Maintenance of Network Connectivity Across Process Migration

Mehdi Bahrbegi *m.bahribayli@gmail.com*
Department of Computer
Islamic Azad University, Shabestar Branch
Tabriz, East Azerbaijan, Iran

Hadi Bahrbegi *hadi.bahrbegi@gmail.com*
Department of Computer
Islamic Azad University, Shabestar Branch
Tabriz, East Azerbaijan, Iran

Amir Azimi Alasti Ahrabi *amir.azimi.alasti@gmail.com*
Department of Computer
Islamic Azad University, Shabestar Branch
Tabriz, East Azerbaijan, Iran

Elnaz Safarzadeh *elnaz_safarzadeh@yahoo.com*
Department of Computer
Islamic Azad University, Shabestar Branch
Tabriz, East Azerbaijan, Iran

Abstract

Most of processes running on a computer need network connections to communicate with other processes and access resources such as files, databases, and so on. Network addressing protocols are tightly coupled with physical addresses. When a process migrates, all of these connections are torn. While these connections are essential to running a process, the process may crash. And process migration becomes an inapplicable approach. We have developed a new method which maintains network connectivity across process migration. Our solution is based on API interception and removes the need for modification of operating system or applications. It also removes disadvantages of other approaches that are based on the API interception and offers a better performance.

Keywords: Network Connectivity, Process Migration, Twin.

1. INTRODUCTION

Problem of network connectivity in the systems that use process migration as solution to problems such as load sharing has existed from the early advent of process migration. Solutions typically consist of modifications to operating systems or applications. Because of complexity of adding transparent migration to systems originally designed to run stand-alone, since designing new systems with migration in mind from the beginning is not a realistic option anymore [1] at one hand and need to developing new applications because of impossibility of use of existing applications, has made process migration an unpopular approach.

Because of the increasing costs of operating system development and the lack of standard solutions for distributed systems and heterogeneity, middleware level solutions have be-come of more interest [3].

Our approach is to create a process called twin for each remote process. The twin is responsible for redirecting communications of the remote process. Every remote process has logically its own twin. But in practice there is not a one to one mapping between remote processes and twins. A

twin can be responsible for more than one remote process. This approach works with existing operating systems and applications.

2.1 Terminology

A process is a key concept in operating systems [5]. It consists of data, a stack, register contents, and the state specific to the underlying Operating System (OS), such as parameters related to process, memory, and file management. A process can have one or more threads of control. Threads, also called lightweight processes, consist of their own stack and register contents, but share a process's address space and some of the operating-system-specific state, such as signals. The task concept was introduced as a generalization of the process concept, whereby a process is decoupled into a task and a number of threads. A traditional process is represented by a task with one thread of control [1].

Process migration is the act of transferring a process between two machines (the source and the destination node) during its execution [1].

The node that process is instantiated in is called home node and the node that process is migrated to, is called a host node. A process that is running away from its home is referred as remote or foreign process.[1]

For every migrated process there exists a special process running at home which is called twin. It is possible that a twin be responsible for redirecting of communications of more than one remote process.

2. RELATED WORKS

Xiaodong Fu et al. [2] and Tom Boyd et al. [4] have introduced a method to maintain network connectivity for process migration and dynamic cluster environments without need to modify operating system or applications. This work introduces another approach that minimizes disadvantages of previous works.

3. NETWORK CONNECTIVITY

Nearly all of addressing protocols are tightly coupled with physical addresses. File addresses in file systems, NetBIOS names, and IP addresses to name a few. It means that a process is tied to a specific location and if it moves to another location, it will lose all of its connections to the resources which, is connected to. This is in contrariety with migration.

Previous approaches for maintaining network connectivity fall into two broad categories: modification to the OS network protocol stack, and introduction of a new API for accessing underlying resources [2] The first approach results in modifications of OS structures to support migration [6, 7, 8, 9], and second approach results in modification or rewriting the application and requiring the use of a new application programming interface (API)) [10, 11, 12, 13] whose implementation isolates the application from the consequences of migration . Moreover, most such solutions do not address the issue of adapting to changes in resource characteristics.[2] which are not desirable solutions.

