Freie Universität Berlin

Bachelor-Arbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Software Engineering

Evaluation von Verfahren zur Realisierung von Hochverfügbarkeit und Skalierbarkeit einer kritischen Serveranwendung, am Beispiel des Leitstellensystems der Berliner Feuerwehr

Alexander Kurth
alexander.kurth@kabelmail.de

Betreuer: Prof. Dr. Lutz Prechelt
Zweitgutachter: Prof. Dr. Adrian Paschke
Betreuer bei Sopra Steria Consulting: Klaas Hagemann

Berlin, 26.05.2015

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben. Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Datum	Unterschrift
26.5.2015	Alexander Kuhl

Alexander Kurth

Sperrvermerk

Die vorliegende Arbeit enthält vertrauliche Daten und Informationen über das Unternehmen Sopra Steria Consulting. Es dürfen keinerlei Kopien oder Abschriften, auch nicht in digitaler Form, ohne schriftliche Genehmigung des Unternehmens Sopra Steria Consulting angefertigt werden. Die Ausarbeitung ist nur den betreuenden Referenten sowie den Mitgliedern des Prüfungsausschusses zugänglich zu machen. Abweichend hiervon darf der Inhalt der Arbeit während einer Sperrfrist von 2 Jahren ab dem Abgabedatum nur mit ausdrücklicher Genehmigung des Unternehmens Sopra Steria Consulting an Dritte weitergegeben werden. Nach Ablauf der Sperrfrist ist diese Genehmigung nicht mehr erforderlich.

Berlin, 26.5.2015
Ort, Datum

Alexander Kurth
Unterzeichner

Alexander Kurth

Zusammenfassung

Aus gegebenem Anlass befindet sich zur Zeit das neue Leitstellensystem der Berliner Feuerwehr in Entwicklung. Für solch ein System müssen mehrere wichtige Elemente beachtet werden. In dieser Arbeit sollen dazu Verfahren evaluiert werden die zur Sicherstellung der Hochverfügbarkeit und Skalierbarkeit von zwei kritischen Systemkomponenten dienen können.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Zielsetzung 1

1.2 Aufbau und Verlauf 1

1.3 Ausgangssituation 1

2 Theoretische Grundlagen **2**

2.1 Einleitung 2

2.2 Hochverfügbarkeit 2

2.2.1 Allgemein 2

2.3 Skalierbarkeit 4

2.3.1 Allgemein 4

3 Allgemeine Beschreibung des Systems **5**

3.1 Einleitung 5

3.2 Fremdsysteme 6

3.2.1 Bezug der Fremdsysteme auf die Hochverfügbarkeit . . 8

3.2.2 Bezug der Fremdsysteme auf die Skalierbarkeit 8

3.3 IGNIS Plus 9

3.3.1 Applikation Server 10

3.3.2 Datenbank 11

4 Detailbeschreibung der Systemteile **11**

4.1 Einleitung 11

4.2 Client Server Kommunikation 12

4.3 Server Client Kommunikation 12

4.4 Suchen über den Datenbestand 13

4.5 Zusammenfassung 13

5 Auswahl der Lösungsverfahren **14**

5.1 Einleitung 14

5.2 Java Message Service 14

5.2.1 Clients melden sich bei allen Servern an 15

5.2.2 Zentraler Nachrichtenprovider 16

5.2.3 Verteilung der Nachrichten an alle Server 17

5.2.4 HornetQ in Jboss Clusterconfiguration 18

5.3 Apache Lucene 19

5.3.1 Apache Lucene Master/ Salve 20
5.3.2 Apache Lucene Replicated-Cache 21

6 Implementierung und Tests **21**
6.1 Konfiguration HornetQ 21
6.2 HornetQ in Jboss Clusterconfiguration, Tests 24
6.2.1 Testszenario 1: Verteilung der Nachrichten 24
6.2.2 Testszenario 2: Verteilung der Nachrichten, bei Ausfall eines Servers 28
6.3 Apache Lucene Infinispan, Implementation 32
6.4 Apache Lucene, Infinispan Testszenarien 36
6.4.1 Testszenario 1: Clusterweiter zugriff auf Index-Files . 36

7 Fazit **38**

8 Anhang **41**
8.1 Sender.java 41
8.2 Empfaenger.java 43
8.3 EmpfaengerT2.java 45

Abbildungsverzeichnis

1 Gesamtübersicht IGNIS Plus 5
2 IGNIS Plus Verteilungsdiagramm 9
3 IGNIS Plus Serverkomponenten Sicht 10
4 Clients melden sich bei allen Servern an 15
5 Zentrahler Nachrichtenprovider 16
6 Verteilung der Nachrichten an alle Server 17
7 HornetQ in Jboss Clusterconfiguration 18
8 Master Slave für Apache Lucene 20
9 Testaufbau HornetQ 1 24
10 Testaufbau HornetQ 2 28
11 Testaufbau Infinispan 36

1 Einleitung

1.1 Zielsetzung

Ziel dieser Arbeit ist die Evaluation ausgewählter Verfahren zur Realisierung von Hochverfügbarkeit und Skalierbarkeit in einer kritischen Client Server Anwendung.

1.2 Aufbau und Verlauf

Zuerst soll eine kurze Übersicht über den Aufbau der Arbeit geben werden. Das erste Kapitel widmet sich den Theoretischen Grundlagen zur Hochverfügbarkeit und Skalierbarkeit. Danach erfolgt eine allgemeine Beschreibung des Systems, der verwendeten Technologien und Frameworks. Im darauf folgenden Kapitel erfolgt eine Detailbeschreibung der Systemteile die für die Sicherstellung der Hochverfügbarkeit und Skalierbarkeit beachtet werden müssen. Als nächstes folgt die Auswahl möglicher Lösungen unter der gegeben Problemstellung. Es folgen dann die Implementationsdetails und die Funktionstest sowie deren Ergebnisse.

