

EJB Clustering in a Nutshell

Praxisbeispiel JBoss

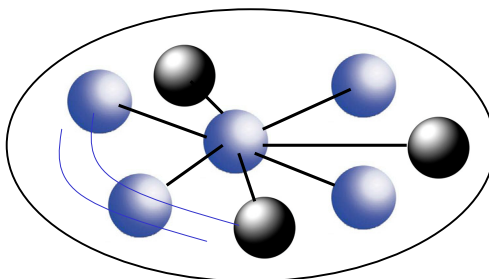
Überblick

- Was ist Clustering ?
- Was ist EJB Clustering ?
- Clustering mit JBoss
 - Implementierung
 - HA JNDI
 - Session Beans
 - Entity Beans
 - Message Driven Beans
 - Einsatz von Clustern

Was ist Clustering ?

Was ist Clustering ?

- Definition:
 - Ein Cluster besteht aus einer Gruppe von gekoppelten Komponenten, die zusammen nach Außen einen einheitlichen Service anbieten



Wozu Clustering ?

- High availability
 - Maximierung der Verfügbarkeit eines Services durch ausweichen auf Alternativkomponenten bei Ausfall einer Komponente
 - Die geclusterten Komponenten des Service müssen in verschiedenen gleichartigen Umfeldern zur Verfügung stehen
- *Scalability*
 - Die Fähigkeit, durch dynamisches Hinzufügen von Komponenten transparent für die Anwender auf Änderungen im Kapazitätsbedarf zu reagieren

Womit wird dies erreicht?

- Grundprinzip: Redundantes Vorhalten sensibler Komponenten
- Failover
 - Eine Komponente springt bei Ausfall einer anderen automatisch für diese ein. Hierzu wird für die rechtzeitige Aktivierung einer neuen Komponente und die Übergabe des letzten Status der ausgefallenen Sorge getragen
- Load balancing
 - Sorgt für eine gleichmäßige Verteilung der Kommunikation- und Rechenlast über mehrfach vorhandene Komponenten eines Gesamtsystems

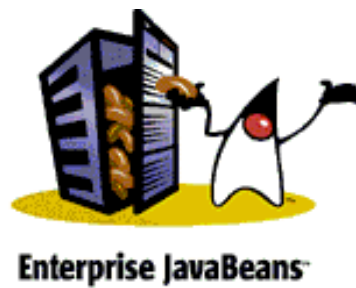
Is something clustered there ?

- Datenbanken
- Linux-Cluster
- Frühstück ?



Clustering von J2EE Komponenten

- Webkomponenten
 - Servlets
 - JSPs
- RMI Objekte
- JMS Destinationen
- JDBC Connections
- **Enterprise JavaBeans**



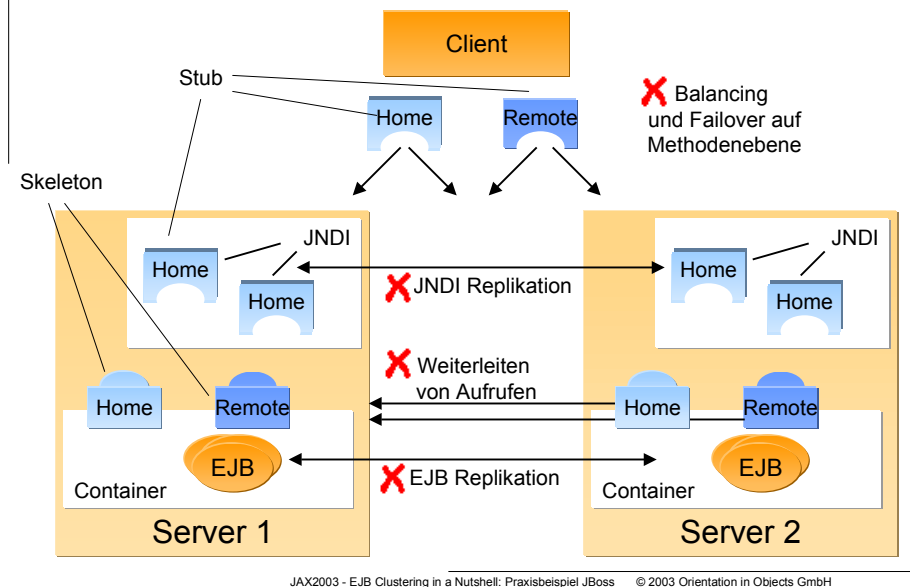
Was ist EJB Clustering

Ansatzpunkte

- EJB Container
- Home stub
- Remote stub
- JNDI (Java Naming Directory Interface) Naming Server

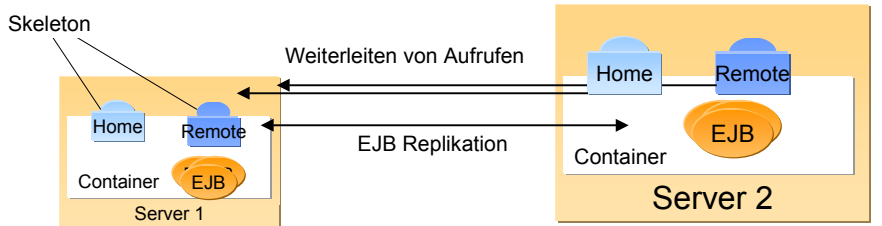
- Charakteristika:
 - Diese Stellen besitzen Logik und Möglichkeiten, die zum Clustern von EJB benutzt bzw. erweitert werden kann.

