



Christian Spann, **Jörg Domaschka**, Franz J. Hauck |
joerg.domaschka@uni-ulm.de | September 10, 2009 |
Aspectix Research Team, Institute for Distributed Systems,
Ulm University, Germany

Reliability and Availability in the XtreamOS Project

XtreamOS Summer School 2009

XtreemOS vs Reliability

Targeting large peer-to-peer grids

- ▶ Off-the-shelf computers
- ▶ Connected via the Internet
- ▶ No central infrastructure, fully decentralised
- ▶ Churn, unreliable nodes

Unreliable environment \Rightarrow ? \Leftarrow Need for reliable services
(e.g. security, monitoring, ...)

Questions to answer

- ▶ How can reliability be achieved?

Reliability

Snapshots:

- ▶ Save state of application from time to time
- ▶ In case of failures: load snapshot

But:

- ▶ May invalidate client state
 - ▶ User may experience downtime
 - ▶ Bad for login or security services
- ⇒ Reliability \neq Availability
- ▶ Which entity monitors the application?
 - ▶ Has to be reliable and available
 - ▶ Has to be distributed
- ⇒ Self-containment

Questions to answer

- ▶ How can reliability be increased?
- ▶ How can availability be increased?
- ▶ Is there a self-contained solution?

Availability

Replication:

- ▶ Availability by redundancy
- ▶ Provide identical entities at multiple sites
- ▶ Contains snapshots as special case
- ▶ Consistency protocol ensures **reliability**

Outline

Motivation

Replication: An Introduction

Virtual Nodes

Distributed Servers with Virtual Nodes

Integration

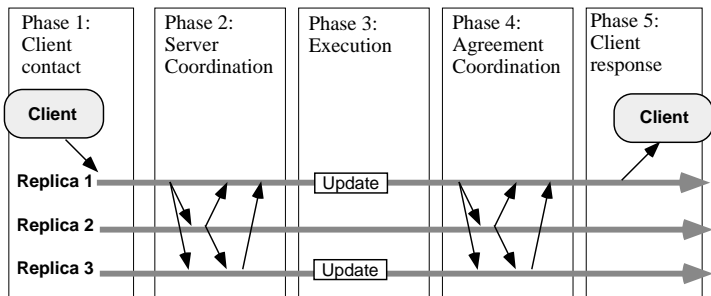
Discussion

Conclusions

What to replicate?

- ▶ Data object
 - ▶ Database
 - ▶ Computing task
 - ▶ Remote object
 - ▶ Service
- ⇒ Sophisticated algorithms for all fields ...
- ⇒ ... and a general model

General Replication Model



Replication protocol:

- ▶ Decides on the use of a phase
- ▶ Different approaches per phase
- ▶ Different demands to the code

General Replication Model

5 Phases

1. Request: client submits operation
2. Server coordination: synchronize the execution
(e.g., message ordering)
3. Execution: operation is executed
(by one or more replicas)
4. Agreement coordination: result of the operation
(e.g., guarantee atomicity)
5. Response: send outcome back to client

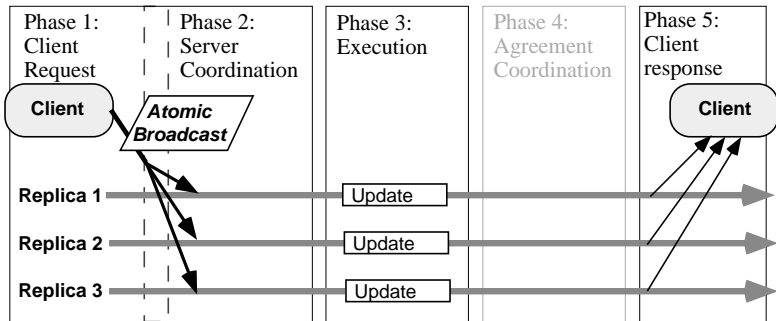
Classification

Active replication:

