Process Migration: Effects on Scientific Computation.

Gerald Q. Maguire, Jr.
Jonathan M. Smith

Columbia University Computer Science Department
450 Computer Science, Columbia University, NY, NY 10027

Technical Report No. CUCS-268-87

ABSTRACT

This report describes the notion of process migration, and points out certain vulnerabilities to architectural assumptions.

Computationally intensive processes, typified by scientific computations, are among the best candidates for migration. In architecturally heterogeneous systems, these processes face particular problems due to their dependence on architectural features such as "machine precision". We point out some of these problems, and suggest some solutions.

Process Migration: Effects on Scientific Computation.

Gerald Q. Maguire, Jr.
Jonathan M. Smith

Computer Science Department
Columbia University, New York, NY 10027

1. Introduction

Process migration has been proposed or implemented as a feature of several operating systems[1] [2] [3]. A discussion of such mechanisms is found in Tanenbaum and Van Renesse[4]; the point of these designs has typically been load-sharing[5] in distributed systems. These systems have typically been composed of *homogeneous* nodes; some interesting questions arise if we attempt to extend the notion of process migration to *heterogeneous* nodes.

The remainder of the introduction provides background material on process migration and the mechanics of migration.

1.1 Process Migration

An *image* is a description of a computation which can be *executed* by a computer. A *process* is a computation in some state of execution, with the initial state given by the *image*. At any given time, the state of the process can be represented as two components: the initial state and the *changes* which have occurred due to execution. The total information, that is, the initial state together with the changes, gives us the *state* of a process. If we *transfer* the state of a process from one processor to another (Figure 1),

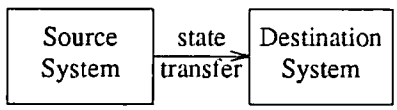

Figure 1. State Transfer

so that an ongoing computation can be correctly continued (Figure 2),

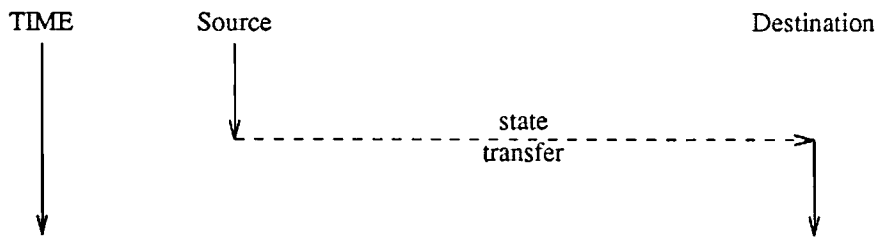

Figure 2. Flow of execution of migrating process

we have *migrated* the process.

Process migration is most interesting in systems where the involved processors do not share main memory; in the case where they do the state transfer is trivial. A typical environment where process migration *is* interesting is autonomous computers connected by a network.

1.2 External Data Representation

An environment where the computer systems are exactly alike is called *homogeneous*; differing processor types or operating systems define a *heterogeneous* environment.

Process migration between processors of the same type is really a problem of *saving* the state information on the local processor, and *restoring* it on the remote processor; since the interpretation of the data is consistent on the sender and the receiver, there is no *translation* problem.

Consider the approach of data *translation* between two processor types. Each processor type must be able to convert the data from the foreign processor type into a format which it can use. If a third processor type is added, each processor must be able to translate between its own representation and that of two other processors. In general, the complexity of the translation software, in terms of the number of data representations which must be supported, grows with n processor types as follows:

- Each processor must be able to translate from the representations of *n-1* other processor types,
- This is multiplied by the number,*n*, of processor types,
- For a total of $O(n^2)$ pieces of translation software.

This is undesirable, as adding a new processor type becomes a more difficult task over time. A way to reduce the software complexity of the translation process is to use a standard representation for the transport of data. A new processor then needs only to be able to convert data to and from the standard form; this bounds the complexity of the software. Thus, process migration schemes between heterogeneous processors will require an *external data representation*. The *external data representation* must successfully address the problem of different representations for data such as characters, integers, and floating point numbers.

One important issue that should be dealt with in such an *external data representation* scheme is the different precisions and magnitudes of floating point representations. If this issue is not dealt with intelligently, users performing scientific computation may not be able to trust their results.