Implementierung - Take a closer look !



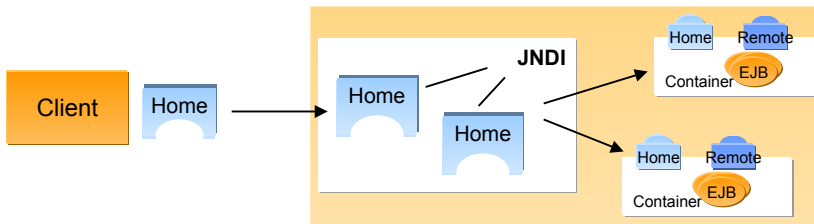
EJB Container Strategien

- EJB Container
 - Call bereits direkt an dem Punkt, wo auf einen Zustand ausweichend reagiert muss
- Load Balancing
 - Weiterleiten von Calls bei vollem Cache
- Fail over
 - State Replikation



Clustering auf JNDI Basis

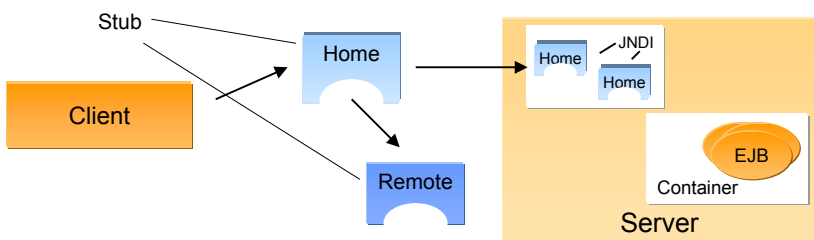
- JNDI (Java Naming Directory Interface) Naming Server
 - Ansatzpunkt als „Zwischenschicht“
 - Abhängigkeit von Applikation Server und Implementierung
- Einheitliche Schnittstelle zu Namens- und Verzeichnisdiensten
- Basisdienst von J2EE basierten Applikationsservern



JAX2003 - EJB Clustering in a Nutshell: Praxisbeispiel JBoss © 2003 Orientation in Objects GmbH

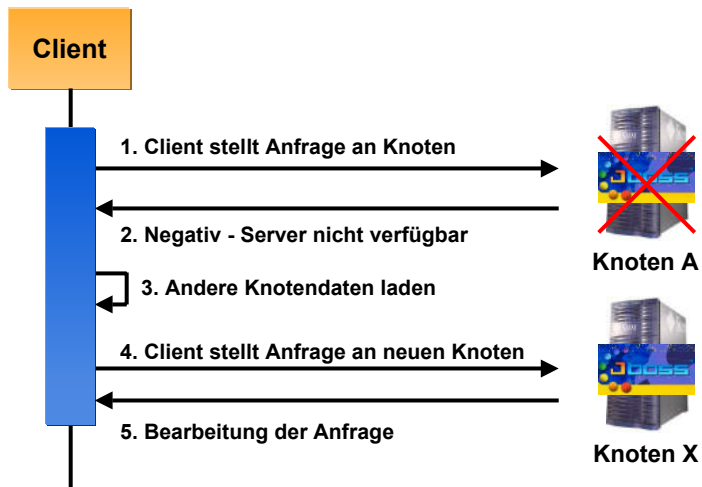
Home und Remote Stub

- Home und Remote Aufrufe zu unterschiedlichen Zeitpunkten - verschiedene Strategien möglich
 - Logik muss bereits im Code implementiert sein
 - Unterschiedliche Betrachtung bei verschiedenen EJB Arten

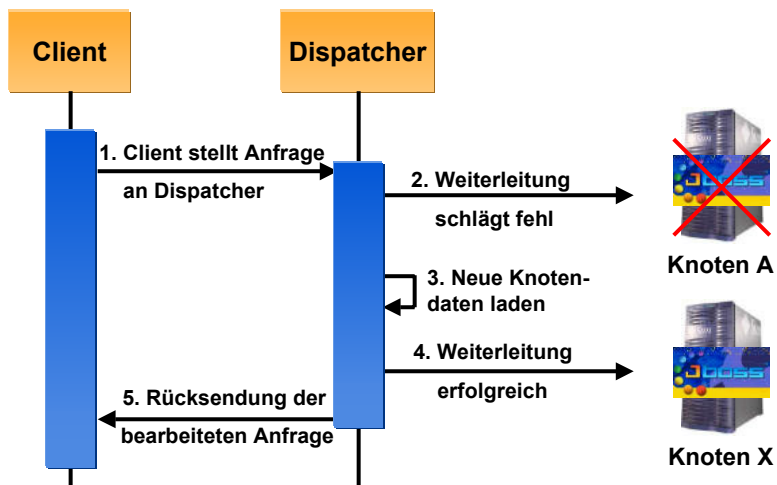


JAX2003 - EJB Clustering in a Nutshell: Praxisbeispiel JBoss © 2003 Orientation in Objects GmbH

