Preprint

IN-61-CR

146

11P

CORRECTNESS CRITERIA FOR PROCESS MIGRATION

AN EXTENDED ABSTRACT

Chin Lu and J. W. S. Liu

1304 W. Springfield Avenue
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Contact Person: Chin Lu
Address: same as above
Phone: (217)-333-8741
Net Address: chine@cs.uiucdcs.edu

(NASA-CR-182939) CORRECTNESS CRITERIA FOR PROCESS MIGRATION (Illinois Univ.) 11 p CSCL 09B

N88-26823

Unclas
G3/61 0146703

This work was partially supported by the NASA Contract No. NAG 1-613 and a grant from AT & T Information Systems.

CORRECTNESS CRITERIA FOR PROCESS MIGRATI N

AN EXTENDED ABSTRACT

INTRODUCTION

A process is a program in execution. Process migration is the relocation of a process fron the host (***the source host***) on which it is executing to another host (***the de ination host***) [1] Process migration has been proposed as a way to improve response time [1-] and availabilit [1,3,6,7] in distributed systems. DEMO/MP [1], System V [2], and ACCENT] are well-knowı systems that support process migration. Process suspension and resumption p ckages have beeı implemented to support process migration in UNIX[+] environments [8,9].

In a homogeneous distributed system containing hosts that have i entical hardwar features, run functionally identical system software, and provide identica services to use processes, a user process can be migrated freely from one host to another in principle. W assume that every user process executes correctly on every host if it is nev r migrated. Th result produced by a process that is never migrated is referred to as its *norma result.* A proces that is migrated may produce a result different from its normal result. This result ma nevertheless be correct according to some adequate criteria, and hence its m gration is correct. In general, a migrated process may execute incorrectly or produce an unacce table result on it destination host. For example, a process that accesses host dependent variab es such as time f the day or process identification number on a UNIX[+] host may produce an in orrect result afte it is migrated. Simple migration packages described in [8,9] do not handle th migration of hos variable dependent processes correctly.

[+] UNIX is a trade mark of AT & T.

This paper defines a set of correctness criteria. These criteria are chosen to maximize the types of processes that are allowed to migrate. A process is not allowed to migrate when its migration cannot be done correctly according to them. Means to support correct migration and to identify processes that cannot be migrated (correctly) are discussed briefly.

MORE ON THE PROBLEM

Typically, a set of system server processes resides on every host and provides basic services to user processes on that host. We refer to these system server processes as *local-server processes* [10]. Conceptually, it is more convenient to envision that local-server processes never migrate. When a host fails, local-server processes die. Hereafter, the term *process* refers to a user process unless stated otherwise. We refer to the local-server process that handles the migration of processes on each host as the *migration handler* on that host.

From the stand point of any process, a system consists of two entities: the process itself and the environment of the process. The term *environment* loosely refers to the rest of the system with respect to that process. There are at least two aspects in the notion of correct process migration. One aspect is concerned with maintaining the internal state of the migrating process. Another aspect is concerned with the interaction between the process and its environment.

It is obvious that when a process is migrated, its internal state must be preserved. In other words, the internal state of the process at resumption time must be the same as the state of the process at suspension time. (The internal state information includes the process's stack, text segment, data segment, register contents, instruction pointer, etc.) We refer to this criterion as *state consistency criterion.* In all process migration implementations [1-3,8,9], this criterion is satisfied by making the information on the internal state of the migrating process at suspension time available on the destination host and by having the migration handler on the destination host restore the state of the process based on this information.

It is also obvious that the migration of any process should not leave its source host in an inconsistent state. The migration handler on the source host is responsible for this job. For instance, on a UNIX host, the files that are opened by the migrated process must be closed; the resources held by the process must be released, including locks for exclusive access of shared resources and temporary storage, etc.. In addition, during the execution and migration of any process, all consistency constraints of its environment must be satisfied. These consistency constraints are both application– and implementation–dependent. Typically different consistency constraints are enforced by different components of the system regardless. We need not be concerned with this problem here.

Any knowledge a process has of its environment is obtained via its interactions with some local–server processes on its host and with other processes in the system. In particular, through its interaction with the local–server processes on its source host, a process inherits some properties local to that host. If it interacts with the corresponding local–server processes on the destination host after it is migrated, it will inherit some properties local to the destination host. These properties may be inconsistent and the inconsistency may cause the migrated process to execute incorrectly.

