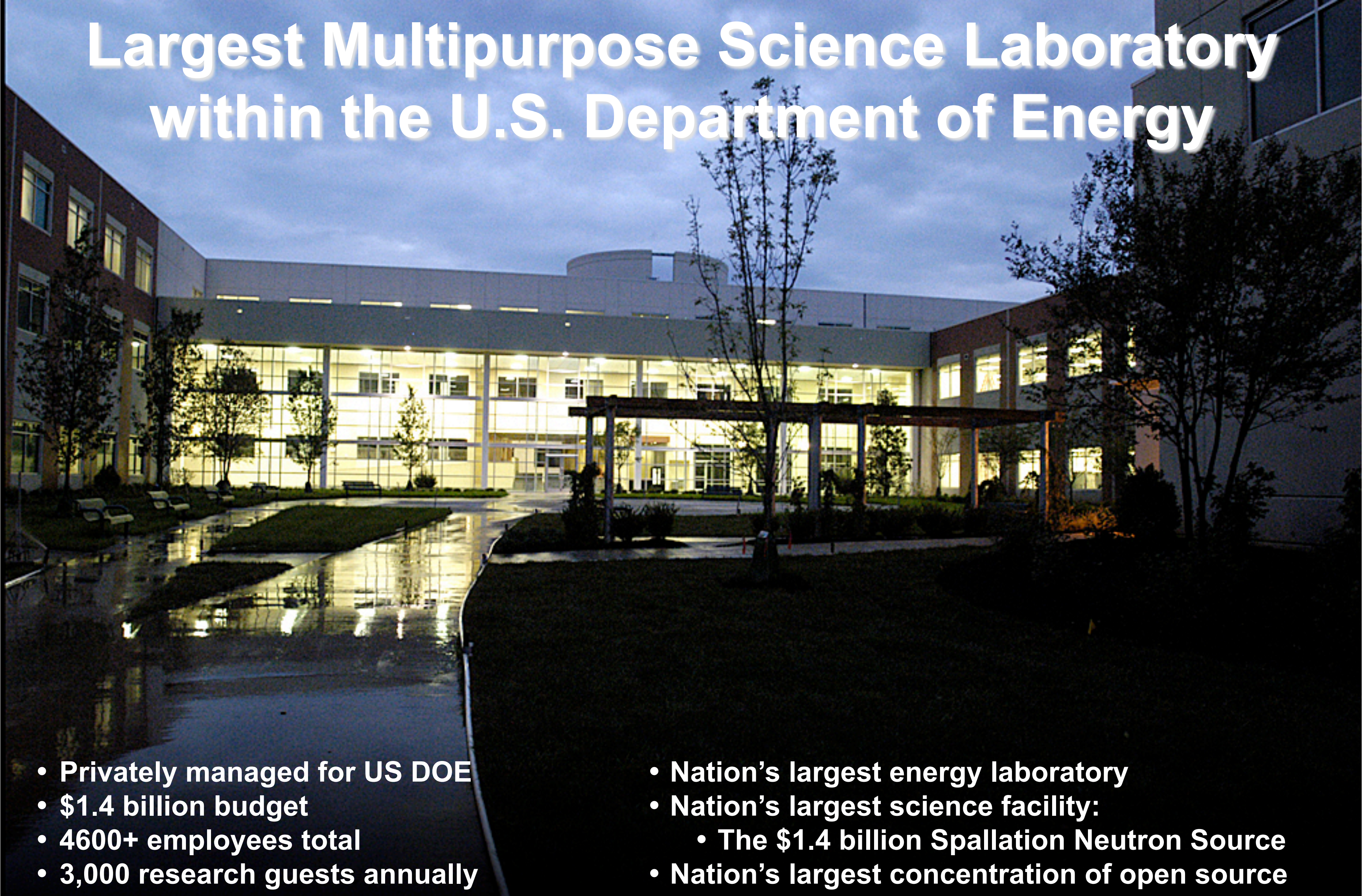
High-Performance Computing Research Internship and Appointment Opportunities at Oak Ridge National Laboratory

Dr. Christian Engelmann

Computer Science and Mathematics Division
Oak Ridge National Laboratory

Largest Multipurpose Science Laboratory within the U.S. Department of Energy

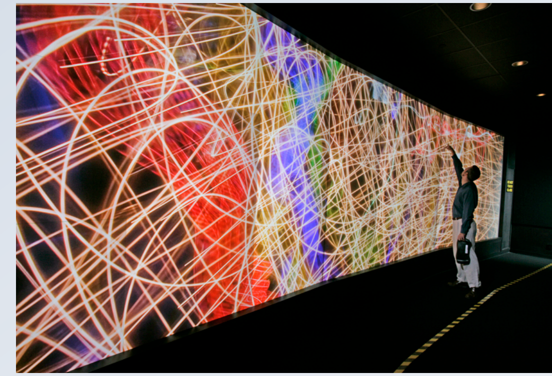


- Privately managed for US DOE
- $1.4 billion budget
- 4600+ employees total
- 3,000 research guests annually
- 30,000 visitors each year
- Total land area 58mi^2 (150km^2)

- Nation's largest energy laboratory
- Nation's largest science facility:
  - The $1.4 billion Spallation Neutron Source
- Nation's largest concentration of open source materials research
- Nation's largest scientific computing facility

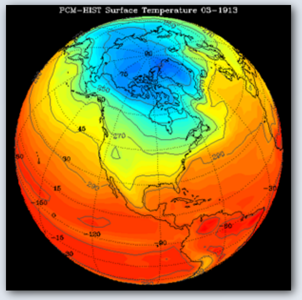
ORNL East Campus: Site of World Leading Computing and Computational Sciences

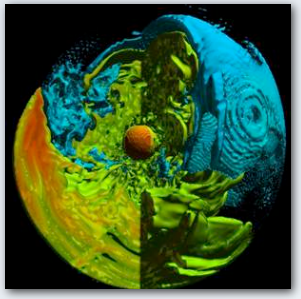
National Center for Computational Sciences

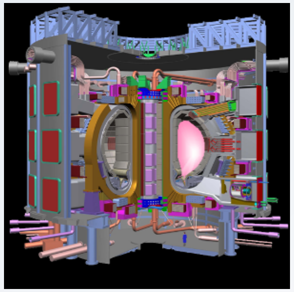


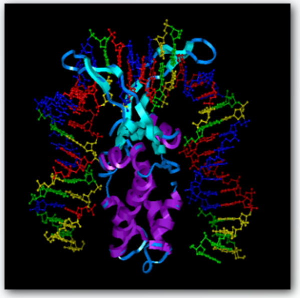
- 40,000 ft^2 (3700 m^2) computer center:
  - 36-in (~1 m) raised floor, 18 ft (5.5 m) deck-to-deck
  - 36 MW of power with 6,600 t of redundant cooling
  - High-ceiling area for visualization lab: 35 MPixel PowerWall
- 5 systems in the Top 500 List of Supercomputer Sites:
  - 1. Jaguar XT5: Cray XT5, with 224,162 processor cores at 2,331 TFlop/s peak
  - 3. Kraken: Cray XT5, with 98,928 processor cores at 1,028 TFlop/s peak
  - 16. Jaguar XT4: Cray XT4, with 30,976 processor cores at 260 TFlop/s peak
  - 30. Athena: Cray XT4, with 17,956 processor cores at 165 TFlop/s peak
  - 370. Eugene: IBM BGP, with 8,192 processor cores at 28 TFlop/s peak

At Forefront in Scientific Computing and Simulation





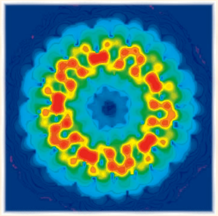


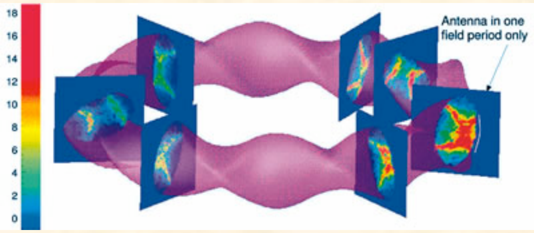


- **Leading partnership in developing the National Leadership Computing Facility**
  - **Leadership-class scientific computing capability**
  - **Currently planning for 10-20 PFlop/s in 2012**
  - **On the path toward:**
    - **100 PFlop/s in 2015 (10- 100 million cores)**
    - **1,000 PFlop/s in 2018 (100-1,000 million cores)**
- **Attacking key computational challenges**
  - **Climate change**
  - **Nuclear astrophysics**
  - **Fusion energy**
  - **Materials sciences**
  - **Biology**
- **Providing access to computational resources through high-speed networking**

Computer Science Research Groups

- Computer Science and Mathematics (CSM) Division.
  - Applied research focused on computational sciences, intelligent systems, and information technologies.
- CSM Research Groups:
  - Climate Dynamics
  - Complex Systems
  - Computational Chemical Sciences
  - Computational Materials Science
  - Future Technologies
  - Statistics and Data Science
  - Computational Mathematics
  - Computer Science Research (23 researchers & postdocs)





Computer Science Research Group Projects

- Parallel Virtual Machine (PVM)
- MPI Specification, FT-MPI and Open MPI
- Common Component Architecture (CCA)
- Open Source Cluster Application Resources (OSCAR)
- Scalable cluster tools (C3)
- Scalable Systems Software (SSS)
- Fault-tolerant metacomputing (HARNESS)
- High availability and resilience (RAS, FAST-OS 1 & 2)
- Super-scalable algorithms research
- Distributed file and storage systems (Freeloader)