1.3 Ausgangssituation

Vorab soll an dieser Stelle aber die Ausgangssituation und die eigentliche Problemstellung spezifizieren werden. Aus gegebenem Anlass befindet sich zur Zeit das neue Leitstellensystem der Berliner Feuerwehr in Entwicklung. Für solch ein System müssen mehrere wichtige Elemente beachtet werden. In dieser Arbeit sollen dazu Verfahren evaluiert werden die zur Sicherstellung der Hochverfügbarkeit und Skalierbarkeit von zwei kritischen Systemkomponenten dienen können. Zuerst soll aber das System als ganzes betrachtet werden. Zur Umsetzung des Leitstellensystems der Berliner Feuerwehr wird eine Client-Server basierte Architektur verwendet. Bei der der Server gegen Ausfälle geschützt werden muss und wegen plötzlich auftretender Lastspitzen ebenfalls gut Skalierbar sein soll.

2 Theoretische Grundlagen

2.1 Einleitung

An dieser Stelle sollen vorab nun die Theoretischen Grundlagen erläutert werden. Die dem Verständniss der Anforderungen dienen, welche den zu evaluirenden Verfahren zu grunde liegen. Dazu soll als erstes die Hochverfügbarkeit (High-Avability) und darauf folgend die Skalierbarkeit (Scalability) vorgestellt werden.

2.2 Hochverfügbarkeit

2.2.1 Allgemein

Betrachtet man ein System als Blackbox, so kann dieses als Hochverfügbar eingestuft werden wenn es mindestens 99% der Laufzeit verfügbar ist. Um eine genauere Einteilung zu ermöglichen werden entsrechende Systeme anhand ihrer Prozentualen Verfügbarkeit in die folgenden Verfügbarkeitsklassen eingeteilt.

- Verfügbarkeitsklasse 2: 99,0% darf in einer Zeitspanne von 365,25 Tagen (ein Jahr) nur eine Ausfallzeit (SZeit in der das System nicht Verfügbar ist") von 438 Minuten pro Monat besitzen.[1]
- Verfügbarkeitsklasse 3 mit 99,9%, was einer Ausfallzeit von ca. 43:48 Minuten pro Monat entspricht[1]
- Verfügbarkeitsklasse 4 mit 99,99%, was einer Ausfallzeit von ca. 4:23 Minuten pro Monat entspricht[1]
- Verfügbarkeitsklasse 5 mit 99,999%, was einer Ausfallzeit von ca. 26,3 Sekunden pro Monat entspricht[1]
- Verfügbarkeitsklasse 6 mit 99,9999%, was einer Ausfallzeit von ca. 2,63 Sekunden pro Monat entspricht[1]

[1] [Hoc15]

Die Verfügbarkeit A (Avability) eines Systems errechnet sich dabei wie in [Hoc15] angegeben aus:
$A = \left(1 - \frac{Ausfallzeit}{Produktionszeit + Ausfallzeit}\right)$
beziehungsweise in Prozent aus:
$A = \left(\frac{Produktionszeit}{Produktionszeit + Ausfallzeit}\right) \cdot 100$
Was aber bedeutet nun Hochverfügbarkeit für ein System und dessen Komponenten? Für ein entsprechendes System teilt sich die Hochverfügbarkeit in drei Teilbereiche auf:

- Zuverlässigkeit (Reliability):

  > "degree to which a system, product or component performs specified functions under specified conditions for a specified period of time"[1]

- Verfügbarkeit (Availability):

  > degree to which a system, product or component is operational and accessible when required for use[1]

- Fehlertoleranz (fault tolerance):

  > degree to which a system, product or component operates as intended despite the presence of hardware or software faults[1]

Soll nun die Verfügbarkeit eines Systems erhöht werden so gibt es dafür zwei Ansätze:

1. Erhöhung der Zeit vom Starten des Systems bis zum Systemversagen
2. Minimierung der Zeit die zum beheben des Systemversagens benötigt wird

Dabei spiegelt Ansatz 1 den traditionellen Ansatz wieder und Ansatz 2 das "Resilient Software Design"[2]. Eine genauere Ausführung beider Konzepte würde allerdings den Rahmen dieser Arbeit sprengen, daher soll dies für unsere Zwecke genügen.

[1] [AVI15]
[2] [Fri15], p. 33

2.3 Skalierbarkeit

2.3.1 Allgemein

Unter Skalierbarkeit versteht man im Allgemeinen das die Leistungsfähigkeit eines Systems proportional zu der Menge der hinzugefügten Ressourcen ansteigt. Um die Eigenschaften eines solchen Systems genauer zu Beschreiben unterscheidet man zwischen der Vertikalen und der Horizontalen Skalierbarkeit. Dabei bedeutet:

Skalierbarkeit:
das die Leistungsfähigkeit bei Verbesserung der Hardware des Systems (hinzufügen von RAM, Erhöhung der Prozessohrenanzahl usw.) ansteigt.

Horizontale Skalierbarkeit:
das die Leistungsfähigkeit des Systems beim Hinzufügen von zusätzlichen Knoten ansteigt.