For example, suppose that a simulation process P on host A uses explicit time reference to count the numbers of events during different time periods. Each host has a local clock; time readings at the same time on different hosts may be different. Suppose that P is suspended at t_1 and is resumed at t_2 according to host A's local time, and B's local time is $t_2^{'}$ when P is resumed. Since t_2 is usually not equal to $t_2^{'}$, the result obtained by P after migration may be different from the normal result of P. (Whether this different result is acceptable depends on the application.) In fact t_2' is not necessarily larger than t_1. In this case, P sees ar inconsistent viev of its environment against monotonic constraint of time as a result of it migration. Thi example illustrates that the correctness of migration depends the proc ss's view of it environment. Therefore, another criterion for correct migration should be that a migratin process sees a consistent view of its environment at all times. In other words the properties o its environment that are visible to a process must satisfy a set of consistenc constraints. W call this ***property consistency criterion*** and will return later to define it formall

Unlike the state consistency criterion, property consistency constraints a mainly semanti constraints on certain system properties and hence are typically systen dependent. Th monotonic property of system clock and the uniqueness of process identifi r are two simpl examples. In distributed systems where several versions of a file may exist n different hosts requiring a migrated process to use the same version (not necessarily the sar copy) is anothe example. Property consistency constraints can be specified at system level. n particular, the can be put in the specification of migration handlers as additional invarian conditions of th process environment that must be maintained on the behalf of a migrating pro ss.

We say that a process is migrated correctly if both the state consist cy and propert consistency criteria are satisfied. One way to ensure correct migration a process is t "encapsulate" its environment; the migrated process interacts with the same set of local-serve processes before and after the migration [1]. The migrated process continues be dependent o the source host. (For example, in the DEMOS/MP system, a migrated pro ss depends on it source host to forward messages.) This is not a feasible solution in gener . If migration carried out for the purpose of increasing host availability, when the host is down local-serve processes on the source host will die making it necessary for a migrated proce s to rely on a ne set of local-server processes on the destination host. Problems similar to the ne encountered i migrating processes that access location dependent variables will arise under this circumstance. In past studies on process migration, processes that access location dependent variables or rely on location dependent features are simply not migrated [1–3]. Methods to identify such processes are typically not given in these studies. We note that this restriction on the types of processes is often unnecessary. For example, the process P in the previous example cannot be migrated under this restriction. However, P can be migrated correctly either after it no longer needs to access the system clock or before it makes any access to the system clock. Alternatively, if the migration handlers can maintain a consistent view of its environment for P by compensating for the difference in system clocks on the two hosts, P can be migrated correctly at any time. We describe later a way to detect inconsistency between properties already inherited by a migrating process from its source host and properties it will inherit from its destination hosts. This inconsistency is resolved by the migration handler whenever it is possible to do so. By making sure that all property consistency constraints are satisfied on the behalf of a migrating process the migration handlers can ensure that some specific attributes of system features inherited by the process are kept consistent as the process migrates.

FORMAL DEFINITIONS

In order to define state consistency and property consistency constraints rigorously, we use the following formal model [11] to specify the interactions between a process and its environment. The former is an object P. The latter consists of a set of objects referred to as *environment objects.* An object consists of a name, a representation of data structures stored in the object, and a set of operations on the data structures in the object [12,13]. The process interacts with its environment by invoking the operations of the environment objects.

An object is modeled as a state machine M [14] represented by a 4-tuple $M = < \mathbf{S}, S_{0,}\ O,\ T >$, where **S** is the state space represented by a set of s ate variables; $S_($ denotes the initial state of the state machine; **O** is a set of transition operat ›ns; and T is th‹ transition function. Each operation in **O**, when executed, causes the object to change state. It: effect is expressed by the transition function T: $\mathbf{O} \times \mathbf{S} \longrightarrow \mathbf{S}$. More specific: ly, there are tw‹ basic types of functions — *V-functions* and *O-functions*. Each *primitive* V-fu ıction returns th‹ value of a state variable. The values returned by the set of all primitive V- ınction calls at : particular moment specify the state of the object at that moment. A V-funct on that returns : value computed from the values of primitive V-functions is called a *derived* V -function. An O function performs an operation that changes the state of the object. The stat‹ transitions calle‹ effects are described by assertions relating new values of primitive V-funct ›ns to their prio values. The set **O** of transition operations of M is the set of O-functions define l for the object.

The interface between an object and its outside world is the set of e ternal operation : consisting of all its derived V-functions and O-functions. A process ıteracts with it ; environment by invoking the external operations of the environment objects. M ore specifically, i : invokes either a derived V-function to examine the state of an environmen object or an O function to change the state of the environment object. Similarly, the state of the process objec : can be changed or examined by an environment object.