- ▶ State-machine replication
- ▶ Decentralised approach
- ▶ Request processed by all replicas
- ▶ Simple due to symmetry
- ▶ Quick reaction to failures
- ▶ Demanding with respect to determinism (most of the time)
 - ▶ Message ordering
 - ▶ Execution order

Classification

Active replication:



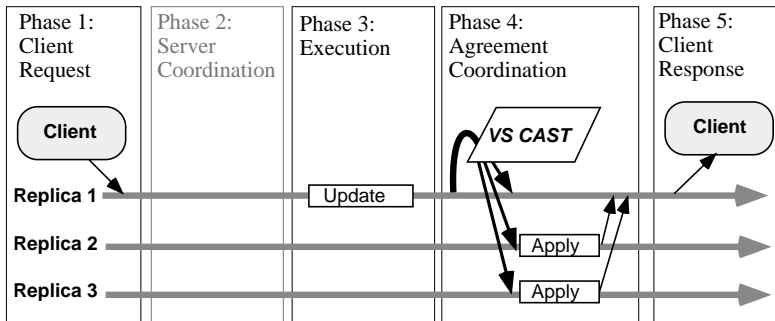
Classification

Passive replication:

- ▶ Primary backup replication
- ▶ Centralised approach
- ▶ Request processed by a single replica (primary)
- ▶ New state/state changes transferred to backups
- ▶ Failure of primary requires re-election
- ▶ Can handle nondeterminism (sometimes)

Classification

Passive replication:



Outline

Motivation

Replication: An Introduction

Virtual Nodes

Overview

Deterministic Scheduling

Example

Distributed Servers with Virtual Nodes

Integration

Discussion

Environment

What to replicate?

- ▶ Data object
- ▶ Database
- ▶ Computing task
- ▶ **Remote object/Service**

Why Objects and Services?

“Can’t you just use databases?”

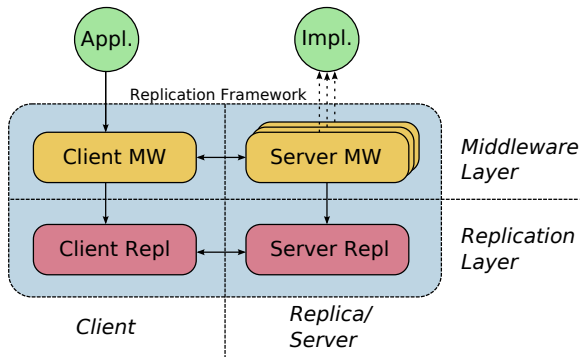
- ▶ Applications do not need/use stable storage
- ▶ Uniform programming model
- ▶ Support for legacy applications

Virtual Nodes: XtremOS Approach to Reliability

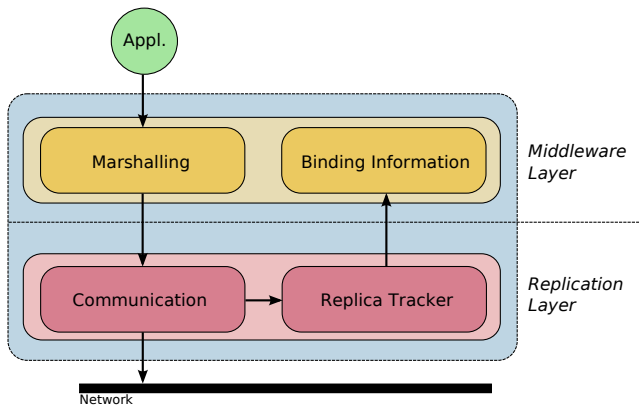
Replication Framework

- ▶ Java-based, highly configurable
- ▶ Support for **changing replica groups**
- ▶ Support for deterministic **multithreading**
- ▶ Multiple middleware interfaces (CORBA, J-RMI, SOAP, ...)
- ▶ Support for nested invocations (SOA)
- ▶ Optimization for *read-only* invocations
- ▶ **Self-contained:** independent of other nodes and services
- ▶ **Service implementation orthogonal to replication**
 - ▶ Except for non-deterministic methods
 - ▶ Except for state transfer
 - ▶ ...