Skalierungstyp
Um die Skalierungseigenschaften eines Systems genauer zu beschreiben, können diesem verschiedene Skalierungstypen zugeordnet werden. Dazu unterscheidet man zwischen vier Verhaltensarten. Wobei auch mehrere auf ein System zutreffen können.[1]

Lastskalierbarkeit
Die Lastskalierbarkeit beschreibt das Verhalten bei einer steigenden Anzahl von Anfragen an das System. Wenn ein System eine Lastskalierbarkeit aufweist, bedeutet dies, dass bei steigender Anfragemenge die Bearbeitungszeit pro Anfrage konstant bleibt.[1]

Räumliche Skalierbarkeit
Ein System Skaliert Räumlich wenn der Speicherbedarf bei größer werdender Anfragemenge nicht inakzeptabel hoch ansteigt.[1]

Zeitlich-räumliche Skalierbarkeit
Ein System weist eine Zeitlich-räumliche Skalierbarkeit auf wenn sich die Anzahl von Objekten die das System verwaltet nicht auf die Performance auswirkt.[1]

Strukturelle Skalierbarkeit
„Strukturelle Skalierbarkeit zeichnet ein System aus, dessen Implementierung das Erhöhen der Anzahl von Objekten innerhalb eines selbst definierten

[1] [Ska15]

Bereichs nicht maßgeblich behindert.“[1]

3 Allgemeine Beschreibung des Systems

3.1 Einleitung

Dieser Abschnitt beinhaltet die allgemeine Beschreibung der Systemkomponenten, dabei soll weitestgehend auf Implementationsdetails verzichtet werden. Er soll hauptsächlich dazu dienen das System, im speziellen den Server, zu Analysieren und die Technologien zu identifizieren die unter den Gesichtspunkten der Hochverfügbarkeit und Skalierbarkeit relevant sind.

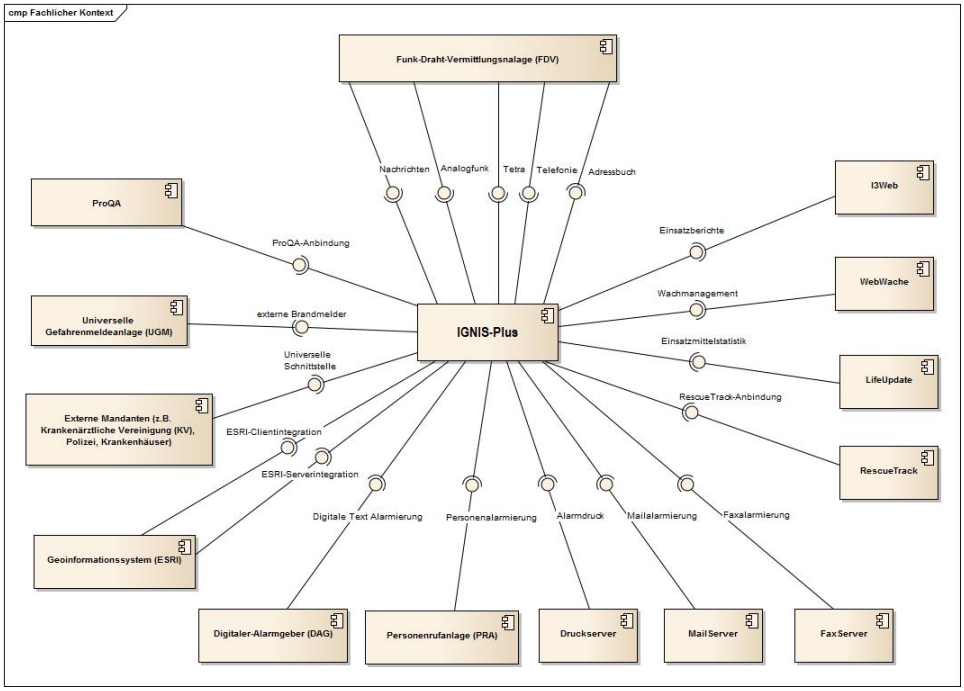

Abbildung 1: Gesamtübersicht IGNIS Plus [2]

Betrachten wir IGNIS Plus aber zunächst als Geschlossenes System. Wie in Abbildung 1 dargestellt kommuniziert IGNIS Plus mit verschiedenen anderen Systemen, im folgenden Fremdsysteme genannt. Es gibt folgende Fremdsysteme:

[1] [Ska15]
[2] [TS13, S. 6]

3.2 Fremdsysteme

Funk-Draht-Vermittlungsanlage (FDV) Die Funk-Draht-Vermittlungsanlage verbindet IGNIS Plus mit dem Analogfunk und Digitalfunk sowie der Telefonanlage der Berliner Feuerwehr. [1]

Digitaler Alarmgeber (DAG) Der Digitale Alarmgeber ist ein System zur Kommunikation mit externen Empfangsgeräten, sogenannten Pagern. Über die Pager ist IGNIS Plus befähigt, Einsatzkräfte mittels Textnachricht zu alarmieren.[1]

Personenrufanlage (PRA) Eine Personenrufanlage ist ein System zur Alarmierung von Trupps beziehungsweise Einsatzkräften mittels Textnachricht oder Sprachdurchsage. Jede Wache hat eine eigene PRA, entsprechend kommuniziert IGNIS Plus mit mehreren Personenrufanlagen. Bei einer Alarmierung werden die Einsatzdaten durch IGNIS Plus an die entsprechenden PRA übergeben.[1]

Druckserver Der Druckserver ermöglicht die Alarmierung von Einsatzkräften mittels Ausdruck der Einsatzinformationen durch am Druckserver angeschlossene Drucker auf den Wachen.[1]

Mailserver Der Mailserver ermöglicht die Alarmierung von Einsatzkräften mittels Email. Die Funktionsweise ist analog zur Alarmierung über den Druckserver.[1]

