Exploiting Global View for Resilience (GVR)

Andrew A. Chien (PI), Hajime Fujita, Guoming Lu, and Zachary Rubenstein, *University of Chicago;*
Pavan Balaji (co-PI), Pete Beckman, James Dinan, Jeff Hammond, Kamil Iskra, *ANL;*
Robert Schreiber, *Hewlett-Packard Labs*
http://gvr.cs.uchicago.edu/

Background

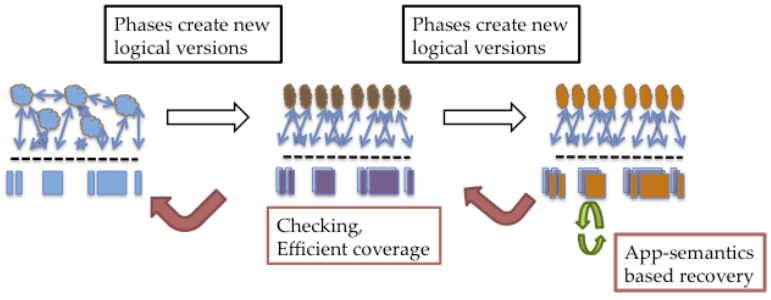
The energy, parallelism, and error rate projections (≈ 2×10^9 FITS/billion hours or 30 minutes MTTI) for exascale systems represent extraordinary challenges for applications and programming models. Resilience is a particular challenge at extreme scale due to system size and computations of interest, but also critical to the incorporation of volume computing technologies engineered for less-demanding markets.

For inspiration, we draw on examples of resilient large-scale computing systems with 10^5 to 10^{10} active elements, including scalable internet services, batch internet-scale data processing, and the internet itself. These systems share three key architectural features: a foundation of reliable data, programmer-managed, nonuniform reliability, and application-managed consistency. These architectural features are used to respond to faults and failures of many types.

Approach

Global View Resilience (GVR) is a new programming approach that exploits a global view data model (global naming of data, consistency, and distributed layout), adding reliability to globally visible distributed arrays. Such globally-visible data is widely considered helpful for programming irregular, adaptive applications and dealing with variability in future extreme scale systems. GVR adopts all three proven architectural elements for large-scale resilience: reliable data, nonuniform reliability, and application-managed consistency. Global naming of distributed data yields programmability benefits that include simpler expression of algorithms and decoupling of computation and data structure across increasingly complex (irregular, variable, degraded) hardware.

In the GVR programming model, applications indicate reliability priorities—which data is more important to protect—allowing application management of reliability overheads. Because the distributed array abstraction is portable, GVR enables said management to be flexible and portable whilst tapping programmers' deep scientific and application code insights. GVR will be developed as a library, allowing it to be used alongside other programming tools and models, or targeted by compiler back-ends of emerging higher level programming models.



Global View Resilience builds computation reliability on a foundation of data reliability. In the new model, computation is viewed as a sequence of data transformations from one global view data object to the next (or to a new state for the data object), as shown above. The implementation of the global view object, which can be made reliable in various ways (checkpoint, replication, encoding, etc.), is handled by the GVR runtime: the programmer simply knows that it is resilient and that earlier versions will remain available if needed, even in the face of transient errors and failures to individual nodes or storage elements. On that foundation of data reliability, applications

can tolerate a wide variety of errors and system failures, continuing steadily forward, albeit with some loss of effort, but without lapsing into inconsistency.

To support efficient, deep error checking and coverage, the GVR project will study algorithms as well as runtime and architecture mechanisms that implement, map, and adapt the reliability deployment based on application-specified reliability priorities. The project will work with the community, to create a flexible, efficient, cross-layer error management architecture called "open reliability" that allows applications to describe error detection (checking) and recovery routines and inject them into the GVR stack for efficient implementation. This enables application and system to work in concert, exploiting semantics (algorithmic or even scientific domain) and key capabilities (e.g., fast error detection in hardware) to dramatically increase the range of errors that can be detected and corrected. The GVR project is collaboration across programming model, runtime, hardware architecture, and operating system with strong partnerships to CESAR and NWChem as driving applications.

Objectives

- Understand and create application-system partnership for flexible management of deep error detection and resilience, including level of resilience, cost, and recovery.
- Explore efficient implementations of resilient and multi-version GVR data including runtime and architecture support opportunities
- Empirical understanding of GVR's effectiveness for applications and higher-level programming systems as well as GVR performance requirements

Impact

- Incremental, portable approach to resilience for large-scale applications
- Flexible, application-managed cost and coverage for efficient resilience

Research Challenges

- Understand application needs for flexible, proportional resilience
- Design of API suitable for use by application developers, library developers, and programming tool back-ends
- Efficient GVR runtime implementations for multi-version memory and flexible resilience (acceptable to HPC applications). Understand of application performance reqs for GVR
- Understand and create evidence for architecture support and its benefits (i.e. intelligence in memories such as HMC or PIM)
- Explore opportunities created by GVR abstractions and underlying implementation technologies

Research Products and Artifacts

- Design of GVR API for flexible resilience and multi-version global data
- Research prototype software to evaluate GVR API with key mini-apps and libraries
- Research prototype software of GVR novel runtime techniques to evaluate cost and effectiveness
- Assessment of opportunities and quantitative benefits of GVR architecture support

THE UNIVERSITY OF CHICAGO

Argonne NATIONAL LABORATORY

U.S. DEPARTMENT OF ENERGY

Office of Science