Let $S_p(t)$ be the internal state of the process P at time t. (In thi section, all tim e references are in terms of a clock external to the system.) $S_p(s)$ and $S_p(r)$ ar‹ the states of P a t suspension time s and resumption time r, respectively. The state consisten y criterion mea s that $S_p(s) = S_p(r)$.

Let $E_X(P)$ denotes the set of environment objects with which process P nteracts on host X. Suppose that there are n environment objects in $E_X(P)$, each of them i denoted by E_i, $i = 1, 2, .., n$. Then $E_X(P) = \{ E_1, E_2, .., E_n \}$. The state space of $E_X(P)$, den ted by $\mathbf{S}_X(P)$, is the product of the state spaces of all environment objects in $E_X(P)$. Suppose hat process P is migrated from host X to host Y. Let $S_X(P, s)$ be the state of $E_X(P)$ at susp ısion time s and $S_Y(P, r)$ be the state of $E_Y(P)$ at resumption time r. Clearly, if

$$S_X(P, s) = S_Y(P, r) \tag{1}$$

the migration is correct.

In a message-based system, the condition in (1) can be satisfied for a mi rating process P by having $E_X(P) = E_Y(P)$ and by allowing no messages between P and its e ıvironment to be exchanged during migration. This in fact is how correct migrations are done in the DEMOS/MP system [1] and in [10]. Sometimes, it will be too expensive or impossible to satisfy the stric condition (1). (For example, in a procedure-oriented system where a process nteracts with it environment through system calls.) Insisting that (1) is to be satisfied for every migrating process will make it impossible for many processes to migrate. Fortuna ely, (1) is not a necessary condition for the the property consistency criterion to be satisfied.

To state the property consistency criterion rigorously, we introduce he e the notion of view in the state machine context. A process P's *view* $V_X(P, t)$ of its envir nment $E_X(P)$ on host X at time t, is defined as the properties that P has observed about $E_X(P)$ in the tim interval starting from the creation time t_0 of P to the time t. $V_X(P, t)$ is par of the knowledg P has at t about its environment $E_X(P)$ [15]. Since the set of derived V-functions of al environment objects defines exactly what a process can observe about its envi onment, we defin $V_X(P, t)$ to be the set of values returned by all the derived V-functions of $E_X(P)$ invoked by P

in the time interval (t_0, t).

We express all property consistency constraints in terms of assertions rel ting the values o external V–functions before and after the migration. Let t_1 and t_2 be two ti ne instances wit $t_2 \geq t_1$. When a process migrates from host X to host Y between t_1 and t_2, its view $V_Y(P, t_2)$ o $E_Y(P)$ on host Y at t_2 is said to be consistent with its view $V_X(P, t_1)$ of $E_X(P$ on host X at t_1 i all the property consistency constraints specified for the system are satisfied. We say that P ha seen a *consistent view* of its environment at t_2 with respect to $V_X(P, t_1)$, or sin ply the two view are consistent; we use $V_X(P, t_1) => V_Y(P, t_2)$ to denote this fact. In other w rds, the propert consistency constraints serve as invariant conditions that are true for bo h $V_X(P, t_1)$ an $V_Y(P, t_2)$. Since a process runs correctly on a single host, $V_X(P, t_1) => V$ (P, t_2) is alway true. The consistency of different views has the transitive property. Hence, for different tim intervals $t_1 < t_2 < t_3$, if $V_X(P, t_1) => V_Y(P, t_2)$ and $V_Y(P, t_2) => V_Z(P, t_3)$, the $V_X(P, t_1) => V_Z(P, t_3)$ also holds.

The property consistency criterion is

$$V_X(P, s) \; => \; V_Y(P, r) \qquad (2)$$

where s and r denote suspension time and resumption time, respectively. In tead of the stric condition (1), we only require that (2) be satisfied during the migration c a process. Thi criterion simplifies the implementation of migration handlers and allows nore processes t migrate since the states of the environment objects on two hosts may be diffe ent so long as th two views are consistent.

Finally, the importance of migration transparency has been discusses at ength in literatu [1–6]. Migration tranparency requires that any names related to process i entifiers that a passed to user processes in the system including the migrated process itself not be changed when a process is migrated. We note that migration transparency can be specified as one of the state and property consistency constraints.

SUMMARY

Two correctness criteria, the state consistency criterion and the property consistency criterion for process migration are discussed in this paper. The state machine approach is used to model the interactions between a user process and its environment. These criteria are defined in terms of this model. We introduced the idea of environment view to distinguish what a user process observes about its environment from what its environment state really is and argue that a consistent view of the environment must be maintained for every migrating process.