Xiaodong Fu et al. [2] and Tom Boyd et al. [4] introduced a method for transparent network connectivity .They have developed a layer that operates transparently between the application and the underlying communication layer or operating system. This layer interfaces with the application and the operating system using API interception techniques [14, 15] to create a middleware to redirect communications transparently [2]. This middleware was called 'agent' and removed the need for modification of application or operating system. Disadvantage to this approach is that the middleware possibly becomes a bottleneck. It means that a single process (the agent) is unable to undertake redirecting communication for all remote processes. It is also prone to the famous problem of distributes systems, 'single point of failure' – if agent crashes all connections will be lost.

Twins use the same technology to intercept API calls, but because there exist theoretically a twin for each remote process the problems of 'bottleneck' and 'single point of failure' is removed.

4. TWINS

When a process asks for a network connection, it is bound to a specific address (for example an IP and Socket in TCP/IP protocol). After migration the packets are still sent to the old address and will be lost. A twin is a ready made lightweight process which is instantiated when a process asks for a network connection for first time. After the process is migrated packets which are sent to the old address are redirected to the new address by its twin. The main idea is taken from Mobile IP protocol. (For more detail about Mobile IP Protocol refer to [1, 16]).

4.1 Twin Implementation

Implementation of twins is very similar to the implementation of the agents in [2]. It is illustrated in Figure 1.

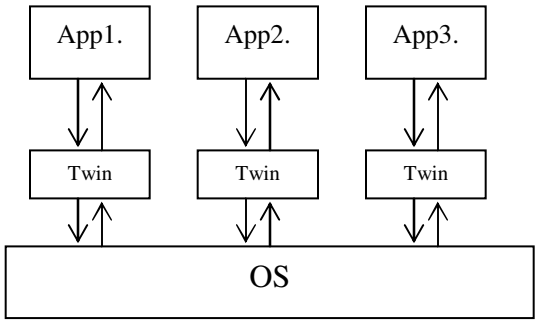

FIGURE 1: Structure of twins.

There is just one twin for each process but inverse is not necessarily true. After twin is created and connection is establish between twin and process every thing goes normally. Twin plays a dual role. It acts as the process at other end of connection for its twin and acts as its twin for the process at the other end.
Every twin listens to a specific port called control port. This port is used by migration manager.

4.2 Whole Story

Scenario of Creating a Network Connection

Process calls the API to create a network connection.
Migration Manager intercepts the API call and looks up in the Table of Twins. If there is not a twin for that process, it creates a twin for that process and inserts handle of process and control port of its twin in the Table of Twins. Then Migration Manger sends a CREATE_NEW_CONNECTION message to the twin's control port. Twin returns a network connection handle. Migration Manager hands this handle to the process.

Scenario of Migration

Migration Manager decides to send a process – which has one or more active network connections – to another node.
Migration Manager looks up in the Table of Twins for its twin then sends a MIGRATING message to its twin. While the twin has not received MIGRATED message it will buffer all incoming packets.
After migration ends Migration Manager moves corresponding record of the Table of Twins to the destination node then sends a MIGRATED message to the twin. One of parameters for MIGRATED message is new address for remote process.

5. FUTURE WORKS

Creating a separate twin for each process consumes a large amount of resources; so it is a good practice to design twins in a way that a twin can be responsible for more than one process. This idea can be combined with other techniques to gain even a better performance. For example twins can be pooled. Hence there is no need to kill and create twins over and over.

6. CONCLUSION

Figure 2 illustrate the process migration's grows tree .