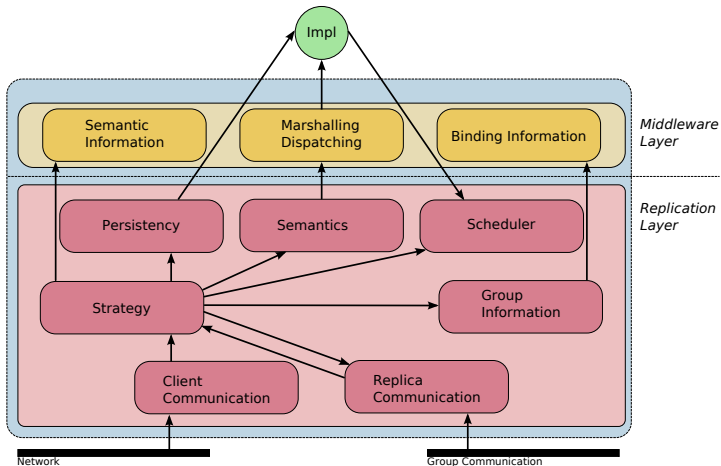
Architecture: Overview



Architecture: Client-side



Architecture: Server-side



Deterministic Scheduling

Active Replication Requires Determinism

- ▶ Multithreading is non-deterministic
- ▶ Single-threaded execution
 - ▶ Slow and dead-lock prone
 - ▶ Denies use of condition variables (`wait`, `notify`)
 - ▶ Does not make use of multi-cpu/-core machines

Deterministic Multithreading

- ▶ Deterministic thread switching: limited concurrency
- ▶ Four algorithms with different properties
 - ▶ Single active thread (SAT, Reiser et al.)
 - ▶ Multiple active threads (MAT, Reiser et al.)
 - ▶ Lose synchronization algorithm (LSA, Basile et al.)
 - ▶ Preemptive deterministic scheduling (PDS, Basile et al.)
- ▶ No one-size-fits-all solution

Scheduler Integration

Intercept Java Synchronisation Statements:

- ▶ synchronized methods and blocks
 - ▶ synchronized instance methods
 - ▶ synchronized static methods
 - ▶ synchronized blocks
- ▶ wait(), notify(), and notifyAll() calls

Interception: Replace Statements by Calls to Scheduler

- ▶ synchronized: pair of lock/unlock invocations
- ▶ All other: simple replacement
- ▶ On source code or byte code level
- ▶ Transparent to service developer
- ▶ Appropriate also for legacy applications

Interception by Code Transformation

```
public class Queue extends ... {  
    public synchronized  
        String remove()  
    {  
        while(data.size()==0)  
            wait();  
        return data.remove(0);  
    }  
}
```

```
public synchronized  
    void append(String x)  
    {  
        data.add(x);  
        notify();  
    }  
}
```

⇒

```
public class Queue extends ... {  
    public String remove() {  
        _scheduler.lock(this);  
        try {  
            while(data.size()==0)  
                _scheduler._wait(this);  
            return data.remove(0);  
        } finally {  
            _scheduler.unlock(this);  
        }  
    }  
    public void append(String x) {  
        _scheduler.lock(this);  
        try {  
            data.add(x);  
            _scheduler._notify(this);  
        } finally {  
            _scheduler.unlock(this);  
        }  
    }  
}
```

Example

Replicated dictionary

```
class DictionaryService {  
    private HashMap<String,String> entries = new ...;  
  
    public String getEntry(String key){  
        return entries .get(key);  
    }  
  
    public void addEntry(String key, String value){  
        return entries .put(key, value);  
    }  
}
```

Example

Steps to replicate a service

- ▶ Create interface(s)

```
public interface Dictionary {  
    public String getEntry(String key);  
    public void addEntry(String key, String value);  
}
```

- ▶ Let class implement interface(s)
- ▶ Make class `Serializable`
- ▶ Annotate methods (optional step)

```
class DictionaryService implements Dictionary, Serializable {  
  
    @ReadOnly  
    public String getEntry(String key) { ... }  
    ...  
}
```