MSc Internship Basics

- 1-2 students (max. 4) for max. 6 months at Oak Ridge National Laboratory in Oak Ridge, Tennessee, USA
- Full-time (40 hours/5 days per week) internship supervised by a research staff member
- Individual leading-edge projects that include background investigation, design, and development
- Includes MSc thesis and draft research paper write-up as part of the final MSc project
- $1500 per month stipend plus travel costs depending on student qualifications
- Subcontracts through the University of Tennessee, Knoxville, USA

MSc Internship Timeline (Spring)

- Early Dec.: Application process
  Specify area of interest/project
  Submit resume/CV to Vassil
- Dec./Jan.: Acceptance notification
  Background Check/Subcontracts
  J-1 (Student) Visa application
- February: Visa issued through U.S. Embassy
- 1. March: Start of internship
- 31. August: End of internship
- September: Presentation at the University of Reading

MSc Internship Timeline (Fall)

• Early June:	Application process Specify area of interest/project Submit resume/CV to Vassil
• Mid June:	Acceptance notification Background check/subcontracts J-1 (Student) visa application
• August:	Visa issued through U.S. Embassy
• 1. September:	Start of internship
• 28. February:	End of internship
• March:	Presentation at the University of Reading

Further Practical Information

- **Driver license is a must: No public transport to work.**
- **\$3500 (2500€) in initial min. funds needed for:**
  - **First rent and various deposits**
  - **One-week car rental (reimbursed afterwards)**
    - **Under 25? Car rental & insurance is more expensive**
  - **Used car, car sales tax, registration, and insurance**
- **Break-even point:**
  - **1 student after 4-5 months, 2 students after 2-3 months**
  - **Most students leave with a net plus despite extra expenses for: high-speed Internet, cable TV, and weekend trips**

Possible Projects (see next slides for details)

- Proactive fault-tolerance
  - Extending the scalable monitoring data aggregation system
  - Integration with the existing fault tolerance framework
- ADDAPT (successor of Harness Workbench)
  - Development of an scientific application execution assistant
  - Development of plug-ins for: job & resource management, data staging tools and/or workflow engines
- IAA simulator
  - Adding enhancements to simulate time-accurate application runs on millions of processors with fault tolerance tests
- Soft Error Resilience
  - Developing diskless checkpoint caching, diskless checkpointng or modular redundancy prototypes

Proactive Fault Tolerance Using Preemptive Migration

Christian Engelmann

Computer Science and Mathematics Division
Oak Ridge National Laboratory

Motivation

- Large-scale PFlop/s systems have arrived:
  - #1: ORNL Jaguar with 224,162 processor cores
  - #2: LANL Roadrunner with 129,600 processor cores
- Other large-scale systems exist
  - LLNL @ 212,992, ANL @ 163,840, TACC @ 62,976
- The trend is toward larger-scale systems
  - Up to 1,000,000,000 cores in the next 10 years
- Significant increase in component count and complexity
- Expected matching increase in failure frequency
- Checkpoint/restart is becoming less and less efficient

Reactive vs. Proactive Fault Tolerance

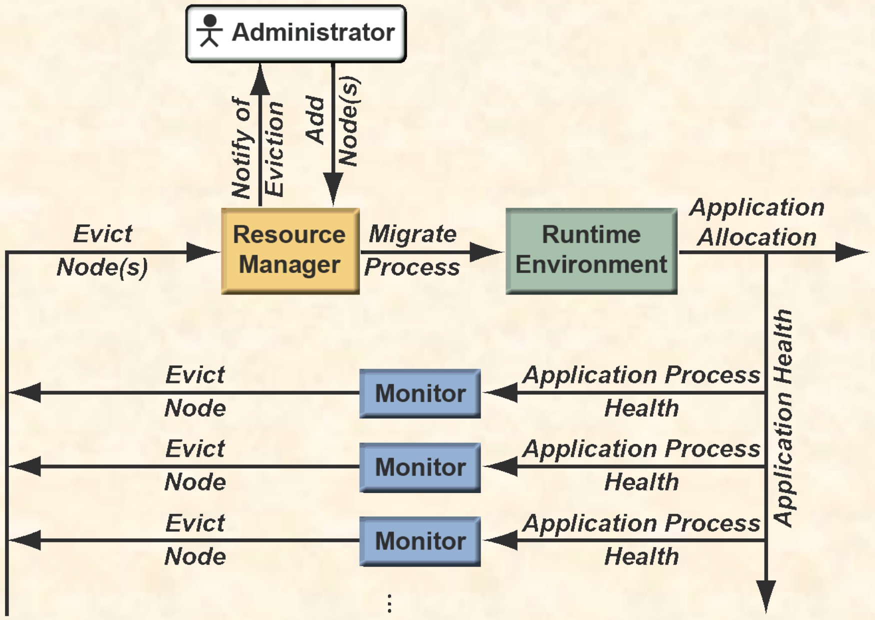
- Reactive fault tolerance
  - Keeps parallel applications alive through recovery from experienced failures
  - Employed mechanisms react to failures
  - Examples: Checkpoint/restart, message logging/replay
- Proactive fault tolerance
  - Keeps parallel applications alive by avoiding failures through preventative measures
  - Employed mechanisms anticipate failures
  - Example: Preemptive migration

Proactive Fault Tolerance using Preemptive Migration

- Relies on a feedback-loop control mechanism
  - Application health is constantly monitored and analyzed
  - Application is reallocated to improve its health and avoid failures
  - Closed-loop control similar to dynamic load balancing
- Real-time control problem
  - Need to act in time to avoid imminent failures
- No 100% coverage
  - Not all failures can be anticipated, such as random bit flips

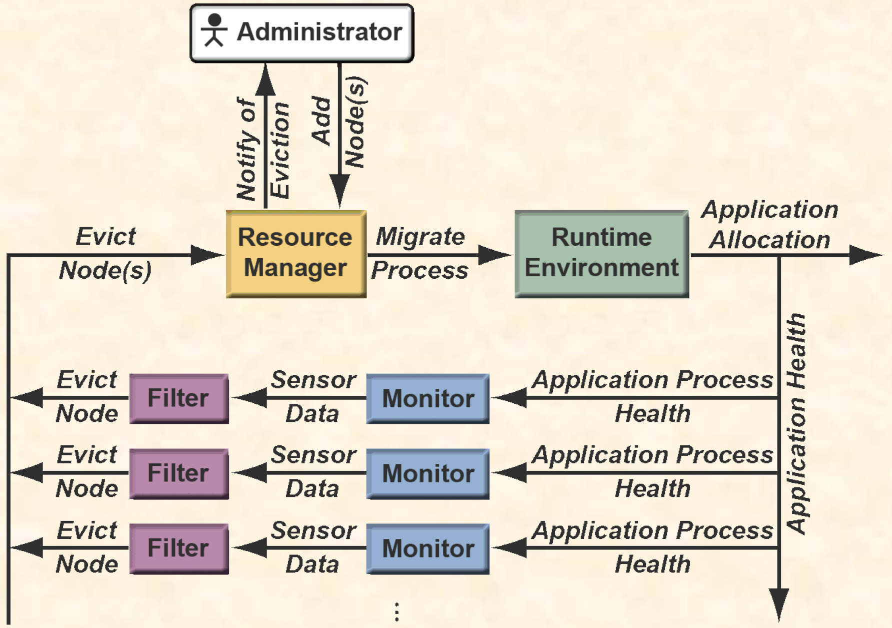
Type 1 Feedback-Loop Control Architecture

- Alert-driven coverage
  - Basic failures
- No evaluation of application health history or context
  - Prone to false positives
  - Prone to false negatives
  - Prone to miss real-time window
  - Prone to decrease application heath through migration
  - No correlation of health context or history



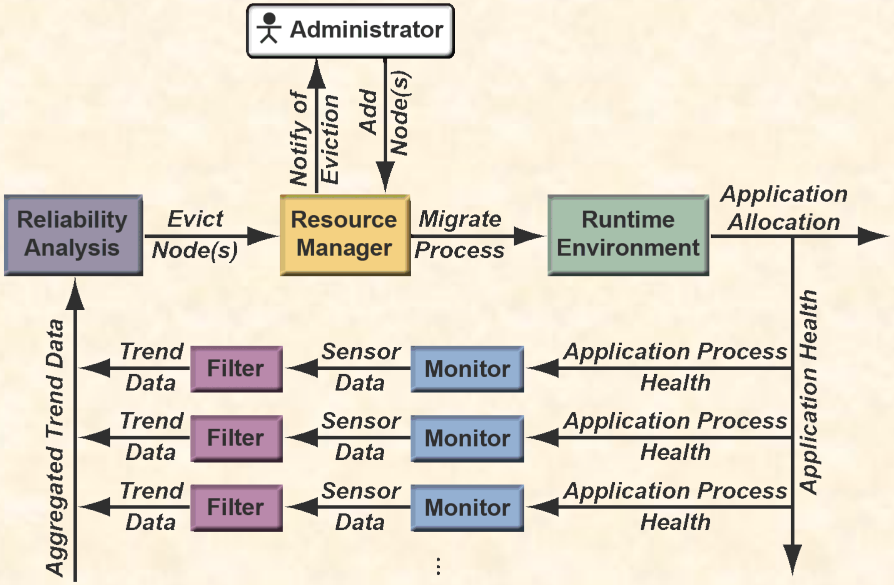
Type 2 Feedback-Loop Control Architecture

