A case for Virtual Machine based Fault Injection in a High-Performance Computing Environment

Thomas Naughton, Geoffroy Vallée,
Christian Engelmann and Stephen L. Scott

Oak Ridge National Laboratory
Computer Science and Mathematics Division
Oak Ridge, Tennessee, USA

HPCVirt 2011 *Bordeaux, France* *August 29, 2011*

HPC machines are large systems

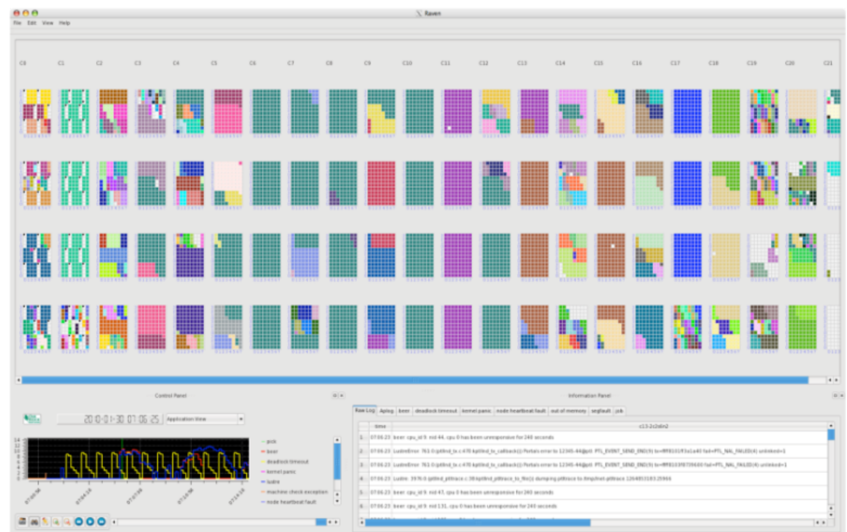
Resource	Size
Compute Nodes	18,688
Compute Cores	224,256
Total Memory	300TB
Interconnect Peak Bandwidth (SeaStar2+)	57.6GB/s
Peak Performance	2.3 pflops/s

"*Jaguar*" – Cray XT5 at ORNL

* Image courtesy of the National Center for Computational Sciences, Oak Ridge National Laboratory.

Scalability challenges

- Large-scale system raise many challenges
  - Performance of applications & system software
  - Complex resource usage/interaction patterns
  - Growing failure rates due to huge component counts



Screen capture from "*Raven*" log analysis tool
Contact: Hoony Park (ORNL)

Failures at Scale

- Failures in current petascale and future exascale
  - Occurrence of failure is much more common
    - *"Failure rates vary widely across systems … and depend mostly on system size and less on the type of hardware."*

      Schroeder & Gibson [DSN2006]

    - *"In some cases, the overall system mean time between failure (SMTBF) is under two hours."*
    - *Previous work similarly suggests a system mean time to failure (SMTTF) constraint of 5-6 hours, or 4 failures per day, for current HEC systems [TaeratHAPCW2008]."*

      Whitepaper by DeBardeleben et al. [2009]

HPC Fault-Tolerance/Resilience

- Scalability driving new research in FT/R
  - Log analysis to identify types/modes of failure
  - Fault & recovery coordination frameworks (e.g., CIFTS)
  - Enhancements to MPI (e.g., MPI3-FT WG)
- New mechanisms for HPC applications
  - Algorithm based FT (e.g., FTLA)
  - Advanced checkpoint/restart techniques (e.g., stdchk)
  - Modular redundant MPI (e.g., MrMPI, rMPI)

Miles to go before we sleep…

- Applications to use available FT/R capabilities
  - Still preliminary and much work to be done
- Infrastructure to provide FT/R capabilities
  - Still lots of work to cope with failures at scale
- Tools for experimentation
  - Need tools to support this development & testing

Introduction

- Large-scale computing platforms
  - Increased size & performance
  - Increased complexity
    - Usability: More work & effort to use system (scientist & admin)
    - Resilience: More failures
- Dealing with complexity
  - *System-level virtualization* – assist users & admins
  - *Fault-tolerance/Resilience* – cope with failures

Motivation for Tools

- Study failures in large-scale systems
  - Identify faults (origins of failure)
  - Controlled environments for experimentation (testbed)
- Explore ways to deal with failure
  - Which technique? (mechanism)
  - How to use mechanisms? (policy)
  - What level of the software stack? (effectiveness)

Fault Injection Tools

- Provide tools to support FT/R experimentation
  - Monitoring & logging
  - Distributed task control
  - Fault injectors, etc.
- Provide environment for controlled tests
  - Experiment config/setup
  - Startup and execution