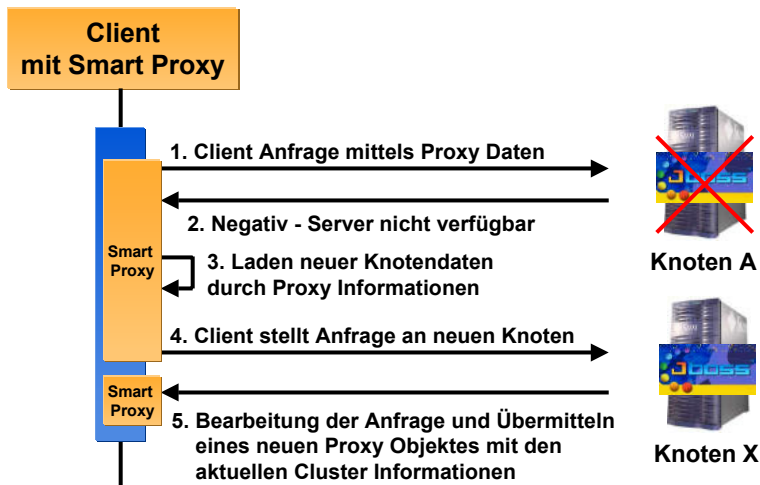
Failover - Client managed



Failover - Dispatcher managed



Failover mit dynamischem Proxy



Idempotenz Problem

- Konzept:
 - Jeder Methodenaufruf kann wiederholbar ausgeführt werden, ohne das sich am Verhalten oder Zustand der Applikation etwas ändert
- Transaktionale Methode - nicht automatisch idempotent !
- Nie Datenmanipulation
- Verantwortung beim Entwickler und Container Vendor
 - Exceptions
 - Deklaration

JBoss

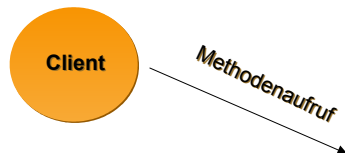
JBoss Clustering Solutions

- Verwendung von Smart Proxies
 - Zur Laufzeit geladene Stubs Objekte für Remotereferenzen
 - Informiert Client über verfügbare Nodes
 - Enthält Code für Failover und Load-Balancing
 - Für Client transparent
- Load-Balancing Algorithmen sind pluggable

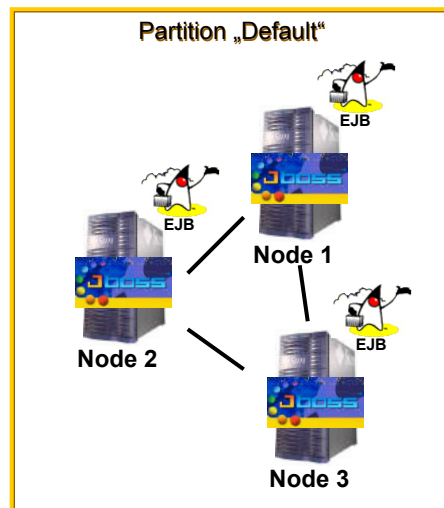
JBoss Clustering - Wichtige Features

- Automatisches Erkennen
 - Knoten in einem Cluster finden sich selbst - ohne zusätzliche Konfiguration
- Fail-over und Load balancing für:
 - JNDI, RMI, Entity Beans, Stateful / Stateless Session Beans
- Dynamisches JNDI Erkennen
 - JNDI Clients können automatisch den JNDI InitialContext erkennen
- HTTP Session Reproduktion
 - für Tomcat und Jetty über JBoss

Ein JBoss Cluster



- Demo-Cluster:
 - JBoss Version 3.2.1
 - 3 Rechner
 - Betriebssystem:
Windows 2000



Cluster einrichten

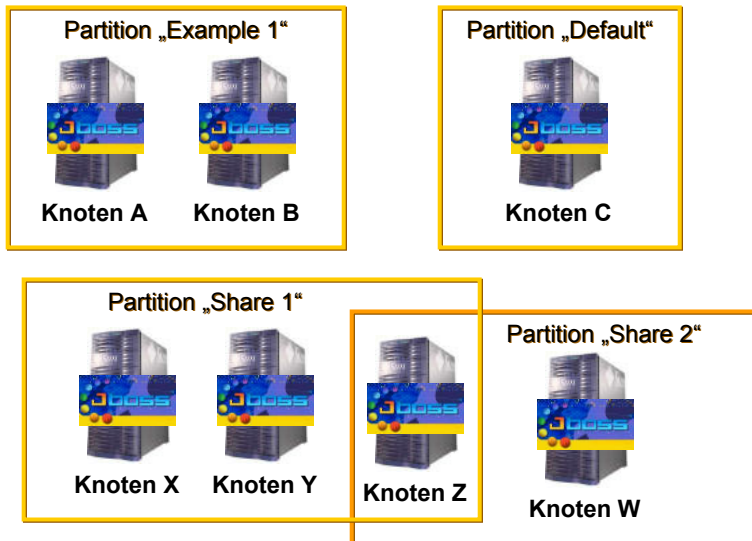
- JBoss Server im Cluster
 - Starten in der „all“ - Konfiguration bei allen JBoss Servern des Clusters
 - `[JBOSS_HOME]\bin\run -c all`
- Enterprise Java Beans
 - Die Beans „cluster-fähig“ machen
 - Auf allen Knoten des Clusters deployen in das Verzeichnis `[JBOSS_HOME]\server\all\deploy`
- Client - JNDI Properties Konfiguration
 - `java.naming.provider.url=[Leer-String]`



