Application Performance Management

Roland Hediger

4. Juni 2014

Inhaltsverzeichnis

Listings

Teil I.

Theorie Teil 2

1. Wildfly Application Server

Der Programmierer soll sich auf funktionale Probleme konzentrieren können. Die immer wiederkehrenden nicht-funktionalen Aspekte werden vom Applikationsserver verwaltet. Ein Java Enterprise Edition (JEE) konformer Applikationsserver kann somit als eine Art Betriebssystem für JavaEE-Applikationen gesehen werden. Enterprise Software unterscheidet sich von anderer Software im Prinzip nicht. Man spricht aber von Enterprise Software meist dann, wenn es sich um eine Server-Applikation handelt, die mehreren Benutzern gleichzeitig zur Verfügung steht. Dies impliziert einige Probleme auf die geachtet werden muss:

Transaktionen Transaktionen: eine Benutzerin darf nicht Zustände sehen, welche durch Interaktionen mit anderen Benutzern entstanden sind, bzw. sie darf solche Zustände nur konsistent und zu definierten Zeitpunkten sehen.

Last LDie Belastung des Systems ist abhängig von der Anzahl und Art der Clients. Das System muss mit Lastspitzen umgehen können.

Verteilte Architektur: Die Applikation ist über mehrere physische Systeme verteilt. Sie besteht aus GUI- Komponenten, Business-Logik und Datenbanksystemen. Eventuell werden auch sogenannte Legacy- Systeme (also ältere Systeme) oder Host-Systeme als Zuliefer- oder Abnehmersysteme verwendet.

1.1. Wildfly/JBoss Einführung

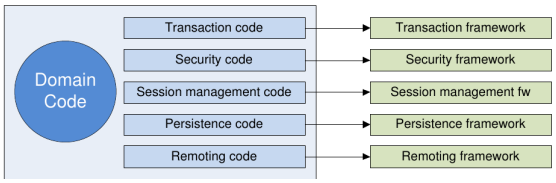

Abbildung 1.1.: figure

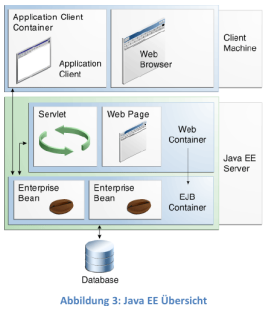

Abbildung 1.2.: figure

Web Container und die darin verwendeten Technologien werden im Modul Web Frameworks behandelt. Der EJB Container wird im Modul Enterprise Application Frameworks besprochen. Application Performance Management geht vertieft auf die Hintergrundtechnologien vom Java EE Server ein (ausgenommen Security, was im Modul Applikationssicherheit abgedeckt wird).

1.2. Wichitge Verzeichnisse + Modi

bin Starten der AS + verschiedene tools.

bin/client bin/client Libraries (jar-Files) die benötigt werden um mit JBoss direkt von einer Client-Applikation aus zu kommunizieren. Es handelt sich dabei um sogenannte Standalone-Clients, also z.B. herkömmliche Java-Applikationen mit Swing-GUIs, welche via RMI auf den Applikationsserver zugreifen wollen.

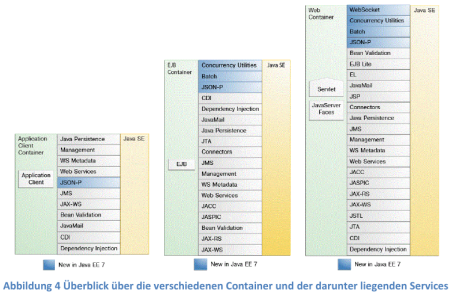

Abbildung 4 Überblick über die verschiedenen Container und der darunter liegenden Services

Abbildung 1.3.: figure

docs/schema DTDs für XML.

docs/examples/cnfigs Konfigurationsbeispiele für häufige Anwendungsfälle

domain Konfigurationen, Deployments und schreibbarer Bereich, der von dem Domain-Mode verwendet wird

modules Wildfly basiert auf einer modularen Architektur. Die verschiedenen Module des Servers sind hier gespeichert

standalone ndalone Konfigurationen, Deployments und schreibbarer Bereich, der von dem Standalone-Mode verwendet wird

welcome content Welcome Page

1.2.1. Modi

Standalone Im Standalone Modus ist jede Wildfly-Instanz ein unabhängiger Prozess mit eigener Konfiguration, Deploymentbereich und schreibbarem Bereich. Einzelne Standalone Instanzen können aber weiterhin geclustert werden.

Domain Im Domain Modus kontrolliert eine einzige Domain-Konfiguration eine ganze Gruppe von virtuellen oder physischen Wildfly-Instanzen. Diese Instanzen werden von einem Host-Controller Prozess gesteuert und kontrolliert. Wir werden diesen Modus im Folgenden nicht weiter vertiefen.

1.2.2. Microkernel Architektur

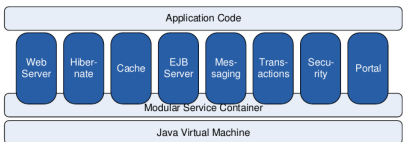

Abbildung 1.4.: figure

1.2.3. Standalone Verzeichnisstruktur

configuration onfiguration: Enthält sämtliche Konfigurationen für den Betrieb im Standalone Modus. Mit der Installation kommen vier verschiedene Serverkonfigurationene: standalone.xml, standalone-ha.xml, standalone-full.xml und stanalone-full-ha.xml. Sie unterscheiden sich in der Verfügbarkeit von High-Availability (die ha-Variante) sowie dem vollen JEE Umfang (full- Varianten) nur das Web-Profil (Varianten ohne full).

data : Hierhin schreibt Wildfly Informationen, die einen Server-Neustart überleben müssen. Kann auch von Applikationen genutzt werden die Zugriff auf das Dateisystem benötigen.

deployments Hierhin werden Applikationen und Services deployed. Ein Deployment besteht darin, z.B. ein .ear- oder ein .war-File in dieses Verzeichnis zu kopieren. JBoss scannt dieses Verzeichnis regelmässig nach Veränderungen und führt dynamisch ein Deployment, bzw. ein Redeployment durch.

lib/ext Libraries (.jar-Files), die von allen Applikationen der Server-Konfiguration geteilt und mit dem Extension-Classloader geladen werden sollen.

log Log files.

tmp und tmp auth Enthält temporäre Daten von Services. Das Unterverzeichnis auth wird auch genutzt um Authentifikationstokens mit lokalen Clients auszutauschen. So können sie beweisen, dass sie lokal ausgeführt werden.

2. Caching

Wie eingangs erwähnt, besteht der Grundgedanke des Caching darin, dass man ein Resultat in einem Zwischenspeicher ablegt. Ziel dabei ist, dass der Zugriff auf den Zwischenspeicher schneller ist, als die Neuberechnung oder der erneute Zugriff auf die nicht gecachten Daten.

- Kopieen von Originaldaten : Hard Disk Cache Browser Cache JVM : Klassen
- Aggregationen von Originaldaten : Wetterprognosen, Reporting : Charts im Cache
- Aggregierte Kopien von Originaldaten Ajax / Html 5 App :
- Beispiele von Caching : Mehrfacher zugriff, zigriff auf gecachte Daten soll zu mindenstens in eine Gross Ordnung schneller sein als auf nicht gecachte Daten.

  Hit Rate : Anzahl Treffer pro Anzahl anfragen. Muss hoch sein.

  u.a bei Aggregation von Original Daten - statisch, selten ändern.

