HARNESS: Heterogeneous Adaptable Reconfigurable NEtworked SystemS

Jack Dongarra – Oak Ridge National Laboratory and
University of Tennessee, Knoxville
Al Geist – Oak Ridge National Laboratory
James Arthur Kohl – Oak Ridge National Laboratory
Philip M. Papadopoulos – Oak Ridge National Laboratory *
Vaidy Sunderam – Emory University

March 3, 1998

Abstract

This paper describes our vision, goals and plans for HARNESS, a distributed, reconfigurable and heterogeneous computing environment that supports dynamically adaptable parallel applications. HARNESS builds on the core concept of the personal virtual machine as an abstraction for distributed parallel programming, but fundamentally extends this idea, greatly enhancing dynamic capabilities. HARNESS is being designed to embrace dynamics at every level through a pluggable model that allows multiple distributed virtual machines (DVMs) to merge, split and interact with each other. It provides mechanisms for new and legacy applications to collaborate with each other using the HARNESS infrastructure, and defines and implements new plug-in interfaces and modules so that applications can dynamically customize their virtual environment.

HARNESS fits well within the larger picture of computational grids as a dynamic mechanism to hide the heterogeneity and complexity of the nationally distributed infrastructure. HARNESS DVMs allow programmers and users to construct personal subsets of an existing computational grid and treat them as unified network computers, providing a familiar and comfortable environment that provides easy-to-understand scoping. Similarly, a particular site could use HARNESS to construct a virtual machine that is presented and utilized as a single resource for scheduling within the grid.

Our research focuses on understanding and developing three key capabilities within the framework of a heterogeneous computing environment: 1) Techniques and methods for creating an environment where multiple distributed virtual machines can collaborate, merge or split; 2) Specification and design of plug-in interfaces to allow dynamic extensions to services and functionality within a distributed virtual machine; and 3) Methodologies for distinct parallel applications to discover each other, dynamically attach, collaborate, and cleanly detach.

1 Introduction

Current software systems, like PVM and MPI, provide a utilitarian model for personally managing a collection of computing resources into a single virtual machine (VM). Users

*Corresponding author: Philip M. Papadopoulos, (423) 241-3972, Oak Ridge National Laboratory, 1 Bethel Valley Rd. Oak Ridge, TN 37831-6367, phil@msr.epm.ornl.gov

are comfortable with the virtual machine abstraction and the encapsulation of resources that it provides. However, current VM implementations have limited degrees of support for handling faults and failures, utilizing heterogeneous and dynamically changing communication substrates, and enabling migrating or cooperative applications. Virtual machines have the benefit of providing information scoping that is independent of an application and insulates a program from the entire universe. We see the VM approach as a practical mechanism for a wide variety of, but certainly not all, applications that can utilize an Internet or computational grid environment. For example, The VM approach is not appropriate for anonymous communication services like web servers, because the VM builds a wall where none is needed. In the space of dynamically adaptable (migrating, cooperating, or fault-tolerant) applications, monitoring agents must exist to watch for events that affect a particular application. VMs are particularly compelling in this application domain because they allow one to design several types of monitoring agents. These agents are part of the virtual construction and can be used by all applications as a guaranteed service. The distributed virtual machine (DVM) scoping of resources allows these agents to narrow their focus (and thereby reduce complexity) from the "entire" Internet to a manageable subset.

While the encapsulation provided by a virtual machine is very compelling, it becomes clear that sometimes entities from two different virtual machines would like to communicate across VM boundaries and share resources. Processes within one virtual machine cannot communicate using messages to tasks within another virtual machine without some sort of sharing. While, TCP sockets allow physical machines to inter-communicate, we believe a more powerful notion is to enable merging of distinct VMs into a single VM (plug together) and, just as easily, to split a single VM into multiple independent VMs (pull apart). HARNESS will design and implement mechanisms to accomplish these goals.

Our research agenda is to explore the dynamic needs and capabilities of the extended virtual machine environment. HARNESS is not about reinventing message passing APIs (initially, both PVM and MPI semantics will be supported), but rather how to construct a virtual environment that can dynamically change (almost) anything at runtime. To this end, our focus is on

- Extension of the present network and cluster computing model to include multiple distributed virtual machines, each with multiple users, where the virtual machines can dynamically merge, split and in general collaborate with each other. Today's cluster computing frameworks typically involve the static configuration of a single distributed virtual machine on which a user executes multiple applications.

- Development of a generalized plug-in paradigm for distributed virtual machines that allows users or applications to dynamically customize, adapt and extend the distributed computing environment's features to match their needs. This is analogous to the plug-in interfaces in use today for serial applications (e.g. web browsers, graphics editors, multimedia players), but extended to distributed systems.