General: Constraints on Tools

- Operate in HPC environments
  - Low-overhead (high-performance) requirements
  - Resource managers and batch allocation systems
- Highly specialized platforms
  - Customized execution environment
  - Tool chain tailored for platform (even if it is “Linux”)
  - Specialized hardware (and software)
- Result
  - This limits use of some existing FI tools

Terminology

- Fault Injection (FI)
  - Purposeful introduction of faults (errors) into target
  - SWIFI: Software Implemented Fault Injection
  - SUT: System Under Test ("target")
- Fault, Error & Failure [Laprie Taxonomy, DSC'04]
  - Fault: defect in a service, may be "active" or "dormant"
  - Error: an "active fault" in a service
  - Failure: unsuppressed error, visible outside the service
- Virtualization
  - VMM: virtual machine monitor (aka *hypervisor*)
  - VM: virtual machine
  - HostOS: operating system run on physical machine
  - GuestOS: operating system run in virtual machine

Fault Injection

- Existing work
  - Techniques: Environmental, Hardware, Software
  - Widely used to test FT mechanisms
  - Lots published, few general/publicly available tools
- Important points
  - Representative failures
  - Representative system (e.g., hardware vs. model-based)
  - Transparency & (low) overhead
  - Detector / Injector pairing
  - Placement & triggering

Where to inject?

- Key challenge for FI experiments
  - Identify “good” target locations
  - Source code driven
  - Runtime usage driven
  - Random
  - Isolate target to avoid mistaken outcomes
    - Clobber FI infrastructure (“self”)
    - Application code vs. linked library (MD vs. libmpi)
- Accounting
  - Record where/when injection took place
  - Record injection events in non-volitile (safe) region

Virtualization

- Virtual Machines
  - Commonly used in testing/development
  - Offer consistent execution environment
  - Provide strong isolation capabilities
- Virtualization for HPC
  - Prior work on Virtual System Environments (VSE)
  - Embeddable hypervisor for HPC (V3VEE/Palacios)

Virtualization: Advantages for FI

- Customizable
  - User can build application as appropriate in VM
  - VMM has access to virtualized hardware of VM
    - Full access to memory & other resources used by VM
- Isolation
  - Separation of SUT and FI infrastructure
- General
  - VM pause/resume, “snap shots”
  - Can over-subscribe resources to simulate more nodes
  - VM offers good system representativeness

Virtualization: Advantages for FI (2)

- Experiment management & packaging
  - VM aids in creating reproducible experiments, and configuration archiving
  - Reuse VM image with FT/R support for different apps
- Other: Future
  - Emulate hardware not available on local/current machine
  - Record/replay VM capabilities for repeatable exp.

Example FI: User-space approach

- Developed memory corruptor
  - Injector based on ptrace()
  - Random address in dynamic memory (heap) region
- Target application
  - LAMMPS molecular dynamics code
  - Inject bit-flips into memory of single MPI rank
- Comments
  - Focus of experiment was on application self-monitoring
  - Con: less representative of bit-flips in real system (e.g., ECC) and not usable for OS-level injections

Ex.: Changes for a VM approach

- Change memory corruptor
  - Injector based on memory access (trigger inject)
  - Target random address in dynamic memory (heap)
  - Could also target recently used memory locations
    - Easier to access this information from system level approach
- Target application (same)
  - LAMMPS molecular dynamics code
  - Inject bit-flips into memory of single MPI rank
- Comments
  - Pro: more representative of bit-flips in real system (ECC)