2.1. Caching in Wildfly

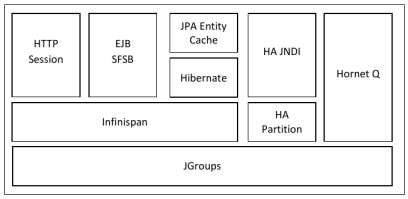

Abbildung 2.1.: figure

Der SFSB Container verwendet den Cache um den gesamten Session State aller SFSBs zu speichern, der Persistence Context nutzt den Cache als second-level cache für seine Enitities. Der JBoss Webserver wiederum speichert http sessions im Cache.

2.1.1. Strategieen

Local Local: Die Cacheeinträge werden lokal abgelegt, egal ob der Knoten einem Cluster angehört oder nicht. Es findet keine Replikation auf andere Knoten statt.

Replication Die Cacheeinträge werden auf alle Knoten im Cluster repliziert. Sind viele Knoten im Cluster dann verringert sich durch diese Cachestrategie die Performance deutlich. Zudem nimmt auch der zur Verfügung stehende Speicher drastisch ab. Replication ist der Default, wenn sonst keine Strategie angegeben wird.

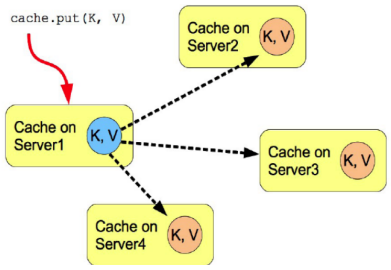

Abbildung 2.2.: figure

Distribution Die Cacheinträge werden nur auf eine Untermenge der Knoten im Cluster verteilt. Dabei kann in der Konfiguration angegeben werden auf wie viele Knoten verteilt wird und ein Hash- Algorithmus berechnet dann pro Eintrag, wohin dieser gespeichert wird. Hier entsteht ein typischer Trade-Off zwischen Ausfallsicherheit (Distribution auf viele Knoten) und Performance (Distribution auf wenige Knoten).

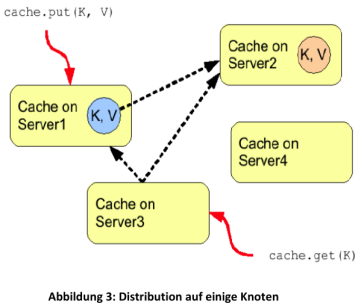

Abbildung 2.3.: figure

Invalidation item Jeder Knoten befüllt seinen Cache lokal, es finden keine Replikationen statt. Die Einträge des Caches werden auch noch in einen zentralen Cache-Store (z.B. in eine Datenbank) gespeichert. Wird in einem Cache ein Eintrag erneuert, dann werden an andere Nodes nur noch Invalidierungsmeldungen verschickt, so dass diese ihre Instanzen verwerfen und neu laden.

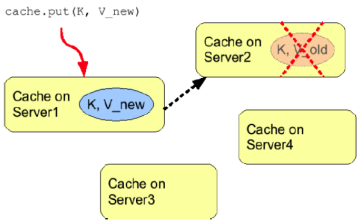

Abbildung 2.4.: figure

2.1.2. Synchronisation

Synchrone Meldungen sind sehr teuer, da hier bei jedem Kopiervorgang auf ein Acknowledge-Signal gewartet wird. D.h. bei jedem Schreibzugriff in den Cache werden die Daten kopiert und auf die Bestätigungen gewartet. Dieser Aufwand lohnt sich selten. Er kann gerechtfertigt sein, wenn sehr hohe Ansprüche an die Cache- Konsistenz gestellt werden. Dies kann z.B. der Fall sein, wenn die Cacheeinträge Resultate von aufwändigen Berechnungen sind. Fehlerhafte Datenübertragungen werden mit synchronen Meldungen sofort entdeckt.

Asynchrone Meldungen Asynchrone Meldungen hingegen blockieren beim Schreiben nicht. Es wird keine Bestätigung zurück gesendet, sondern nur ein Log-Eintrag gemacht. Dies ist natürlich weniger sicher wie synchrone Meldungen, reicht aber in der Praxis meist aus. Besonders geeignet sind asynchrone Meldungen z.B. bei sticky http-sessions. Hier wird bei jeder Veränderung eine Kopie auf andere Knoten geschrieben, das Ergebnis jedoch nicht abgewartet. Erst im Eintreten eines Versagens eines Knotens werden die Daten aus den verteilten Caches gelesen. Bei asynchronen Meldungen ist es also möglich, dass Daten verloren gehen. Dies aber nur dann, wenn exakt bei der Replikation/Distribution der Daten der Quell-Knoten versagt. Zu allen anderen Zeitpunkten sind die Daten konsistent und bei einem Versagen können andere Knoten sofort übernehmen.

2.1.3. Isolationsstrategieen

REPEATABLE_READ: dies ist der Default-Isolationslevel. Es werden Lese-Sperren auf alle gelesenen Daten gehalten. Phantom-Reads sind aber immer noch möglich.

READ_COMMITTED: ist signifikant schneller als REPEATABL_READ, aber Daten die durch ein Query gelesen wurden können von anderen Transaktionen verändert werden.

2.1.4. Konfiguration

- Nicht Teil der JEE Spezifikation.
- JBOSS = Verteilten Caches. Der Cache ist ein Modul (Infinispan System)

Da Infinispan eine zentrale Rolle spielt, werden viele Details in der Cache-Konfiguration festgelegt. Diese Konfiguration findet man in den Konfigurationsdateien im configuration-Verzeichnis. Egal welches standalone*.xml sie auch anschauen:

Listing 2.1: Standalone.xml Caching

```
  standalone*.xml sie auch anschauen:
<subsystem xmlns="urn:jboss:domain:infinispan:2.0">
<cache-container name="web" ... >
...
</cache-container>
<cache-container name="ejb" ... >
...
</cache-container>
<cache-container name="hibernate" ... >
...
</cache-container>
</subsystem>
//Beuspiel:
<cache-container name="server" default-cache="default" aliases="singleton cluster"
module="org.wildfly.clustering.server">
<transport lock-timeout="60000"/>
<replicated-cache name="default" batching="true" mode="SYNC">
<locking isolation="REPEATABLE_READ"/>
</replicated-cache>
</cache-container>
<cache-container name="web" default-cache="dist"
module="org.wildfly.clustering.web.infinispan">
<transport lock-timeout="60000"/>
<distributed-cache name="dist" batching="true" mode="ASYNC" owners="4" l1-lifespan="0">
<file-store/>
</distributed-cache>
</cache-container>
<cache-container name="ejb" default-cache="dist" aliases="sfsb"
module="org.wildfly.clustering.ejb.infinispan">
<transport lock-timeout="60000"/>
<distributed-cache name="dist" batching="true" mode="ASYNC" owners="4" l1-lifespan="0">
<file-store/>
</distributed-cache>
</cache-container>
<cache-container name="hibernate" default-cache="local-query" module="org.hibernate">
<transport lock-timeout="60000"/>
<local-cache name="local-query">
<transaction mode="NONE"/>
<eviction strategy="LRU" max-entries="10000"/>
<expiration max-idle="100000"/>
</local-cache>
<invalidation-cache name="entity" mode="SYNC">
<transaction mode="NON_XA"/>
<eviction strategy="LRU" max-entries="10000"/>
<expiration max-idle="100000"/>
</invalidation-cache>
<replicated-cache name="timestamps" mode="ASYNC">
<transaction mode="NONE"/>
<eviction strategy="NONE"/>
</replicated-cache>
</cache-container>
```