Faxserver Der Faxserver ermöglicht die Alarmierung von Einsatzkräften mittels Fax. Die Funktionsweise ist analog zur Alarmierung über den Druckserver.[1]

ProQA Das externe System ProQA ergänzt die Einsatzdatenerfassung für Einsätze im Kontext Brand sowie medizinische Notfälle.[1]

Universelle Gefahrenmeldeanlage (UGM) Die Universelle Gefahrenmeldeanlage ist ein System der Firma Bosch, welches mit einer Vielzahl von

[1] [Con15, S. 6-9]

externen Brandmeldern (ausgelöst durch automatische Sensoren oder durch manuelle Betätigung) verbunden ist.[1]

Mandatensystem (z.B. Krankenärztliche Vereinigung (KV), Polizei, Krankenhäuser) Die Systeme externe Mandanten (z.B. Kranken kassenärztliche Vereinigung (KV), Polizei, Krankenhäuser) benutzen die folgenden fachlichen Schnittstellen für den Datenaustausch mit Ignis Plus [1]

Universelle Schnittstelle (USST) Die Universelle Schnittstelle ist eine fachliche Schnittstelle von IGNIS Plus zur Übergabe von Einsätzen, Einsatzmitteln sowie zur Übermittlung von Informationen bzgl. Einsatzmittel.[1]

I3Web Das I3Web System stellt die Technik bereit um für abgeschlossene Einsätze Berichte erfassen zu können.[1]

WebWache Die WebWache ist das externe System, welches den Mitarbeitern in den Wachen ermöglicht, mit IGNIS Plus zu interagieren, um beispielsweise Informationen über Einsätze und Einsatzmittel abzurufen und um Stammdaten (z.B. Trupps, Einsatzmittel) der Wache im betrieb anzupassen.[1]

LiveUpdate Das System LiveUpdate erstellt Statistiken über Einsätze und Einsatzmittel.[1]

RescueTrack (Internet) RescueTrack ist eine Internetanwendung zur Statusanzeige von Rettungshubschraubern. [1]

GIS (Geoinformationssystem) Das Geoinformationssystem stellt die folgenden Schnittstellen bereit; für: Die Darstellung von Karten mit den entsprechenden fachlichen Inhalten von IGNIS Plus (Einsatz, Einsatzkräfte, etc.) Die Übernahme von Geokoordination aus dem GIS heraus nach IGNIS Plus Die Berechnung von Routen. Für die Disponierung ist insbesondere wichtig zu ermitteln, wie lange ein Einsatzmittels auf dieser Route zu einem Einsatzort benötigt. Die Übernahme der aktuellen Positionen von IGNIS

[1] [Con15, S. 9-10]

Plus-Fachdaten (Einsätze, Einsatzmittel, etc.), so dass diese auf den Karten jeweils die aktuelle Position anzeigen kann.[1]

3.2.1 Bezug der Fremdsysteme auf die Hochverfügbarkeit

Das Wissen um die Fremdsysteme die mit IGNIS Plus Kommunizieren ist in sofern relevant da jede Schnittstelle eine zusätzliche Fehlerquelle ist. Daher muss zu jederzeit sichergestellt werden das IGNIS Plus im Fehlerfall eines der Fremdsysteme weiterhin Einsatzfähig ist, mit Ausnahme der Funktionalität des entsprechenden Fremdsystems.

3.2.2 Bezug der Fremdsysteme auf die Skalierbarkeit

In dieser Hinsicht muss sichergestellt werden das bei einem erhöhten Datenaufkommen durch die Fremdsysteme die Antwortzeiten von IGNIS Plus auf Client anfragen nahezu unverändert bleiben.

[1] [Con15, S. 10]