Example

Start first replica

```
void main(String[] args) {  
    DictionaryService dicts = new DictionaryService();  
    Dictionary proxy = (Dictionary) Exporter.exportObject(dicts).proxy;  
  
    // replica is running in background now  
    // store proxy in registry/database/filesystem  
    ...  
}
```

Example

Start client

```
void main(String [] args) {  
  
    // load proxy from registry/database/filesystem  
    Dictionary proxy = ... ;  
    proxy.addEntry(" hello", " world");  
    ...  
}
```

Example

Start additional replica

```
void main(String[] args) {  
  
    // load proxy from registry/database/filesystem  
    Dictionary proxy = ... ;  
  
    // cast to AdminMethods interface  
    AdminMethods adm = (AdminMethods) proxy;  
    adm.startNewReplica();  
    ...  
}
```

Time to Take a Breath

How far we've come

- ▶ Easy-to-use replication framework
- ▶ Highly configurable
 - ▶ Replication protocol, scheduler, access protocol
- ▶ Deterministic scheduling for high performance
 - ▶ Transparently integrated into system

Where we are going to?

- ▶ Scheduler optimisations
- ▶ Integration of data-centric replication protocol
- ▶ Automatic configuration/re-configuration
- ▶ Increased client-side transparency

Outline

Motivation

Replication: An Introduction

Virtual Nodes

Distributed Servers with Virtual Nodes

Motivation: Client Transparency

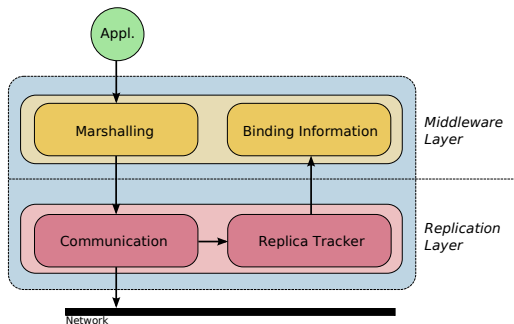
Excursus: Mobile IPv6

Distributed Servers

Integration

Discussion

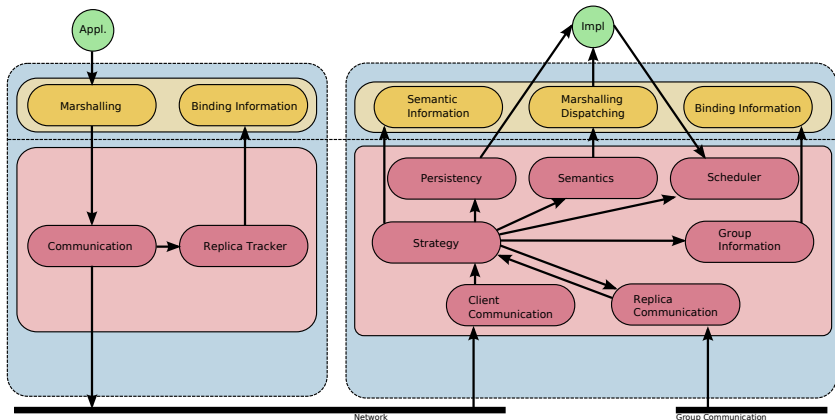
Client Transparency



- ▶ Client has to install additional software
- ▶ Application developer has to be aware of replication
- ▶ Violates the goal of transparency

Increase Client Transparency

Remove Replica Layer



Replica Tracking

Use a location service?

- ▶ How is the location service being tracked?

Use client-side daemon?

- ▶ Will work most of the time
- ▶ Still no guarantee
- ▶ Additional traffic due to polling
- ▶ No fix address: initial contact difficult

Our Approach: Exploit Mobile IPv6

- ▶ Uses standardized techniques
- ▶ Does not require any modifications at client side
- ▶ Provided by XtremOS Distributed Servers

Mobile IPv6 in a Nutshell

Mobile nodes reachable while away from home networks

- ▶ Correspondent node (CN): any node talking to mobile node

Mobile Node: Two Addresses

- ▶ Home address (HoA): identifies mobile node, never changes
- ▶ Careof address (CoA): represents mobile node's current location

Transparency for High-level Protocols:

- ▶ Mobile nodes addressed by HoA
- ▶ IP-level translates HoA to CoA
- ▶ Location changes are announced by the mobile node

”Sounds nice, but how does the IP-level know?”