- Konfiguration besteht aus mehereren Einzelkonfigs für die verschiedene Caches.
- Eigenschaften die früher erwähnt worden sind , sind alle hier konfigurierbar.
- Sync oder Async kann gewählt werden.
- Auffallend hierbei ist, dass der hibernate cache container offenbar drei Strategien spezifiziert. Das default-cache Attribut gibt an, welcher der drei ausgewählt wird. Die anderen beiden müssen explizit angefordert werden. Das <file-store>- Tag spezifiziert, wo der Cache seine Daten speichern soll. Der Default-Pfad ist: <JBOSS_HOME>/standalone/data/web/repl . Er kann aber mit diesem Tag an einen beliebigen anderen Ort versetzt werden: <file-store relative-to="...path="..."/>

2.1.5. Anwendung

Listing 2.2: Anwendung Caching

```
  @ManagedBean
public class MyBean<K, V> {
```

```
@Resource(lookup="java:jboss/infinispan/hibernate")
private org.infinispan.manager.CacheContainer container;
private org.infinispan.Cache<K, V> cache1, cache2;
@PostConstruct
public void start() {
cache1 = container.getCache(); // returns default cache
cache2 = container.getChache("timestamps"); // explicit cache selection
}
}
```

Das ist alles! Es werden keine Wildfly spezifischen Klassen oder Annotationen mehr benötigt. Ein kleiner Haken bleibt noch: da Wildfly ein modulares System ist, ist auch Infinispan als Modul implementiert. Dieses Modul wird aber nicht automatisch geladen. Damit dies geschieht muss noch eine Modul-Dependency deklariert werden und zwar im File META-INF/MANIFEST.MF: Dependencies: org.infinispan export

3. Load Balancing

Load Balancing ist eine Methode zur Lastverteilung auf verschiedene Application Server-Instanzen. Mit Last sind von ausserhalb eintreffende (, gleichzeitige) Requests gemeint. Load Balancing soll eine Applikation skalierbar und hoch-verfügbar machen.

Skalierbarkeit Applikation kann mehrere Requests verarbeiten kann mittels zusätzliche Hardware oder erzeugen von Redundante Insanzen der Applikation `ohne Spirce Code der Applikation zu verändern.` **IdealFall** - Applikation skaliert linear.**Praxis :** Fläschenhalse, die diese Linearität bedrohen: gemeinsame Dienste. Es findet immer ein gewisses Mass an Synchronisation statt.

Bemerkung: Load Balancing ist keine Eigenschaft einer Applikation oder von Applikationsservern.

3.1. Load Balancers

Load Balancers gibt es als Hardware- und Software-Versionen. Hardware Load Balancers sind typischerweise teurer, aber auch schneller und vor allem zuverlässiger. Ein Load Balancer präsentiert sich mit einer einzigen IP-Adresse stellvertretend für eine ganze Gruppe (einem Cluster) von Servern. Er unterhält dabei eine Liste von internen (oder virtuellen) IP-Adressen für jede Maschine des Clusters. Wenn ein Load Balancer einen Request erhält, so passt er dessen Header so an, dass er auf eine Maschine des Clusters verweist.

High Availability Hat immer Maschine bereit Request anzunehmen.

Server Affinity Aufeinanderfolgende Requests an gleichen Maschine - `Sticky Sessions bei stateful Applikationen.`

Software Load Balancers Native Web Servers am besten geeignet.

Load Banalcing Topologie

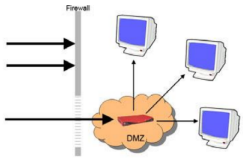

Abbildung 3.1.: figure

Dabei steht der Load Balancer in der Demilitarized Zone1 (kurz DMZ) und die Applikationsserver hinter weiteren Firewalls im Intranet. Eine häufige Kombination ist: ein Apache HTTP Server in der DMZ mit offenem Port 80. Auf Unix-Systemen sind die unteren Ports root vorbehalten, d.h. der HTTP Server muss als root gestartet werden. Somit ist auch klar, dass aus Sicherheitsgründen niemals ein Application Server direkt den Verkehr auf Port 80 entgegennehmen wird. Mit dieser Topologie ergeben sich aber folgende Vorteile:

- Jede seriöse Topologie wird eine DMZ verwenden, häufig wird auch das Intranet noch in verschiedene Sicherheitszonen unterteilt und mit Firewalls abgesichert.
- In der DMZ steht ein leichtgewichtiger LoadBalancer (evtl. sogar ein Hardware-Balancer) den zu hacken sowieso wenig bringt.
- Im Intranet stehen die Application Server, welche dann auch einfacher administriert werden können, als wenn sie in der DMZ stünden.

Zwischen dem Load Balancer und den Applikationsservern kann HTTP verwendet werden, oft wird aber das effizientere AJP2 eingesetzt. Das AJP wurde als binäres Protokoll entwickelt, welches zur Kommunikation von nativen Webservern zu Webcontainern wie Tomcat zur Anwendung kommt. Es ist TCP/IP-basiert und effizienter als HTTP. Zudem bietet es auch Support für SSL. Es gibt von Apache auch ein IIS-Plugin für AJP.

3.1.1. Strategieen

Random Zufällig eine Maschine.

Round Robin Riehe nach in eine Loop.

Sticky Session / First Available Zuerst werden neu eintreffende Requests nach einer Random- oder Round-Robin-Strategie verteilt. Der Load Balancer merkt sich aber für jede Request-Quelle (Client oder User) wohin der Request weitergeleitet wurde. Nachfolgende Requests von derselben Quelle werden dann immer an dieselbe Maschine weitergeleitet. Diese Strategie wird im Zusammenhang mit Server Affinity verwendet.

Sonnstiges Es gibt noch viele weitere Strategien, die z.B. statische Gewichtungen oder auch dynamische, lastabhängige Verteilmechanismen verwenden.

3.1.2. DNS Load Balancers

Einige DNS Server bieten auch Load Balancing an. Dabei wird der DNS Server angewiesen mehrere IP-Adressen für einen einzigen Domain-Namen zu unterhalten. Bei jeder DNS-Abfrage wird dann (meist im Round Robin Verfahren) eine andere IP-Adresse zurückgegeben. Diese Art des Load Balancings ist sehr einfach aufzusetzen, hat aber einige grundsätzliche Probleme:

1. Viele Clients (oder auch andere DNS Server) merken sich IP-Adressen in eigenen Caches um einen weiteren DNS-Lookup zu verhindern. Dies führt dann automatisch zu Sticky-Sessions bei einer einzigen Applikation. Bei einem DNS-Server kann dies aber zu einer Serveraffinität führen, die einen Server überlastet, während die anderen unterbeschäftigt sind.
2. DNS Server kennen keine Server Affinity, d.h. falls sich die Applikation selbst die IP-Adresse nicht merkt, oder ein Server abstürzt, wird der Request trotzdem stur weitergeleitet.

Load Balancing kann ohne Clustering betrieben werden und umgekehrt kann ein Cluster ohne Load Balancing betrieben werden. Da es aber viele überlappende Konzepte gibt, werden beide häufig gemeinsam betrieben.

4. Clustering

Durch das Load Balancing kann die Verfügbarkeit einer Applikation verbessert werden, einfach indem mehrere Maschinen zur Verfügung stehen um die Request zu bearbeiten. Das alleine reicht aber noch nicht aus, denn was passiert, wenn eine Maschine plötzlich ausfällt? Dieses Problem wird durch Clustering gelöst.

Cluster: Eine Gruppe von Computern die durch ein High-Speed-Netzwerk miteinander verbunden sind und so zusammenarbeiten, als ob sie eine Maschine mit mehreren CPUs wären.