- Creation of a collaborative computing framework that allows multiple parallel applications running in a heterogeneous distributed environment to dynamically attach, interact and detach from one another. One existing example of such capabilities is the CUMULVS [8] system, which allows collaborative computational steering. We will provide a framework that supports a wide class of parallel tools and applications that would benefit from being able to attach and detach from each other. The framework will integrate discovery services with an API that defines baseline attachment and detachment protocols between heterogeneous, distributed applications.

This extended abstract will highlight these issues and our plans for building such an environment.

2 Where HARNESS Fits and Why Pluggability?

Although distributed computing frameworks continue to be expanded and improved, the growing need in terms of functionality, paradigms, and performance quite simply are increasing faster than the pace of these improvements. By developing a distributed computing framework that supports plug-ins, it will be possible to extend or modify the capabilities available to parallel applications without requiring immediate changes in the standards, or endless iterations of ever-larger software packages. For example, a distributed virtual machine could plug in modules for distributed shared memory programming support along with message passing support, allowing two legacy applications, each written in their own paradigm, to interoperate in the same DVM. This type of plug-in enables efficient adaptation to many new capabilities, such as new advanced communication protocols or networks, programming models, and encryption methods, without the need for extensive re-engineering of the computing framework. To derive full benefit, HARNESS plug-ins will be dynamic in nature, or "hot-pluggable." Certain features or functionality will plug in temporarily, only while needed by an application, and then unplug to free up system resources. Distributed applications no longer will need to adjust to fit the capabilities of the distributed computing environment. Instead, the environment can be dynamically adapted to the changing needs of the application.

Beyond just plugging small functional components into the distributed environment, this concept of pluggability can be extended to encompass the merging and splitting of DVMs, as well as the attachment of tools and applications in collaborative scenarios. Analysis tools can plug into applications on-the-fly to collect information or steer computations. Peer applications will be able to "dock" with each other to exchange intermediate results or even active objects (e.g., Java bytecode) thereby facilitating collaborative computing at the software level.

3 Technical Design

The HARNESS virtual machine model is a dynamically adapting configuration of heterogeneous machines that presents a unified distributed computing environment to users. Machines, or sub-clusters, are brought into or taken out of the distributed virtual machine by cooperating and collaborating users, either via a user interface or under application program control. The DVM consists of a "software backplane" and a set of HARNESS plug-in modules that execute on each machine to coordinate the interaction among machines and applications. A small bootstrap kernel is initialized whenever a machine actively joins the DVM. Modules are subsequently plugged into this kernel to set the functional level of the new machine (or sub-cluster) or bring it up to par with the rest of the DVM. The kernel itself essentially contains only the facilities necessary to incorporate plug-ins and to ensure that additions on one machine (sub-cluster) are compatible with the rest of the DVM.

When the DVM first starts up, a set of required plug-ins are automatically loaded (Figure 2). These are for communication, process control and resource management. Simple