3.3 IGNIS Plus

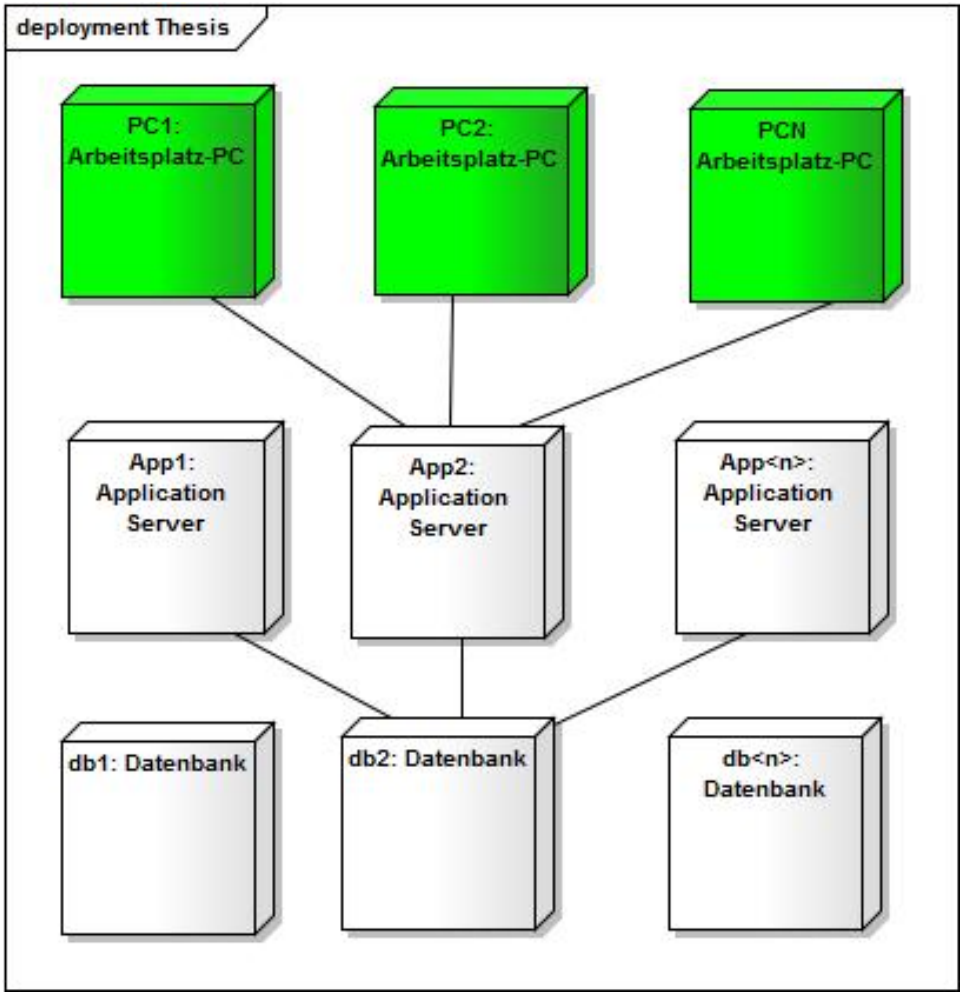

Abbildung 2: IGNIS Plus Verteilungsdiagramm (gekürzte Fassung aus [1])

Wie in Abbildung 2 (Verteilungsdiagramm) dargestellt, besteht IGNIS Plus aus mehreren Systemkomponenten. Bei diesen Handelt es sich um die Clients (Arbeitsplatz-PC 1 bis N), um die Appliaktion Server (App 1 bis N) und um die Datenbank. Diese wird durch ein Oracle Rac realisiert. Für den weiteren Verlauf dieser Arbeit werden ausschließlich der Application Server und dessen Komponenten betrachtet, da die zu evaluirenden Verfahren die Hochverfügbarkeit und Skalierbarkeit dessen sicherstellen sollen.

[1] [TS13, S. 80]

3.3.1 Applikation Server

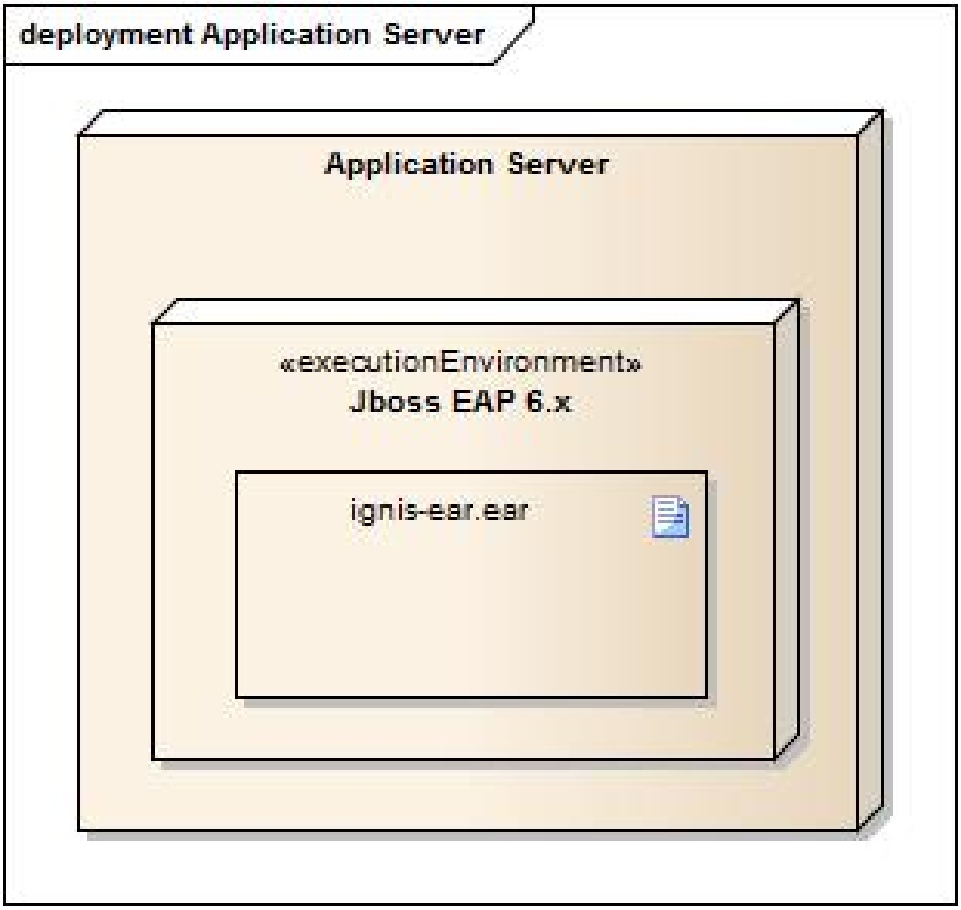

Abbildung 3: IGNIS Plus Serverkomponenten Sicht[1]

Wie in Abbildung 3 zu sehen, besteht der Server aus dem Jboss EAP 6 [2]. Auf diesem ist die eigentliche Anwendung (IGNIS Plus) Deployed. Diese bildet die Geschäftsprozesse der Leitstelle der Berliner Feuerwehr mithilfe von Jave EE ab.
Dabei erfolgt jede Kommunikation in Client-Server Richtung über Webinterfaces, die über die Jboss Serversoftware bereitgestellt werden.
Die aus dieser Kommunikation entstehenden Daten werden innerhalb einer Oracle Datenbank gespeichert, dazu gehören unter anderem die angelegten

