FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Diplomarbeitspräsentation

computer languages

Relaxed non-blocking Distributed Transactions for the eXtensible Virtual Shared Memory

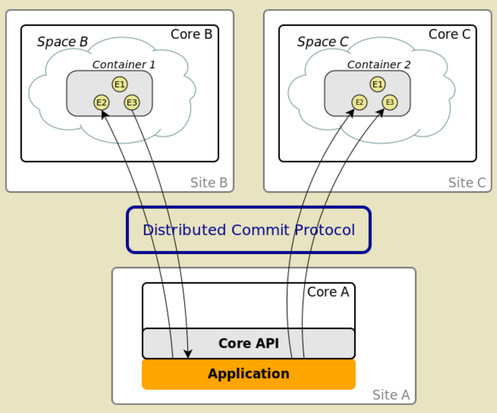
Andreas Brückl

Masterstudium:
Technische Informatik

Technische Universität Wien
Institut für Informationssysteme
Arbeitsbereich: Programmiersprachen und Übersetzerbau
BetreuerIn: A.o. Univ. Prof. Dr. Dipl.-Ing. eva Kühn

Problem Statement

Spaced-based middlewares provide coordination mechanisms for distributed applications by using a virtual shared memory called space. The spaced-based middleware XVSM (eXtensible Virtual Shared Memory) should be extended by distributed transactions so that it can cope with more complex business requirements. The new transaction model has to fulfil the following requirements:



- Support of long lived transactions (LLT)
- Support of asynchronous transactions
- High concurrency and performance
- Focus on transactions for coordination
- Support of distributed, ACID compliant transactions

Related Work and Design Approach

There are a lot of different distributed transaction models in the literature. In general they can be separated into two groups:

atomic commit protocols

- representatives
  - Two-Phase Commit (2PC)
  - Three-Phase Commit (3PC)
  - Paxos Commit
- characteristics
  - Strict ACID compliance
  - Coordination by application

relaxed transaction models

- representatives
  - Sagas
  - Flex transaction model
  - WS-BPEL
- characteristics
  - Relaxed ACID compliance
  - Support of sub transactions
  - Early commit and compensation
  - Coordination by transaction manager

To fulfil all requirements two new transaction models have been developed:

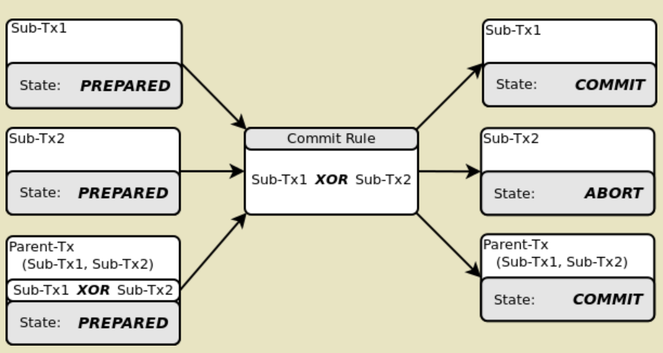
Extended Paxos Commit

- covers ACID compliant distributed transactions

REXX Transaction Model

- covers the relaxed behavior with high concurrency and additional features

Extended Paxos Commit



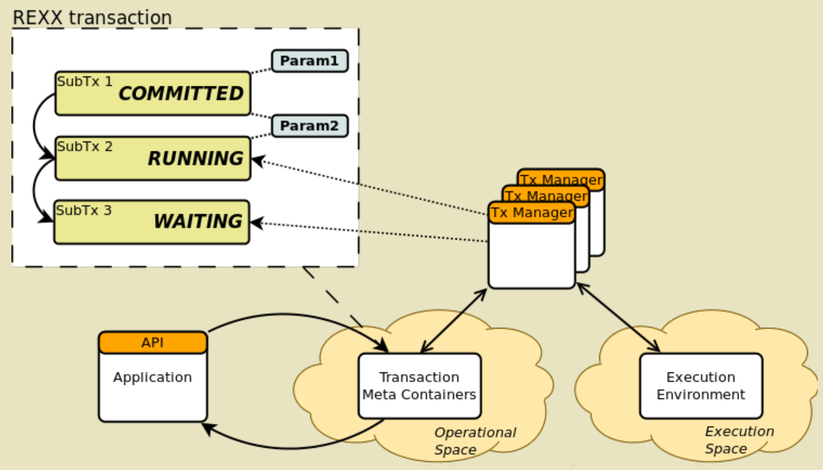
How does it work?

