A Runtime Environment for Supporting Research in Resilient HPC System Software & Tools

Geoffroy Vallee, Thomas Naughton, Swen Böhm, Christian Engelmann
Motivation & Challenges

• High performance computing trends
  – Bigger machines (e.g., TITAN, upcoming exascale systems)
  – More complex architectures (e.g., heterogeneous compute nodes)
  – More failures

• Runtime environment (RTE) is a crucial software component
  – Interface between the operating system and scientific simulation
  – Manage the lifecycle of the scientific simulation

Is it possible to provide building blocks for the study and development of new RTEs?
Scalable RunTime Component Infrastructure – STCI

• Goals
  – Scalable start-up and management of scientific simulations
  – Resilience/fault tolerance
  – Ease the study and development of new system tools and/or applications for HPC

• Key characteristics
  – Modular architecture
  – Provide reusable components

• Illustration with 2 use cases
  – Alternate MPI runtime
  – New fault injection tool
STCI Architecture

• Agents
  – Instantiate both the STCI infrastructure and applications/tools
  – Different “types” of agents
    • *Frontend*: user frontend running on user’s terminal
    • *Controller*: logical agent representing the job from a control point of view
    • *Root agent*: privileged agent for resource allocation; one per node; non-specific to a job
    • *Session agent*: local management of users’ job; one per user and per node
    • *Tool agent*: instantiation of an application or a tool

• Topologies
  – Represent connections between agents
  – Examples: trees, meshes, binomial graphs
STCI Architecture (2)

• Launcher
  – Deploy a job by creating the necessary agents across the HPC platform
  – Two challenges
    • Scalable deployment method: by default, a tree-based topology
    • Method to create the required agents
      – Example: fork, ssh, ALPS
      – On Cray:
        » Torque gives the list of target compute nodes
        » ALPS is used to create the RAs
        » then RAs create other agents

• Event system
  – Support for asynchronous execution model
  – Various progress models available: implicit or explicit progress
**STCI Architecture (3)**

- Two communication substrates
  - One dedicated to bootstrapping
  - One for the implementations of parallel/distributed services

**Bootstrapping communication substrate**

- Requirements
  - Self-bootstrapping
  - Reliable and ordered communications
  - Support sparse connectivity
  - Support fine-grain monitoring of all communication links (agents may be volatile)
  - Support asynchronous communications
- Currently based on a tree topology
STCI Architecture (4)

- Active Message (AM) communication substrate
  - Requirements
  - Reliable communications
  - Blocking/non-blocking send
  - Avoid data copies
  - Sparse connectivity
  - Asynchronous communications
- 3 different AM APIs with different levels of abstraction
  1. Point-to-point, non-routed fragment-based communications
  2. Point-to-point, routed message-based communications
  3. Stream based (based on a topology), routed message-based communications
STCI Architecture (5)

• Fault tolerance
  – Failure detection
    • Inter-node: e.g., mesh topology between compute nodes
    • Intra-node: e.g., signal based detector
  – Fault tolerant topology
    • Topology that tolerates the failure of one or more agent
    • Ex: binominal graph (BMG) based topology providing redundant communication links
  – Failure notification
    • Propagate any local notification from detectors
    • Abstract how the propagation is implemented (ex: broadcast notification, tree-based fan-in/fan-out)
  – Error manager
    • Implement the consensus policy for failure recovery (ex: terminate on failure)
    • Local and global recovery managers
Use Case – Alternate Runtime for MPI

• Based on Open-MPI
  – Replace the default runtime (ORTE)
  – Benefit the RTE abstraction in Open-MPI
    • Out-of-band communications
    • Naming service
  – RTE mainly used for the deployment of MPI ranks
    • STCI communication substrates used during bootstrapping
    • Open-MPI high-performance communication substrates once bootstrapping completed

• Used for the implementation of the fault tolerant MPI prototype
  – Ongoing MPI-3.x standardization
  – Focusing on user-level failure mitigation (ULFM)
Use Case – Fault Injection Tool

• Goal
  – Study the impact of faults
  – Validate mitigation mechanisms

• Development of a new tool
  – Specialized frontend and distributed control
  – Experiment setup/management
  – Monitoring and event logging
  – Fault injection mechanisms, e.g., process kill for process fail-stop
Use Case – Fault Injection Tool (2)

• Users provide a description of the experiment via the frontend
• Session agents implement the target manager, which will apply a fault injection mechanism on the target application
Conclusion

• STCI provides a modular architecture that
  – Tolerates failures at the infrastructure level
    • give users the opportunity to be notified
    • Let users decide the appropriate actions
  – Minimizes the bootstrapping phase during which the entire infrastructure is at risk
  – Eases the design and implementation of HPC tools
  – Provides all the building blocks for supporting research in resilience

• Used at ORNL to enable research related to resilience
  – MPI Fault tolerance Working group – ULFM
  – Resilience tool for HPC via fault injection
Acknowledgment

- Individuals that contributed to the STCI project, including Richard Graham, Wesley Bland, Joshua Hursey, Christos Kartsaklis, Rainer Keller, Gregory Koenig, Pavel Shamis and Chao Wang.

- This research is sponsored by the Office of Advanced Scientific Computing Research; U.S. Department of Energy and performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 and used resources of the Center for Computational Sciences at Oak Ridge National Laboratory.