High-End Computing Resilience: Analysis of Issues Facing the HEC Community and Path-Forward for Research and Development

Christian Engelmann

Computer Science and Mathematics Division
Oak Ridge National Laboratory
Report Background

• National HPC Workshop on Resilience, Arlington, VA, USA, August 12-14, 2009

• Full-day workshop with approx. 60 participants:
  – Session on Data Integrity
  – Session on Collection, Monitoring, and Analysis of Data
  – Session on Metrics and Modeling
  – Session on Resilient Middleware

• Workshop report authors:
  – Nathan DeBardeleben (LANL), James Laros (SNL), John Daly (DoD), Stephen Scott (ORNL, now TN Tech), Christian Engelmann (ORNL), Bill Harrod (DARPA, now OASCR)

• Workshop report was submitted to NSF’s High-end Computing Program
• Motivation:
  − Current resilience methods will be unpractical in the future

• Resilience terminology definitions

• Survey existing HPC resilience technologies

• Identify key areas for future research, development, and standards work, such as
  − Theoretical foundations
  − Enabling infrastructure
  − Fault prediction and detection
  − Monitoring and control
  − End-to-end data integrity
Resilience Terminology Definitions

- **Resilience**: The ability of a system to keep applications running and maintain an acceptable level of service in the face of transient, intermittent, and permanent faults.

- **Fault tolerance**: The ability of a system to continue performing its intended function properly in the face of transient, intermittent, and permanent faults.

- **40+ other frequently used terms**:  
  - Error latency, detection and propagation  
  - Transient, intermittent, and permanent faults  
  - Soft and hard errors

Existing HPC Resilience Technologies

• Checkpoint/restart (C/R)
  – SSD in Cray X/Y-MP (1982/88) and IBM 3090 (1985)
  – Networked disk storage in Intel Paragon XP/S (1992)
  – Local & networked disk storage in ASCI White (2000)
  – Networked disk storage in Cray XT and IBM BG (2000+)

• Application-level C/R dominates in practice

• System-level C/R

• Diskless C/R

• Fault-tolerant message passing
  – PVM 3 (1993), Starfish MPI (1999), FT-MPI (2001), MPI-3 (?)
Existing HPC Resilience Technologies

- **Message logging**

- **Algorithm-based fault tolerance (ABFT)**
  - Huang et al. (1984), Chen et al. (2006), Ltaief et al. (2007)

- **Proactive fault tolerance**
  - Nagarajan et al. (2007), Wang et al. (2008)

- **Log-based failure analysis and prediction**

- **Soft-error resilience**
  - Parity memory in Cray-1 (1977)
  - ECC memory in Cray X-MP (1982)
  - ECC for caches and registers in AMD Opteron (2007)
Key Areas for Future Research, Development, and Standards Work
Theoretical Foundations

- Lord Kelvin: “If you can’t measure it, you can’t improve it!”
- Agreed upon definitions, metrics and methods
  - System vs. application MTTI, MTTR, and availability/efficiency
- Dependability analysis
  - Fault injection studies using modeling and simulation
- Dependability benchmarking (robustness testing)
  - Fault injection studies using experimental evaluation
- Formal methods, statistics, and uncertainty quantification
Enabling Infrastructure

- Programming models & libraries
  - Fault awareness and transparent fault tolerance
- System software
  - Reliable (hardened) system software (OS kernel, file systems)
- RAS systems and tools
  - System and application health monitoring
- Cooperation and coordination frameworks
  - Fault notification across software layers
  - Tunable resilience strategies
- Production solutions of existing resilience technologies
  - Enhanced recovery-oriented computing

Fault Prediction and Detection

- Statistical analysis
- Machine learning
- Anomaly detection
- Visualization
- Data & information collection
Monitoring and Control

• Non-intrusive, scalable monitoring and analysis
  – Decentralized/distributed scalable RAS systems

• Standards-based monitoring and control
  – Standardized metrics and application/system interfaces

• Tunable fidelity
  – Adjustable resilience/performance/power trade-off
  – Variety of resilience solutions to fit different needs

• Quality of service and performability
  – Measure-improve feedback loop at various granularities

End-to-End Data Integrity

- Confidence in getting the right answer and using correct data to make informed decisions
- Protection from undetected errors that corrupt data/code
  - Understanding root causes and error propagation
- Mitigation strategies against silent code/data corruption
  - Application-level checks
  - Self-checking code and ECC
  - Redundant multi-threading and process pairs

Conclusions

• Current resilience methods will be unpractical in the future
• Alternatives need to be developed into practical solutions
• Agreed upon definitions, metrics and benchmarks are needed to measure improvement and to compare fairly
• Root causes and propagation are not well understood
  – No effective fault detection and prediction
• Resilience is needed across the entire software stack
  – System software, programming models, apps and tools
  – Communication/coordination between layers
• Faults and fault recovery will be continuous
• Tunable solutions to adjust resilience/performance/power