2. *The Problem*

The two (not counting the *sign*) parts of a floating point number[1] must be handled differently, as the side effects caused by an *external data representation* affect each of the two components differently. These two parts are the exponent and the mantissa. The problems will be discussed in the two sections following.

2.1 Exponent

Different architectures[2] allocate different numbers of binary digits (bits) in a floating point datum to processor exponents. Assume processor A uses 8 bits, processor B uses 16 bits, and the *external data representation*, designed by users of processor architecture A, provides 12 bits (4 to be safe). Assume, for the sake of this section's discussion, that all three representations use the same number of bits for the mantissa.

A process on processor A which is migrated to processor B should experience no problems in representing its floating data; the *external data representation* has plenty of room for its 8 bits of exponent in its 12 bits, and the 12 bits used by the *external data representation* again easily fit in the 16 bits of processor B's exponent.

However, the process on processor B may face several problems. First, if the exponent of a number requires more than 12 bits, the *external data representation* cannot contain it. If inadequate error handling exists, the process may not have meaningful data by the time it gets to processor A. Thus, the migration *mechanism* must be made aware that this process cannot be migrated, as its floating point data cannot be represented. Note that the problem here is with the *representation*, as processes using such a scheme cannot even be transferred between processors of the same type (B), which can represent the same numbers.

This problem of the representation can only be eliminated by guaranteeing that the *external data representation* have at least as many bits in the exponent as the longest exponent on any processor in the distributed system.

A second problem, not so easily addressable, is the problem of a number on processor B with 8 < # bits <= 12 in the exponent. In this case, the *external data representation* easily handles the number, but processor A cannot; it must raise an overflow or underflow[3] exception upon conversion from the *external data representation* or expect meaningless results.

The only obvious solution to this problem is to guarantee that numbers used by programs which migrate have a smaller exponent value than the smallest processor on the network; this may be intolerable for serious scientific computation. Solutions such as emulating the larger processor's values may be prohibitively expensive in terms of computation time. Defining a subset of processors which can serve as *destinations* in a migration may be a viable solution; the idea here is that the process may only be moved to a *destination* processor if the representable

---

1. Those not familiar with the topic of *floating point representation* may wish to consult a basic text on numerical computation, such as Ralston & Rabinowitz[6] or Cheney & Kincaid[7].
2. A description of architectural decisions relevant to floating point representations can be found in Chapter 2 of Stone[8].
3. Depending upon the sign of the exponent.

exponent is at least as large as that on the *source* processor. In any case, exception handling for overflow or underflow must be performed in some fashion.

2.2 The Mantissa

Another issue is the representation of mantissas; this discussion follows the same reasoning as above, but deals with a rather more subtle problem that may creep in due to multiple process migrations. Assume that processor A and processor B have different architectures; assume that processor A uses 32 bit mantissas, processor B uses 64 bit mantissas, and that the *external data representation* uses 48 bit mantissas; assume that the exponent field is of the same size on all processors.

Once again, the user of processor A has a fine situation, where both the *external data representation* and processor B can represent his data with no loss of precision (in fact, computations on processor B will occur with less loss of precision than those carried out on processor A!).

The user of processor B has several concerns, however. The conversion process from his representation to the *external data representation* throws away 15 bits of data with rounding, 16 with chopping. The conversion process from *external data representation* to processor B's format throws away another 16 bits of data. This means that the computation on processor A will be carried out in "half-precision", a fact that the numerical analyst may not have counted on when designing the program which the process is executing.

The solutions to these problems are similar to those discussed above; the *external data representation* must have sufficient precision to handle the largest mantissa, and the migration will not be transparent without either conformance with some limits on expected precision or emulation of higher precision computation.

2.2.1 Multiple migrations

One item of concern here is the effect of multiple transitions between a set of processors, where our example processors A and B might be a subset. This is a concern only in the mantissa case because loss of data in the exponent is catastrophic, while loss of data in the mantissa can be viewed as an additional perturbation of a computation.

One suspects that the precison loss which we have discussed (in the case of moving from A to B) may be cumulative, and thus a series of migrations may totally invalidate a computation. However, the situation is not that bad, assuming that the *external data representation* is properly designed, i.e. that it is adequate to represent the longest mantissa of any processor encountered in the course of the computation. If we take an alternate point of view, we can see why multiple migrations are not a problem. Informally, postulate a processor C, with a floating point representation such that processor C has precision (number of bits in the mantissa) less than or equal to any other processor that processes migrate to. Then, the precision of our results will never be *worse* than performing the calculation on processor C with no migration. This is because we can view the remote computations as "extra precision" calculations with respect to processor C's floating point.