- Trend-driven coverage
  - Basic failures
  - Less false positives/negatives
- No evaluation of application reliability
  - Prone to miss real-time window
  - Prone to decrease application heath through migration
  - No correlation of health context or history



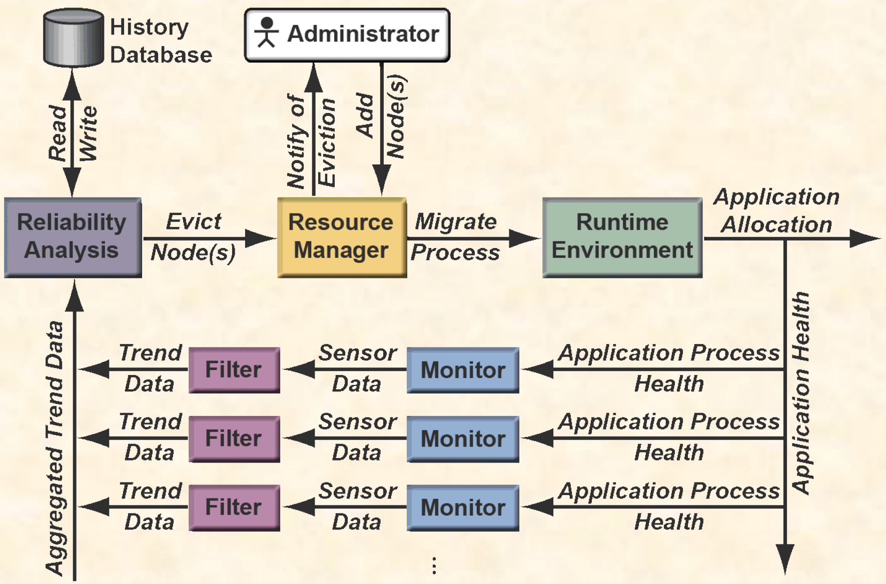
Type 3 Feedback-Loop Control Architecture

- Reliability-driven coverage
  - Basic and correlated failures
  - Less false positives/negatives
  - Able to maintain real-time window
  - Does not decrease application heath through migration
  - Correlation of short-term health context and history
- No correlation of long-term health context or history
  - Unable to match system and application reliability patterns



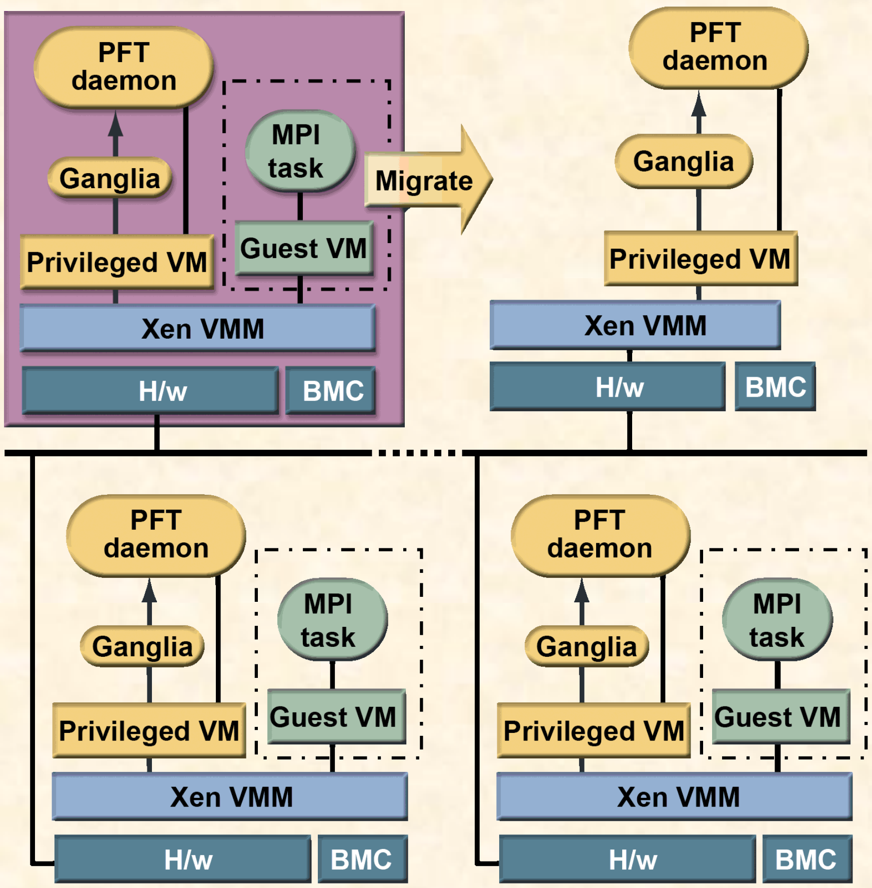
Type 4 Feedback-Loop Control Architecture

- Reliability-driven coverage of failures and anomalies
  - Basic and correlated failures, anomaly detection
  - Less prone to false positives
  - Less prone to false negatives
  - Able to maintain real-time window
  - Does not decrease application heath through migration
  - Correlation of short and long-term health context & history



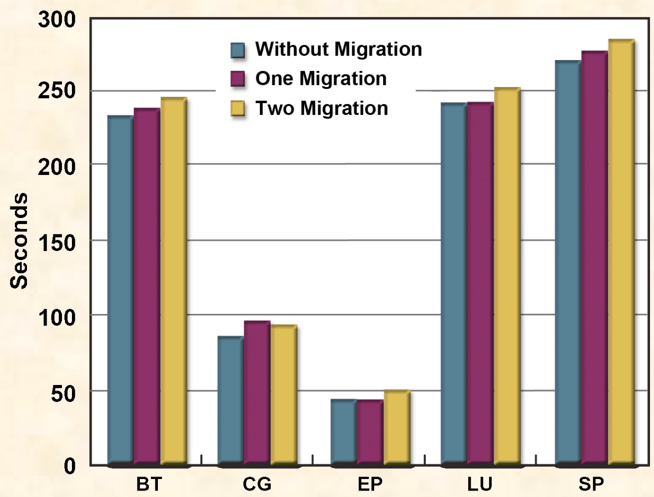
VM-level Preemptive Migration using Xen

- Type 1 system setup
  - Xen VMM on entire system
  - Host OS for management
  - Guest OS for computation
  - Spare nodes without Guest OS
  - System monitoring in Host OS
  - Decentralized scheduler/load balancer using Ganglia
- Deteriorating node health
  - Ganglia threshold trigger
  - Migrate guest OS to spare
  - Utilize Xen's migration facility



VM-level Migration Performance Impact

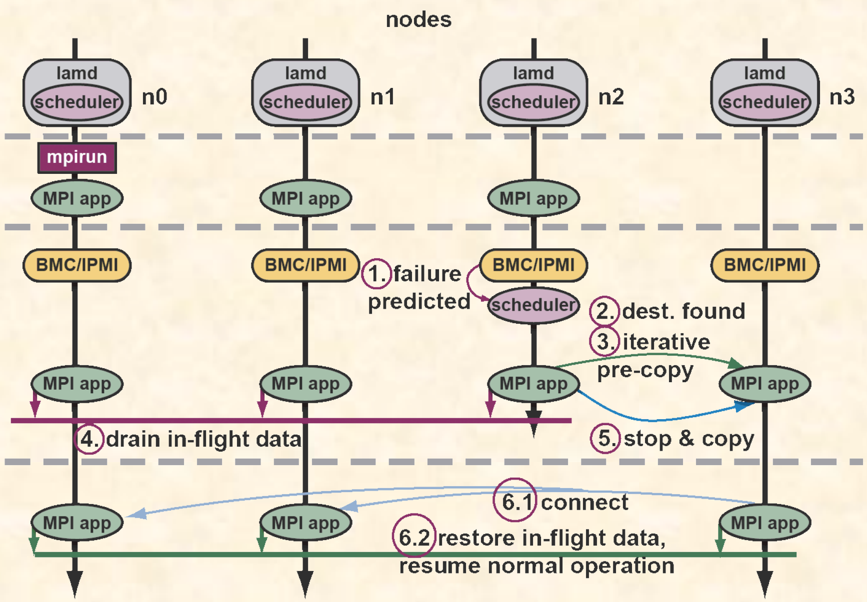
- Single node migration
  - 0.5-5% longer run time
- Double node migration
  - 2-8% longer run time
- Migration duration
  - Stop & copy : 13-14s
  - Live : 14-24s
- Application downtime
  - Stop & copy > Live



16-node Linux cluster at NCSU with dual core, dual-processor AMD Opteron and Gigabit Ethernet

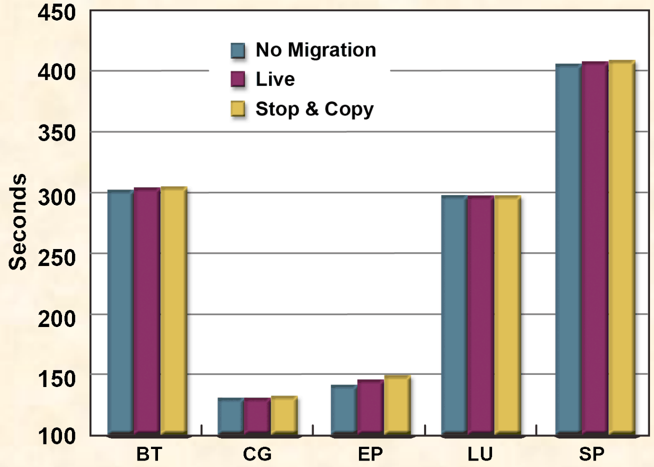
Process-Level Preemptive Migration w/ BLCR