JBoss Sprechweise

- Partition (äquivalent zu einem Cluster)
 - Gruppierung von JBoss Instanzen zum Erstellen des Clusters
 - Eine JBoss Instanz kann mehreren Partitionen angehören
- Sub-Partitionen
 - Werden momentan von JBoss noch nicht unterstützt
 - Interessant für strategische Lastverteilung
 - **Beispiel: Stateful Session**

JBoss Partitionen



JBoss Cluster Konfiguration

- Clustering nur in der „*all*“ Konfiguration möglich
- Im Verzeichnis `.../all/server/deploy` befindet sich die `cluster-service.xml`

```
<mbean code="org.jboss.ha.framework.server.ClusterPartition"  
name="jboss:service=DefaultPartition"/>
```

- Einstellungen für
 - Partitionierung
 - Naming Service
 - SessionState
 - Cache

- <clustered> Tag in der *jboss.xml*

```
<jboss>
  <enterprise-beans>
    <session>
      <ejb-name>SimpleWorker</ejb-name>
      <jndi-name>SimpleWorker</jndi-name>
      <clustered>True</clustered>
    </session>
  </enterprise-beans>
  ...
</jboss>
```

High Available JNDI

JNDI Implementierungen



- JNDI Clustering Möglichkeit - Abhängig von der JNDI-Implementierung des Applikationsservers
- Varianten:
 - Unabhängiger JNDI- Baum
 - Zentraler JNDI- Baum
 - Shared global JNDI- Baum

Unabhängiger JNDI- Baum



- Lokaler JNDI Baum ohne Kenntnis der anderen Server im Cluster
- Vorteil:
 - Einfach zu skalieren (Hinzufügen weiterer Server)
 - Zeitaufwand sehr kurz, alle Clusterkomponenten beim Hochfahren der Server des Clusters zu erkennen = *Konvergenzzeit*
- Nachteil:
 - Externe Umsetzungsstrategien für Ausfallsicherheit
 - **Dispatcher Entwurf, Proxy Dienst**

Zentraler JNDI-Baum



- Cluster mit einem zentralem JNDI-Baum, an den alle Objekte gebunden werden
- Name Server kennt diesen JNDI-Baum, d.h. ein Client erhält „globale“ Referenz, mit der er beim einem Server im Cluster Home- und Remote erhält
- **Nachteil:**
 - Ausfallsicherheit vs. Name Server down !
 - **Einbinden neuer Name Server bedingt neues Ermitteln und Binden aller Daten = hoher zeitlicher Aufwand**
 - Verdopplung der Anzahl der JNDI-Aufrufe
 - Vergrößern des Clusters erhöht die Konvergenzzeit und erfordert evtl. das Einbeziehen mehrerer Name Server

Shared Global JNDI- Baum I



- Verwenden von IP Multicasting
 - Bekanntgabe neuer Server im Cluster - Übertragen des JNDI Baumes
- Objekte werden im shared global JNDI- Baum des Clusters und lokalen JNDI- Baum des Servers gebunden.
 - Failover und schneller JNDI Lookup
- Spezielle Home Objekte im shared global JNDI-Baum, wenn ein Objekt auf mehr als einem Server im Cluster verfügbar ist
 - Kenntnis über alle verfügbaren Objekte