4.1. Topologie

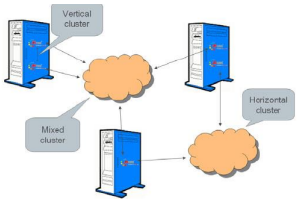

Abbildung 4.1.: figure

Abbildung 1: Cluster mit unterschiedlichen Topologien Es ist offensichtlich, dass der Skalierbarkeit (scaling up) eines vertikalen Clusters Grenzen gesetzt sind. In einer horizontalen Topologie ist es (zumindest theoretisch) immer möglich noch eine weitere Maschine hinzuzufügen (scaling out).

Horizontal vs Vertikal Clustering:
Falls genugend starke HW verfügbar ist dann ist vertikaler Clustering schneller da kein echter Netwerktraffic entsteht. ;eist interessiert die perfomance weniger als die Ausfallsicherheit. Diese kann nur dürch horizontaler Clustering erreicht werden. **IdealFall:** Im Idealfall besteht ein Cluster aus möglichst homogenen Knoten, d.h. gleiche Hardware und auch dieselben Applikationen deployed auf allen Knoten. In der Praxis ist dies nicht immer realisierbar, es kommt häufig vor, dass gewisse Knoten leistungsfähiger sind als andere, oder dass bestimmte Services nur auf spezieller Hardware zum Einsatz kommen.

4.2. Verteilung vs Clustering und andere Definitionen

Verteilung Aufteilen von logisch verschiedenen Applikationskomponenten auf physisch unterschiedliche Maschinen. Man spricht auch von verteilten Applikationen.

Clustering Gleiche App versch Maschine.Es ist möglich sowohl zu clustern als auch zu verteilen. Macht die Verteilung einer Applikation auf mehrere physisch getrennte Tiers Sinn?

Replikation Falls eine Applikation vollständig zustandslos ist, dann ist das Clustering einfach: ein Load Balancer reicht. Leider sind zustandslose Applikationen ein Ausnahmefall. Normalerweise wird in verschiedenen Stellen der Applikation Zustand gehalten: Als Session State im Web-Container oder als SFSB im EJB-Container. Auch Entities im Persistence-Context stellen Zustand dar.

Failover Was soll geschehen, wenn eine Maschine ausfällt, welche noch Zustand gehalten hat, also z.B. Session Context- Objekte oder SFSBs im Speicher hatte? Da der Zustand verloren gegangen ist, kann die Session nicht mehr (vernünftig) fortgesetzt werden. In einem Cluster ist nun die Idee, dass eine andere Maschine die Session übernehmen kann und gegenüber dem Client einfach fortfahren kann – das sogenannte Failover. Ein Client merkt nichts vom Ausfall (ausser einer vielleicht etwas längeren Responsezeit).

Fault Tolerance In einer zustandsbehafteten Applikation, bedeutet Fehlertoleranz (fault tolerance), dass der Zustand auch auf dem Knoten verfügbar ist, der im Notfall Sessions eines ausgefallenen Knotens übernehmen muss. Ein Beispielszenario: Eine Kundin ist am Bezahlvorgang an der virtuellen Kasse eines Webshops. Üblicherweise besteht dieser Vorgang aus mehreren einzelnen Schritten wie Rechnungsadresse und Versandadresse erfassen,

Kreditkartenangaben, Überprüfen der bestellten Waren, Bestätigung die allg. Geschäftsbedingungen gelesen zu haben, dem Abschicken der Bestellung und der „Danke“-Seite inkl. Bestätigungsmail versenden. Diese Schritte werden in einzelnen Request/Response-Schritten behandelt. Was würde die Kundin nun erleben, falls der Server mitten in dieser Kommunikation abstürzt und die Session, nicht aber der State der Applikation auf einen Failover-Server übergeht?

- Warenkorb
- Addresse
- Kreditkarteninfo
- Reservation konsistent mit Bestellung
- Bestätigungen
- Loginzustand
- History im Web Browser

Damit Sessions fehlertolerant sind, muss also eine Kopie des Session-States auf der Maschine zur Verfügung stehen, die den Request im Falle eines Failovers übernimmt. Das Kopieren des Zustandes auf andere Knoten in einem Cluster bezeichnet man als State Replication.

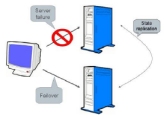

Abbildung 4.2.: figure

4.3. Replikationsarten

Sychrone Replikation	Buddy könnte offline sein. Unproblematisch solange nicht alle Buddies offline sind -> Timeout wählen
Asynchrone Replikation	Nur kitisch falls während Replikation ein Fehler auftritt
Gar keine	Performant kritisch während der ganzen Session.

Total replication Wenn jeder Knoten seine Zustände auf jeden anderen Knoten im Cluster repliziert spricht man von Total State Replication. Diese Art der Replikation bringt zwar die grösste Sicherheit, kostet aber am meisten. Da nun jeder Knoten alle Zustände aller anderen Knoten speichern muss kostet es neben dem Netzwerkverkehr auch Speicher und CPU-Ressourcen, da diese Zustände auch noch verwaltet werden müssen.

Buddy Replication Bei der Buddy Replication versucht man diese Kosten zu senken, indem jeder Knoten mindestens einen Buddy (engl. für Kumpel) erhält. Nun wird der State nur noch zu diesem Buddy repliziert. Im Failover-Fall muss nun dieser Buddy übernehmen.

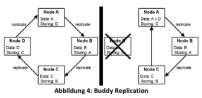

Abbildung 4.3.: figure

Active Replication Jeder Knoten hat alle notwendigen Ressourcen vorgängig Repliziert (inklusive der Datenbank). Ein Request wird an `alle` Knoten des Clusters `gleichzeitig` verschickt. D.h. alle Knoten berechnen simultan eine Response. Hierbei handelt es sich eigentlich nicht um eine Replizierung von Zuständen indem Kopien von einem Knoten zum anderen gemacht werden. Sondern jeder Knoten berechnet autonom eine Response. Danach wird über ein sog. Voting darüber abgestimmt, welche Antworten die richtigen sind. Diese Abstimmungen können nach eigenen Gesetzmässigkeiten erfolgen. Active Replication wird vor allem in sicherheitskritischen Applikationen eingesetzt, beispielsweise in Medizinalsoftware, in Steuersoftware von Flugzeugen oder in Kontrollsoftware von Kernkraftwerken. Die Knoten sind dabei häufig heterogen, denn neben der Ausfallsicherheit

geht es auch darum, nicht systematische Fehler auf allen Knoten zu machen. Es kommt (v.a. in der Flugzeugindustrie) vor dass die Knoten sogar von unterschiedlichen Programmierteams implementiert werden.

4.4. Fallover Unterstützung

State Passivation Es kommt oft vor, dass Sessions lange dauern (Stunden) und auch lange Zeit keine Requests mehr vorhanden sind für eine Session. In diesen Fällen kann der Zustand der Session z.B. auf eine Disk oder in eine DB persistiert werden. Wird dann wieder ein Request geschickt, kann der Zustand von der persistenten Quelle her geladen werden. Dieses Laden kann natürlich auch auf einen anderen Knoten erfolgen, als demjenigen der ursprünglich für die Session zuständig war.