A separate paper [16] discusses the design of migration handlers that ensures correct process migration in procedure-oriented systems. As a local-server process, a migration handler never migrates. When a process is to be migrated, the migration handler on the source host (1) suspends the process; (2) identifies if the process can be migrated correctly according to the criteria given here; and (3) transfers the internal state of the process and its view of the environment to the handler of the destination host for resumption. Upon receiving the information sent by the migration handler on the source host, the migration handler on the destination host (4) resumes the process and (5) presents the migrated process a consistent view of its environment. Ways to specify and implement system calls and to store history on process interactions that allow their views be generated and maintained efficiently are described along with examples illustrating the implementation of key components for the UNIX environment.

REFERENCES

[1] Powell, M. L. and Miller, B. P., "Process Migration in DEMOS/MP", *Operating System Review*, Vol. 17, No. 5, 1983.

[2] Theimer, M. M., Lantz, K. A., and Cheriton, D. R., "Preemptable Remote Executio Facilities for the V-System", *ACM Operating Systems Review*, Vol. 19, N. 5, 1985

[3] Rashid, R. F. and Robertson, G. G., "Accent: A Communication Oriented Networ Operating System Kernel", *Proc. of the Eighth Symposium on Operating ystems Principle*, December, 1981.

[4] Ni, L. M., Xu, C.-W. and Gendreau, T. B., "Draft Algorithm – A Dynamic Proces Migration Protocol", *Proceedings of the Fifth International Confere e on Distribute Computing Systems*, May 13–17, 1985.

[5] Stone, H. S., "Multiprocessor Scheduling with the Aid of Network Flow lgorithms", *IEE Transactions on Software Engineering*, SE-3, No. 1, January, 1977.

[6] Rennels, D. A., "Distributed Fault-Tolerant Computer Systems", *IEE Computer*, Vo. 13, No. 3, March, 1980.

[7] Solomon, M. H. and Finkel, R. A., "The Roscoe Distributed Operating System" *Proceedings of the Seventh Symposium on Operating Systems Principle* 10–12 Decembe 1979.

[8] Cagle, R. P., "Process Suspension and Resumption in UNIX System V Operating System" *Thesis Report UIUCDCS-R-86-1240*, Department of Computer Scie ce, University o Illinois at Urbana-Champaign, January 1986

[9] Chen, A. Y.-C., "An UNIX 4.2BSD Implementation of Process Suspension an Resumption", *Thesis Report UIUCDCS-R-86-1286*, Department of Computer Science University of Illinois at Urbana-Champaign, June 1986

[10] Lu, C., Chen, A. and Liu, J., "Protocols for Reliable Process Migration". to be appeared i *IEEE Infocom '87*, Six Annual Joint Conference of the IEEE Computer ar d Communicatio Societies: Global Networks- Concept to realization, March 30–April 2, 1987.

[11] Guttag, J., "Abstract Data Types and the Development of Data Structure" *Communication ACM*, Vol. 20, No. 6, June 1977

[12] Goldberg, A., and Robson, D., "Smalltalk-80 the Language and its Implementation" *Addison-Wesley Publishing company*, July 1985

[13] Strousptup, B., "Data Abstraction in C", *AT&T Bell Laboratories Tech ical Journal*, Vol 63, No. 8, Part 2, October 1984

[14] Parnas, D. L., "A Technique for Software Module Specification vith Examples" *Communication ACM*, Vol. 15, No. 5, May 1972

[15] Halpern, J. Y. and Moses, Y., "Knowledge and Common Knowledge in a Distribute Environment", *ACM SIGACT-SIGOPS*, Symposium on Principles of Distribute Computing, Vancouver, Canada, August 1984

[16] Lu, Chin and J. W. S. Liu, "Correct Process Migration in UNIX Environ nents." paper t be submitted to the 11th Symposium on Operating System Principle *ACM SIGOPS* Austin, Texas, 1987

DEPARTMENT OF COMPUTER SCIENCE

Request for Copying

REQUESTED BY Chin Lu PHONE NO. 3-9741

ROOM NO. 5109

COPY CENTER ACCOUNT NUMBER 104

DEADLINE, IF ANY by 9am would be much apprecia

NUMBER OF COPIES 11

TYPE OF WORK Conference paper

RUN ON ONE SIDE ONLY ✓

RUN ON BOTH SIDES

ASSEMBLE ONLY

ASSEMBLE/STAPLE ✓

ASSEMBLE/BIND

SPECIAL INSTRUCTIONS AND COMMENTS