- Type 1 system setup
  - LAM/MPI with Berkeley Lab Checkpoint/Restart (BLCR)
  - Per-node health monitoring
    - Baseboard management controller (BMC)
    - Intelligent platform management interface (IPMI)
  - New decentralized scheduler/ load balancer in LAM
  - New process migration facility in BLCR (stop&copy and live)
- Deteriorating node health
  - Simple threshold trigger
  - Migrate process to spare



Process-Level Migration Performance Impact

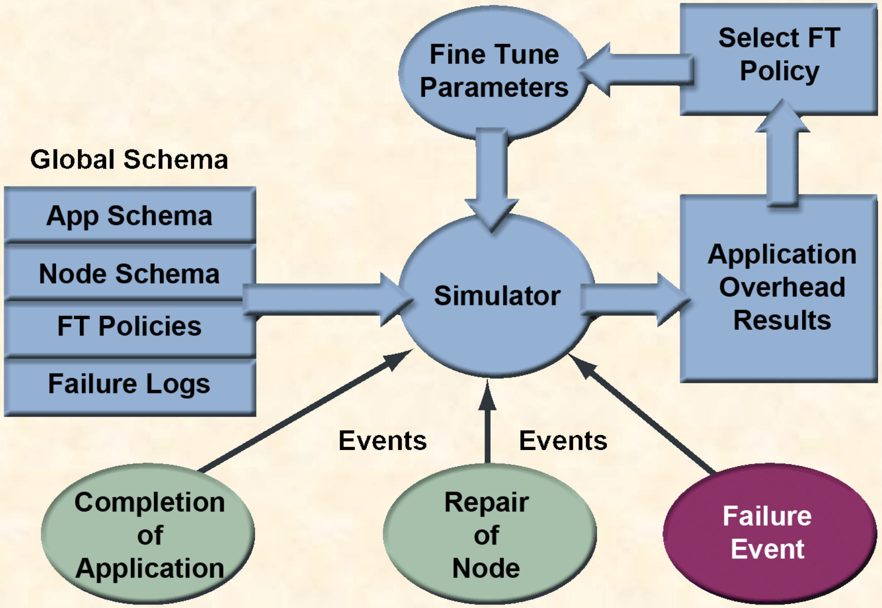
- Single node migration overhead
  - Stop & copy : 0.09-6 %
  - Live : 0.08-2.98%
- Single node migration duration
  - Stop & copy : 1.0-1.9s
  - Live : 2.6-6.5s
- Application downtime
  - Stop & copy > Live
- Node eviction time
  - Stop & copy < Live



16-node Linux cluster at NCSU with dual core, dual-processor AMD Opteron and Gigabit Ethernet

Simulation of Fault Tolerance Policies

- Evaluation of fault tolerance policies
  - Reactive only
  - Proactive only
  - Reactive/proactive combination
- Evaluation of fault tolerance parameters
  - Checkpoint interval
  - Prediction accuracy
- Event-based simulation framework using actual HPC system logs
- Customizable simulated environment
  - Number of active and spare nodes
  - Checkpoint and migration overheads



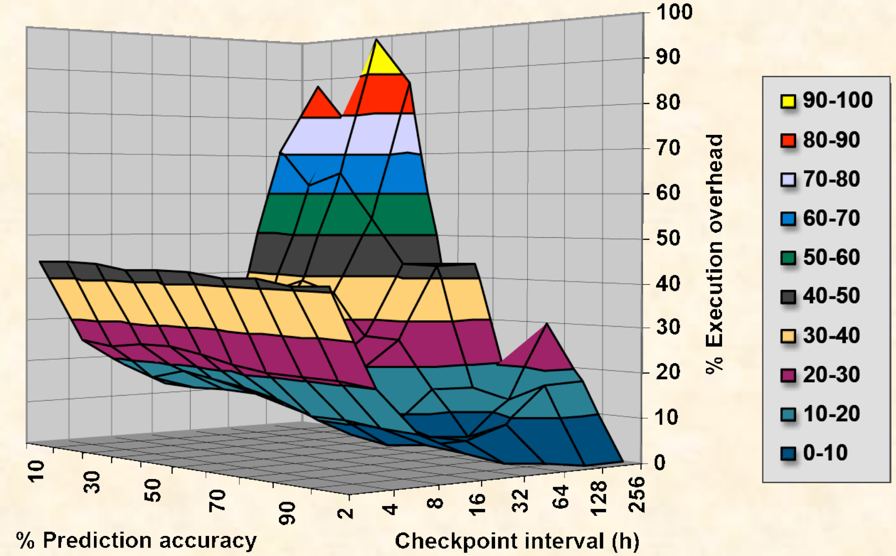
Combining Proactive & Reactive Approaches

- Best: Prediction accuracy >60% and checkpoint interval 16-32h
- Better than only proactive or only reactive
- Results for higher accuracies and very low intervals are worse than only proactive or only reactive

Number of processes	125
Active/Spare nodes	125/12
Checkpoint overhead	50min
Migration overhead	1 min

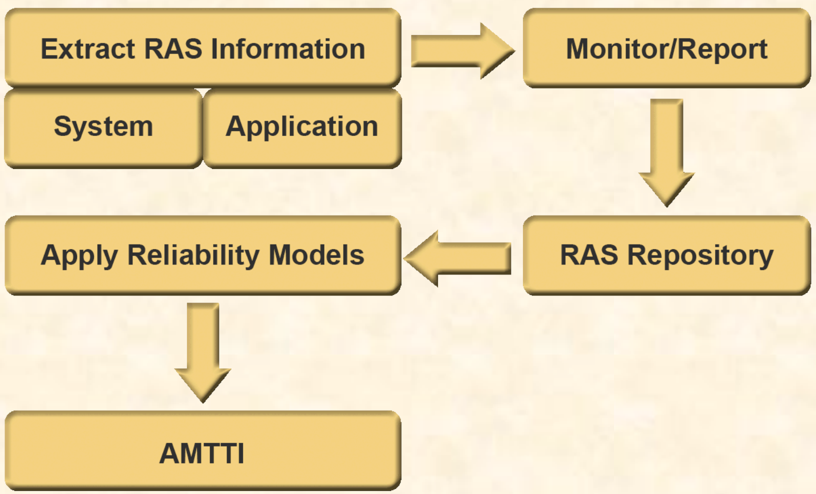
Simulation based on ASCI White system logs (nodes 1-125 and 500-512)

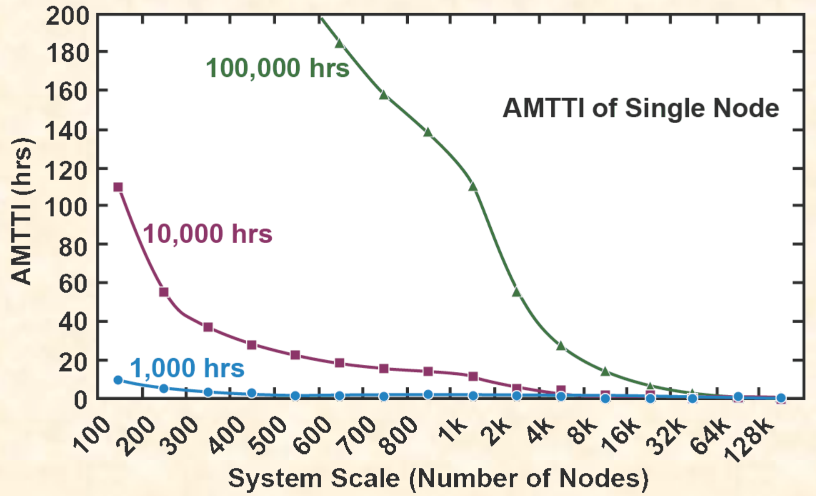
Execution overhead for various checkpoint intervals and different prediction accuracy