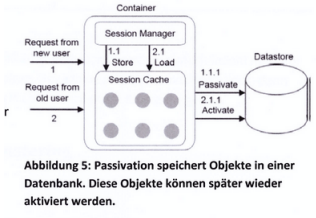

Abbildung 4.4.: figure

Code Invalidierung Eine weitere Möglichkeit effizient zu replizieren besteht darin, Daten zu löschen statt zu kopieren. Wie geht das? Wird ein Objekt im Cache eines Servers verändert, so müsste eigentlich der neue Zustand des Objektes an alle Failover-Maschinen verschickt werden. Jede dieser Knoten müsste dann bei sich im Cache nachschauen, ob das Objekt dort vorhanden ist und falls ja (Cache-Hit) das Objekt auch anpassen. Caches haben aber eine spezielle Eigenschaft: man kann jederzeit auf sie verzichten! Wenn nämlich auf dem Buddy-Knoten das Objekt nicht im Cache gefunden wird, muss auch nichts gemacht werden. Zwischenfrage: warum soll das Objekt bei einem Cache-Miss nicht einfach in den Cache eingefügt werden? Statt also bei einem Cache-Hit das Objekt anzupassen, kann es auch aus dem Cache gelöscht werden. Das spart vor allem Netzwerkbandbreite. Es muss nur noch übermittelt werden, welches Objekt zu löschen ist. Dies sind üblicherweise weniger Daten, als den kompletten neuen Zustand zu übermitteln.

4.5. Programdesign für fallovoer

Was passiert, falls mitten in einer Methode ein Knoten ausfällt? Was erwarten Sie? Problematisch wird es, wenn beim Failover Teile des Codes nochmals und damit doppelt ausgeführt werden. Davor sollte man sich schützen, indem man folgende Techniken anwendet:

1. Idempotente Operatinen : $f(x) = f(x)\dot{f}(x)$ Ein Beispiel für eine idempotente Funktion ist: setLocation(int x, int y). Nicht idempotent hingegen wäre: move(int dx, int dy).
2. Transaktionen Acid eigenschaften. Erorberung der Indempotenz - Interface redesign hilft - `move(dx,dy)` -> `move(ddx,ddy,dx,dy)` Verwendung von eine ünique Invocation ID"

5. Performanz messen

5.1. Aspekten der Messung

Applikation Performanz Algorithmen, Resourcenverbrauch selbst.

AS Performance Server, VM, Resourcen von Betriebsystem optimal benutzen.Prozessor,Memory,IO

Platform Stehen dem Betriebsystem alle Resourcen zur Verfügung die es benotigt. (Mem,CPU IO Graphics).

Extern Untersysteme,verbindungsqualitäten.

5.2. Erfassung der Messdaten

JMX Java Management Extension - erlaubt es Anwendungen und Systemobjekte zu verwalten und zu überwachen

MBean Managed Java Bean - Interface mit Namensmuster `xxxMBean` und eine Klasse xxx. Nachdem man eine Instanz erzeugt habe, kann man es beim JMX Server registrieren. Falls nötig ist es also möglich selbstentwickle MBeans zur Performanzmessungen oder überwachung einsetzen.

Ablauf:

Abbildung 5.1.: figure

JBoss verfolgt einen POJO Ansatz. Ersetzen von Services ist schwierug, da ein Client womöglich noch direkte Referenzen uf den Service hält. Dafür entfällt der JMX Server mit seinem Verwaltungsoverhead. Trotzdem ist es wierterhin angeboten weil Management tools darauf basiert sind.

5.2.1. JVMTI /JVMPI

- Natives Interface
- JVM Zugriff durch C++
- Bytecode modifizieren, Bytecode Instrumentierung.

  Instrumentierung auf 3 Arten

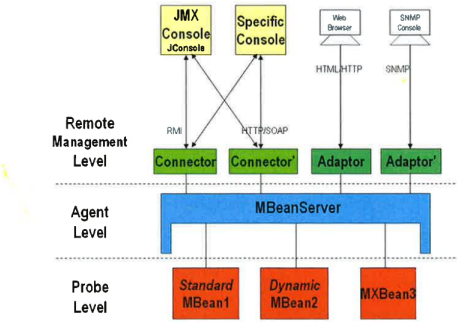

Abbildung 5.2.: figure

- Arten von Instrumentieren:

  Statisch - Bevor JVM Klasse ladet, Post build process.

  Zur Loadezeit mittels spezielle Classloader.

  Laufzeit : Hot Spot Compiler (dynamische Instrumentierung)

5.3. MessMethodik

1. Zeitbasierte Messung : Ausführungs-Stacks aller Threads einer Anwendung in einem definierten Intervall (Sample) abgefragt und analysiert. Je häufiger eine Methode auf Stack erscheint - höhere Ausführungszeit. Keine genauen Angaben über Ausführungszeit hat. Kurzlebige Methoden könen leicht übersehen werden.
   Sample Graph:

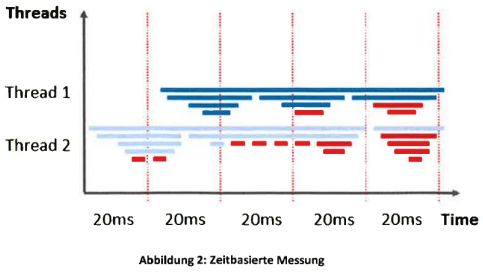

Abbildung 5.3.: figure

2. Eventbasierte Messung: Anstatt periodisch Snapshots vom Stack machen, werden einzelne Methodenaufrufe analysiert. Dabei wird für jenen Aufruf der Eintritts und Austrittszeitpunkt protokolliert. Um vorzunehmen brauchts Bytecode Instrumentierung.

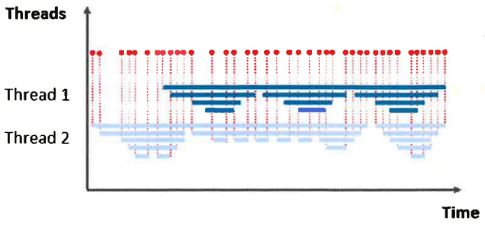

Abbildung 5.4.: figure

5.4. Zeit vs Eventbasiert

Zeitbasiert	Eventbasiert
- nur statistische Daten	- mehr Overhead
+ keine Änderung am anusgeführten Code	- modifizierte Code verhält sich anders.
+ geringer Impact da in der Regel weniger Messungen durchgeführt werden	+sehr detailierte Infos
-> grobe Lokalisierung von Performanzproblemen	detailierte Problemanalyse
geeignet für produktive Systeme	Reihenfolge von Events erkennbar

5.5. Overhead und Messdatenverfälschung

Messungen selber die Messungen beinflussen. Heisenbugs.
subsectionAntwortzeit Overhead

5.5.1. CPU und speicher Overhead

CPU sollte < 1% sein und Speicher auch zu beachten wenn wir Resultaten abliegen.

5.5.2. Netzwerk Overhead

Je detailierter eine Anwendung ausgemessen wird, desto mehr Daten müssen zi einem Profiling/Monitoring Tool übertragen werden. Als beispiel soll in einem eventbasierten Messverfahren in einer Serverlandschaft durch 50000 User je 5000 Methodenausführungen simuliert werden, dessen Name und die Ausfürhungsdauer bemerkt insegesamt 4GB.

5.6. Theroretische Grundlagen

Queueing Theorie Resourcen in Pool verwaltet. Eine Anfrage hohlt sich die benötigten Resourcen aus dem Pool oder wartet bis der Pool wieder freie Resourcen hat. Falsch dimensionierten Resourcenpools sind oft die Ursache für Performanzprobleme.