[1] [TS13, S. 81]
[2] Jboss Referenzen: [Jbo15b, jbo15a]

Einsätze.
Die Server-Client Kommunikation, sofern es sich nicht um Antworten auf Client anfragen handelt, wird über eine Message Oriented Middleware Realisiert. Jeder Client bucht dabei nur die für ihn Relevanten Nachrichten und erhält diese dann sobald sie vorliegen.
Jeder Client ist zu einem bestimmten Zeitpunkt nur mit einem Server verbunden. Zur besseren Lastverteilung sind alle vorhandenen Clients auf die vorhandenen Server aufgeteilt. Daher muss jede von einem Server produzierte Nachricht auf alle anderen Server verteilt werden.
Die Bearbeitung aller eingehenden Nachrichten erfolgt in Abschnitten. Dies erlaubt eine strikte Trennung aller Arbeitsschritte. Für die Verbindung der einzelnen Arbeitsschritte wird ebenfalls die Message Oriented Middleware verwendet.
Zwangsläufig sind für ein Leitstellensystem in einer Stadt wie Berlin sehr große Datenmengen vorrätig zu halten, darunter fallen zum Beispiel das Straßen- und das Hausnummernverzeichnis. Da bei der Aufnahme von Einsätzen ständig über dieses gesucht werden muss, hat man sich dazu entschlossen diese im Speicher vorzuhalten.
Im Gegensatz zur Erwartung sind weder das Straßen- noch das Hausnummernverzeichnis vollständig. Durch bauliche Maßnahmen und durch das Städtewachstum unterliegen diese regelmäßigen Änderungen.

3.3.2 Datenbank

Wie bereits erwähnt müssen für die Funktionsfähigkeit von IGNIS Plus gewisse Informationen stets verfügbar sein. Dazu gehören unter anderem die bereits erwähnten Lokalitäten, wie Straßen und Hausnummern. Die Zugriffsmöglichkeit auf diese muss daher zu jederzeit sichergestellt werden. Dies wird mit Hilfe eines Oracle Datenbankclusters sichergestellt.

4 Detailbeschreibung der Systemteile

4.1 Einleitung

In diesem Abschnitt wird genauer auf die genutzten verfahren und Technologien eingegangen. Es soll hauptsächlich dazu dienen die Technologien zu

Identifizieren welche Auswirkungen auf die Hochverfügbarkeit beziehungsweise auf die Skalierbarkeit haben.

4.2 Client Server Kommunikation

In diesem Abschnitt wird die Client Server Kommunikation genauer betrachtet. Welche, wie bereits erwähnt, Webschnittstellen nutzt. Spezifischer wird hier die JAX-WS 2.2[1] Implementierung verwendet die im JDK 6 enthalten ist und SOAP[2] abbildet. Vereinfacht kann man sagen das bei SOAP die Daten in XML Strukturen abgebildet werden und diese dann über die Anwendungsschicht (TCP/ IP Schichtenmodel) übertragen werden. Was einen gewissen Overhead zur Folge hat. Für die Client Server Kommunikation bedeutet die nutzung von Webinterfaces aber auch, dass sich Mithilfe eines Lastverteilers die Anfragen der Clients sehr gut auf mehrere Server verteilen lassen. Dadurch ist die Skalierbarkeit dieser Komponente von IGNIS Plus sichergestellt. Die Authentifizierung der Clinets wird an dieser Stelle nicht genauer betrachtet.

4.3 Server Client Kommunikation

Als nächstes wird die Server Client Kommunikation betrachtet, bei der es sich nicht um Antworten auf die Webaufrufe der Clients handelt. Dazu soll an dieser Stelle nur die Kommunikation, die zwischen einem Server und einem Client entsteht genauer untersucht werden. Wenn der Server den Client über Änderungen informieren muss gibt es verschiedene Wege dies zu tun. Ist der Client dem Server bekannt, kennt er also dessen IP, kann er, sofern vorhanden, diesen über entsprechende Schnittstellen informieren. Umkehrschluss muss Server eine Liste über alle Clients vor- und aktuell halten. Dies würde umso komplizierter werden, je mehr Server und Clients zum System gehören, da festgelegt werden muss wer wem was mitteilt. Da dies einen unnötigen Aufwand bedeutet, hat man sich für eine Nachrichtenbasierte Technologie entschieden, bei der sich jeder Client nur die für ihn wichtigen Nachrichten abholt.
Dazu wird die Java Message Service (JMS) Api genutzt, welche sich aus dem Message Oriented Middleware (MOM) Konzept ableitet. Diese ist dabei

[1] [jax15]
[2] [W3C15, Wik15]

durch JSR 914 genau Spezifiziert[1].
Zur verwendung der JMS Api wird ein Provider benötigt, der die Client Anmeldungen und Subscripstions verwaltet. Der Jboss Server bietet einen solchen Provider Service in Form von HornetQ[2] an, dieser ist standardmäßig Teil des Jboss EAP 6 Servers.