Research in Reliability Modeling

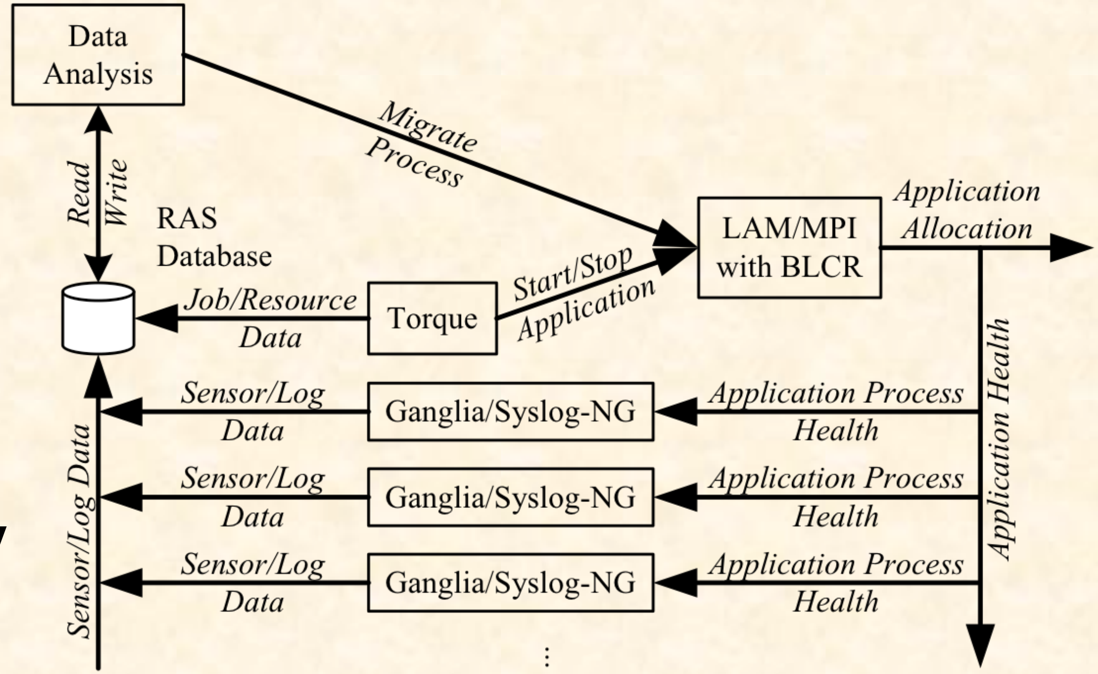
- Type 3 system setup
  - Monitoring of application and system health
  - Recording of application and system health monitoring data
  - Reliability analysis on recorded data
  - Application mean-time to interrupt (AMTTI) estimation
- Type 4 system setup
  - Additional recording of application interrupts
  - Reliability analysis on recent and historical data





Proactive Fault Tolerance Framework

- **Unified interfaces between components**
- **Extendable RAS engine core interfacing with**
  - **Monitoring data aggregation/filtering component**
  - **Job and resource management service**
  - **Process/VM migration mechanism**
  - **Online/offline reliability modeling**
- **Previous Reading MSc student (A. Litvinova)**



Ongoing and Future Work

- Research in scalable monitoring data aggregation/filtering
  - Scalable, fault tolerant overlay reduction networks
  - In-flight monitoring data aggregation
  - Current MSc student (Swen Boehm)
- Research in scalable monitoring data filtering
  - Extend the current prototype with in-flight data filtering
  - Enhance filters with statistical analysis techniques
- Research in scalable syslog data aggregation/filtering
  - Extend the current prototype with log message aggregation
- Integrate scalable monitoring prototype with proactive fault tolerance framework

Challenges Ahead

- Health monitoring
  - Identifying deteriorating applications and OS conditions
  - Coverage of application failures: Bugs, resource exhaustion
- Reliability analysis
  - Performability analysis to provide extended coverage
- Scalable data aggregation and processing
  - Key to timeliness in the feedback control loop
- Need for standardized metrics and interfaces
  - System MTTF/MTTR != Application MTTF/MTTR
  - System availability != Application efficiency
  - Monitoring and logging is system/vendor dependent

Acknowledgements

- **Investigators at Oak Ridge National Laboratory:**
  - ***Stephen L. Scott [Lead PI]*, Christian Engelmann, Geoffroy Vallée, Thomas Naughton, Anand Tikotekar, George Ostrouchov**
- **Investigators at Louisiana Tech University:**
  - ***Chokchai (Box) Leangsuksun [Lead PI]*, Nichamon Naksinehaboon, Raja Nassar, Mihaela Paun**
- **Investigators at North Carolina State University:**
  - ***Frank Mueller [Lead PI]*, Chao Wang, Arun Nagarajan, Jyothish Varma**
- **Funding sources:**
  - **U.S. Department of Energy, Office of Science, FASTOS Program**

Office of Science U.S. DEPARTMENT OF ENERGY

OAK RIDGE National Laboratory

NC STATE UNIVERSITY

LOUISIANA TECH UNIVERSITY®

UT-BATTELLE

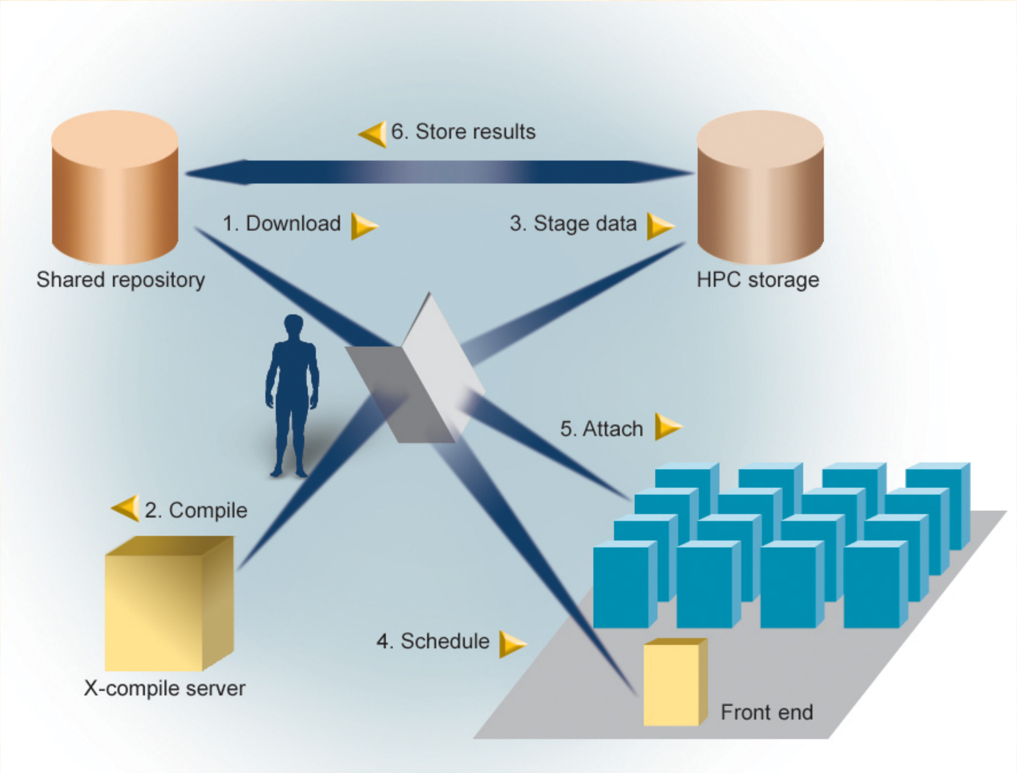
ADDAPT: Assisting Application Development, Deployment, and Execution

Christian Engelmann

Computer Science and Mathematics Division
Oak Ridge National Laboratory

Research and Development Goals

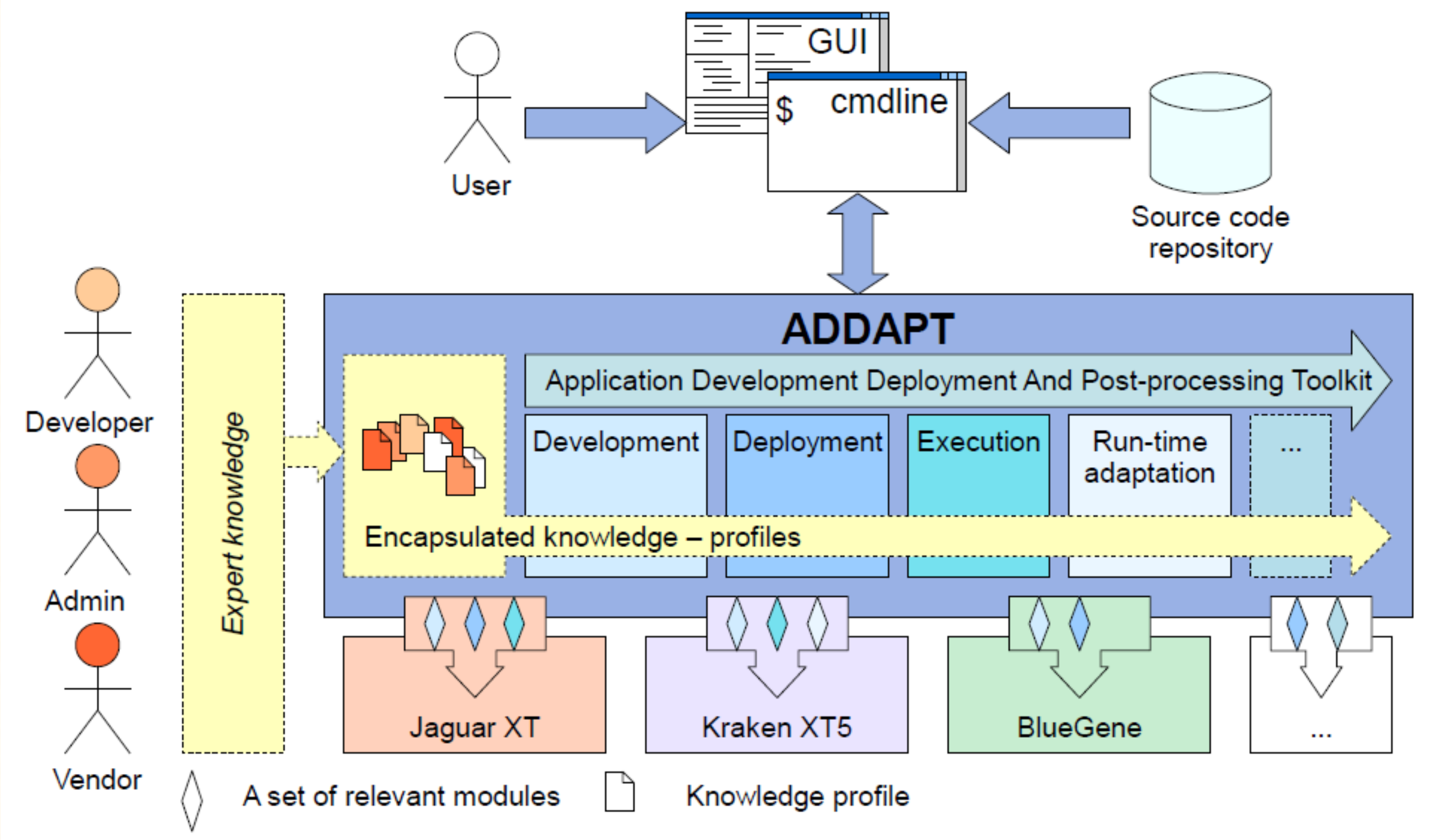
- Increasing the overall productivity of developing and executing computational codes
- Optimizing the development and deployment processes of scientific applications
- Simplifying the activities of application scientists, using uniform and adaptive solutions
- “Automagically” supporting the diversity of existing and emerging high–performance computing architectures