Mobile IPv6 in a Nutshell (II)

Home Agent (HA)

- ▶ Router in home network
- ▶ Mobile node informs HA about CoA
- ▶ Knows mapping from HoA to CoA

"Hey, wait a second! You do use a central entity! Isn't this cheating!?!"

Yes, but ...

- ▶ Routers are not switched off spontaneously
- ▶ Routers run a small software system and tend to be less buggy
- ▶ No network depends on a single router

Distributed Servers

Distributed Server (Vrije Universiteit Amsterdam)

- ▶ Group of nodes pretending to be a mobile node
- ▶ Identified by home address
- ▶ Node addresses represent careof addresses

Features

- ▶ One node registers at home agent (contact node)
- ▶ Nodes can hand back and forth single connections (cooperatively)
- ▶ Contact node can change (cooperatively)

Outline

Motivation

Replication: An Introduction

Virtual Nodes

Distributed Servers with Virtual Nodes

Integration

Discussion

Conclusions

Integrated Approach

Benefit:

- ▶ Virtual Nodes: fault-tolerance for Distributed Servers
- ▶ Distributed Servers: anycast mechanism for Virtual Nodes

Facts:

- ▶ Handover requires an old socket state
⇒ Replication of state
- ▶ Only reasonable with active replication

Failure Detection:

- ▶ Minimize experienced downtime: change contact node quickly
- ▶ Minimize false positives: exclude group members slowly

Invocation

1. Client sends request to contact node
2. Contact node copies socket state
3. Contact node broadcasts request and socket
4. All nodes process request
5. Contact node sends reply to client
6. Contact node broadcasts new socket state

Discussion

Fault-tolerance:

- ▶ No fault-tolerance during steps 1 and 2
- ▶ Steps 3 – 5: Handover reveals #bytes sent and received
 - ▶ Allows to send remaining bytes of reply

Minimal overhead (copying socket)

- ▶ Step 6 purely for garbage collection
- ▶ Piggyback on other requests

Changing contact node

- ▶ No effect on client
- ▶ Other replicas need to know
- ▶ Causes an additional group message

Conclusions

XtreemOS:

- ▶ Challenge for reliability and availability
- ▶ Replication can solve both issues

XtreemOS Virtual Nodes:

- ▶ Configurable replication framework for fault-tolerance
- ▶ Support for multiple middleware systems at client-side
- ▶ Deterministic multithreading

XtreemOS Distributed Servers:

- ▶ Anycast due to mobile IPv6
- ▶ Group of nodes pretends to be a mobile node
- ▶ Handing over of connections

Integration:

- ▶ Both systems are orthogonal
- ▶ Increases client-side transparency

Papers

- ▶ Matthias Wiesmann et al: *Understanding Replication in Databases and Distributed Systems*. ICDCS '00
- ▶ Hans P. Reiser et al: *Consistent Replication of Multithreaded Distributed Objects*. SRDS'06
- ▶ Hans P. Reiser et al: *Deterministic Multithreading for Replicated CORBA Objects*. PDCS'06
- ▶ Claudio Basile et al: *Active Replication of Multithreaded Applications*. Transactions on Parallel and Distributed Systems, May 2006
- ▶ Michal Szymaniak et al: *Enabling Service Adaptability with Versatile Anycast*. Concurrency and Computation: Practice and Experience, September 2007.
- ▶ Jörg Domaschka et al: *Multithreading Strategies for Replicated Objects*. Middleware '08.
- ▶ Jörg Domaschka et al: *Virtual Nodes: a Re-Configurable Replication Framework for Highly-available Grid Services*. Middleware (Companion) '08.