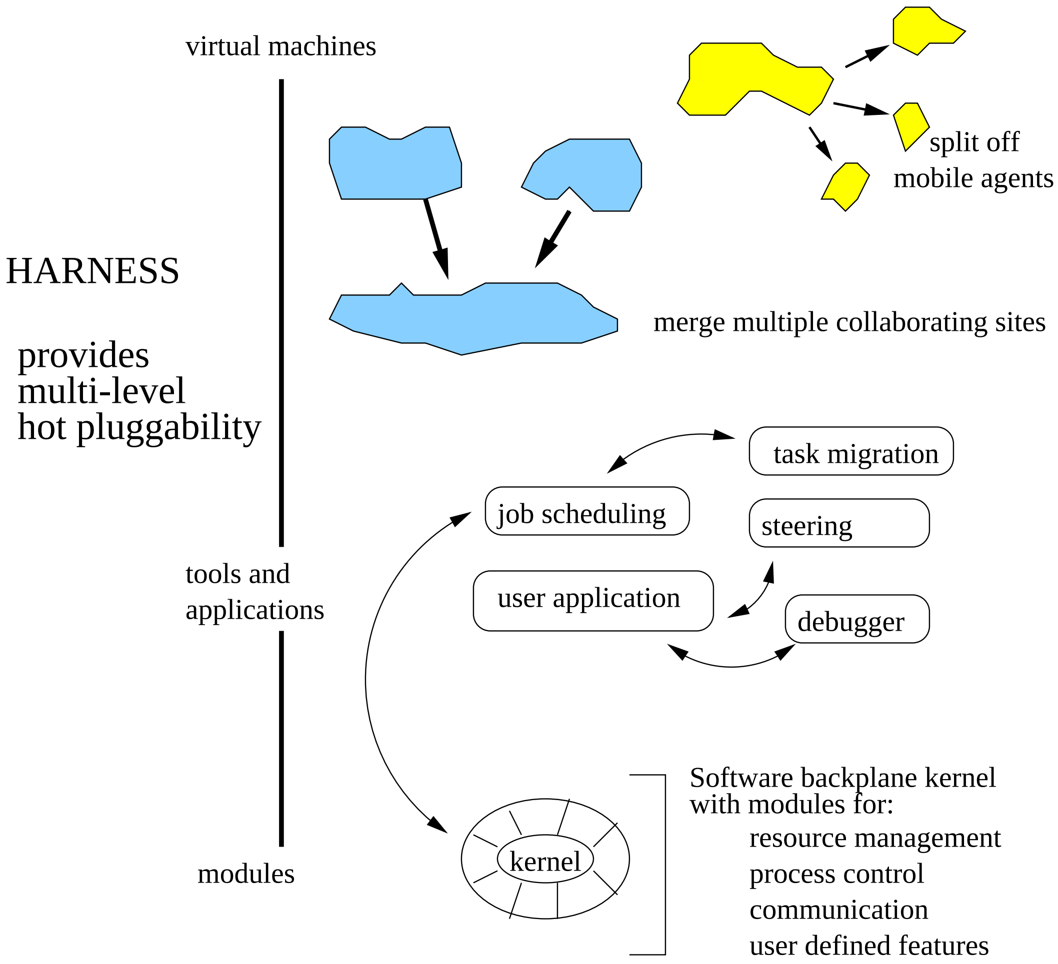

Figure 1: HARNESS allows dynamic plug-ins at module, tool, and virtual machine levels

versions of each of these will be developed and supplied with the initial prototype. The user can exchange any of the system-supplied plug-ins to customize the basic DVM features. The DVM can also dynamically load additional plug-ins to take on capabilities that are needed for a particular problem or mission.

HARNESS exhibits pluggability at three levels as illustrated in Figure 1.

3.1 Required kernel plug-ins

At the lowest plug-in level are HARNESS communications modules. The most basic service provided by a DVM is an abstract communication method among programs, tools, and virtual machine components. Depending upon the facilities required and the programming environment to be supported in a given incarnation of HARNESS, different communication plug-ins might be used. However, reliable, ordered, multi-way communication likely will be a minimal requisite of all kernel-level communication plug-ins, which will be required to deliver untyped messages to identifiable end-points within the executing DVM. Interfaces will be defined that categorize plug-ins so that HARNESS can (simultaneously) utilize several communication substrates such as TCP, UDP, Nexus, ST, and the Virtual Interface Architecture (VIA). By rigidly defining inquiry and service interfaces, the HARNESS kernel

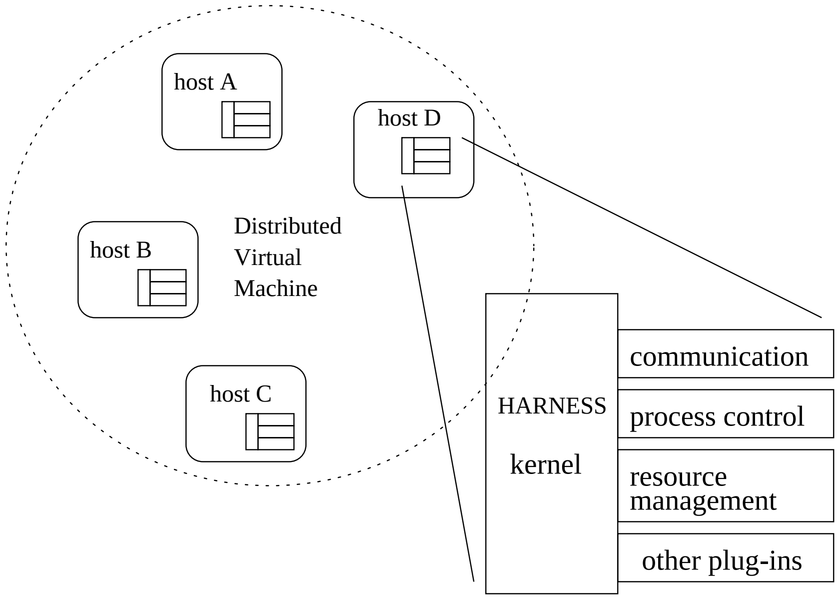

Figure 2: Distributed Virtual Machine is composed of kernels running on each computer and each kernel is composed of 3 required modules (plus possibly others)

can determine if requested plug-ins meet the requirements of modules that already exist in the protocol stack. The research challenges in this regard will be to evolve a methodology for the semantic definition of the interfaces that each plug-in will provide, in a manner that permits interchange and negotiation.

