REXEC: A Decentralized, Secure Remote Execution Environment for Clusters

Brent N. Chun and David E. Culler
University of California at Berkeley
Computer Science Division, Berkeley, CA 94720
Tel: +1 510 642 8299, Fax: +1 510 642 5775
{bnc,culler}@cs.berkeley.edu
http://www.cs.berkeley.edu/~{bnc,culler}

Abstract. Bringing clusters of computers into the mainstream as general-purpose computing systems requires that better facilities for transparent remote execution of parallel and sequential applications be developed. While much research has been done in this area, most of this work remains inaccessible for clusters built using contemporary hardware and operating systems. Implementations are either too old and/or not publicly available, require use of operating systems which are not supported by modern hardware, or simply do not meet the functional requirements demanded by practical use in real world settings. To address these issues, we designed REXEC, a decentralized, secure remote execution facility. It provides high availability, scalability, transparent remote execution, dynamic cluster configuration, decoupled node discovery and selection, a well-defined failure and cleanup model, parallel and distributed program support, and strong authentication and encryption. The system is implemented and is currently installed and in use on a 32-node cluster of 2-way SMPs running the Linux 2.2.5 operating system.

Keywords: Clusters, Remote execution, Distributed systems, Decentralized control

1 Introduction

We have designed and implemented a new remote execution environment called REXEC 1 to address the lack of a sufficient remote execution facility for parallel and sequential jobs on clusters of computers. Building on previous work in remote execution and practical experience with the Berkeley NOW and Millennium clusters, the system provides decentralized control, transparent remote execution, dynamic cluster membership, decoupled node discovery and selection, a well-defined error and cleanup model, support for sequential programs as well as parallel and distributed programs, and user authentication and encryption. It takes advantage of modern systems technologies such as IP multicast and mature OS support for threads to simplify its design and implementation. It is implemented almost entirely at user-level with small modifications to the

1 Our REXEC system has no relation to the 4.2 BSD rexec function, nor does it have any relation to the rexec command used in the Butler [13] system or the rexec function in NEST [1].
Linux 2.2.5 kernel. The system is currently installed and in use on a 32-node cluster of 2-way SMPs as part of the UC Berkeley Millennium Project.

The rest of this paper is organized as follows. In Section 2, we state our design goals and assumptions for the REXEC system. In Section 3, we describe the REXEC system architecture and our implementation on a 32-node cluster of 2-way SMPs running the Linux operating system. In Section 4, we discuss three examples of how REXEC has been applied to provide remote execution facilities to applications. In Section 5, we discuss related work. Section 6 describes future work and in Section 7 we conclude the paper.

2 Design Goals and Assumptions

In this section, we describe our design goals and the assumptions made in designing REXEC. Our design goals are based on several years of practical experience as users of the Berkeley NOW cluster, a thorough examination of previous systems work in remote execution, and a desire to combine and extend key features in each of the systems into a single remote execution environment. Our goals are as follows:

- High availability. The system should be highly available and provide graceful degradation of service in the presence of failures.
- Scalability. As more nodes are added and more applications are run, remote execution overhead should scale gracefully.
- Transparent remote execution. Execution on remote nodes should be as transparent as possible.
- Minimal use of static configuration files. The remote execution system should rely on as few static configuration files as possible.
- Decoupled discovery and selection. The process of discovering which nodes are in the cluster and what their state is should be separated from the selection of which nodes to run an application on.
- Well-defined failure and cleanup models. The system should provide well-defined models for failure and cleanup.
- Parallel and distributed program support. The remote execution environment should provide a minimal set of hooks that allow parallel and distributed runtime environments to be built.
- Security. The system should provide user authentication and encryption of all communication.

Our assumptions are typical of remote execution systems and not overly restricting or extensive. Modern clusters built using off-the-shelf hardware and contemporary operating systems are easily configured to satisfy these assumptions. Our assumptions are as follows:

- Uniform file pathnames. We assume that all shared files are accessible on all nodes using the same pathnames and that most local files on each node are also accessible under the same pathnames (e.g., /bin/ls).