COVERT: Configurable Virtual Redundancy with Transparent Availability on Commodity Software

Overview

Scaling integrated circuit technology into the deep submicron regime is expected to increase both soft and hard error rates significantly [1]. Therefore, providing high availability in the presence of relatively unreliable components is likely to become an increasingly important requirement for a diverse set of systems, including general-purpose commodity systems. Traditionally, high availability systems have used specialized hardware and software that cater to a small subset of the application domain like banking and mission critical applications. For example, the HP NonStop system [2] uses a combination of enhanced commodity hardware and specialized system software to create and manage the synchronization of redundant processes. The IBM zSeries systems [3] and Stratus [4] use custom designed hardware, system software, and firmware to provide high availability.

Custom designed hardware and/or software is not a viable solution for the cost-competitive commodity market. Both commodity hardware and software are resistant to significant changes, particularly if they affect the performance in non-fault tolerant configurations. Systems that use specialized system software are restricted by their ability to run only applications written for a particular OS, e.g, the NonStop kernel [1] and VOS [3]. In this poster, we present a technique for enabling the use of "off the shelf" general purpose software in high availability systems running on general purpose Chip Multiprocessors (CMPs).

We propose enhancing a virtual machine monitor (VMM) to enable it to perform the same functions as a specialized high availability operating system, e.g. NonStop, with no changes to any commodity software, including the OS, runtime software, and application software. We call this solution "COVERT" – Configurable Virtual Redundancy with Transparent Availability. The COVERT software manages the creation, synchronization and output comparison (at I/O level similar to NonStop OS) of redundant threads and performs recovery in the event of a failure (Fig. 1(a)). The VMM approach used by COVERT provides transparent high availability to all conventional software – legacy, current, and future. The software is aided by configurable hardware that can provide fault isolation to the redundant threads [5].

The VMM creates a redundant replica by cloning a guest virtual machine that needs high availability. Then, it uses a state machine based approach to synchronize the duplicate VMs. The VMM needs to ensure that all inputs to both copies are identical and are delivered to the VMs at an identical state. Given that the two VMs are identical and the inputs are identical, their outputs should be identical, except for the occurrence of a hardware error, which can be detected by comparing the outputs. A significant design issue is maintaining identical inputs. We classify all the inputs to the VMs as deterministic or non-deterministic. The deterministic inputs can be passed directly to the guest VMs. However, the non-deterministic inputs must be coordinated by COVERT to appear identical to the duplicated VMs. COVERT is also responsible for creating checkpoints and output comparison to detect hardware errors. The poster will present a detailed design of COVERT software. We have evaluated the feasibility of the proposed design with an analytical model (Fig. 1(b)). Based on this model, we show that overheads for several compute intensive and I/O intensive benchmarks can be restricted to less than 20% (Fig. 1(c)).

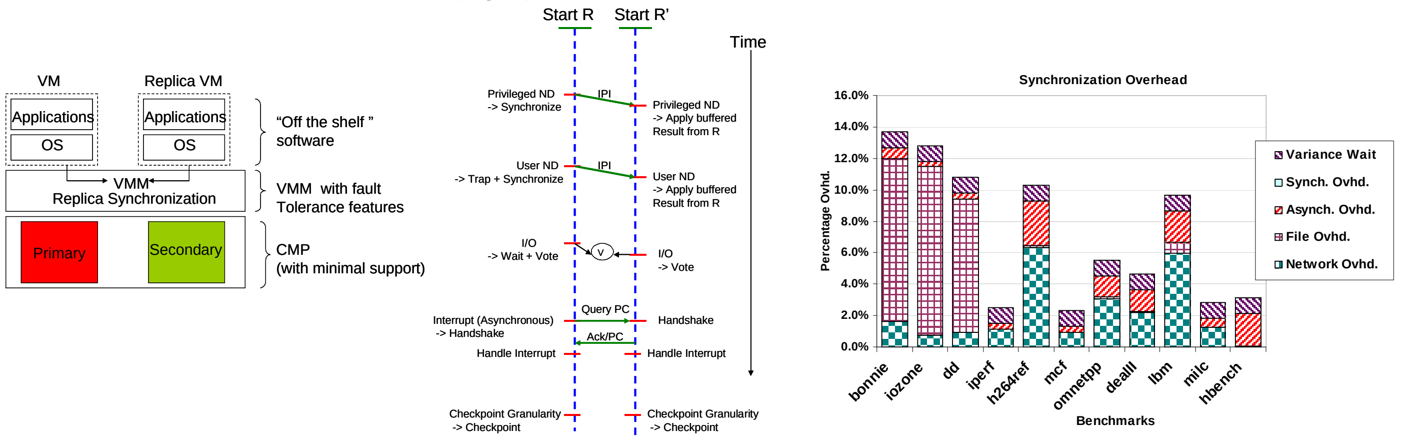

Figure 1. (a)VMM based replica co-ordination. (b) Analytical model events with overhead (c) Synchronization overhead for various benchmarks

Why the poster is appropriate for ASPLOS audience?

The poster tackles the important problem of increasingly common hard and soft errors [1] that would result in significant downtime and loss in revenue. We believe that the poster is particularly suited for an ASPLOS audience because it presents an integrated hardware and software technique to build high availability systems using commodity hardware and software. As soft error rates become increasingly relevant for commodity multicore platforms, integrated hardware and software approaches that provide configurable and transparent availability on commodity implementations are likely to become very important.

Novelty

This poster presents a novel method to provide transparent availability to all conventional software on commodity multicore platforms while requiring only small changes in the hardware and virtual machine software. This is in contrast to traditional fault tolerant systems that use custom and proprietary hardware and software to provide high availability. It is also a major improvement over extensively modifying an existing OS to have high availability features. The closest related prior work is hypervisor based fault tolerance by Bressoud et al. [6]. However, their solution only handled hard errors and did not consider CMPs. To the best of our knowledge, ours is the first proposal to provide detection and recovery from both hard and soft errors for commodity software running on general purpose CMPs.

References

1. Borkar, S. *Challenges in Reliable System Design in the Presence of Transistor Variability and Degradation*. IEEE Micro, vol. 25, no. 6, 2005, pp. 10-16.
2. Bernick, D., Bruckert, B., Vigna, P. D., Garcia, D., Jardine, R., Klecka, J., and Smullen, J. *NonStop® Advanced Architecture.* Conf. on Dependable Systems and Networks, 2005.
3. Fair, M.L. et al. *Reliability, Availability, and Serviceability (RAS) of the IBM eServer z990.* IBM Journal of Research and Development, Nov, 2004.
4. Webber, S. and Beirne, J. *The Stratus Architecture.* International Symposium on Fault Tolerant Computing, 1991.
5. Aggarwal, N. et al. *Configurable Isolation: Building High Availability Systems with Commodity Multi-Core Processors*. International Symposium on Computer Architecture, 2007
6. Bressoud, T. C. and Schneider, F. B. *Hypervisor-based fault tolerance.* ACM Trans. Computer Systems 14, 1 (Feb. 1996), 80-107.