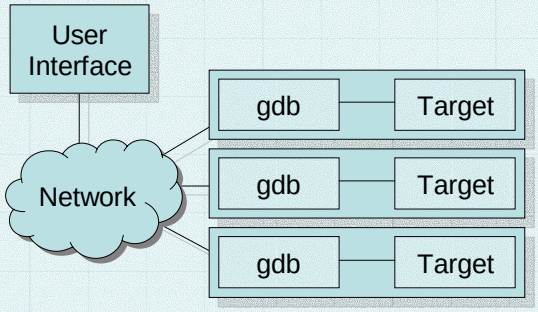
Pervasive Debugging

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

The Problem

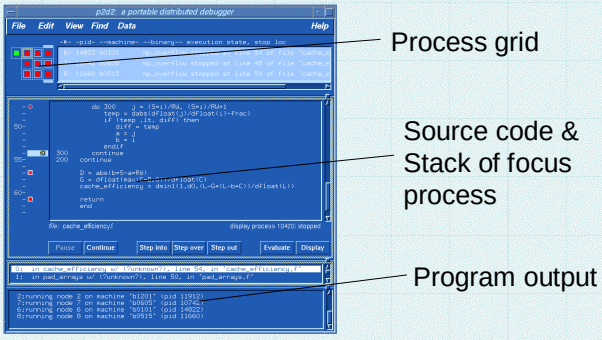
Understanding and correcting the behaviour of parallel and distributed applications is hard. It is difficult to control the processes spread out over multiple nodes. It is difficult to investigate the operating system kernels, network links and other devices which are used.

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



A traditional debugger or monitor runs on each node

- A central console issues command to them.
- Features are limited by the back-end debuggers.
- This approach is taken in popular tools such as p2d2.



These systems can provide good data visualization, consistent user interfaces between machines, abstraction over PVM, MPI etc.

They do not provide:

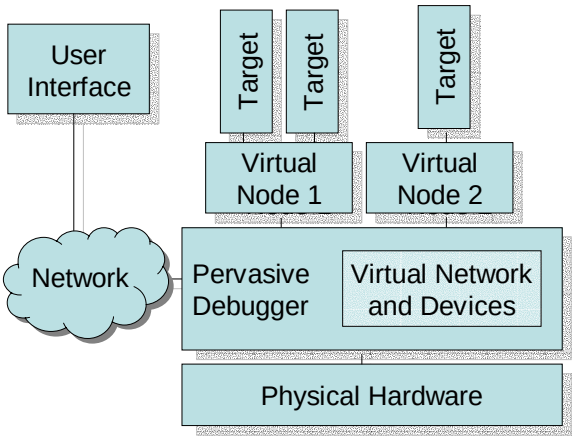
- Multi-node breakpoints.
  "Stop if more than one node thinks it is the leader."
- Network-related or device-related breakpoints.
  "Stop if a packet is dropped."
- Atomic suspension of entire systems.
- Deterministic re-execution.
- Debugging across multiple layers of abstraction.
- Evaluation of performance outside the physical system's parameters.

Our Approach

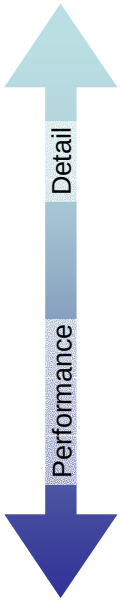
Pervasive debugging conceptually places the debugger below the entire distributed system. The debugger genuinely operates at a whole-application level rather than a per-node level:

- The entire system can be stopped atomically.
- Breakpoints can relate to multi-node conditions.
- An arbitrary inter-node network can be emulated.

This is complementary to existing tools: where appropriate re-use their visualization features and control interfaces.



The entire system executes in a virtual environment that is under the control of the debugger. A range of implementation techniques are possible with different trade-offs:



Run all of the virtual nodes within a single hardware-level or instruction-level simulator, e.g. Bochs. Suitable for small systems with fine-grained concurrency – e.g. mutex or work-queue implementations.