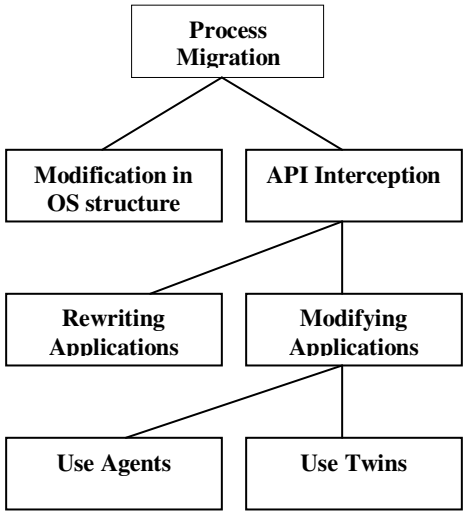

FIGURE 2: Process migration's growth tree.

Modifying stack protocol to implement process migration is not ignorable .This modification results in modification to the core of operating system and it is not desirable in process migration [2].Using API interception solves above problem . This solution operates in two distinct categories, rewriting applications and rebuilding (modifying) applications. High cost of rewriting application cause to this approach also is not desirable to use in process migration. The recent development in rebuilding applications cause to process migration using agents (this paper and [2, 4]). This approach also have problems that we explain in this paper .The suggested approach in this paper solved two major problems in process migration(bottleneck and single point of failure) and this approach cause to commonality and reduce costs of process migration.

7. SUMMARY

We have described a new technique to maintain network connectivity across migration of process. This approach has removed disadvantages of previous solutions.

8. REFERENCES

[1] Dejan S. Milojicic et al., "Process Migration", ACM Computing Surveys, Vol. 32, No. 3, September 2000.

[2] Xiaodong Fu et al., "Transparent Network Connectivity in Dynamic Cluster Environment", Proceedings of the 4th International Workshop on Network-Based Parallel Computing: Communication, Architecture, and Applications, 2000.

[3] Bernstein, P. A., "Middleware: A Model for Distributed System Services", Communications of the ACM, Vol. 39, Issue 2, pp. 86–98, 1996.

[4] Tom Boyd et al., "Process Migration: A Generalized Approach Using a Virtualizing Operating System", 22nd International Conference on Distributed Computing Systems, , 2002.

[5] Tanenbaum, "Modern Operating Systems", Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[6] Douglis, F. and Ousterhout, J. "Process migration in the sprite operating system", In Proceedings of 7th International Conference on Distributed Computing Systems, pp. 18–25, 1987.

[7] Paoli, D. and Goscinski, A. "The RHODOS Migration Facility", Technical Report TR C95/36, School of Computing and Mathematics, Deakin University, 1995.

[8] Rozier, M., Abrossimov, V., Gien, M., Guillemont, M., Hermann, F., and Kaiser, C. Chorus "Overview of the Chorus distributed operating system". In Proceedings of USENIX Workshop on Micro-Kernels and Other Kernel Architectures, pp. 39–70, 1992.

[9] Milojicic, D., Zint, W., Dangel, A., and Giese, P. "Task migration on the top of the mach microkernel", In Proc. of the 3rd USENIX Mach Symp., pp. 273–289, 1993.

[10] Baratloo, A., Dasgupta, A., and Kedem, Z. "Calypso: A novel software system for fault-tolerant parallel processing on distributed platforms", In Proc. of 4th IEEE Intl. Symp. on High Performance Distributed Computing, 1995.

[11] Blumofe, R., Joerg, C., Kuszmaul, B., Leiserson, C., Randall, K., and Zhou, Y. Cilk: "An efficient multithreaded runtime system", In 5th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming, pp. 207–216, 1995.

[12] Birman, K. "Replication and fault-tolerance in the ISIS system", In Proc. of 10th ACM Symp. on Operating System Principle, pp. 79–86, 1985.

[13] Hayden, M. "The Ensemble System", Technical Report TR98-1662, Cornell University, 1998.

[14] Hunt, G. and Brubacher, D. "Detours: Binary interception of Win32 functions", Technical Report MSR-TR- 98-33, Microsoft Research, 1999.

[15] Balzer, R. "Mediating Connectors", In Proc. of ICDCS Middleware Workshop, 1999.

[16] Milojicic, D., Douglis, F., Wheeler, R. "Mobility: Processes, Computers, and Agents", Addison-Wesley Longman and ACM Press, 1999.