Integrating Transaction and Replication Management Services in CORBA

Hadi Salimi
Computer Engineering Department
Iran University of Science and Technology
Tehran, Iran
h_salimi@mail.iust.ac.ir
January 14, 2006

Where Are We?

Object Orientation 60s
Distributed Systems 80s
Fault Tolerant Software 70s
CORBA 90s
Availability
Safety
FT-CORBA
OTS
We Are Here!!!
CORBA

- It stands for Common Object Request Broker Architecture.
- OMG, (a consortium over 800 companies) has produced the CORBA standard.
- CORBA [1] allows objects to invoke services from other objects, hiding differences in location, programming language or platform.

CORBA, A Bird-View

Adopted from [2]
CORBA Features

- Uniform view of the world: everything is an object
- Uniform view of communication: via request invocation
- Communication between objects independent of:
  - Programming languages
  - Physical locations
  - Platform types
  - Networking protocols
- Provides useful Object Services
  - change management, naming, transactions, security, query, etc.

Interoperable Object Reference (IOR)

- IORs are used by the ORB to transparently locate objects
Software safety can be defined [7] as features and procedures which ensure that the likelihood of an unplanned event is minimized and its consequences are controlled.

- Thereby preventing accidental injury or death, whether intentional or unintentional.
Availability

- Measure of the time that the system is available for use
- \[ A = \frac{MTBF}{MTBF + MTTR} \]
  - MTBF is Mean Time Between Failures
  - MTTR is Mean Time to Repair

Software Techniques

- Dependable Software Attributes
  - Safety
  - Availability
  - Redundancy
    - Time
    - Data
    - Software
  - Transactions
Transaction

- It was originally developed in the context of database management systems.
- From a database point of view it is a very elegant way to keep the data consistent even in the presence of highly concurrent data accesses [4].

There are two mistakes one can make along the road to truth. Not going all the way and not starting.

---Buddha

Transaction (cont.)

- Distributed Systems’ point of view
  - They allow a process to access and modify multiple distributed resources as a single atomic operation [8].
  - If the process backs out halfway during the transaction, everything is restored to the point just before the transaction started.
Redundancy

- Software redundancy includes additional programs, modules, functions, or objects used to support fault tolerance [9]
  - Data
  - Time
  - Software

CORBA & FT

transactions CORBA Redundancy

Object Transaction Service Fault Tolerant CORBA
OTS Architecture [10]

Transaction Client

Transaction Server

Recoverable Server

Context

Begin or End Transactions

Transactional Object

Recoverable Object

Resource

Context

Not Involved in Transaction Completion

Registers Resources

2-Phase Commit Protocol

Transaction Service

Transaction Coordinator

Context

FT-CORBA [11]

- At 1998 OMG issued the RFP
- In early 2000 the first version released
- The last version, December 2001
- Using
  - Replication
  - Object Groups
Replication

- The various approaches to fault-tolerant CORBA are alike on their use of replication [12].
- The behind idea is to mask the failure of an object by making extra objects.
- In the case of a failure, the fault tolerant ORB transparently redirects a failed request to a live replica.

Replication Models

*How to keep replicas consistent?*

- **Object Replication**
  - **Active**
  - **Passive**
    - **Warm**
    - **Cold**

Active:

- Rep 1
- Rep 2
- Rep 3

Passive:

- Primary
- Backup 1
- Backup 2
Object Groups

- Object group represents a replicated object and the group members represent the individual replicas of the object.
- Each object group has an Interoperable Object Group Reference (IOGR).
- Using an IOGR, a fault tolerant ORB is capable of accessing each object replica.

FT-CORBA Architecture
FT-CORBA Implementations

- Delta-4 [13]
  - 1990 – supported by CEC through ESPRIT Project
- Arjuna [14]
  - 1994 – Newcastle University
- Orbix-Isis [15]
- Electra [16]
  - 1995 – Zurich University
- DOORS [17]
  - 1997 - Bell Labs Research
- OGS [31]
  - 1998 – Swiss Federal Institute of Technology
- IRL [18]
  - 1999 – Rome University
- AQuA [30]
  - 1999 – Illinois University
- Eternal [19]

Why Integration?

| Replication     | Transaction
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>FT-CORBA</td>
</tr>
<tr>
<td>Tier</td>
<td>Control Tier</td>
</tr>
<tr>
<td>Attribute</td>
<td>Availability</td>
</tr>
<tr>
<td>Technique</td>
<td>Roll-forward</td>
</tr>
</tbody>
</table>
Roll-Forwarding

- Replication has brought a new termination style into transaction literature.
- Roll-forwarding means although there are errors during the execution of a transaction, the transaction can be committed safely, just by redirecting the failed request to a fresh replica.

Related Work

- The new models of replication and transaction are a bit different from their traditional concepts.
- As an example:
  - Concurrency is considered separately.
  - The support of transaction server variety is essential.
  - The granularity of transactional elements is coarser.
Related Work (cont.)

- So, the related works can be categorized into:
  - Integration of traditional concepts
    - Is not interested here, you can refer to [21], [22], [23], [24], [25] and [26].
  - Integration of new concepts
    - Have mainly focused on reconciling replication and transaction in distributed systems, specially the ones that are built my means of CORBA.

