

# Smart Restartable Snapshots on top of a Distributed Transactional Memory

Steffen Gerhold<sup>1</sup>, Michael Schöttner<sup>2</sup>, Markus Fakler<sup>1</sup>, Michael Sonnenfroh<sup>2</sup>, Peter Schulthess<sup>1</sup>  
steffen.gerhold@uni-ulm.de  
Student: no

<sup>1</sup> University of Ulm, 89069 Ulm, Germany

<sup>2</sup> University of Duesseldorf, 40225 Düsseldorf, Germany

Fault tolerance for cluster operating systems (OSs) can be achieved by relying on redundant hardware modules to compensate for malfunctioning devices. A well-known software approach is to store periodically distributed consistent snapshots of single applications or the entire cluster on durable storage. If a critical error occurs, a node is restarted from a former consistent snapshot without losing all its computation work before this checkpoint. These backward error recovery techniques require on the one hand saving a consistent distributed snapshot including application and operating system state information and on the other hand the ability to restart a cluster from such a snapshot.

Checkpointing an application in traditional OSs like Microsoft Windows or Linux is a non-trivial task [1] [2]. Solely saving the user context of the target is not sufficient, as processes depend on additional information stored inside the kernel space, e.g. the process id or file descriptors. When the application is restarted, the entire process context must be restored, which includes the user context as well as the affected kernel space structures. In a distributed cluster system each participating node must also store its relevant local application and system states in a coordinated way to guarantee a cluster-wide consistent snapshot.

We have implemented smart restartable snapshots within our cluster operating system named "Plurix". A key feature of Plurix is the distributed transactional memory (DTM) storing applications, the kernel, and device drivers [3]. All nodes execute speculative transactions operating on the DTM. Write operations are bundled within transactions transforming the cluster system to the next consistent state. This simplifies the implementation of checkpointing as there is no need for an additional global coordination. When a new snapshot is to be taken, it is sufficient to write asynchronously the modified parts of the DTM on a hard disk without considering consistency and dependency aspects.

Because of the speculative execution transactions may be aborted in case of conflicts requiring all operations to be reset. This is an easy task for memory but other hardware does typically not support "transactional consistency" [4] and especially no transparent abort. Smart buffers bridge the gap between restartable-transactional and interrupt-driven space guaranteeing transactional consistency of the DTM image while at the same time avoiding lost device data in case of a transaction abort [3].

Although smart buffers make most device accesses restartable they are not sufficient for restarts caused by a fallback. Internal device states are typically not subject to transactional consistency thus raising functional inconsistencies if the current device states differ from the states during the checkpointing. A graphics adapter for example accepts different drawing commands depending on it being in text mode or in graphics mode and will not render the desired results while running in an inappropriate mode. To avoid this each device driver reflects its internal state in a special fallback information object which resides inside the DTM and therefore becomes part of the checkpoint. If a fallback occurs, the driver reinitializes its device to enter the state specified in this object, thus extending the consistent image to critical device states as well.

Restarting the system from a previously recorded checkpoint is not difficult because (re-)starting over from a former state is an inherent feature of our transactional task model and occurs transparently to applications and most system modules during normal execution. Even in the worst case where a reboot of the entire cluster is necessary the quick boot capabilities of Plurix ensure a ready-for-use cluster within several 100 milliseconds.

Currently, we are designing and implementing a fault-tolerant distributed checkpointing solution together with dependency tracking support to minimize the number of nodes which have to be reset after a failure. This ongoing work is funded by the German Science Foundation (DFG).

## References:

- [1] G. Vallée, R. Lottiaux, D. Margery, C. Morin and J.-Y. Berthou. G., „Ghost Process: a Sound Basis to Implement Process Duplication, Migration and Checkpoint/Restart in Linux Clusters“, in: Intl. Symposium on Parallel and Distributed Computing, Lille, France, 2005.
- [2] Paul H. Hargrove and Jason C. Duell, „Berkeley Lab Checkpoint/Restart (BLCR) for Linux Clusters“, in: SciDAC 2006.
- [3] [www.plurix.de](http://www.plurix.de)
- [4] M. Wende, M. Schoettner, R. Goeckelmann, T. Bindhammer, P. Schulthess, „Optimistic Synchronization and Transactional Consistency“, 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2002.