system was used[3]. Even it also allows programs written to function in the real mode to operate in a protected mode system. A relocate-able program and data are program and data that can be placed in any area of memory and used without any change in their contents. The segment and offset addressing scheme allows both programs and data to be relocated without changing the instructions or contents of program or data, respectively. This is ideal for use in a general-purpose computer system in which not all machines contain the same memory areas. Because memory is addressed within a segment by an offset address, the memory segment can be moved to any place in the memory system without changing the offset addresses[3]. This is accomplished by moving the entire program, as a block, to a new area and then changing the contents of the segment registers.

Process migration occurs when a process is transferred between two machines which differ in software environments such as compiler, operating system or software tools. Presented study defines that, if memory

Corresponding Author: Muhammad Kamran Naseem, Department of Computer Science, International Islamic University, Islamabad, Pakistan E-mail: kamran_naseem@yahoo.com

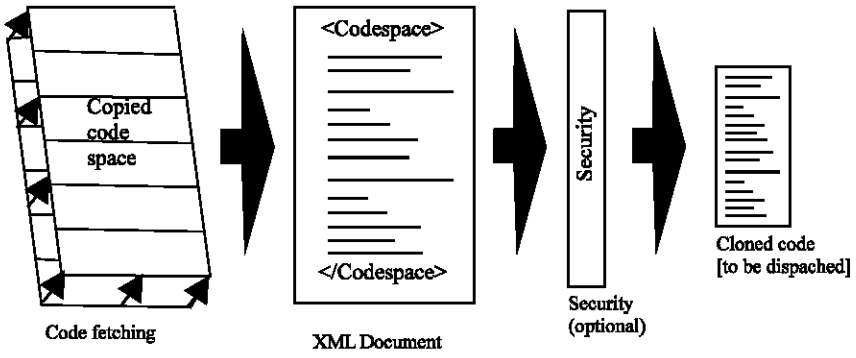

Fig. 1: Mechanism from code fetching to code dispatching

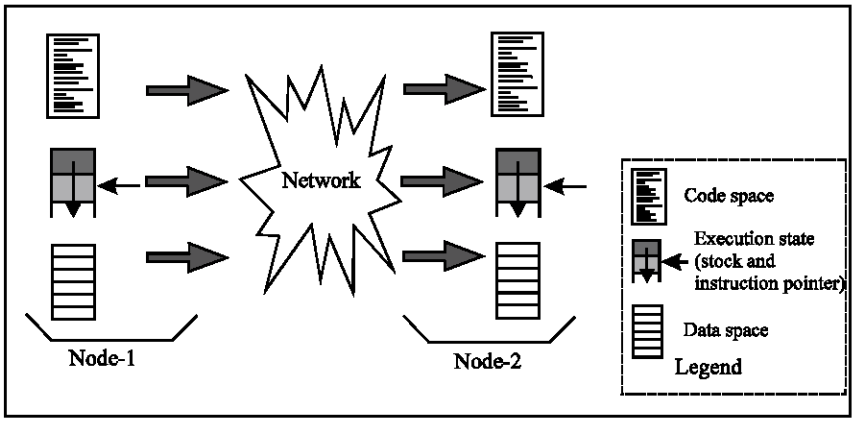

Fig. 2: Abstract model of code migration

contents can be cloned or relocated on the same system then they can also be cloned on another system's memory over network. This network can be a set-up from p2p to internet. In this way the programs picked from real mode can also be instantiated in protected mode.

Fetching data: In the approach to process migration, there is a need for efficient methods to identify, assemble and restore data contents of a process. From now inwards we refer data, code and state as process. For migrating a process, all data necessary for future execution of the process was collected (Fig. 1) and then restored in the segments of the new process on another machine. We copy the binary contents of process from the system memory into an information stream and store them into a buffer. Since there are two basic types of data objects that can be contained in the memory space of a process: the storage object and the memory reference object[4].

Due to different operating system memory management and loading operations, a memory address used in a process on one computer may be worthless to a process on a different computer. When the executable file is loaded into the computer's main memory, its contents are placed at particular memory locations, depending on the memory management scheme used by the operating system of that node. Therefore, while a memory address in one process refers to particular data, most probably the same memory address might be undefined or may refer to any unrelated data when used by a process on another machine. For this purpose a Value-to-Pointer table was maintained to store the data addressed by any pointer. These pointers can be identified while scanning the memory for shipment of data. Each pointer entry in the table was assigned with pointer's value and offset, which makes easy to assign data values to pointers on the destination machine.

Migrating the fetched data: The fetched data, code, execution state along with required register values was transferred to the destination machine (Fig. 2). We use to dispatch the copied data (Code and data etc. from the computer memory) in extensible Markup Language (XML) tags from one machine to the other (Fig. 1).

