Distributed Transactional Memory for Clusters and Grids

EuroTM, Paris, May 20th, 2011

Michael Schöttner

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Distributed Transactional Memory for Clusters
The Plurix Operating System

- **Plurix** project (1996-2004) at Ulm University
  - with Prof. Peter Schulthess
  - http://www.plurix.de
- **Distributed Operating System** for PC-based clusters
- **Language-based approach using Java**
  - Everything (including kernel and drivers) written in Java

Transactional Distributed Shared Memory

- **DSM with transactional consistency = DTM**
- Data exchange between nodes

  through DTM not through message passing
- **Almost everything stored in the DTM**
  - Application data and code
  - Including kernel and drivers data and code (except driver buffers)
  - Smart buffers for device access within transactions

Transactional Distributed Shared Memory

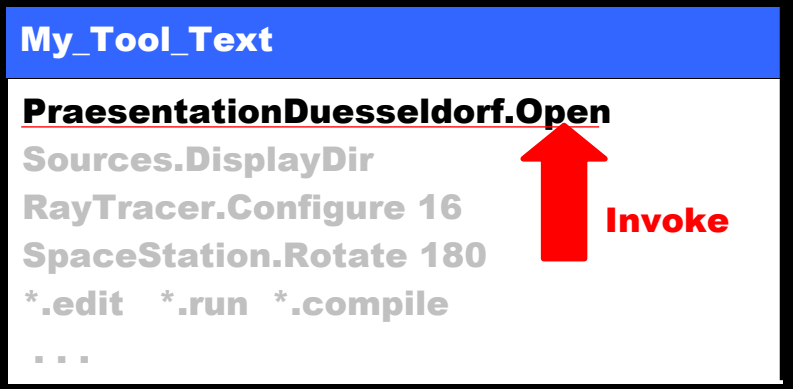
- **MMU-based memory access detection**
  - Automatic shadow copy creation
- **Restartable transaction with optimistic synchronization**
  - First wins strategy
  - Serializazion implemented by a token
  - Write sets send to other nodes → forward validation
- Commit numbers stored in token used to recover from missed commits

Distributed Transactional Memory for Clusters
Programming Model

- **Event-driven architecture**

  → implicit transactions

  - Programs provide commands
  - Users execute commands
  - System automatically inserts BOT and EOT



- **Explicit transactions**

  - For parallel computations
  - Reduce conflict probability

Distributed Transactional Memory for Grids
Object Sharing Service

- **XtreemOS** Project (2006-2010)

  EU FP6 project

- **Linux-based Operating System** for grids

- **Object Sharing Service**
  - Includes a DTM
  - Implemented in C (shared library)
  - Objective: Simplify distributed state management

Object Sharing Service

- **Replica management**
  - Shared objects are automatically replicated
  - Update vs invalidate (adaptive approach)
- **Consistency management**
  - Supporting different consistency models
  - Including a **distributed transactional memory**

Distributed Transactional Memory for Grids

Programming model

- Shared objects explicitly allocated (PGAS-model)
- Speculative transactions defined by the programmer
- MMU-based access detection

  (millipages avoiding false conflicts)
- Deferred abort when in transaction-unsafe code

Distributed Transactional Memory for Grids
Commit protocols

- P2P-based commit protocol
  - Using a token, first wins strategy & forward validation
  - but optimized by request queue and holder prediction
- Coordinated commit protocol
  - Backward validation on coordinator
  - Combined with forward validation on clients
- Super-peer-based commit protocol
  - Allowing group-local commits,
  - If data is not replicated outside groups

Lessons learned

- Evaluation performed on clusters and grids
  - Up to 128 nodes on the Grid'5000 platform
  - Parallel benchmarks (SOR, ray tracing, pi, ...)
  - MMVE: Wissenheim → http://www.wissenheim.de
- Unfortunately, conflicts are **not** seldom
  - Requires programer to reason about access patterns, transaction boundaries, local and shared data, ...
- False conflicts
  - Reduce conflict unit size ←→ bulk network transfers

Distributed Transactional Memory for Clouds
Lessons learned

- Even if there are no conflicts we have the commit overhead
  - Local commits help (partially)
- Difficult to adapt existing applications/algorithms
- It is easier to write new transactional apps but ...
- Non-transparent optimizations, e.g. allowing to read old versions make development harder

  → Same story as for consistency models in DSM systems?

DTM perspectives for cloud applications

- DTM useful for distributed applications running in data centers where strong consistency is required
- Server-based MMVEs
- Extended MapReduce applications
- Transactional caches for Web applications (see CloudTM, TxCache, CumuloNimbo)