Layered on low level communication (but at a functionally equivalent level in the application interface) are the machine configuration and process control plug-in modules. For machine configuration, module functionality consists primarily of initialization functions and architecture reconciliation with the rest of the DVM. However, hooks will be provided to enable security modules for incorporating authentication and access control features. Our initial HARNESS module for resource management will provide a means to add and delete hosts, and to detect host failures within a single DVM. Additional functionality will be developed to add the capability of merging two DVMs based on direct user input, or based on a configuration file that specifies access restrictions.

Process management plug-ins will constitute the infrastructure for spawning application task units, and for naming and addressing tasks in the dynamic DVM. Process control modules are under development that provide functions for spawning and terminating groups of tasks across the DVM, using a simple algorithm for task placement. Alternative plug-ins, such as those provided by a computational grid environment, will use more sophisticated task placement algorithms designed to load balance the work across the DVM, and will also include support for basic resource usage accounting functions.

3.2 User-defined plug-ins

The real power of HARNESS will be derived from users themselves (or third parties) who will be able to construct plug-in modules to reconfigure DVMs with different capabilities (e.g., alternative load balancing schemes and situation-specific protocol stacks), or to create programming environments of their choice, such as distributed shared memory and tuple-space models. However, plug-in developers will have to keep in mind that, unlike a simple sequential plug-in as found in web browsers, a distributed computing plug-in can have wide-ranging effects on tasks that are scattered across the DVM. Distributed computing plug-ins, therefore, fall into a taxonomy classified by the type of coordination that must occur across the DVM while loading the plug-in:

- Module plugs into a single host (local daemon) and supplies features only to processes running on that host. In this category no coordination is required between the system of daemons that make up the DVM. An example of this class of plug-in is a local process startup module.
- Module plugs into a single host, but supplies features that affect a set of hosts or tasks across the DVM. Coordination for this class of plug-in requires at a minimum a broadcast and acknowledgment from affected entities. An example is a third party resource manager that wants to control the placement of tasks across the DVM to optimize utilization of the resources.
- Module plugs into a set of (possibly all) the hosts in the DVM, and affects tasks across the DVM. Coordination for this class of plug-ins requires careful synchronization between the components that make up the DVM. An example is the replacement of a low-level communications plug-in, perhaps in response to traffic pattern changes.

3.3 Plug-in Administration

One of the research areas is to understand and develop the necessary coordination algorithms for distributed computing plug-ins. Complicating factors are the need to make these algorithms efficient, fault tolerant, and robust across heterogeneous machines. If one puts aside the mechanics of how to dynamically load new code into an existing program (which is done successfully in the Linux kernel, Netscape Plug-ins, and Java applets), then finding, enabling, and using a plug-in (allowing two applications to plug together), or enabling two virtual machines to merge, all go through the following steps:

1. Naming/Lookup – finding the plug-in, application, or virtual machine.
2. Inquiry - determine whether the plug-in is compatible, the application supports particular message semantics, or whether two VMs can merge/split.
3. Instantiation – loading and starting the code, application, or VM operation.
4. Negotiation – provide the basic attachment semantics to enable communication.
5. Communication – move information or messages across the plug-together interface.
6. Shut Down – “unplugging” the connection.
7. Unload – stopping a program, clearing loaded code, stopping part of a virtual machine.

These steps may be elaborate, simple, or non-existent, depending on the particular plug-in. The critical thing to note is that whether one plugs in a low-level communication substrate, higher-level tuple-space message semantics, or entire virtual machines, the basic mechanism of enabling a plug-in is straightforward. What is not quite so simple, is exactly what it means, for example, for two applications to plug together and how the programs can benefit from this new capability. Again, the goal is to build a useful, dynamic, heterogeneous environment to explore fundamental issues of how users and programs can benefit from the dynamics that HARNESS enables.

3.4 Distributed Control and Arbitration

Unlike their physical counterparts, virtual machines require mechanisms to build, dismantle, monitor, and modify their current configuration. Static models, like those found in MPI implementations, have very simple virtual machine semantics: the DVM either exists or does not exist. However, even in the static model, mechanisms are required to detect faults so that the current virtual collection can be maintained or destroyed. A dynamic environment like PVM employs a model where individual hosts are either part of the virtual collection or they are not. However, the collection may change over time. The dynamic nature of these environments and of HARNESS means that some mechanism must be employed to keep track of membership. One difficulty with existing systems is the reliance on a single ultimate arbitrator for machine reconfiguration. For example, there is the *master pvmd* in PVM and the *legion class* object in Legion [7]. This asymmetric, almost client-server, configuration and control mechanism has a single point of failure, and can't address the issue of multiple DVMs (each with their own control) trying to merge together. HARNESS is designed around symmetric peer-based control to eliminate this single point of failure.

