Pervasive Debugging

A Fresh Approach to Understanding Distributed Applications

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

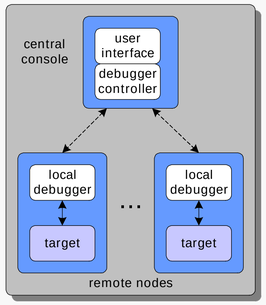
Alex Ho
alex.ho@cl.cam.ac.uk

http://www.cl.cam.ac.uk/netos/pdb/

The Problem

Understanding and correcting the behavior of distributed applications is hard. It is difficult to control processes spread out over multiple nodes.

Existing solutions can be characterized as *peer debugging* and utilize a conventional architecture: "just a bunch of debuggers".

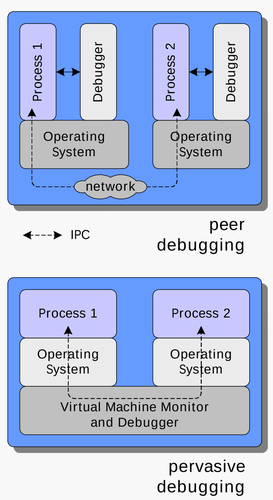


"Just a Bunch of Debuggers"

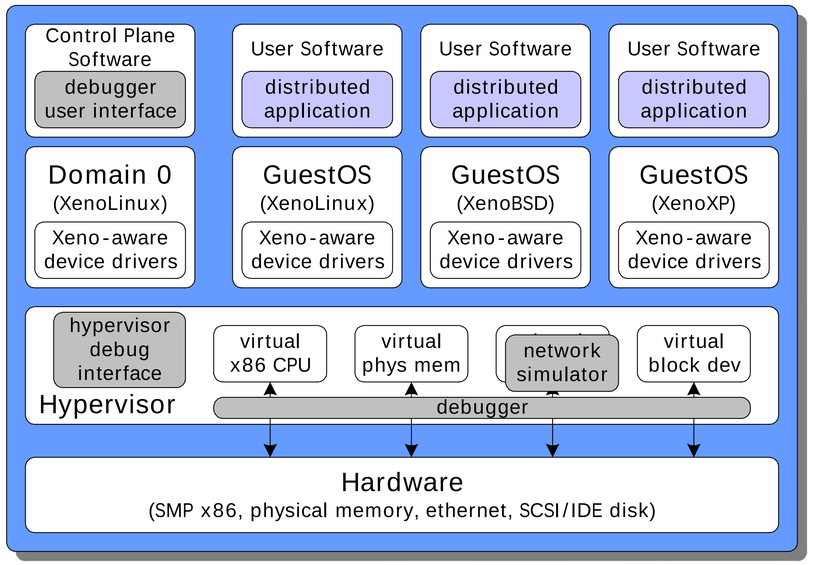
- An invasive debugger, monitor, or additional thread runs on each node.
- A central coordinator messages each node over the network. Unpredictable communication delays make synchronous operations impossible.
- It is impossible to stop the computation atomically on each node.

Our Approach

Pervasive debugging maps the entire distributed computation onto a single virtual machine monitor.



- Each node runs in a separate virtual machine.
- No changes are required to the application, and no custom libraries are needed.
- Any network topology between the nodes can be enabled with a network simulator in the virtual machine monitor.



System Design

The system leverages the Xen hypervisor (virtual machine monitor) from the XenoServers project. Multiple operating systems execute concurrently, each in a protected domain. User applications run unmodified within each guest operating system.

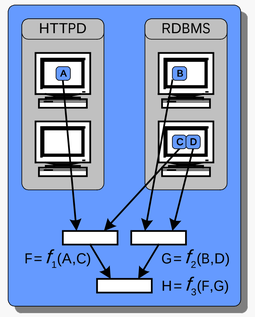
- Debug functionality is embedded within the hypervisor.
- A user-space debugger communicates via a hypervisor debug interface.

Benefits

- It is possible to view the entire computation in a consistent state. There is no need for a distributed snapshot algorithm.
- The pervasive debugger controls the entire execution environment. User processes, the operating system, application libraries, system resources (disk or network), and their interactions can be debugged.

Distributed Event Detection

- User breakpoints and software exception trigger dataflow primitive events. Event triggers include processes' state (stack, registers, etc) and inter-process communication.
- Primitive events can be arbitrarily combined to form high-level events that represent application actions.
- A language for recognizing complex event patterns that supports "near miss" matches and not just simple pattern matching is used.
- Active event processing extends passive event recognition with the ability to change the state of the system.



Fault Injection

- Hardware faults such as memory bit errors; node, disk, and network failures can be simulated.
- Software faults can be introduced at various levels: from random memory page writes to process failure to programmer errors.

References

- T. Harris, "Dependable Computing Needs Pervasive Debugging",Proceedings of the 2002 ACM SIGOPS European Workshop, September 2002.
- P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization", Proceedings of the 19th ACM Symposium on Operating Systems Principles, October 2003.