4.4 Suchen über den Datenbestand

Als letztes wird die Suche über den Datenbestand genauer betrachtet. Wie bereits erwähnt werden alle Daten innerhalb einer Datenbank abgelegt. Wenn nun innerhalb dieser Daten gesucht werden muss, zum Beispiel beim aufnehmen eines Einsatzes, dann müssen alle relevanten Daten aus der Datenbank in den Server übertragen werden. Um dies und die daraus folgenden Nachteile für die Skalierbarkeit zu vermeiden hat man sich dazu entschlossen alle häufig benötigten Daten im Server vorzuhalten. Dazu wird Apache Lucene[3] verwendet, dabei handelt es sich um eine Bibliothek zur Volltextsuche. Mit dieser werden Indexfiles angelegt, die dazu dienen die Zeit die zum Suchen benötigt wird zu verkürzen.
Dies führt allerdings bei Systemen, die aus mehr als einem Knoten bestehen zu einem Konsistenzproblem. Da bei einer Änderung am Datenbestand die Indexfiles auf veraltete Informationen verweisen würden.

4.5 Zusammenfassung

Aus den in diesem Kapitel behandelten Komponenten vom IGNIS Plus ergeben sich nun die folgend zwei Problemstellungen. Erstens, Auswahl eines Verfahrens zur Sicherstellung der Hochverfügbarkeit und Skalierbarkeit des JMS Providers. Zweitens, Auswahl eines Verfahrens zur Sicherstellung der Konsistenz aller Indexfiles innerhalb des Serverclusters von IGNIS Plus. Die Konsistenz der Indexfiles ist im Bereich der Skalierbarkeit des Systems entscheidend, da nur so Sichergestellt wird, dass Suchanfragen bei allen Knoten des Systems das gleiche Ergebnis liefern.

[1] JMS Referenz: [JSR15]
[2] HornetQ Referenz: [Hor15a]
[3] Lucene Referenz: [Apa15]

5 Auswahl der Lösungsverfahren

5.1 Einleitung

In diesem Abschnitt werden die Verfahren vorgestellt, die zur Sicherstellung der Hochverfügbarkeit und Skalierbarkeit genutzt werden können. Diese beziehen sich auf die im vorherigen Abschnitt ausgewählten Teile des Servers. Dazu werden die möglichen Lösungsansätze zunächst Theoretisch betrachtet. Die Verfahren sollen dabei auch auf ihre Wartbarkeit hin untersucht werden. Wartbarkeit meint in diesem Zusammenhang die Komplikationen, beziehungsweise den Aufwand der nötig ist, um eine neue Ressource, zum Beispiel einen weiteren Applikation-Server, zum bestehenden Cluster hinzu zu fügen. Begonnen wird mit den Verfahren die dazu genutzt werden die JMS Topics auf allen Servern verfügbar zu halten. Danach sollen die Verfahren untersucht werden die zur Verteilung der Indexfiles vom Lucene dienen.

5.2 Java Message Service

In diesem Abschnitt sollen nun die möglichen Lösungsansätze zur Verteilung der Nachrichten untersucht werden. Es haben sich für die Verteilung der JMS Topic die folgenden vier möglichen Lösungsansätze ergeben.

- Clients melden sich bei allen Servern an

  Bei diesem Ansatz melden sich alle Clients bei allen Server an und Buchen das entsprechende Topic.

- Zentraler Nachrichtenprovider

  Bei diesem Ansatz gibt es nur einen einzigen Server bei dem die Clients das Topic buchen müssen.

- Verteilung der Nachrichten an alle Server

  Bei diesem Ansatz müssen die Server allen anderen Servern die Nachrichten zukommen lassen, beziehungsweise müssen sie die Nachrichten aller anderen Server Buchen.

- HornetQ in Jboss Clusterconfiguration

  Bei diesem Ansatz wird der in Jboss integrierte HornetQ Server für ein Cluster Konfiguriert.

Diese vier Lösungsansätze sollen nun auf ihre Tauglichkeit im Bereich der Hochverfügbarkeit, Skalierbarkeit und Wartbarkeit hin untersucht werden.