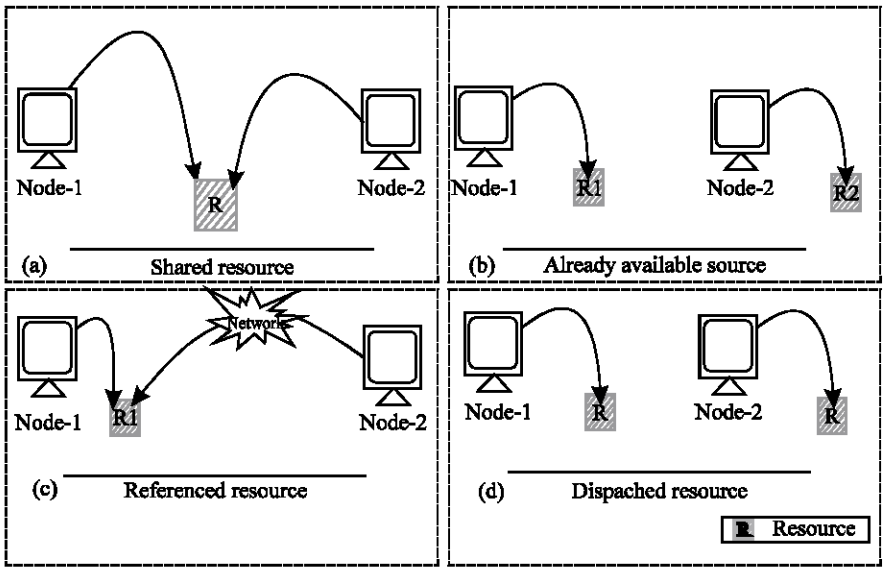

Fig. 3: Resource management

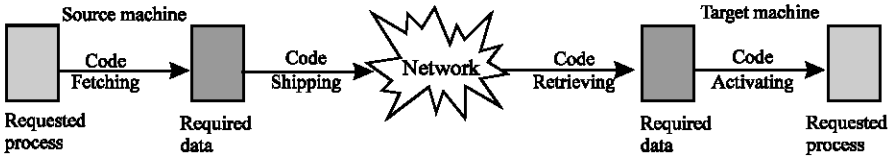

Fig. 4: Four steps of process migration

The reason to use XML for shipment and maintenance is that it provide several potential advantages:

1. A general, open and self-describing data format
1. 2. An open and interoperable environment for distributed applications which rely on the concept of code mobility. This opens up the possibility to separate the maintenance from the design and implementation aspects of code mobility.
3. Hierarchical and complex relationship between or within dispatched data segments.
4. Creating structured documents in flexible ways.
5. Designed for use on the internet and for exchanging data.

All the above mentioned points are directly related to the requirements and it was observed that XML was most suitable for transferring data. As XML based data transfer was not secure over a network, we apply optional security measures before transferring the process (Fig. 1).

Protocol encapsulation: Though TCP/IP for communication was used but we did not bother about the protocol to be used while transferring the data.

The technique or protocol to be adopted for dispatching or receiving the information stream was not concerned but may be decided by the programmer on the basis of operating system and software environment to be used. The process can be transferred via direct network-to-network communication (network migration) or by any other means of communication.

Code retrieval and reactivation: To restore the contents to the memory space on a different machine, the restoration mechanism must be able to extract the collected data from the received information stream and reconstruct the data structure into the memory space. When we receive information stream of the migrating process on destination machine, we restore the data from the information stream and reactivate it until the end of its execution. The reverse of (Fig. 1) was practiced on destination machine.

Resource management: When a process was migrated from one machine to another there might be a case that the migrated process was using a resource at the previous node. We believe either that resource is already available at new node (Fig. 3b) or resource on previous node is

shared (Fig. 3a). If the resource is not available at new node then that resource (Fig. 3d) or a reference to that resource (Fig. 3c) is dispatched along with the code, if possible. Here by resource we mean device (s) or file (s). The process rebinds with the resource already available at new node or previous node, if possible.

Steps of strong code mobility implementation: Each operation involving code mobility is divided into following steps:

(I) Determine the requested code in the system memory
(ii) Fetch the required code along with state into a stream or buffer
(iii) Transfer code
(iv) Integrate code into the target system: I-e
(a) Activate the instance of the code
(b) Connect it to the existing data or code or resource
© Continue its transfer over the network to yet another node if required.

These steps of implementing strong code mobility have been elaborated in Fig. 4.

DISCUSSION

Applications of process migration include

Load distribution: migrating processes from overloaded machines to under loaded machines to exploit unused computing cycles.

Fault resilience: Migrating processes from those machines that may experience partial failure.

Resource sharing: Migrating processes to machines with special hardware or other unique resources such as databases or peripherals required for computations.

Data access locality: Migrating processes toward the source of data.

Mobile computing: Migrating processes from a host to a mobile computer.

A technique for collecting memory contents of a process on one computer into a platform-independent information stream and for restoring the data content from that information stream to the memory space of a new process on a different computer was presented. The mechanisms of data collection and restoration enable complicated data structures such as pointers to be migrated properly. This mechanism examines the current program state for migration of process and can be used in process migration, as well as in sequential and parallel distributed computing. These procedures may be used in any general solution of process migration over a network to carry out the following tasks automatically and effectively.

- Recognize the complex data structures like pointers of a migrating process for process migration
- Replicate the data into a machine-independent format
- Transmit the buffered information stream for a new process on the destination node
- Decode the transmitted information stream and retrieve the data in the memory space of the new process and reactivate it on the destination machine

REFERENCES

1. Fuggeta, A., G.P. Picco and G. Vigna, 1998. Understanding Code Mobility: Transactions on Software Engineering. IEEE., 24.
2. Riggs, R., J. Waldo, A. Wollrath and K. Bharat, 1996. Pickling State in the Java System, Computer Sys., 9: 313-329.
3. Barray, B.B., 2000 The Intel Microprocessors: 5th Edn., pp: 55-57.
4. Sun, Xian-He and K. Chanchio, 1998. Data collection and restoration for homogeneous or heterogeneous process migration. Software Practice and Experience, 32: 1-27.