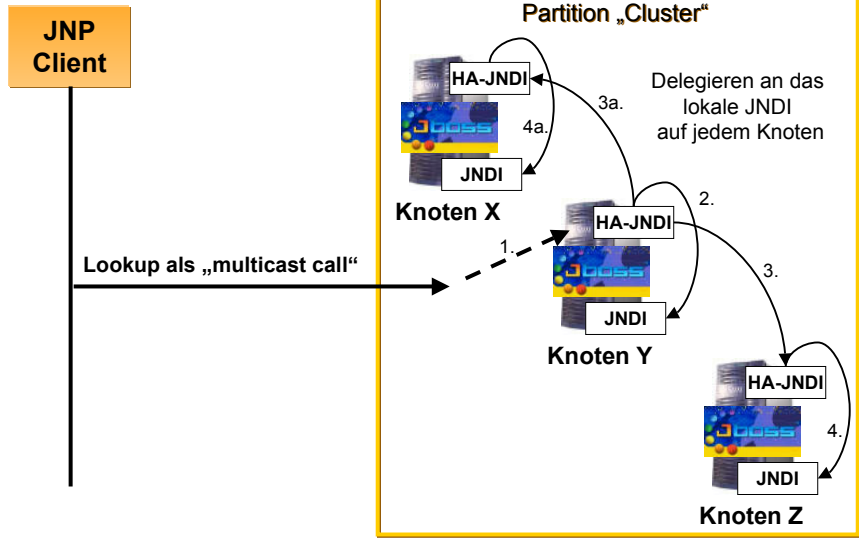
Shared Global JNDI- Baum II

- Vorteile
 - Einfachere Skalierbarkeit sowie höhere Verfügbarkeit als im zentralen Cluster
 - Kein Cluster-Tuning der Name- Server
 - Netzwerkbelastung im Betrieb geringer als im zentralen Cluster - pro Lookup nur einen Netzwerkaufruf
- Nachteile
 - Hohe Netzwerkbelastung beim Hochfahren der Server -höhere Konvergenzzeit , abhängig von der Anzahl der Server des Clusters
- Shared Global JNDI teilweise in JBoss

HA-JNDI

- HA-JNDI
 - Ein cluster-weiter, globaler JNDI Context zum Binden und Abfragen von Objekten
 - Durch Replikationsmechanismus sind Objekte, die an diesen Context gebunden sind, auch beim Ausfall eines Knotens erreichbar
 - Serverseitiges Binden von Objekten wird über „normales“ JNDI nur lokal ausgeführt, d.h. ein entfernter Client Aufruf wird vom HA-JNDI an das lokale JNDI delegiert
 - Konfiguration möglich in der *cluster-service.xml*

HA-JNDI Regeln zum Lookup und Binden

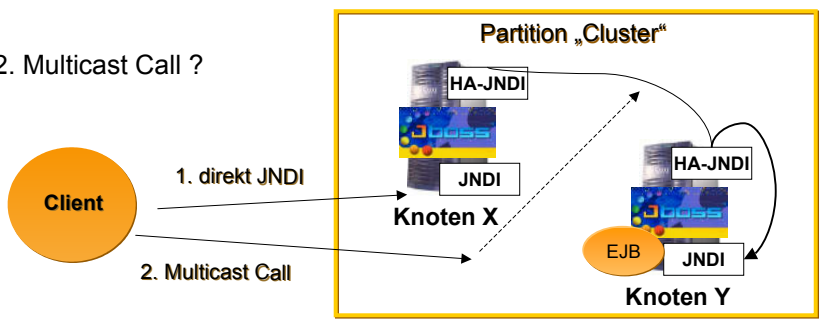


EJB im lokalen JNDI

- EJB wird ohne <clustered> Tag im lokalen JNDI gebunden
- Server im Cluster

1. Was passiert, wenn ein Client direkt einen anderen Server im Cluster aufruft ?

2. Multicast Call ?

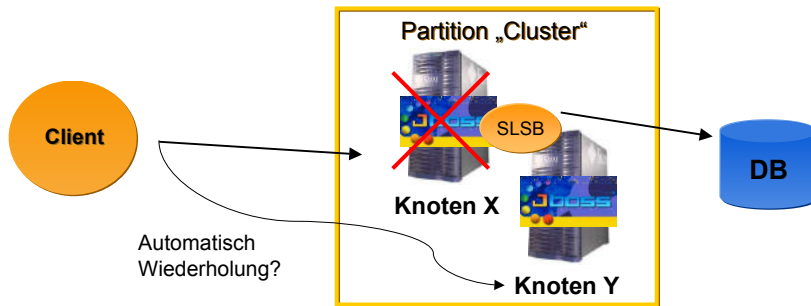


- Client Konfiguration im Cluster:
 - 1. Möglichkeit:
 - **Leerer Property String** für `java.naming.provider.url`
 - **JNP Client produziert einen „multicast call“ im Netzwerk**
 - **Benutzte Adresse: 230.0.0.4:1102**
 - 2. Möglichkeit:
 - **Einzelnes Anmelden aller Nodes des Cluster (Komma separiert)**
 - `java.naming.provider.url=server1:1100,server2:1100,server3:1100,server4:1100`
 - **Client versucht der Reihe nach, sich mit einem Server zu verbinden**
 - **Beim ersten Connect mit einem Server, erhält er von diesem Server den Stub mit den Informationen über die Cluster-Topologie**

Stateless Session Beans

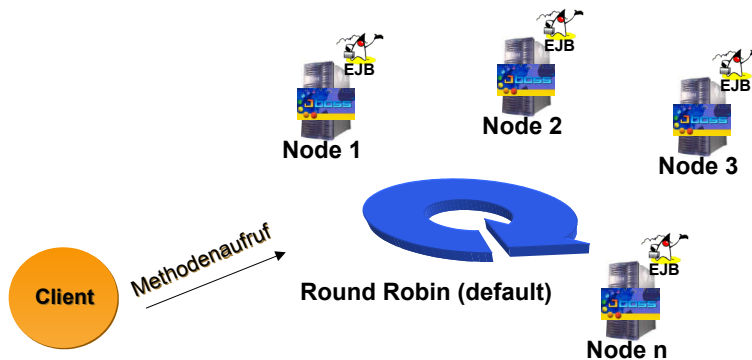
Stateless Session Beans - Fail over

- Home Stub: trivial
- Remote Stub:
 - Idempotenz Eigenschaft
 - no state - no problem - just try again ! (JBoss)