- Paxos Commit is used for all sub transactions
- Several coordinators (acceptors) guarantee consistency and availability
- The Paxos leader collects the results of the sub transactions and applies the commit rule
- The commit rule determines the final outcome (*commit* or *abort*) of the parent transaction and of all sub transactions

Characteristics

- ACID compliant
- Support of sub transactions
- Different types of Commit Rules
  - Default Commit Rule
  - Threshold-based Commit Rule
  - Logical Commit Rule
- Support of function replication and low priority transactions

The REXX Transaction Model



How does it work?

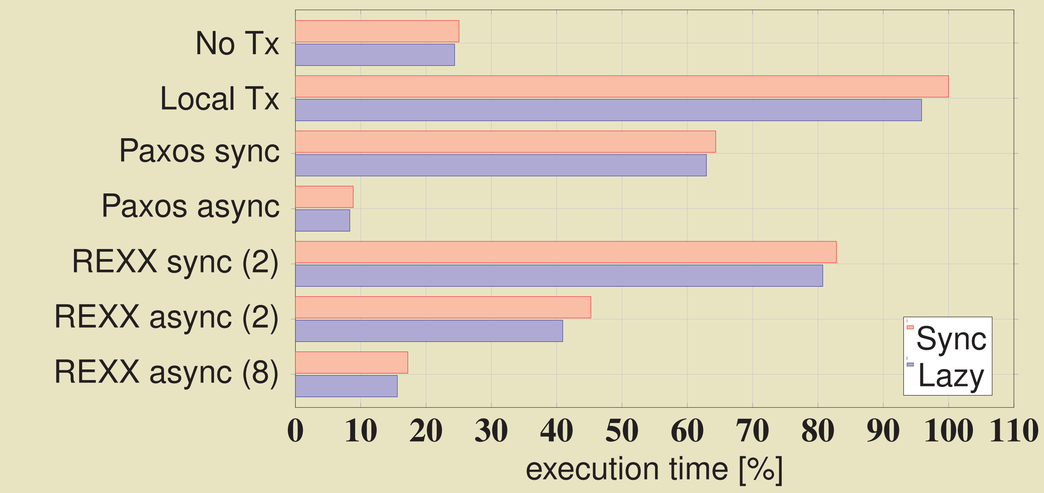
- REXX transactions consist of sub transactions and dependencies
- All sub transactions are processed individually
- Every transaction manager processes exactly one sub transaction
- High parallelism by using several transaction managers
- Parent transactions aggregate the states of their sub transactions
- Compensation of committed sub transactions can be scheduled automatically

Characteristics

- Support of LLTs
- Internally uses Extended Paxos Commit in sub transactions
- Asynchronous and synchronous execution
- Support of nested sub transactions
- Semantic compensation
- Dependencies between sub transactions
- Parameter passing between sub transactions
- Flexible timeout handling

Evaluation

The two presented transaction models have been implemented in MozartSpaces, which is the reference implementation of XVSM. For the comparison of the performance the following scenario has been used: An application located on *Site A* writes one entry to two spaces which are located on *Site B* and *Site C*. The scenario has been executed with different transaction models and different persistency profiles (*Lazy* and *Transactional Sync*).



Notes

- *sync* - The application waits until a transaction has been completed before it starts the next one
- *async* - All transactions are asynchronously committed but their results are collected synchronously
- The number in the parentheses specifies the number of parallel running transaction managers
- Implementation without transactions is slower since MozartSpaces uses an implicit transaction for every operation if no explicit transaction is specified

Conclusion

MozartSpaces now supports ACID compliant distributed transactions as well as long running business processes. The execution time of distributed transactions is reduced by up to 90%. This improvement is caused by the expensive local commit operations at the remote spaces. The implementation using local transactions executes the commit operations sequentially whereas the presented models execute them in parallel by using sub transactions. The more sub transactions are used the higher is the performance improvement.

Kontakt: brueckla@gmx.net