- Vergrösserung des Thread Pools bei gleicher CPU würde mehr Durchsatz bringen.
- Eine Ressource die im Einsatz ist, steht anderen Requests nicht zur Verfügung. Je länger sie im Einsatz ist desto länger mussen andere Request darauf warten(stehen also stil)

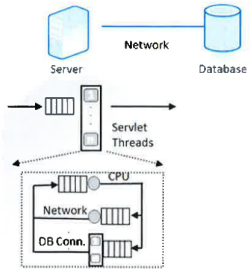

Abbildung 5.5.: figure

Little Gesetz

$N_s = \lambda t_s$ N_s die # Kunden.
λ die Ankunfsrate
t_s Verweildauer $t_s = t_w + tp$ wo w = wait and p = processing. Ein System ist dann stabil wenn die Anzahl neuer abfragen nicht grösser ist also die Anzahl abfragen die im gleichen Zeitraum maximal bearbeitet werden können.

- Abschätzen Pool grössen
- Validierung von Lasttest-Setups

Amdahl

$S = \frac{1}{(1-p)+\frac{p}{n}}$ S = speedup
P = Anteil Parallel Operation aus gesamt. N = Einheiten

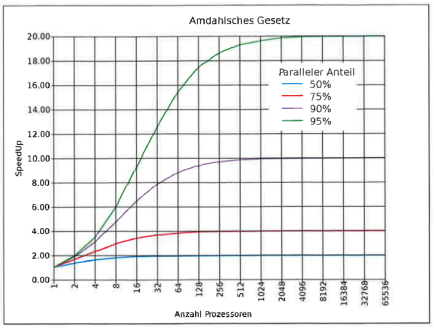

Abbildung 5.6.: figure

Erlang

$P_w = \frac{\frac{A^N}{N!}\frac{N}{N-A}}{\sum_{i=0}^{N-1}\frac{A^i}{i!}+\frac{A^N}{N!}\frac{N}{N-A}}$
P_w Wahrscheinlichkeit Kunde muss warten. A Last vom System in Erlang N Anzahl Telefonisten.

Benutzt für Poolgrössen und notwendige Ressourcen.

Teil II.

Arbeitsblätter

1. Wildfly

1. Was passiert, mit der Applikation HelloWorld, wenn Wildfly gestoppt und neu gestartet wird? Die Applikation wird wieder gestartet mit dem Reboot des AS.

2. Löschen Sie HelloWorld.war aus dem deployments Verzeichnis. Was passiert? Die Konsole vermeldet ein undeployment und ein HelloWorld.war.undeployed File ersetzt das *.deployed File.

3. Was ist ein exploded deployment? Statt einem war-File wird die Applikation als Verzeichnis deployed, d.h. der ausgepackte Inhalt des war-Files wird ins deployment-Verzeichnis kopiert.

4. Wann kann es Sinn machen exploded zu deployen? Für Entwickler, die nur einzelne Files ersetzen wollen kann es bequemer sein, nicht immer ein war-File erzeugen zu müssen.

5. Warum sollte man keine auto-deployment für exploded deployments einrichten? Weil ein deployment-Scanner mitten in der Kopieroperation des Verzeichnisses eine Änderung feststellen könnte und dann versuchten würde ein deployment durchzuführen während das Applikationsverzeichnis noch in einem inkonsistenten Zustand ist.

6. Was sind Markerfiles und wozu sind sie notwendig? Markerfiles zeigen den Zustand eines Deployments einer Applikation an. Sie tragen den Namen der Applikation (inkl. der .war-Endung) und fügen dann je nach Zustand noch eine weitere Endung hinzu. Da Files atomar verändert werden können, können Markerfiles auch in einem Multitaskingumfeld Auskunft über den Zustand eines Deployments geben.

7. Was passiert, wenn HelloWorld über die Management-Console entfernt wird? Die Applikation wird undeployed, und im deployment-Verzeichnis wird wieder ein HelloWorld.war.undeployed gesetzt.

8. Stoppen Sie den AS, löschen Sie die HelloWorld-Applikation und eventuelle .undeployed oder .deployed Dateien aus dem deployments-Verzeichnis und starten Sie den AS erneut. Nun deployen Sie HelloWorld über die Management-Console. Was passiert im Verzeichnis deployments? Schauen Sie auch in das oben erwähnte Konfigurationsfile standalone.xml! Wohin wurde deployed? Hinweis: beobachten Sie das tmp- und das data-Verzeichnis! Es ist kein war-File sichtbar! Im standalone.xml wird das deployment aber explizit am Schluss erwähnt. Der Eintrag dort erwähnt einen sha1-Schlüssel, der auf ein Verzeichnis im data/content Folder hinweist. Und zwar setzt sich der endgültige Dateiname aus dem SHA1-Schlüssel zusammen (<erste beiden Stellen>/<restliche Stellen>/content). Das eigentliche Deployment findet im tmp/vfs- Verzeichnis statt! Und zwar bei jedem Neustart des AS erneut. Das tmp-Verzeichnis kann also gelöscht werden und die Applikation wird einfach nochmals dorthin deployed.

9. Wozu diese beiden unterschiedlichen Deployment-Methoden? Hot-Deployment ist während der Entwicklung praktisch, in der produktiven Phase wird „offiziell" deployed. Hot-Deployment kann jeder machen, der Schreibzugriff auf das Verzeichnis hat, „offiziell" deployen können nur Administratoren.

10. Wie kann das Hot-Deployment abgestellt werden?In der Management-Console unter Profile/Core/Deployment Scanners kann das deployment deaktiviert werden. Dort kann übrigen auch ein anderer Pfad angegeben werden, damit z.B. unter Linux das Deploymentverzeichnis nicht unter dem Installationsverzeichnis stehen muss (Konflikt mit Schreibrechten).

11. Definieren Sie einen Logger, der auch den FINEST-Level-Log unseres Servlets in einem Log-File ausgibt. Log-Files können schnell sehr gross werden. Einerseits können Sie die Log-Geschwätzigkeit durch den Log-Level steuern (pro Logger falls nötig). Andererseits verwendet JBoss sogenannte RollingFileAppender um ein Log-File bei Erreichen bestimmter Kriterien zu schliessen und ein neues Log-File zu eröffnen. Wie müsste die Log-Konfiguration aussehen, damit höchstens 10 solcher Log- Files erzeugt werden, und keines grösser als 1kB ist? Erweitern Sie die Log-Konfiguration so, dass Logs vom HelloWorldServlet in ein eigenes File (z.B. apm.log) geschrieben werden.

Listing 1.1: wildlfly log config

```
Folgendes sollte im standalone.xml im subsystem logging drin stehen:
<size-rotating-file-handler name="APM">
<level name="FINEST"/>
<file relative-to="jboss.server.log.dir" path="apm.log"/>
<rotate-size value="1k"/>
```

```
<max-backup-index value="10"/>
</size-rotating-file-handler>
<logger category="HelloWorldServlet">
<level name="FINEST"/>
<handlers>
<handler name="APM"/>
<handler name="CONSOLE"/>
</handlers>
</logger>
```

2. Caching Puzzle

Cache grösse Was sind die Vorraussetzung für das löschen aus einem Cache mit limitierte grösse. (Wobei die alle limitierte Grösse haben) Das problem wird gelöst mit periodischen Testen für die notwendigen bedingungen oder ein Predikat für wenn man testen soll - bzw nur bei 50% Voll.
Dies setzt doch vorraus das man die Grösse von Objekten wissen kann in Bytes oder dass wir Annahmen darüber zuverlässig machen können. Ist aber in der Praxis nicht war und kann nur mittels JVM commandline Optionen für Heapsize geregelt werden - Cache in seperater JVM laufen lassen.
Eviction: Auch replacement Strategie : Wie soll ich entscheiden welche Elemente aus dem Cache gelöscht werden sollen. **Expiration:** Wann Objekte zu ältßind - wann sind die zu lange in dem Cache - Most Recently Used, Least Frequently Used usw.