Stateless Session Beans - Load Balancing

- Home und Remote per Default - Round Robin
 - andere Strategien möglich
- Konfiguration in der *jboss.xml*



Balancing Strategien

- Konfiguration für Home und Remote Stub
 - Round Robin
 - Jeder Call wird einem neuen Knoten im Cluster zugewiesen
 - FirstAvailable
 - Verfügbarer Stub sendet Calls immer auf einen Knoten
 - FirstAvailableIdenticalAllProxies
 - Wie FirstAvailable, der Zielknoten wird von den Proxies der gleichen „family“ geteilt (ab 3.2 und höher)
- Strategien erweiterbar
 - Laufzeitinformationen der VM
 - Verfügbarer Speicher
 - Anzahl der Threads, die gerade warten/laufen

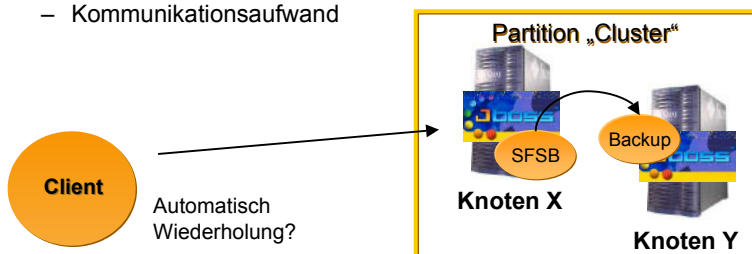
Balancing Konfiguration *optional*

```
jboss>
<enterprise-beans>
  <session>
    <ejb-name>SimpleWorker</ejb-name>
    <jndi-name>SimpleWorker</jndi-name>
    <clustered>True</clustered>
    <cluster-config>
      <partition-name>DefaultPartition</partition-name>
      <home-load-balance-policy>
        org.jboss.ha.framework.interfaces.RoundRobin
      </home-load-balance-policy>
      <bean-load-balance-policy>
        org.jboss.ha.framework.interfaces.RoundRobin
      </bean-load-balance-policy>
    </cluster-config>
  </session>
</enterprise-beans>
<enforce-ejb-restrictions>false</enforce-ejb-restrictions>
</jboss>
```

Stateful Session Beans

Stateful Session Beans - Fail over I

- Home Stub und Remote Stub: Idempotenz !
 - Home: create - nicht idempotent !
 - Remote
 - **Nur getter, keine Zustandsveränderung herbeiführend**
- Fail over über Replikation / Back up
 - Kommunikationsaufwand

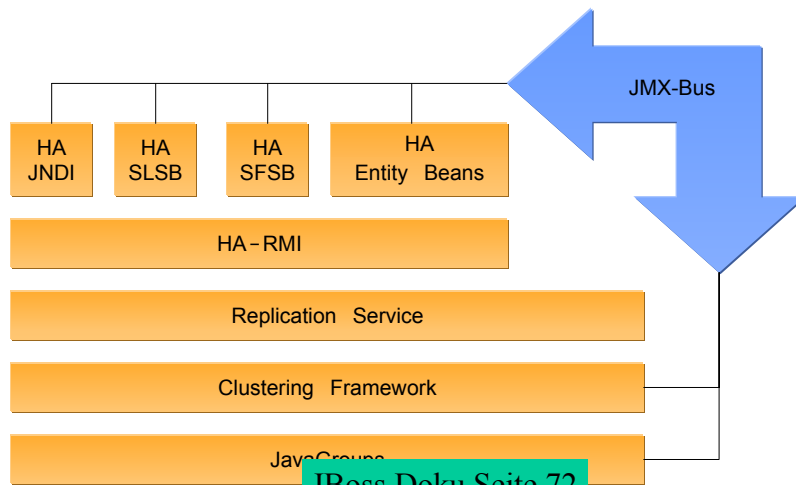


- Verwalten von Zustand
 - In-memory state replication - JBoss
 - **Back up über gesamte Partition**
 - Datenbank
- Zeitpunkt ?
 - Nach jeder Methode (*JBoss*)
 - Nach jeder Transaktion
 - `isModified()` ab Version JBoss 3

JBoss Clustering Architektur

Exkurs

JBoss Clustering Architecture



JBossClustering - JavaGroups

- JavaGroups – Open-source group communication Toolkit
 - Lossless transmission of messages through multicast
 - Guaranteed message ordering
 - Atomicity: all or nothing
 - Knowledge of group membership
 - Membership change notification
 - Automatic state transfer

JBossClustering – Clustering framework



- JavaGroups abstraction layer
- RPCs:
 - Call methods on object in every cluster node
- State replication
 - Used to develop any new HA service that needs to share some common state between its instances
- Dynamic cluster composition events
 - Used to know when cluster topology changes or a new service/bean is made available on a node