Typical scientific application development, deployment, and execution activities

Office of Science U.S. DEPARTMENT OF ENERGY
OAK RIDGE National Laboratory
EMORY UNIVERSITY

ADDAPT Architecture

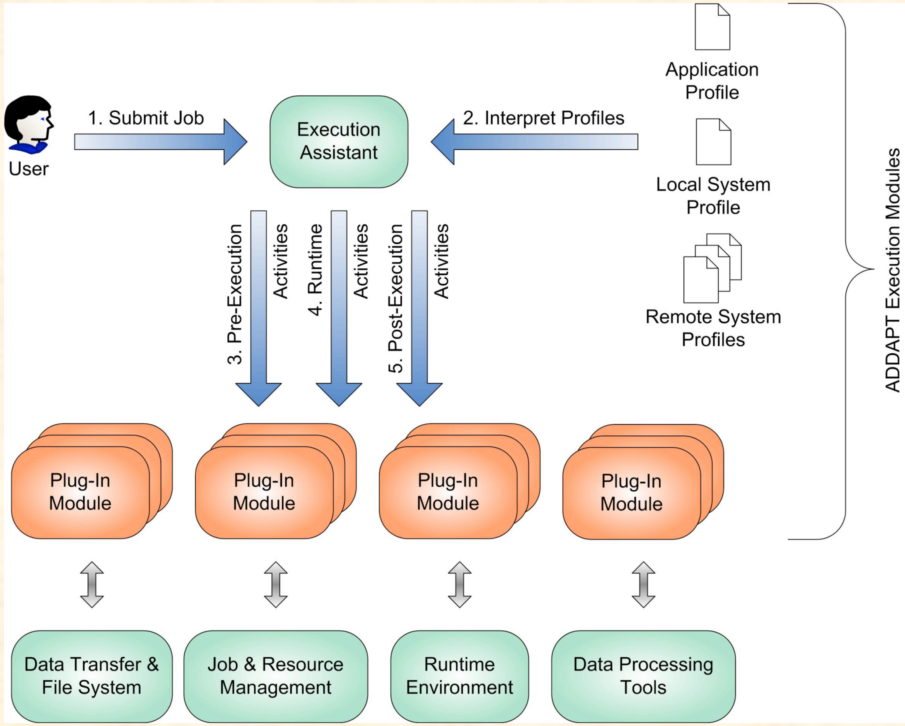


ADDAPT Components

- **Porting assistant**
  - to help port new libraries and kernels into legacy codes,
  - to identify incompatibilities with the system software stack,
  - do automatic source-to-source translation, and
  - identify areas to improve performance or fault tolerance.
- **Build assistant**
  - to adapt the application's build script to the site-specific versions of compilers, libraries, and flags, and
  - to help resolve problems at the link stage.
- **Execution assistant**
  - to assist in data staging, fast application launch, runtime support, and post-execution data off loading.

Ongoing and Future Work: ADDAPT Execution Assistant Component

- Combine knowledge from application and site profiles
  - Match application properties with system needs using ontologies and reasoning
- Assist the scientist in running his/her application
  - Adapt system configuration to application needs
- Automate data staging and pre-/post-processing activities
  - Interface with respective tools through plug-ins



UT-BATTELLE

Institute for advanced Architectures and Algorithms (IAA): Simulation Efforts at ORNL

Christian Engelmann

Computer Science and Mathematics Division
Oak Ridge National Laboratory

Objectives

- Simulation of system architectures at scale
- To investigate scalability, performance, and fault tolerance of algorithms at extreme scale
- ORNL’s earlier work was already able to run up to 1,000,000 simulated processes (JCAS)

Java Cellular Architecture Simulator (JCAS)

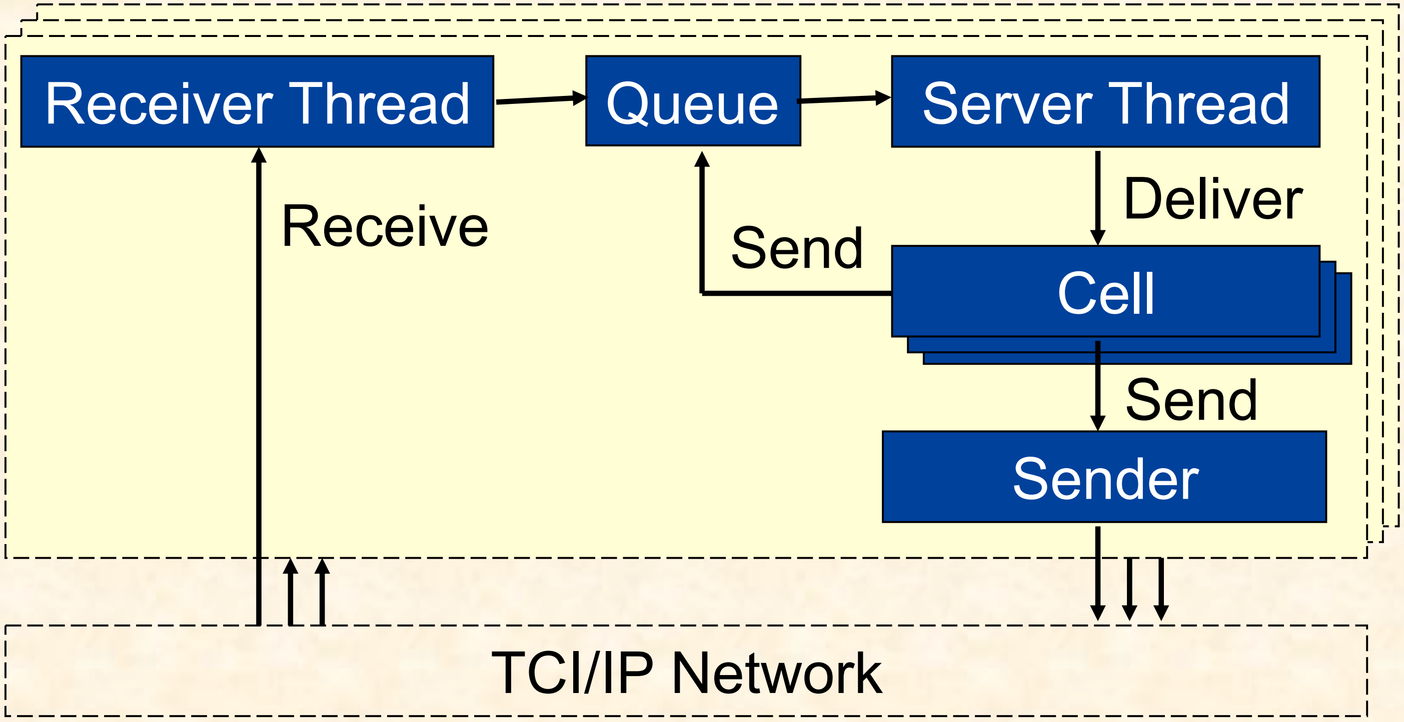
- Developed at ORNL in Java
- Native C and Fortran application support using JNI
- Runs as standalone or distributed application
- Lightweight framework simulates up to 1,000,000 lightweight virtual processes on 9 real processors
- Standard and experimental network interconnects:
  - Multi-dimensional mesh/torus
  - Nearest/Random neighbors
- Message driven simulation without notion of time
  - Not in real-time, no time-accurate discrete event simulation
- Primitive fault-tolerant MPI support
  - No collectives, no MPI 2