Schnittstelle und Implementation Cache hit/Miss: Wann in der Cache etwas gefunden wird bzw bei Miss wenn ich oder der Framework von der Originalquelle hohlen muss.
Expliziter/Impliziter Cache : Explizit Cache - Ich bekomme null oder ein Exception zurück bei Cache miss und sollte selber um das hohlen aus der Originalquelle kummern. Implizit - Das bedeutet mehr oder weniger das wir ein Persistence Framework implementieren wie JPA sodass der Programmierer nicht bemerkt ob Caching benutzt wird oder nicht, er bekommt das Resultät so oder so und es wird hinter der Kulisse behandelt.

Memory Leaks Referenzen im Hash sind problematisch weil der Garbage Collector nicht da drin rekursiv durchgeht - Strong References. Man sollte die etwas abschwächern (wrap mit Weakreference) damit wenn die nicht mehr im Hash referenciert sind die aufgeräumt werden können.
Key Value Probleme: Equals methode muss überschrieben werden - d.h wenn die Keys gleich sind sind die entsprechende Objekte dann gleich.
(Angaben hier ohne gewähr)

Performance Keys sollten in der Hash eigentlich immutable sein damit die obere gGeichheitsprüfung OK wäre. Einsetzung von IdentityCache? Multithreading - mann sollte auf Konsistenz der Cache achten und entsprechendes Locking einsetzen. (Update im Original - Update im Cache) (Angaben hier ohne gewähr)

3. Load Balancing + Clustering

1. Load Balancing und asynchrone Requests. Wir haben Load Balancing immer nur im Zusammenhang mit synchronen Requests (http-Requests) betrachtet. Warum soll man nicht auch Load Balancing für asynchrone Requests verwenden? Überlegen Sie sich welche Konsequenzen es hat, wenn asynchrone Requests von einem Load Balancer auf einen Cluster verteilt werden.

   NOTE One thing to keep in mind is that load balancing is a mechanism for scaling applications that synchronously execute code. Applications that asynchronously execute requests don't necessarily need to load balance requests. In fact, often, you want to make sure that there's only a single instance of an asynchronous application or service running to ensure that it's managing all the requests. Running a single instance allows the application or service to manage ordering and priority over all the incoming requests.

2. Welche Nachteile haben Software Load Balancers gegenüber Hardware Load Balancers? Überlegen Sie z.B. warum Softwarelösungen weniger robust sind als Hardwarelösungen.

Load Balancer	Vorteile	Nachteile
Software Load Balancing	Billiger als Hardwarelösungen.Einige Balancer haben mehr Konfigurations- und Customization-Optionen. Kann besser an spezifische Bedürfnisse angepasst werden.	Die meisten Produkte können nicht mit grossen Sites oder komplexen Netzwerktopologien umgehen. Produkte die es können, haben immense Anforderungen an die Hardware. Load von anderen Applikationen auf der Load- Balancer Maschine kann Performance des Load Balancing kompromitieren.Angreifbar, da mögliches Ziel von Hackern.
Hardware Load Balancing	Typischerweise robuster Verarbeitet Datenverkehr auf Netzwerkebene. Das ist effizienter als Software "Entschlüsselung". Daten müssen nicht durch alle Netzwerklayers. Funktioniert mit allen Betriebssystemen. Bietet kein Angriffsziel für Hacker	Teurer als Softwarelösungen Komplizierter in Konfiguration und Unterhalt

3. High Availability in Zahlen: Rechnen Sie die Downtime pro Jahr für folgende Tabelle aus. Downtime ist die Zeit in der ein System nicht verfügbar ist. In welche Kategorie gehört Ihr Internet-Banking?

Table 12.1 How the various uptime percentages relate to actual time

Uptime	Availability based on number of nines	Allowed downtime per year
98%		7.3 days
99%	2-nine	87.6 hours
99.5%		43.8 hours
99.9%	3-nine	8.8 hours
99.95%		4.4 hours
99.99%	4-nine	53 minutes
99.999%	5-nine	5.3 minutes
99.9999%	6-nine	31 seconds
99.99999%	7-nine	3.1 seconds

Abbildung 3.1.: figure

4. High Availability kann nicht alleine durch Load Balancing erreicht werden. Warum nicht? Was wird sonst noch benötigt? Erklären Sie anhand eines Szenarios.

   Falls eine Maschine abstürzt sollten die anderen Maschinen in der Lage sein den Request zu übernehmen. Reines Load-Balancing kann dies nicht leisten, da beim Absturz einer Maschine deren Zustand nicht auf eine andere Maschine übertragen wird. Clustering leistet genau dies, in JBoss z.B. durch einen verteilten Cache gelöst.

5. Welche Alternativen gibt es zum Clustering?

   Applikation nur auf einer Maschine deployen. Dafür aber

   Algorithmen optimieren, Applikation tunen damit sie schneller läuft

   Stabile und redundante Algorithmen einsetzen um die Fehleranfälligkeit zu mindern.

   Schnellere Hardware einsetzen um die Performance zu steigern

   Sicherere Hardware einsetzen um das Absturzrisiko zu minimieren

   Verteilte Algorithmen. Applikation übernimmt dann die Verteilung selbst. Dezentrale und redundaten Datenhaltung und Berechnung sorgen für Failover.

6. Was müssen Sie tun, wenn Sie einen Cluster nur für Scalability haben wollen?
   Da reicht ein einfaches Load Balancing. Weil in diesem Fall keine veränderten Zustände auf andere Nodes im Cluster kopiert werden müssen wird die Applikation zusätzlich schneller.

7. Kann ein Cluster eine Applikation gleichzeitig skalierbar und zuverlässiger machen?
   Ja, das ist meistens der Sinn eines Clusters. Allerdings wird dies am besten erreicht, wenn nicht jeder Knotem im Cluster als Fail-Over-Knoten aller anderen Knoten dient. Daher wird mit Buddy- Replication für jeden Node nur ein Fail-Over-Node bestimmt. Der Kopieraufwand ist somit bedeutend verringert worden. Trotzdem ist der gesamte Cluster in der Lage mehr Last aufzunehmen als ein einzelner Server – die Skalierbarkeit wurde dadurch gewährt. Beachte: Skalierbarkeit heisst dass eine System mehr Last tragen kann, nicht dass eine einzelne Transaktion / ein einzelner Request schneller bearbeitet würde.

8. Cache-Invalidierung: Warum soll das Objekt bei einem Cache-Miss nicht einfach in den Cache eingefügt werden? (Diese Frage steht im Skript und ist im dortigen Zusammenhang zu verstehen!)
   Eine Cache-Invalidierung läuft so ab: statt den Inhalt wird eine Id des veränderten Objektes an die Buddy-Knoten geschickt. Jeder der Empfänger prüft nun, ob er das Objekt im Cache hat. Falls ja, wird das Objekt aus dem Cache gelöscht, Falls nein, muss gar nichts gemacht werden. Benötigt der Knoten das Objekt später wieder, muss er es von der Datenbank nachladen und kriegt somit wieder eine aktuelle Kopie.

   Die Frage bezieht sich nun auf eine Buddy-Knoten, der eine Id erhält und feststellt, dass er das Objekt gar nicht im Cache hat (also ein Cache-Miss). Achtung dies ist eine Fangfrage. Die Abfrage ob das Objekt im Cache ist, ist keine „echte“ Abfrage im Sinne, dass das Objekt auch benötigt wird. Würde der Buddy jedesmal das Objekt in den Cache nachladen, wenn er es nicht bei sich findet, würde unnötig viel Verkehr auf die DB erzeugt. Zudem wäre der Cache unnötigerweise mit Objekten gefüllt, die der Buddy-Knoten selber gar nicht braucht. Ein Knoten muss in der Lage sein den Zustand seines Buddys zu rekonstruieren im Fail-Over Fall. Er muss aber nicht dessen Caches bei sich nachführen, denn dadurch würde er die Leistung des Caches für seine eigenen Aufgaben vermindern.