3.5 Extending Group Semantics to Multiple Virtual Machines

In addition to distributed membership algorithms, group semantics will be extended to virtual machine collections to add more expressive power for the management and control of HARNESS DVMs. Peer-based control helps in building a more robust environment but helps little in abstracting virtual collections to simplify merging and splitting of DVMs. HARNESS will use group operations as a unifying abstraction to manage the complexity of multiple tasks, hosts, and virtual machines. These will be the three fundamental entities that can be grouped. Current message passing libraries like PVM and MPI provide fundamental operations like broadcast, reduction, and synchronization, for groups of tasks. MPI goes a step further and defines set intersection, union and difference for groups of tasks. We will extend these ideas to the notion of hosts and virtual machines. Hosts can be grouped together to form a virtual machine set. Set operations on virtual machines will be defined so that the user can compose new virtual machines, or use set operations to split or modify a current configuration. Performing the set union of two DVMs to form a new one will be analogous to adding a single host to an existing virtual machine.

4 Summary

The HARNESS distributed virtual machine model presents a dynamic heterogeneous adaptable computing platform for the user. It can be used as a flexible and general purpose computing environment that is *shared* among multiple users, each of whom contributes a subset of the resources. Each host in the DVM can be dynamically configured using plug-ins for various communication protocols, resource management policies, and process control features. The cooperating set of users can coordinate custom features across a given DVM.

Users can reconfigure contributed resources portion as warranted by specific situations. Applications can "dock" with one another using the HARNESS plug-in infrastructure to enable cooperation like visualization and steering. HARNESS is focused on providing a framework where essential building-block components can be dynamically plugged together to form an *application-specific* operating environment. The architecture is modular from the low-level internal functionality to high-level merging and dividing of entire virtual machines.

References

[1] F. Berman, R. Wolski, S. Figeuria, J. Schopf, and G. Shoa. "Application-level Scheduling on Distributed Heterogeneous Networks." In *Proceedings Supercomputing '96*, Pittsburgh, PA, November 1996.

[2] J. Dongarra and E. Grosse. "Distribution of Mathematical Software Via Electronic Mail." *Comm. ACM*, 30(5):403–407, May 1987.

[3] I. Foster and C. Kesselman. "Globus: A metacomputing infrastructure toolkit." *Internat. J. Supercomputing Applic.*, 11(2):115–128, May 1997.

[4] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, B. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine A User's Guide and Tutorial for Networked Parallel Computing.* MIT Press, Cambridge, MA, 1994.

[5] A. Geist, J. Kohl, R. Manchek, and P. Papadopoulos. "New Features of PVM 3.4 and Beyond." In Dongarra, Gengler, Tourancheau, and Vigouroux, editors, *EuroPVM'95*, pages 1–10. Hermes Press, Paris, 1995.

[6] P. Gray and V. Sunderam. *The IceT Project: An Environment for Cooperative Distributed Computing*, 1997. (http://www.mathcs.emory.edu/ gray/IceT.ps).

[7] A. Grimshaw, W. Wulf, J. French, A. Weaver, and P. Reynolds. "Legion: The Next Logical Step Toward a Nationwide Virtual Computer." Technical Report CS-94-21, University of Virginia, 1994.

[8] J. Kohl, P. Papadopoulos, and A. Geist. "CUMULVS: Collaborative Infrastructure for Developing Distributed Simulations." In *Proc. Eigth SIAM Conf. on Par. Proc. and Sci. Comp.*, Minneapolis, MN, March 1997.

[9] B. Miller, M. Callaghan, J. Cargille, J. Hollingsworth, B. Irvin, K. Karavanic, K. Kunchithapadam, and T. Newhall. "The Paradyn Parallel Performance Measurement Tool." *IEEE Computer*, 28(11):37–46, November 1995.

[10] J. Pruyne and M. Livny. "Parallel Processing on Dynamic Resources with Carmi." In *Proc. Workshop on Job Scheduling Strategies, IPPS'95* , April 1995.

[11] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference.* MIT Press, Cambridge, MA, 1996.

[12] Emory University. *Collaborative Computing Frameworks for Natural Sciences Research*, 1997. (http://wwwccf.mathcs.edu/ccf/overview.ps).