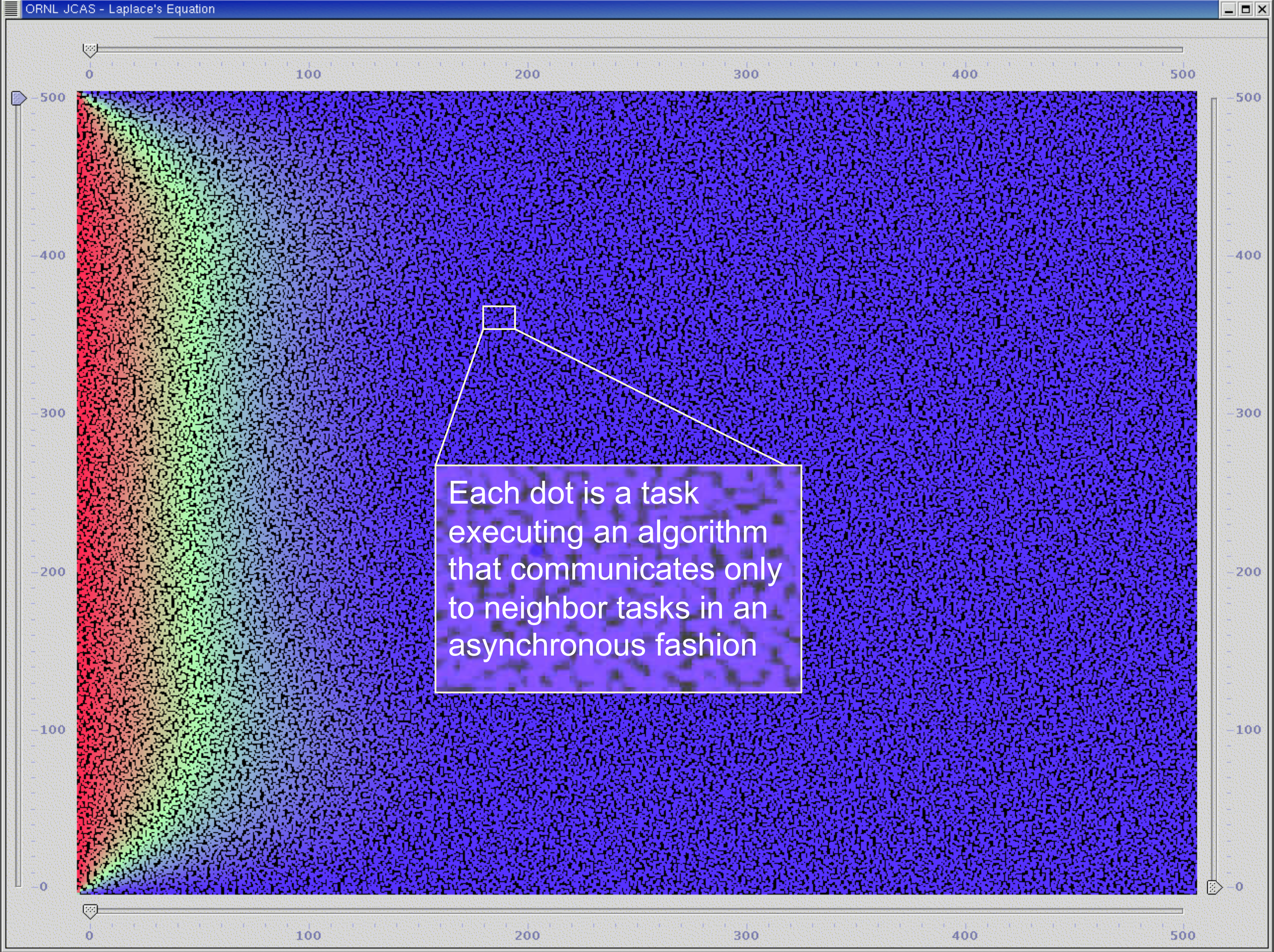
Technical Approach

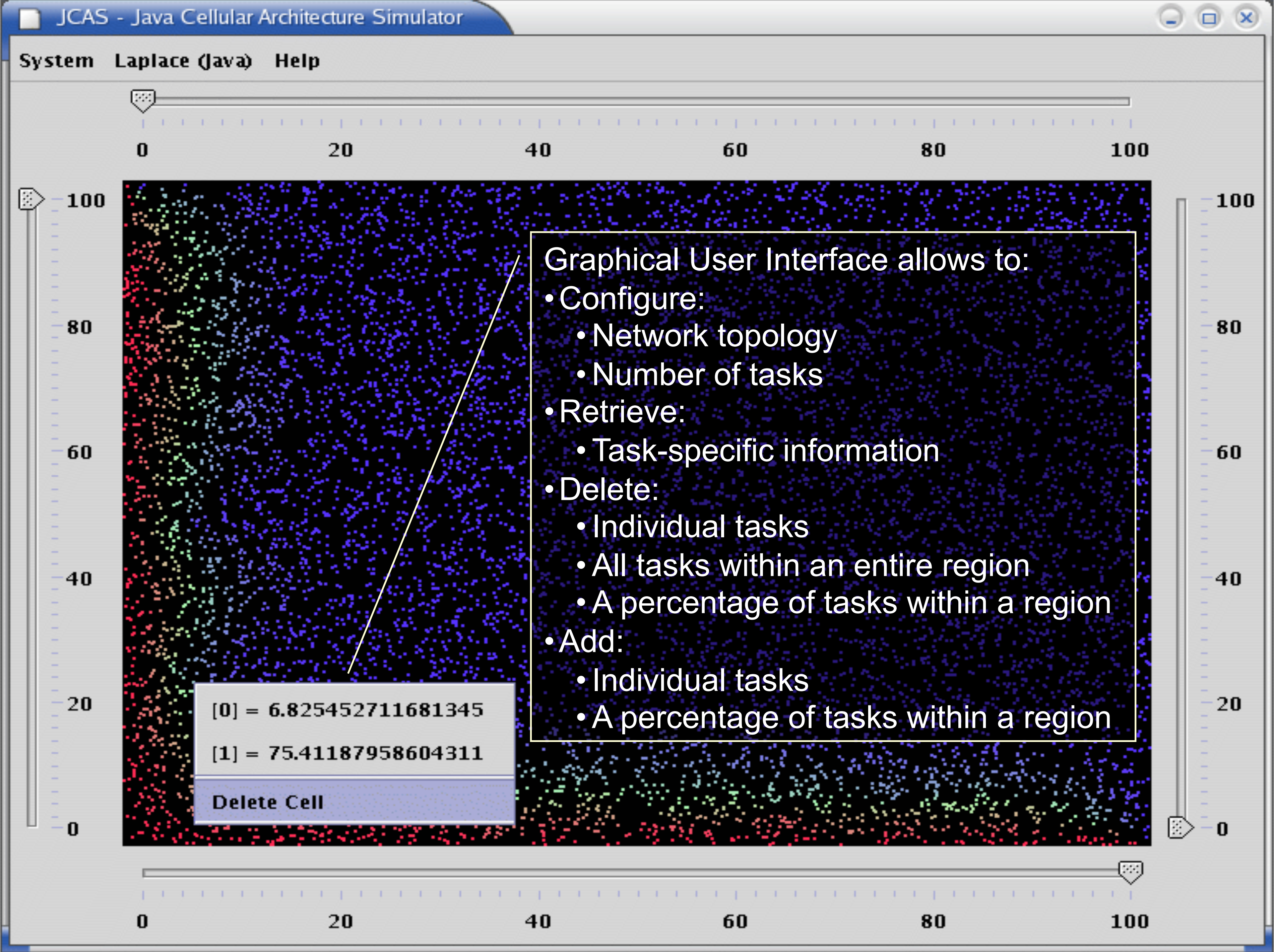
- **Distributed set of discrete event simulators with node-local message queues**
- **Simulation of virtual MPI processes for parallel app.**
- **Virtual processes run on real hardware with virtual MPI**
- **No virtual process time**
- **Fault injection capability**
- **Interactive graphical user interface as front-end**
- **TCP servers as back-ends**

Implementation

- Every cell has own code, memory and neighbors list
- Server hosts cells and initiates the context switch
- Cells communicate asynchronously using messages




ORNL JCAS - Laplace's Equation
Each dot is a task executing an algorithm that communicates only to neighbor tasks in an asynchronous fashion
0
100
200
300
400
500