Run the virtual nodes over a high-performance virtual machine monitor, e.g. Xen.

Explore the federation of multiple virtual machine monitors on separate physical machines and the use of timewarp-simulation style execution.

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

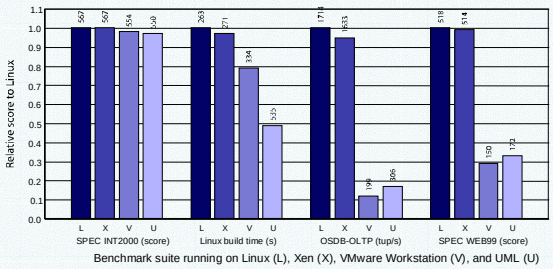
Steven Hand, Tim Harris, Alex Ho, Rashid Mehmood
{firstname.lastname}@cl.cam.ac.uk

Implementation

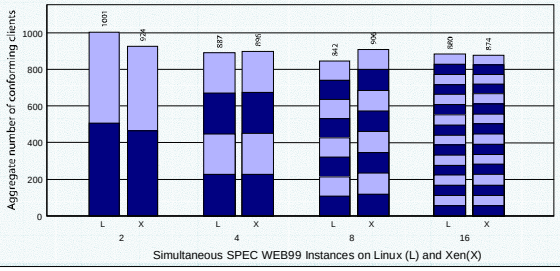
We're building a prototype pervasive debugger using the Xen virtual machine monitor (developed under the Programmable Networks programme).

- z/VM-style virtualization on an uncooperative x86 architecture.
- Support full-featured multi-user multi-application OSes.
- OSes are *ported* to a new 'x86-xeno' architecture.
  - Similar to x86, but call to Xen for privileged operations.
  - Most kernel code executes directly.
  - Porting requires access to kernel source code.
  - Exposing real resources important for correctness and performance.
- *Retain compatibility* with OS ABIs.
  - All application code & libraries execute directly and without recompilation.
  - Run unmodified Linux, Windows XP and BSD application binaries and libraries.

Performance of Xen versus native execution



Scalability of Xen hosting multiple virtual nodes



Current directions

Our research is broadly split into two categories.

The first is developing implementation methods for high performance pervasive debugging. The second is exploring the new techniques enabled by pervasive debugging.

Integration with existing tools and resources from existing serial and distributed debuggers

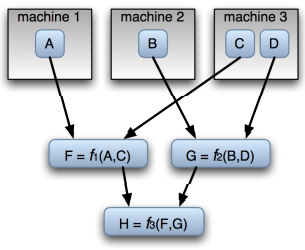
- To enable re-use of user-interfaces from p2d2 while executing the system under test within a fully controllable environment.
- To enable controlled communication outside the virtual environment, e.g. access to name services.

Network virtualization

- Description of the topology and behaviour of the network connecting the virtual nodes.
- Validation of behaviour observed within the pervasive debugger against behaviour in real systems.

System-wide assertions

- Definition of primitive events generated within each node (e.g. temporal events such as breakpoints, or spatial events such as watchpoints triggered by changes to shared memory or disk storage).



- Event-composition to build up system-level events.

Stress testing and fault injection

- The virtual network could introduce extreme load or packet-reordering: allows testing of corner-cases rarely seen in practice.

Live debugging

- There's actually no need to suspend the virtual nodes while the debugger is attached
- Can use the debug interface for general visualization and inspection during execution.

Federating execution over multiple physical machines

- The controllable environment provided by Xen should allow timewarp-simulation style execution.
- Speculatively run nodes in parallel.
- Use checkpointing and re-execution to deal with causality problems.

References

- T. Harris, "Dependable Computing Needs Pervasive Debugging",Proceedings of the 2002 ACM SIGOPS European Workshop, September 2002.
- A. Ho, "Pervasive Debugging: A Fresh Approach to Understanding Distributed Applications", 8th Cabernet Radicals Workshop, October 2003.
- 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.