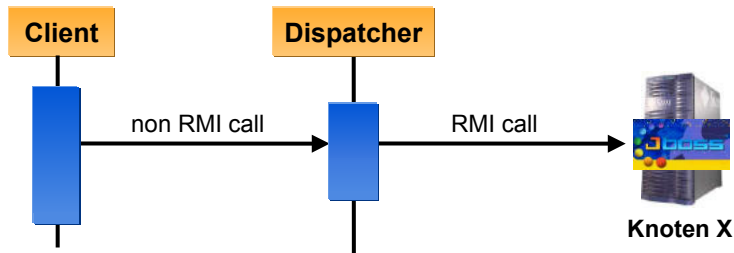
Javagroups



- Problemstellungen
- Paketgröße, Paketverlust, Paketreihenfolge
- Verschlüsselung, Ablaufkontrolle, Gruppenbildung, Fehlererkennung, Zustandsreplikation verschiedener Transportprotokolle

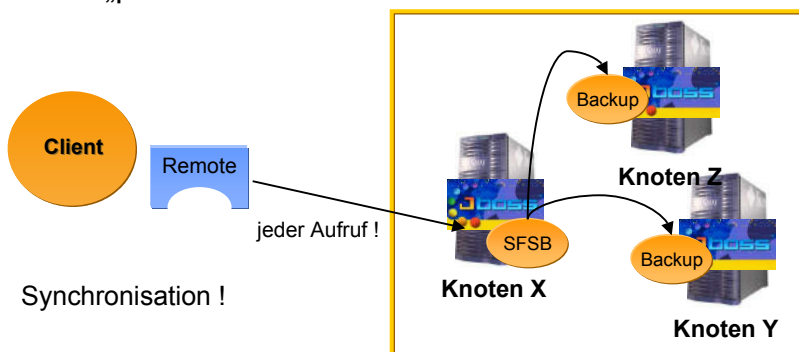
Und ohne RMI ?

- Seitens JBoss wird RMI zur Kommunikation genutzt
- Dispatcher-manager fail-over
 - Mittels Dispatcher kann ein clientseitig verwendetes Protokoll übersetzt werden (SOAP, CORBA)



Stateful Session Beans - Load Balancing

- Konfiguration wie bei Stateless Session Bean
 - Home stub: Round Robin
 - Remote stub: FirstAvailable
 - „pinned“



- Synchronisation !

Entity Beans

Entity Beans - Fail over

- Remember: Idempotenz !
 - idempotent: getter, finder, evtl. home business Methoden
- Keine Notwendigkeit von Replikation bei Einsatz einer Datenbank
 - Caching !
 - Fail over nur am Ende einer Transaktion
- Local Interfaces
- Session Facade

Entity Beans Load Balancing

- Konfiguration in *jboss.xml*
 - Home stub: Round Robin
 - Remote stub: FirstAvailable
- Local Interfaces
- Session Facade

Caching und Synchronization

- Cache
 - Vor der Datenbank - Synchronisation
 - *distributed shared object cache*
 - **Cache und DB synchron und umgekehrt**
- Kommunikations- und Verwaltungsaufwand
- JBoss - zur Zeit kein *distributed shared object cache*
 - Ziele für Version JBoss 4

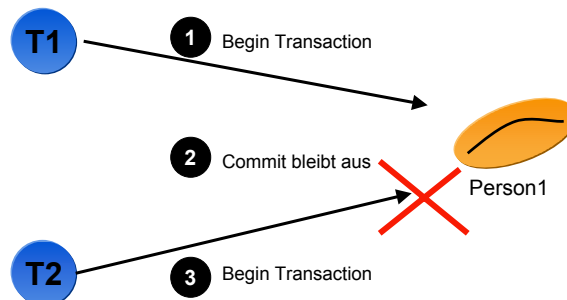
Commit Time Options (Container Option)

	Instanz Status in DB schreiben	Instanz bleibt im „ready“ Zustand	Instanz-Zustand wird ungültig (vor Transaktion Synchronisation)
Option A	Ja	Ja	Nein
Option B	Ja	Ja	Ja
Option C	Ja	Nein	Ja
Option D	Ja	Ja	Nein

Periodisches Update!

Entity Locking Problem

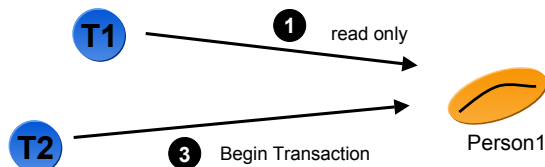
- Optimistic vs. pessimistic locking
- Verantwortung bei Datenbank im JBoss
 - Konfiguration in `jbossjdbc-cmp.xml` - Tag `<row-locking>`