Related Work 1

- *Felber* and *Narasimhan* who are pioneers in FT-CORBA, have issued a new publication [27] recently.
- They have indicated that reconciling replication and transaction concepts in CORBA is still an open issue.
Related Work 2

- Froland and Guerraoui have claimed that [28] the current standards cannot be integrated due to some limitations:
  - OTS does not support fine-grained replication.
  - Duplicated elimination cannot be guaranteed.
  - Output determinism problem.

Related Work 3

- Felber and Narasimhan have introduced a protocol [20] in order to use transactional operations on replicated objects.
- They have clarified their protocol with a simple bank balance transfer operation example.
Related Work 3 (cont.)

Roll-Forward Approach

Roll-Back Approach
Their proposed approach

Zhao and Moser [29] have shown that sending a multicast message to all transaction resources can decrease the overhead of 2-phase commit protocol.
Why they do not add up?

- Almost all of them suppose that middle tier objects are stateless.
- All of them use total order multicast protocols. The drawback of these protocols will be highlighted.
- All of them suppose the existence of nested transaction.
Total Order Multicast Protocols

- Suppose that O1 intends to make an atomic call to some objects.
- What will happen if O2 fail during this operation?

Total-Order Multicast Protocols

- What will happen if C fails after updating G1, but just right before updating G2?
Nested Transaction

- A nested transaction can be committed successfully if a failure occurs in one of its sub-transactions.

Replication-Aware Transactions (RATs)

- From another point of view, any failure in the scope of a transaction that is executing on a group of replicated objects can be easily ignored by using Replication-Aware Transactions (RATs):
  - In the case of stateless objects, redirecting a failed request to a live replica is the remedy.
  - But in the case of statefull objects, this technique is not enough.
In the case of statefull objects, omitting the object from its group and recreating it will solve the problem.

We call this termination style roll-over.

For each object recreation, the state of the object should be transmitted.
OTS Extension

```cpp
void ResourceManager::registerResource(
    CosTransactions::Resource_ptr r,
    const char* name) throw()
{
    ResourceRecord record;
    record.name = name;
    //... other initializations for record
    if (strstr(name, "~$Replicated$~"))
        record.setReplicated(true)
    else
        record.setReplicated(false);
    resources_.push_back(record);
}
```

```cpp
CosTransactions::Vote
ResourceManager::prepare()
{
    ....

    switch(v)
    {
        case CosTransactions::VoteRollback:
            if (res->isReplicated())
                resources_.remove(res);
            else return v;
    //other cases come here
    }
}
```

Fault Detection

- How to detect the:
  - Crash of H
  - Crash of Replicated Object
  - Crash of Resource Object
Group Failure

- What will happen if all replica objects fail due to a:
  - Network Failure
  - Rare but possible group failure
- In this case, the resource objects that belong to an object group should be distinguishable.

Replication Style Support

- Active Replication:
  - To update a group of active replicated objects, RATs can be helpful.
Replication Style Support (cont.)

- **Warm-Passive Replication:**
  - In this case, the primary object should update the backup objects in the scope of a RAT.

![Diagram of Warm-Passive Replication](image)

- **Cold-Passive Replication:**
  - Checkpointing should be performed in the scope of the transaction that the primary object participates in it.

![Diagram of Cold-Passive Replication](image)
Implementation

- My Implementations:
  - Extending OTS in such a way that can support roll-over approach.
  - Implementing a light-weight Replication Manager.
  - Implementing a prototype to evaluate our model.

Replication Manager

- Our implemented replication manager:
  - Keeps an object factory for each host
  - Recreates objects whenever they fail
  - Keeps object groups as a bunch of replica object
  - Assigns each object group a set of properties (e.g. minimum number of replica objects)
The Prototype

Adopted from [29]

Measurement Parameters

- We ran the implemented prototype with the change on:
  - Number of Replica Objects (N)
  - Failure Probability of a replica object (P)
  - Transaction Model (TM)
- And for each run we measured:
  - The overall transaction throughput (T) in terms of the number of committed transaction per second.
Performance Evaluation

- We compared the transaction throughput of our approach with the transaction throughput of:
  - Felber’s approach [20], which uses nested transactions
  - Zhao’s approach [29], that uses roll-back approach.

Experimental Results

- N=2

![Graph showing experimental results](image)
Experimental Results (cont.)

- N=3

![Graph showing experimental results for N=3]

Experimental Results

- N=4

![Graph showing experimental results for N=4]
Experimental Results

- N=5

![Graph showing experimental results for different values of N (2, 3, 4, 5) with three types of behavior: Roll-Back, Roll-Over, and Nested. The graphs illustrate the relationship between P and T for each case.]
Object Size

- The object size can affect the performance of RATs, because the object state needs to be transmitted on each object recreation.

Conclusion

- We showed that current replica consistency techniques that are based on total order multicast protocols cannot guarantee system safety.
- We presented a new transaction model that can be applied to replicated objects.
- In this model, a failure in the scope of a transaction that is running on a group of replicated objects can be ignored.
Conclusion (cont.)

- We implemented a prototype to evaluate this extension.
- The experimental results show that this model is particularly beneficial:
  - When dealing with crowded object groups
  - In faulty environments
  - For light weight objects

Other Open Issues

- Active_With_Voting (Quorum-Based or Consensus-Based) Replication [33]
- Applying transactions on active with voting replication style
- Fault Tolerant CCM
- Modeling of this approach [32]
- Transaction and Concurrency Integration
References


References (cont.)

References (cont.)


Thanks ...

Hadi Salimi
h_salimi@mail.iust.ac.ir
January 14, 2005