Ganymed / Satellites / DB Farm

Seminar in Distributed Computing
Table of content

- Motivation
- Ganymed
  - RSI-PC
  - Architecture
  - Evaluation
- Satellites
- DBFarm
Motivation

- Data grids, large scale web applications etc. overcharge DB engines
- Replication is the solution to scalability issues
- Existing solutions do not behave like a single database
  - Asynchronous (lazy) replication
  - Consistency is abandoned in favour of scalability
RSI-PC

- Replicated Snapshot Isolation with Primary Copy
  - Scheduling algorithm
  - One primary copy as master
  - Arbitrary number of replicated slaves

- Implemented as transaction scheduler
  - Scheduler distinguishes read-only and update
Snapshot Isolation (SI)

- Every transaction sees a consistent data state
- Conflicting updates are resolved by the first-committer-wins rule
First-committer-wins

- T4 is aborted since \( \text{start}(T4) < \text{end}(T3) \) and T3 and T4 both update Y
- No other conflicts here
Isolation levels

- **SERIALIZABLE**
  - Uses row level write locks
  - Only sees updates committed before transaction start
  - Transaction that commits later is aborted
  - Potential deadlocks, resolved by the database

- **READ COMMITTED**
  - Sees all committed updates
  - No aborting
  - Fuzzy reads possible
  - Default in Oracle and PostgreSQL
R S I-PC updates

- All updates are handled by the master
- After a successful commit the master extracts the *writeset* of the transaction
- The *writesets* are sent to all slaves in the same order
- *Writesets* are tagged with a global database version number
Writeset

- Optimization of the *update-everywhere* approach
- Propagation of database changes instead of SQL-statements
**RSI-PC reads**

- The scheduler distributes reads to the slaves
- Round robin
- If the slave has not yet applied all committed updates, the read is delayed

- Client is able to define a staleness threshold to shorten the delay
RSI-PC

- No SQL statement parsing
- No concurrency control operations
- No locking at scheduler level

-> Very fast and „thin“ middleware
Ganymed
Ganymed

- **Scheduler**
  - Implements RSI-PC
  - JDBC driver
  - Behaves like a single SI based database
  - Distributes transactions over master and slaves

- **Manager component**
  - System monitoring
  - Configuration
  - Graphical interface
Ganymed scheduler

- JDBC Driver
  - Standard interface
  - Easy integration
  - Easy migration from centralized systems
  - Client application has to mark read-only transactions
    - By `Connection.setReadonly();`
  - No support for writeset extraction
    - Has to be done in the database

- Behaves like a single database
  - Transparent to clients
Ganymed scheduler

- Implements RSI-PC algorithm
  - No support for loose consistency models
  - Read-only transactions always assigned to a slave
    - Even if the master replica has free capacity
  - A FIFO queue for each replica is used to distribute writesets

- Supports PostgreSQL and Oracle replicas
- Heterogenous environment (also OS-wise)
- Dynamic management of replicas
**Ganymed manager**

- **System monitoring**
  - Replica loads
  - Component failures
    - Replica failure is handled by the scheduler
    - On scheduler failure, the manager tries to start a backup scheduler

- **Configuration**
  - Adding and removing of replicas
  - System startup
    - Choice of master replica
Ganymed evaluation

- Different Ganymed configurations tested against single PostgreSQL DB
  - Load generator simulating TPC-W application server

- Testing of component failure
TPC-W benchmark

- **Transactional Processing Council**
- Simulates real world transactional web applications
- Three different types of workload
  - WIPS: shopping
    - 80% read-only
  - WIPSb: browsing
    - 95% read-only
  - WIPSo: ordering
    - 50% read-only
Performance and scalability

- Throughput and response time measurements
Component failure

- Reaction to slave and master failure
**Extending DBMSs with satellite databases**

- Extension of Ganymed
- Dynamically create satellites
- Satellites can add functionality
Satellites

- Primary satellites are used to create secondary satellites without bothering the master
- Different load distribution policies can now be selected: *round-robin*, *least-pending-requests-first*, *least-loaded*
Satellites

- Several isolation levels are now supported on the master
  - Slaves respond to queries always using SI
- Satellites can appear and disappear without causing any loss of data
- SQL parsing is done in case of single queries to optimize routing
  - Slaves with only part of the tables can be used
Dynamic creation of satellites

- **Copy (used in Ganymed)**
  - Just copy a snapshot from the master or a ready satellite and start applying writesets

- **Writeset replay**
  - If a history of writesets has been kept, they can be applied to an old copy of the system
  - Especially useful if a satellite has left the setup and is then reentering

- **Hybrid**
  - Decide for each object (table) whether to copy or to replay its writesets
Dynamic creation of satellites

- **Physical copy (used in Ganymed)**
  - Directly transferring DBMS table space files

- **Logical copy**
  - Extracting and importing data using queries
The PITR approach (point-in-time-recovery)

- Take a copy of the primary satellite
- Submit a marker transaction
- Copy the REDO logs
- Apply all transactions that occurred after the first step (get writesets from primary satellite)
- Apply all writesets received since activation of the queue
DBFarm: A Scalable Cluster for Multiple DBs

- Primary copy, read-only satellites scenario with multiple DB instances
- For use on clusters
- Potential use as a service provider
  - Clients see a single consistent image
DBFarm

- Scalability up to several hundred DB instances
- Again, load distribution is done by having one master for updates and read-only slaves
- Later versions use Ganymed as single SI database
Summary

- Idea is the same in all three papers
  - Write one, read many
  - Thin middleware-layer for load distribution
Thank you. Questions?