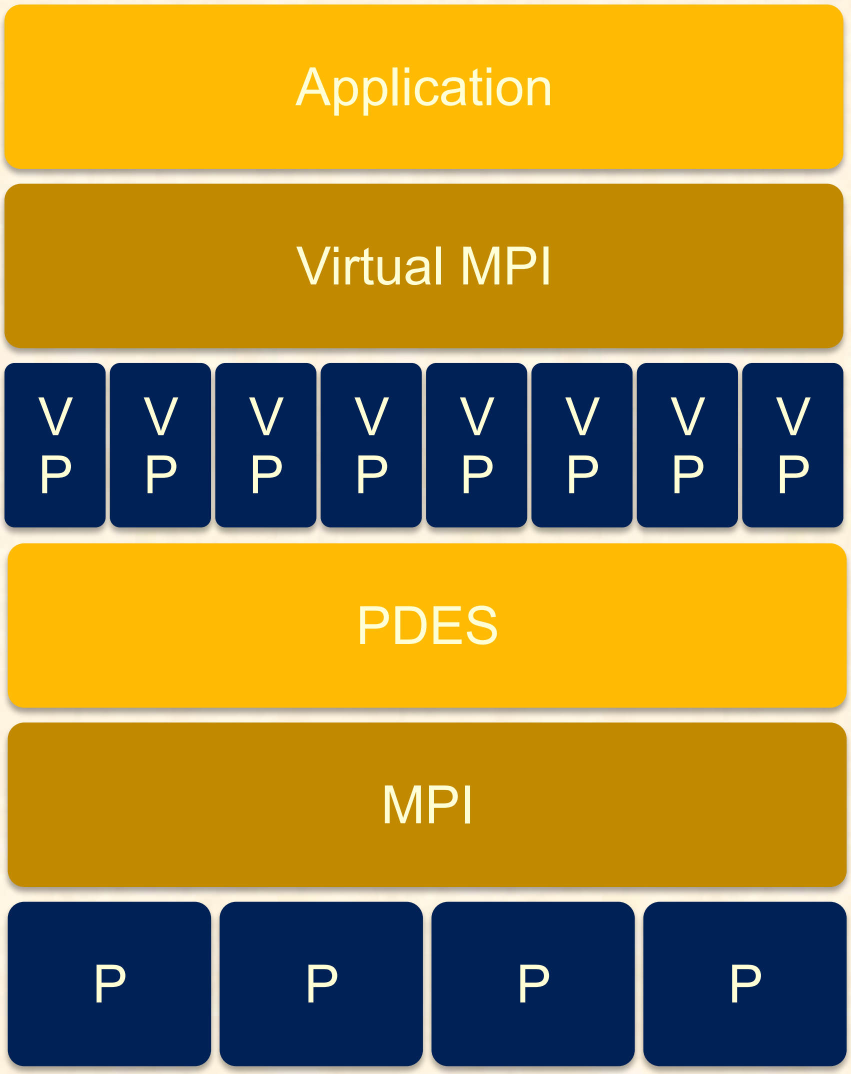

JCAS - Java Cellular Architecture Simulator
System
Laplace (Java)
Help
0
20
40
60
80
100
Graphical User Interface allows to:
• Configure:
• Network topology
• Number of tasks
• Retrieve:
• Task-specific information
• Delete:
• Individual tasks
• All tasks within an entire region
• A percentage of tasks within a region
• Add:
• Individual tasks
• A percentage of tasks within a region
[0] = 6.825452711681345
[1] = 75.41187958604311
Delete Cell

IAA Simulation Efforts at ORNL

- Investigate scalability, performance and fault tolerance of algorithms at extreme scale through simulation
- Extending the JCAS simulation capabilities
  - Simulating more processes (~10,000,000)
  - Running more complex and resource-hungry algorithms
  - Support for unmodified MPI applications
- Evaluation of algorithms at extreme scale
  - Notion of global virtual time and virtual process clocks
  - Accounting for resource usage, such as processor and network
  - Gathering of scalability, performance & fault tolerance metrics
  - Parameter studies at scale

Technical Approach

- Parallel discrete event simulation (PDES) atop MPI
- Simulation of virtual MPI processes for parallel app.
- Virtual processes run on real hardware with virtual MPI
- Consistent virtual process clock from PDES
- Virtual process clock can be scaled by PDES via model
- Virtual interconnect latency is set by PDES via model



Ongoing and Future Work

- Ported JCAS to C/C++ to improve scalability/performance
- Replaced TCP/IP with (native) MPI communication
- Replaced distributed set of DESs with PDES
  - Conservative synchronization only, need optimistic and time-warp synchronization
- Extend virtual MPI capabilities
  - Asynchronous, collectives, process control (spawn), …
- Extend fault injection and notification mechanisms
  - Injection based on failure distributions and application state
- Add simulated machine model (for network)
- Gather scalability, performance & fault tolerance metrics

Soft-Error Resilience for Future-Generation High-Performance Computing Systems

Christian Engelmann

Computer Science and Mathematics Division
Oak Ridge National Laboratory

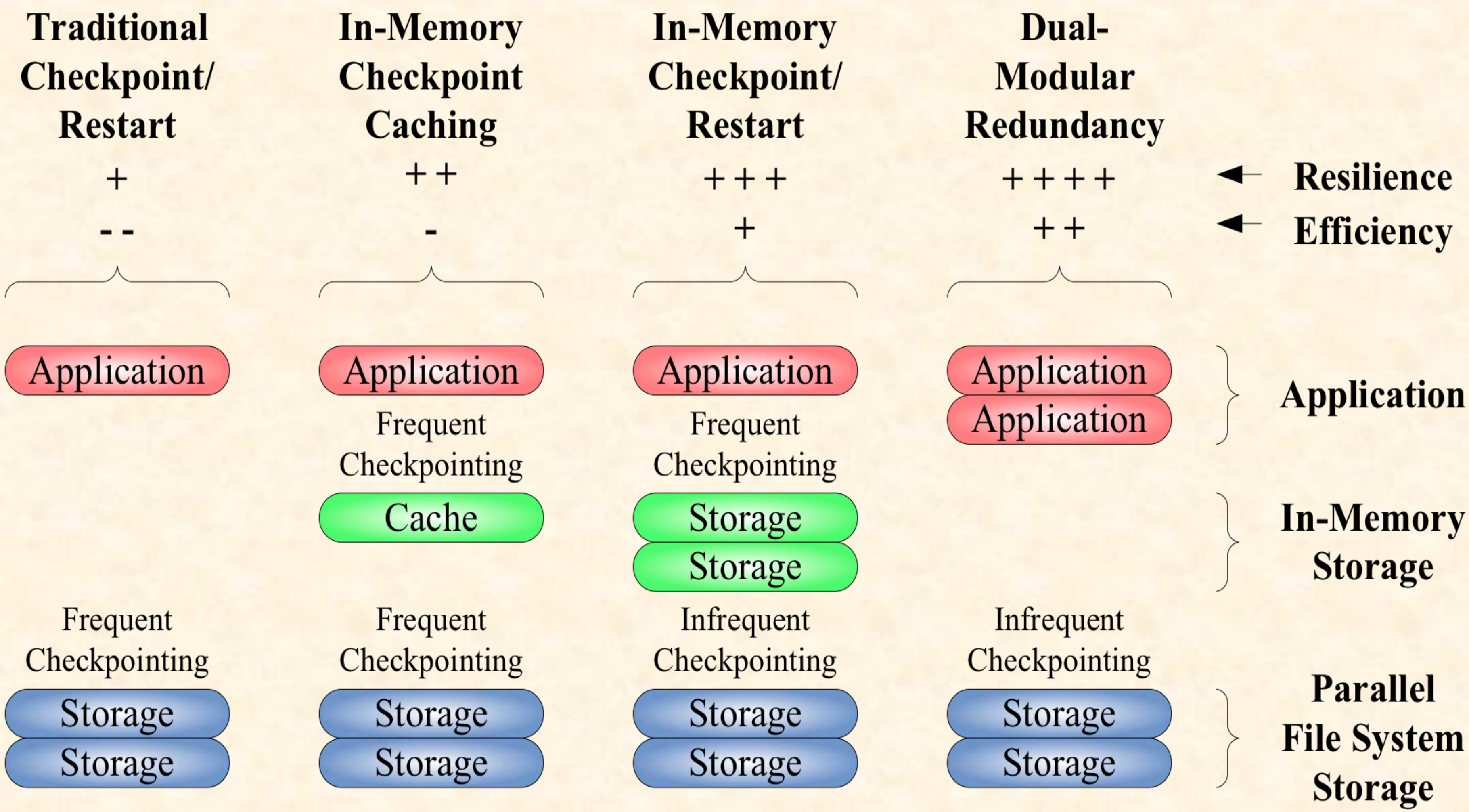
Motivation

- Next-generation HPC systems will have
  - More frequent failures in general
  - More frequent soft errors in particular
  - Less efficient parallel file system checkpoint/restart
- Existing fault tolerance approaches an ongoing research efforts do not cover soft error resilience
  - ECC double-bit errors require node/process restart
  - Silent data corruption remains undetected
- Lack of soft error resilience strategy is preventing deployment of GPUs and FPGAs at scale

Technical approach

- Compute-node in-memory checkpoint caching
  - Short-term solution
  - Improving parallel file system checkpoint/restart
- Compute-node in-memory checkpoint/restart
  - Near-term solution
  - Replacing parallel file system checkpoint/restart
- Dual-modular redundancy (DMR)
  - Long-term solution
  - Replacing rollback recovery schemes in HPC

Comparison of traditional and proposed technologies (1/2)



Comparison of traditional and proposed technologies (2/2)

Solution	**Processor**	**Memory**	**Price**	**Power**
Current: Lustre checkpoint/restart	1xAMD Opteron 2356	2x4GB Micron DDR2-800	\$500 +\$750 = \$1250	$75W$ $+2W$ $= 77W$
Short-term: Compute-node in-memory checkpoint caching	1xAMD Opteron 2356	4x4GB Micron DDR2-800	\$500 +\$1500 = \$2000	$75W$ $+4W$ $= 79W$
Near-term: Compute-node in-memory checkpoint/restart with possibly new boards	1xAMD Opteron 2356	2x4GB Micron DDR2-800 4x4GB Kingston DDR2-800	\$500 +\$750 +\$600 > \$1700	$75W$ $+2W$ $+4W$ $> 81W$
Long-term: DMR with possibly new boards and/or more racks	2xAMD Opteron 2356	4x4GB Kingston DDR2-800	\$1000 +\$600 > \$1600	$150W$ $+4W$ $> 154W$

Ongoing and Future Work

- Develop compute-node in-memory checkpoint caching
  - User-space (FUSE) front-end for storage virtualization
  - User-space (FUSE) backend for seamless integration
  - Asynchronous draining of cache to parallel file system
- Develop compute-node in-memory checkpoint/restart
  - Checkpoint data replication for fault tolerance
  - Integration with application- and system-level C/R solutions
- Develop dual-modular redundancy
  - Design modular redundancy models and algorithms
  - Implement static modular computation redundancy prototype
  - Experiment with I/O & file system access under redundancy
  - Implement dynamicmodular computation redundancy prototype
- Create trade-off models

UT-BATTELLE

Questions?

OAK RIDGE NATIONAL LABORATORY
U. S. DEPARTMENT OF ENERGY