2.3 Other issues

Other issues which the *external data representation* designer should address may arise. Two of these are *signed infinity* and *signed zero*. Some architectures support a value which indicates that a result too large (or too small) to store has been generated; this is the result of an *overflow* (*underflow*), and is called *signed infinity*.

Other architectures may use the sign bit of a value which would otherwise be zero, thus giving rise to a *signed zero*.

The problem that we have is that these representations may not be supported on all systems; decisions have to be made in the translation algorithms about how to deal with these situations, but the *external data representation should* provide support for these values, so that a given processor can either take advantage of the extra information or discard it.

3. Conclusions

The important point of this discussion for the *external data representation* designer is that he must be extremely knowledgeable and conservative in the design of the *external data representation*; for example, precision loss can also be caused by conversion between bases, e.g. 0.10_{10} can not be represented exactly in base 2. Such conversion could occur if, for example, an American Standard Code for Information Interchange (ASCII) base-10 representation was chosen.

For the designer of a process migration facility, however, the problems of heterogenity are somewhat more difficult than just the *external data representation* design. For example, performance considerations may prohibit using an otherwise satisfactory technique such as a virtual machine with the required precision[4]. These problems must be addressed for such a facility to be useable.

The issues here are real, and can affect the users of facilities; for examples, SUN's XDR[9] uses IEEE standard floating point format[5] which is insufficient to represent the floating point values of several architectures, including several floating point data types[6] supported by the VAX-11 Architecture[10] which is often found in networks containing SUN workstations.

---

4. E.g., a mechanism which migrates LISP processes may convert a running program into a symbolic representation, which can be copied to a remote processor and interpreted by the LISP on that processor. BIGNUM behavior will be independent of the underlying processor.

5. A number is described by $(-1)^{S}*2^{E-Bias}*1.F$, where:
   S is the *sign*. 0/1 encode positive/negative.
   E is the base 2 exponent. 8 bits for floats, 12 for doubles. The corresponding *Biases* are 127 and 1023, respectively.
   F is the base 2 fractional part of the mantissa, with 23 bits for floats and 52 for doubles.

6. More specifically, the *H_floating* data type has an excess-16384 binary exponent (15 bits), and a 112 bit field devoted to the mantissa. The 112 bits are used to represent a 113 bit quantity by not representing the redundant most significant fraction bit.

REFERENCES

1. *Process Migration in DEMOS/MP*
   Michael L. Powell and Barton P. Miller
   Proceedings of the Ninth ACM Symposium on Operating Systems Principles
   1983

2. *Network Tasking in the LOCUS Distributed UNIX System*
   David A. Butterfield and Gerald J. Popek
   USENIX Conference Proceedings
   Summer 1984
   pp. 62-71

3. *Preemptable Remote Execution Facilities for the V-System*
   Marvin M. Theimer, Keith A. Lantz, and David R. Cheriton
   10th ACM Symposium on Operating Systems Principles
   1985
   pp. 2-12

4. *Distributed Operating Systems*
   Andrew S. Tanenbaum and Robbert Van Renesse
   ACM Computing Surveys
   Volume 17
   Number 4,
   December 1985
   pp. 419-470

5. *A Distributed Load-balancing Policy for a Multicomputer*
   Amnon Barak and Amnon Shiloh
   SOFTWARE - PRACTICE AND EXPERIENCE
   Volume 15(9), pp. 901-913
   (September 1985)

6. *A First Course in Numerical Analysis (2nd Edition)*
   Anthony Ralston and Philip Rabinowitz
   McGraw-Hill
   1978

7. *Numerical Mathematics and Computing (2nd Edition)*
   Ward Cheney and David Kincaid
   Brooks/Cole
   1985

8. *Introduction to Computer Architecture (2nd Edition)*
   Harold S. Stone
   SRA
   1980

9. SUN XDR Protocol Specification (Release 2.0)
   SUN Microsystems, Inc.
   1985

10. *VAX Architecture Handbook*
    Digital Equipment Corporation
    1982-1983