Exploiting Global View for Resilience (GVR) – Progress Report (March 2013)

Andrew A. Chien, University of Chicago and Argonne National Laboratories
Pavan Balaji, Argonne National Laboratory

Staffing

We have made rapid progress in staffing, and building a coherent team across the project sites. As planned we have held monthly "All-Hands" meetings, alternative between the University of Chicago and Argonne (our 6th is April 3, 2013). In September 2012, the University of Chicago team included Dr. Hajime Fujita (postdoc), Zachary Rubenstein (graduate students), Professor Guoming Lu (visitor), and is led by Prof. Andrew A. Chien. Since January 2013, we have added Dr. Ziming Zheng (postdoc), and Aiman Fang (graduate student). At Argonne, since October 2012, the team included Dr. Kamil Iskra, as well as a small fraction of time of Pete Beckman, Jim Dinan, and is led by Pavan Balaji. In April, a new postdoctoral researcher, Wesley Bland, will join the team and work full-time on GVR.

Technical Progress

1. Design and Evaluation of Three Generations of GVR APIs

Working closely across the UChicago and Argonne teams we tapped collective creativity and deep experience to develop a set of use cases then design and evaluate, and rapidly improve 3 generations of GVR API's. These design iterations focused on three major design elements – globally-visible distributed structures, efficient versioning, and error signaling/recovery.

Creating Globally-Visible Distributed Structures Critical choices include whether distributed structures are composed bottom up, based on node-local allocation, enabling local control of data layout and addressing structure or allocated globally with the global data structure runtime managing distribution and node-local layout. The federation approach has significant advantages for efficiency and library interoperation; the latter for resilience and flexibility of decoupling memory servers from clients. After careful evaluation of interface options, we decided both modes of use are essential, settling on a hybrid interface that allows globally-visible distributed structures to be created in both fashions. In the federated case, we provide GDS_access to allow a process direct access to its portion of the global array. We also discussed whether GVR's distributed structures would be accessed exclusively as binary data, or support a typed interface and higher dimensional arrays. To enable the library to optimize multi-dimensional structures,
Efficient Versioning GVR's approach to resilience depends on application annotations for consistent snapshots, so versions can be created cheaply. However,

to manage the cost of versions, that application annotation should reflect the option for the runtime to create a version, not a requirement. After several iterations ranging from random-access versions to prev-next, our final design defines, access operations (put, get) w.r.t. the current version, and a version_inc() call demarcates logical version boundaries. This careful choice ensures that version_inc() can be implemented without creation of a version. Our design also enables application programmers to use nearly identical code idioms (particularly for synchronization) to traditional global view models such as Global Arrays, enabling easy migration to resilient execution.

Error Signalling/Recovery Perhaps the most difficult issue in the GVR interface (and system) is the application-system interaction for error signaling and recovery. Critical challenges include how to seamlessly incorporate hardware and OS signaled errors (asynchronous), runtime and application signaled errors (synchronous), as well as how to flexibly manage node local recovery, regional recovery, and global recovery. . In general, it appears that the most viable solution is to signal errors asynchronously, but resolve them at specific points in code. In addition, we currently leave the level of coordination up to the programmer to specify, since it appears that both coordinated and uncoordinated strategies are appropriate for different real-world use cases. The latest GVR API (v 0.71) addresses these issues consistently, but has yet to be implemented, much less be used in application experiments – both subjects of future work. Furthermore, the current API is a rather low-level functionality, and to enable most applications, we expect that a collection of "common case libraries" are both needed, and will be widely useful.

2. Design of the GVR Software Architecture To be used in HPC applications, the GVR system must not only deliver a successful resilience partnership but it must also achieve efficient and scalable performance. Our basic design reflects an architecture that can exploit traditional message-passing hardware, advanced features such as RDMA, and both flexible, efficient versioning. Of course, significant research and implementation efforts are needed before this promise can be realized. The GVR runtime system software includes a client side and target side. The client side provides global, consistent, and multi-version view of an array. It presents the API, and manages data distribution, consistency, and versioning across multiple targets. The target side preserves multi-version efficiently, maintains metadata for versions, and restores them on demand.

A critical element of the target side is the local reliable data store (LRDS) which provides node-local management of data, including support for multiple versions, in support of target-side distributed version management. It provides an efficient interface to target-side global data, particularly for data movement, differencing, compression, etc. The current implementation provides two prototype backends for tracking changes to limit storage requirements across multiple versions: one based fully on user-provided hints, the other--currently being integrated—taking advantage of page-based memory protection for a more transparent tracking. We plan to investigate a kernel-based implementation for lower overhead and maximum transparency, and to utilize hardware dirty bit tracking when such

becomes available. Multiple versions of the data are currently maintained in main memory. Preliminary implementation of storage API built on top of a POSIX file system just became available and should be used as the data store for older versions soon; in the future we plan to extend this to utilize NVRAM.

3. Development of Limited, Basic GVR Prototype

We have developed an initial research prototype of GVR runtime system. While still incomplete in several areas, current functionality enables experiments with mini-apps. The current GVR prototype is implemented as a library built on top of MPI-3, and integrates LRDS and a simple memory-based multi-version checkpoint is supported. We have created two GVR-enabled Mantevo mini-apps (miniFE and miniMD), demonstrating modest source code changes can be used to incorporate resilience, application error checking, and application recovery. Limitations of the system are too long to enumerate, including OS error integration, performance, etc.

4. Multi-version Checkpoint Modeling A novel GVR feature is low-cost, application-controlled versioning of data structures. We built a formal model based on Markov chains, and analyzed a range of realistic current and future HPC systems to explore the question – when do latent (or silent) errors require multi-version checkpoints to support robust execution (the ability to run large jobs to completion), and to support high system efficiency (less than 10% loss due to restarts and lost computation). The results show that between 2 to 10 checkpoints are required for realistic exascale system configurations and error rates, and that reductions in checkpoint cost (such as proposed in SCR), are productive but do not obviate this need.

5. Ongoing and Future Efforts

In the next six months, focused GVR project efforts will include:

- Continued GVR implementation efforts, progressing towards a full API implementation, and robust functionality to enable co-design application experiments
- Exploration of efficient implementation of redundant, distributed global-view data structures, including challenges of
- Exploration of efficient multi-version snapshot capture and storage techniques, including consistency, synchronization, compression, and restoration
- Experiments with co-design applications (OpenMC, MD, others) to explore the match of GVR capabilities with common application structures – refine API and demonstrate potential
- Working with OS/runtime community on cross-layer error handling classification and naming

Publications and Key Documents

- tbd, Creating Robust Iterative Solvers Using Global View Resilience, in preparation, expected April 2013.
- Guoming Lu, Ziming Zheng, and Andrew A. Chien, *When are Multiple Checkpoints Needed?*, to appear in Fault Tolerance at Extreme Scale, (FTXS), June 2013.
- Hajime Fujita, Robert Schreiber, Andrew A. Chien, *It's Time for New Programming Models for Unreliable Hardware*, in ACM Conference on Architectural Support for Programming Languages and Operating Systems, March 18-20, 2013. (Provocative Ideas session).
- *The Global View Resilience Application Programming Interface*, Version 0.71, February 2013, contact GVR team at UChicago/Argonne for copies.

Prior relevant work

- Sean Hogan, Jeff Hammond, and Andrew A. Chien, *An Evaluation of Difference and Threshold Techniques for Efficient Checkpointing*, 2nd workshop on fault-tolerance for HPC at extreme scale FTXS 2012 at DSN 2012