Fine-Grained Mobility in the Emerald System*

Eric Jul, Henry Levy, Norman Hutchinson, and Andrew Black†
Department of Computer Science
University of Washington
Seattle, WA 98195

Extended Abstract

Emerald [2,3,4] is a distributed object-based language and system designed to simplify the construction of distributed programs. An explicit goal of Emerald is support for object mobility; objects in Emerald can freely move within the system to take advantage of distribution and dynamically changing environments. We say that Emerald has *fine-grained* mobility because Emerald objects can be small data objects (such as arrays, records, and integers) as well as objects with processes. Thus, the unit of mobility can be much smaller than in process migration systems which typically move entire address spaces [5,7]. Object mobility in Emerald subsumes both process migration and data transfer.

Emerald provides mobility support through a small set of language primitives. Object invocation is location-independent, that is, it is the responsibility of the Emerald kernel to locate the target of an invocation request. However, objects can also utilize distribution explicitly; an Emerald object can *locate* an object, *move* an object to another node, and *fix* an object at a particular node. The Emerald programmer may also wish to specify which objects move together. For this purpose, the Emerald language allows the programmer to *attach* objects to other objects. Attachment is transitive: when Emerald moves object *a*, any object attached to *a* will also be moved. For example, linked structures may be moved as a whole by attaching the link fields.

In addition to mobility, a principal goal of Emerald was support for a single object model suitable for programming both small, local data-only objects and active, mobile distributed objects. All Emerald objects are defined through a single object definition mechanism. The Emerald compiler analyzes object definitions and attempts to produce efficient implementations commensurate with the way in which objects are used. For example, an object that moves around the network will require a very general remote procedure call implementation; however, an object that is completely internal to that mobile object can be implemented using direct memory addressing and inline code or procedure calls.

We wanted to achieve performance competitive with standard procedural languages in the local case and standard remote procedure call systems in the remote case. These goals are not trivial in a location-independent object-based environment. To meet them, we relied heavily on an appropriate choice of language semantics, a tight coupling between the compiler and run-time kernel, and careful attention to implementation.

As an example of Emerald's local performance, Table 1 shows execution times for several local Emerald operations executed on a MicroVAX II[1]. The "resident global invocation" time is for a global object (i.e., one that can move around the network) when invoked by another object resident on the same node. By comparison, other object-based distributed systems are typically over 100 times slower for local invocations of their most general objects [6,1].

The Emerald language uses call-by-object-reference parameter passing semantics for all invocations, local or remote. While call-by-object-reference is the natural semantics for object-based systems, it presents a potential performance problem in a distributed environment. When a

*This work was supported in part by the National Science Foundation under Grants No. MCS-8004111 and DCR-8420945, by Københavns Universitet (the University of Copenhagen), Denmark under Grant J.nr. 574-2,2, by a Digital Equipment Corporation External Research Grant, and by an IBM Graduate Fellowship.

†Author mobility: Eric Jul, DIKU, Dept. of Computer Science, University of Copenhagen, Universitetparken 1, DK-2100 Copenhagen, Denmark. Norman Hutchinson, Department of Computer Science, University of Arizona, Tucson, AZ 85721. Andrew Black, Digital Equipment Corporation, 550 King Street, Littleton, MA 01460.

[1]MicroVAX is a trademark of Digital Equipment Corporation.

Emerald Invocation	Example	Time/μs
primitive integer	$i \leftarrow i + 23$	0.4
primitive real	$x \leftarrow x + 23.0$	3.4
local	localobject.no-op	16.6
resident global	globalobject.no-op	19.4

Table 1: Timings of Local Emerald Invocations

Operation Type	Time/ms
local invocation	0.019
kernel CPU time, remote invocation	3.4
elapsed time, remote invoc	27.9
remote invoc, local reference param	31.0
remote invoc, call-by-move param	33.0
remote invoc, call-by-visit param	37.4
remote invoc, remote reference param	61.8

Table 2: Remote Operation Timing

remotely invoked object attempts to access its arguments, those accesses will typically require remote invocations. Because Emerald objects are mobile, it may be possible to avoid some of these remote references by moving argument objects to the site of a remote invocation. To make argument mobility possible, Emerald provides a parameter passing mode that we call *call-by-move*. Call-by-move does not change the semantics, which is still call-by-object-reference, but at invocation time the argument object is relocated to the destination site. Following the call the argument object may either return to the source of the call or remain at the destination site (we call these two modes *call-by-visit* and *call-by-move*, respectively).

Table 2 shows the elapsed time cost of various remote Emerald operations. In each remote invocation, the argument object is invoked exactly once in the body of the operation. For the simplest remote invocation, the time spent in the Emerald kernel is 3.4 milliseconds. For historical reasons, we currently use a set of network communications routines that provide reliable, flow-controlled message passing on top of UDP datagrams. These routines are slow: the time to transmit 128 bytes of data and receive a reply is about 24.5 milliseconds. Hence, the total elapsed time to send the invocation message and receive the reply is 27.9 milliseconds.

From this table we can compute the benefit of call-by-move for a simple argument object. For this simple argument object, the additional cost of call-by-move was 2 milliseconds while call-by-visit cost 6.4 milliseconds. These are computed by subtracting the time for a remote invocation with an argument reference that is local to the destination. The call-by-visit time includes sending the invocation message and the argument object, performing the remote invocation (which then invokes its argument), and returning the argument object with the reply. Had the argument been a reference to a remote object (i.e., had the object not been moved), the incremental cost would have been 30.8 milliseconds. These measurements are somewhat of a lower bound because the cost of moving an object depends on the complexity of the object and the types of objects it names.

Emerald currently executes on a small network of MicroVAX IIs and has recently been ported to the SUN 3[2]. We have concentrated on implementing fine-grained mobility in Emerald while minimizing its impact on local performance. This has presented significant problems; however, through the use of language support and a tightly-coupled compiler and kernel, we believe that our design has been successful in meeting both its conceptual and performance goals.

References

[1] Guy T. Almes, Andrew P. Black, Edward D. Lazowska, and Jerre D. Noe. The Eden system: a technical review. *IEEE Transactions on Software Engineering*, SE-11(1):43–59, January 1985.

[2] Andrew Black, Norman Hutchinson, Eric Jul, and Henry Levy. Object structure in the Emerald system. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 78–86, October 1986.

[3] Andrew Black, Norman Hutchinson, Eric Jul, Henry Levy, and Larry Carter. Distribution and abstract types in Emerald. *IEEE Transactions on Software Engineering*, January 1987.

[4] Norman C. Hutchinson. *Emerald: An Object-Based Language for Distributed Programming.* PhD thesis, University of Washington, January 1987. Department of Computer Science technical report 87-01-01.

[5] Michael L. Powell and Barton P. Miller. Process migration in DEMOS/MP. In *Proceedings of the Ninth ACM Symposium on Operating Systems Principles*, pages 110–119, ACM/SIGOPS, October 1983.

[6] Eugene H. Spafford. *Kernel Structures for a Distributed Operating System.* PhD thesis, School of Information and Computer Science, Georgia Institute of Technology, May 1986. Also Georgia Institute of Technology Technical Report GIT-ICS-86/16.

[7] Marvin M. Theimer, Keith A. Lantz, and David R. Cheriton. Preemptable remote execution facilities for the V-system. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, pages 2–12, ACM/SIGOPS, December 1985.

[2]SUN is a trademark of SUN Microsystems, Inc.