9. Wie kann eine Methode idempotent gemacht werden. z.B. eine Methode die auf ein Konto einen Betrag x gutschreiben soll?
   Jede Methode kann mit einem eindeutigen Schlüsselwert idempotent gemacht werden. Das Gutschreiben eines Betrages z.B. würde eine Schnittstelle vorsehen, die nicht nur den Betrag enthält, sondern z.B. zusätzlich eine sogenannten unique invocation id. Diese wird z.B. der Webserver für jeden Request vergeben den er erhält. Wird nun (durch ein Fail-Over) die Gutschriftsmethode ein zweites Mal ausgeführt, kann die Methode dies erkennen anhand des Zeitstempels. Dazu muss sich die Methode allerdings die verarbeiteten Zeitstempel merken. Oft kann auch schon durch ein geschicktes Service-Design eine Idempotenz ohne unique invocation id erreicht werden.

4. Prüfungsfragen - Theorie

4.1. Multiple Choice

Stimmt	Stimmt nicht	Aussage
	X	Es ist generell nicht möglich mit einem Cache eine schlechtere Performance zu erreichen als ohne Cache.
X		Caching heisst verstecken. Daten werden versteckt in dem Sinne, dass von aussen nicht unterschieden werden kann, ob sie von der Originalquelle stammen oder von einem (von aussen her gesehen) versteckten Speicher.
X		Round-Robin ist eine Load Balancing Strategie.
X		Load Balancing erhöht die Verfügbarkeit.
X		Clustering erhöht die Verfügbarkeit.
X		Clustering erhöht die Ausfalltoleranz.
	X	Caching ist eine Voraussetzung für Skalierbarkeit.
X		Message Driven Beans können State halten bis zum Ende einer Transaktion.
	X	Der JEE Standard schreibt vor, dass eine Application Server Clustering anbieten muss.
	X	Der JEE Standard schreibt vor, dass Clustering über einen verteilten Cache realisiert werden muss.
X		Performance-Tuning kann immer nur für eine ganz bestimmte Konfiguration gemacht werden. In der Regel ist ein Tuning z.B. für eine geringe Last nicht übertragbar auf ein System mit hoher Last.
	X	Load Balancing erfordert Replikation von Zuständen.
X		JBoss verfügt über einen eigenen Loadbalancer für EJBs, benötigt aber einen externen Load Balancer für Web-Applikationen.
X		Laut dem Gesetz von Amdahl gilt: Der erreichbare Speedup durch Parallelisierung steigt linear mit der Anzahl verfügbarer Maschinen bzw. Prozessorkerne
X		Heisenbugs sind Fehler die entstehen, weil Messungen am Systen / an einer Applikation durchgeführt werden.
X		Das Gesetz von Little lautet: Ein System ist dann stabil, wenn mehr Requests verarbeitet werden können, als Neue im gleichen Zeitraum hinzukommen.
X		JBoss baut auf einer Kernel Architektur auf, welche je nach Bedarf um verschiedene Module (z.B. web, ejb, logging, datasources etc) erweitert werden kann.
X		Beim Deployment wird z.B. ein WAR-File in ein Unterverzeichnis der JBoss-Installation kopiert. Ist das Hot-Deployment aktiviert erkennt JBoss diesen Vorgang und initialisiert die neue Applikation automatisch.
X		Die Verzeichnisse für Hot-Deployment und für Deployment über die Admin-Konsole sind unterschiedlich.
X		JBoss kann mittels Ctrl-C gestoppt werden.
	X	Als „exploded deployment“ bezeichnet man das Fehlschlagen beim Auspacken des WAR-Files und die daraus entstandene Unterordnung im deploy-Verzeichnis.
	X	Hot-Deployment kann nicht ausgeschaltet werden.
X		Jeder AS kann neben den JEE-Standard-Konfigurationsfiles auch proprietäre Konfig-Files anbieten.
X		Beim Starten von JBoss wird ein Konfig-File eingelesen, in welchem unter anderem auch Memory-Grössen und Heap-Ratios für die JVM angegeben werden können.
	X	JBoss ist ein Standard und wird von JEE implementiert.
X		Ein JBoss Server kann einzeln (standalone) oder in einem Verbund (domain) betrieben werden.
X		Eine JBoss-Installation muss über die üblichen Installationsmechanismen des Host-Betriebssystemes angelegt werden. D.h. unter Windows werden z.B. Registry-Einträge gemacht, unter Linux ist der Package-Manager zwingend nötig um die Installation durchzuführen.

4.2. Sonnstiges

Zeitbasierte vs Erregnisbasierte Messung Was wann wozu gemessen wird, und die Nachteile

Zeitbasierte Messung Was : Methodenaufrufe auf Stack
Wann : Vordefinierter Zeitintervall
Wozu: Load Prediction und grobe Lokalisierung von Performanceproblemen
Nachteile: iv) Nur Momentaufnahmen möglich, was zu Informationsverlust führt. Ausserdem ist die Reihenfolge der Methodenaufrufe nicht erkennbar.

Erreignisbasierte Messung Was : Wann eine Methode startet und endet.
Wann :Beim Start und Ende einer Methode.
Wozu Ermittlung von lange laufenden Methoden, und den Abhängigkeiten, welche Methode welche anderen aufruft. Nachteile: Der Impact auf das System ist sehr gross. Dies kann zu anderem Verhalten führen und die Messung verfälschen. Speziell bei kurz laufenden Methoden.

Load Balancing

1. Erklären Sie was eine Sticky-Session ist. Warum und in welchen Fällen ist Stickyness bei Web- oder Enterprise-Applikationen ein Thema? Jede Anfrage durch denselben Client wird immer demselben Server zugeordnet. Hält eine Session Daten, welche nicht persistiert oder an alle Server übertragen wurde und soll der Benutzer auch bei einem erneuten Aufruf darauf Zugriff haben, muss er zum selber Server, welcher eine Session hält, weitergeleitet werden.

2. Was versteht man unter DNS-Loadbalancing? Der DNS-Server löst bei jeder Anfrage einer Domain, diese zu einer anderen IP auf. Meist Round-Robin.

3. Wozu dient Load Balancing? Geben Sie in diesem Zusammenhang auch an, was eine 3-nine-Uptime bedeutet ! Verteilung der Last auf mehrere Systeme. 99,9

Clustering

1. Beim vertikalen Clustering werden zwei Knoten des Clusters auf derselben Maschine deployed.

   Wozu wird dies eingesetzt? Bessere Ausnützung der Hardware.

   Nachteile? Ist die HW defekt, sind alle Teile der Applikation betroffen.

2. Sowohl beim Clustering als auch bei Load Balancing sind die beteiligten Maschinen von aussen gesehen nur als eine Maschine sichtbar. Was unterscheidet nun Clustering von Load Balancing? Beim Clustering werden Daten so unter den Systemen verteilt, dass der Benutzer im Falle eines Ausfalls eines Knotens dies nicht bemerkt, also keine Daten verloren gehen.

3. Erläutern Sie warum...

   eine Applikation mit Clustering immer langsamer ist als eine Applikation ohne Clustering? Zusätzlicher Overhead für das Verteilen der Daten.

   man denn Clustering überhaupt einsetzen soll? Es erhöht die Ausfallsicherheit.