Strategie: Read-only

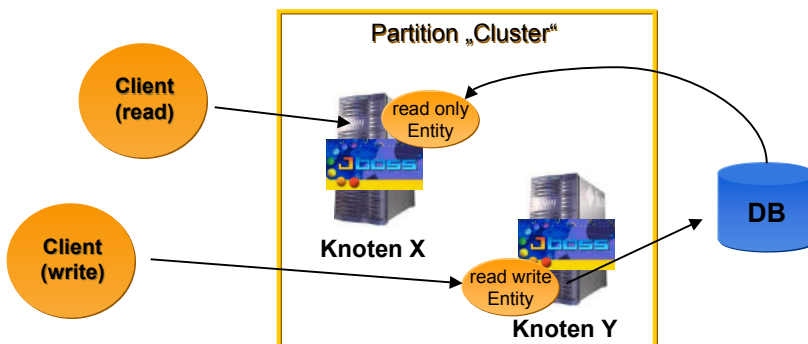
- Konfiguration in jboss.xml

```
...  
<entity>  
  <ejb-name>Person</ejb-name>  
  ...  
  <read-only>true</read-only>  
</entity>  
...
```



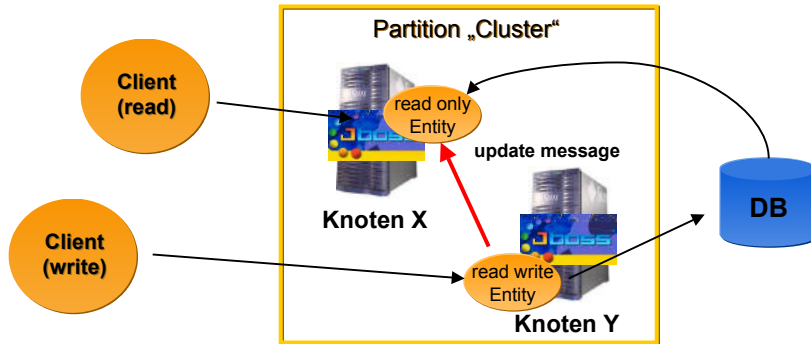
Read only cache

- Update nach Commit Option
- Locking umgangen
 - Wirksam: Cluster und Standalone



Read mostly cache

- Mögliche Lösung für „stale data“ Problem
- Zwei Kommunikationswege
 - synchron / multicast
 - asynchron / JMS



Konfiguration zur Cache invalidation

- JBoss S. 59
- Invalidation Manager
- JBoss Container
- Bean muss als *read only* in jboss.xml angemeldet sein
- Beans müssen Invalidation group zugeordnet werden

Message Driven Beans

MDB - Fail over und Balancing

- Fail over
 - Fault-tolerance durch JMS-Acknowledgement gesichert
- Load balancing
 - MDB sind lastanziehend
 - Automatische Lastverteilung
- Cluster für MDB nötig ?
 - Eigentlich Clustering des JMS-Providers ?

MDB - JBoss

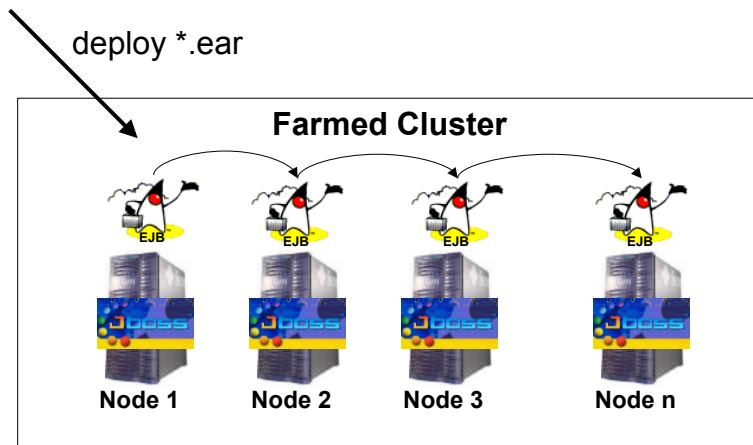
- Kein Message Driven Bean Clustering
- Kein JBossJMS Clustering
- Anbindung anderer Provider nicht trivial
- Migrationsdienst für MDB wäre vorteilhaft
 - Möglichkeit durch JBoss Clustering Architektur gegeben

Wartung von Clustern

- Schwerpunkt der Cluster- Wartung
 - Cluster am Laufen zu halten
 - Änderungen/ Updates einspielen
- Bei großen Anwendungen und vielen Änderungen kann der DeploymentProzess bei File- und Konfigurations-Synchronisation relativ lange dauern

- JMX Konsolen
- JSR 77 Tool
- JBoss Netboot
- Farming !

What is farming ?



Der "Farming" Service

- „hot deploy“ im gesamten Cluster möglich
- Konfiguration in der *farm-service.xml* im Unterverzeichnis
[JBOSS_HOME]\server\all\deploy
- Deployen der „farmed“ Beans in das Verzeichnis
[JBOSS_HOME]\server\all\farm

„To be clustered or not to be ?“

Vielen Dank für Ihre Aufmerksamkeit

Sabine Winkler

winkler@oio.de

Orientation in Objects GmbH
Weinheimer Str. 68
68309 Mannheim
<http://www.oio.de>
info@oio.de