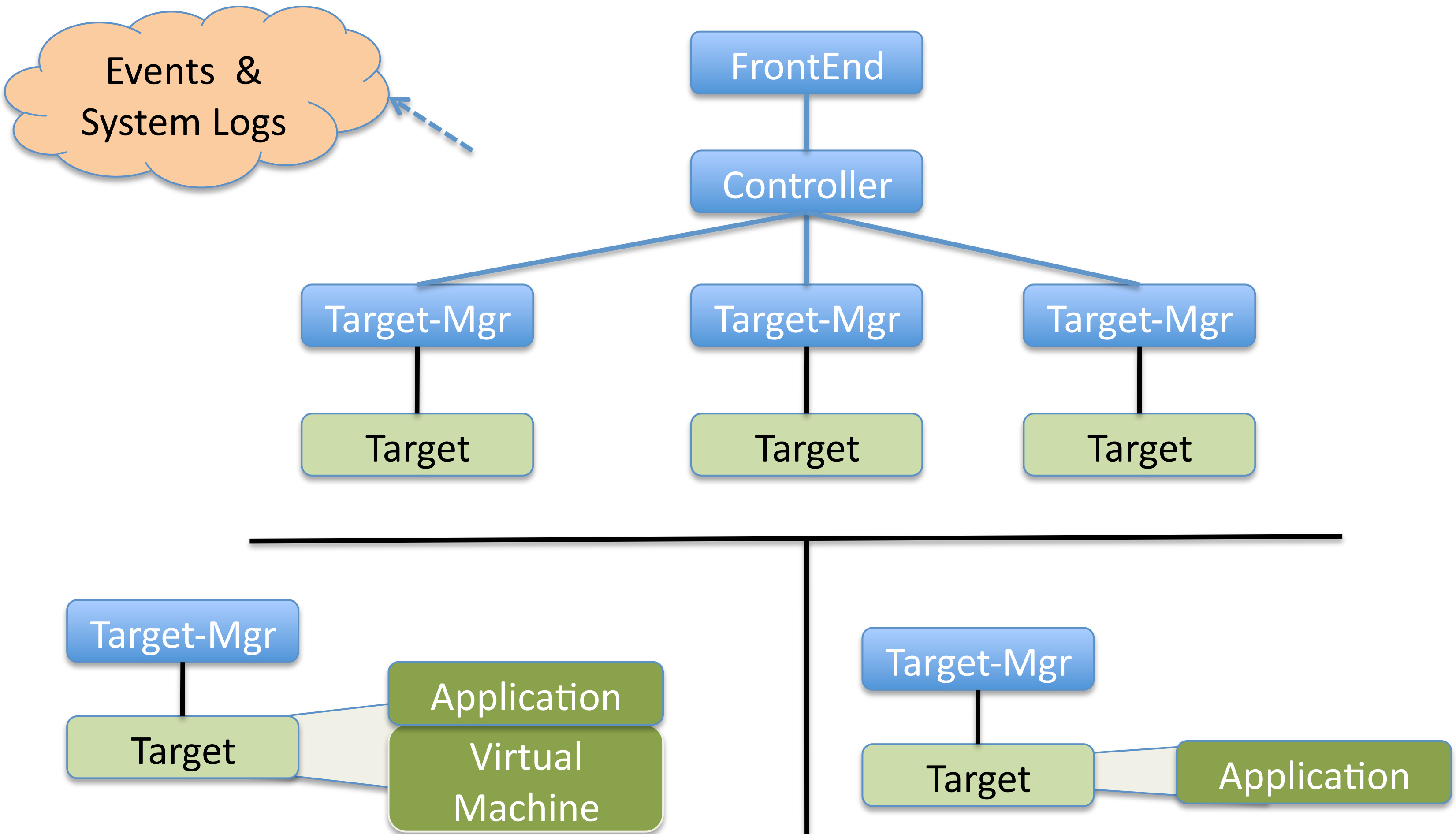
Moving Forward

- Develop fault injection framework for HPC context
  - Tools to simplify failure experiments
- Leverage prior & current work
  - *CIFTS*: Event pub/sub & log analysis tools
  - *STCI*: Distributed runtime control & basic communication
  - *XVirt*: Hypervisor for HPC
  - *VSE*: VM management tools & user environment

Structure for Experiments

- Two parts to an experiment Target
  - Target Manager: local experiment setup/control
  - Target: Victim "application"
- Experiment Event
  - Type
    - Ex. Memory bit-flip
  - Trigger Mode
    - Ex. Trigger on command, on timer, upon access

Basic Structure



(a) Target is Application running in VM

(b) Target is stand-alone Application

Evaluation

- Gather basic statistics
  - Target crash, hang
  - Number of injections
  - Number of detections
- Generate summary reports
  - MTTF for given target/experiment
- Dependability Benchmarking
  - Look to projects like DBench, AMBER, etc.

Some Open Questions

- What is right representation for experiments?
  - Express different types of faults/errors for HPC
- How to provide intuitive "target location info"
  - Usable by end-user and sufficient for backend
  - Also relevant when providing feedback to user and doing post-mortem analysis (mapping)

Related Work

- Xception
  - Leveraged hardware debug/perf. monitoring capabilities
- FIG
  - Errors via shared library interposition (LD_PRELOAD)
- NFTAPE
  - Component-based SWIFI for distributed environments
- Linux-FI
  - In Linux kernel >= v2.6.20
  - For select areas of memory & IO subsystems

Related Work (cont.)

- FAUmachine
  - Simulated faults in a user-space process (like UML)
  - Experiments included HDL perspective
- FI-QEMU
  - Patch to QEMU process emulator for ARM architecture
- Gigan
  - Additions to Xen for virtual machine fault injection
  - Focus was not on HPC
  - Simulating distributed environments on single node

Summary

- Large-scale HPC systems
  - Increased complexity
  - Many resilience challenges
  - Driving research in Fault-Tolerance/Resilience
- Tools for FT/R Experimentation
  - Clear need for ways to test & evaluate techniques
  - Fault injection is widely used to test FT mechanisms
  - Provide fault injection tools for HPC environments
  - Leverage work in HPC virtualization for FI tools

Questions?

- Thank you, and enjoy the conference

ORNL's work was supported by the U.S. Department of Energy, under Contract DE-AC05-00OR22725.