5.2.1 Clients melden sich bei allen Servern an

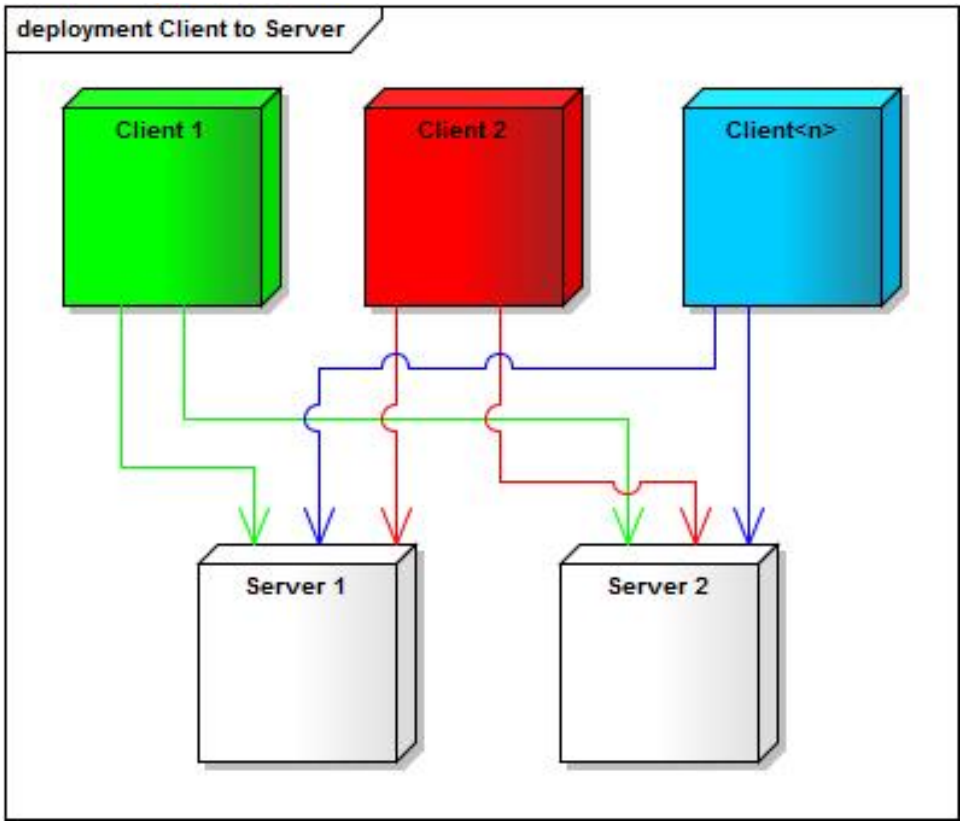

Abbildung 4: Clients melden sich bei allen Servern an

Wie man in Abbildung 4 erkennen kann Steigt der Verwaltungsaufwand dieses Verfahrens mit steigender Anzahl an Clients oder Server exponentiell an. Da jederzeit Clients und Server zum Netzwerk hinzugefügt und aus diesem entfernt werden dürfen, ergibt sich, das bei der Umsetzung dieser Lösung ein dynamischer Ansatz gewählt werden muss. Damit steigt der Aufwand, der zur Sicherstellung der Zuverlässigkeit des System betrieben muss, an. Weiterhin müsste man weitere Verfahren zur Überwachung verwenden, da jederzeit Sichergestellt werden muss das ein Client zu einem beliebigen Zeitpunkt mit allen Servern verbunden ist.

5.2.2 Zentraler Nachrichtenprovider

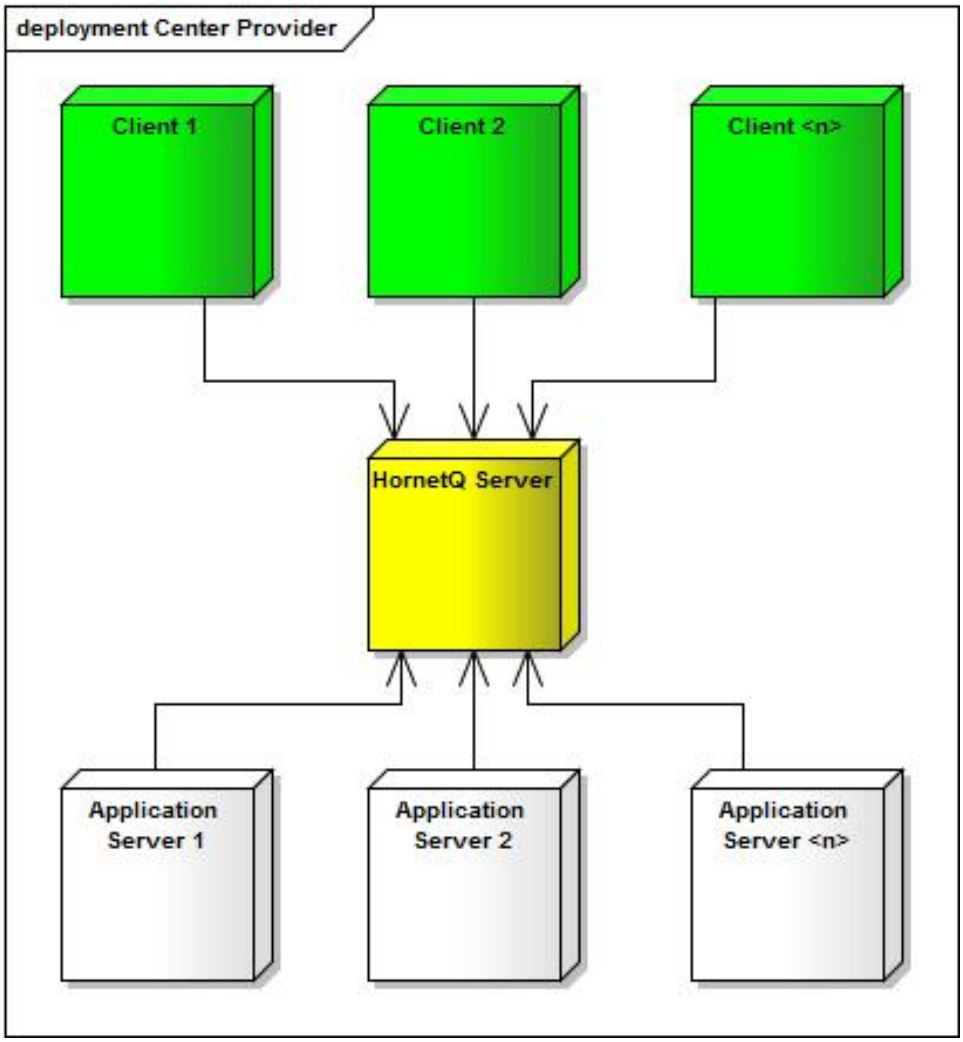

Abbildung 5: Zentrahler Nachrichtenprovider

Eine weitere recht naheliegende Lösung wäre ein zentraler Nachrichtenprovider, bei dem alle Server ihre erzeugten Nachrichten an diesen weitergeben. Die Clients müssen dann nur einen Server kennen bei dem sie die Nachrichten Buchen können.
Dies würde allerdings zu einem Single Point of Failure führen. Daher muss der Nachrichtenprovider durch geeignete Maßnahmen vor Ausfällen geschützt werden. Weiterhin besitzt dieser Lösungsansatz sehr schlechte Skalierungs-

eigenschaften. Da nun alle Clients mit nur einem Server verbunden sind ist die maximale Anzahl an Clients durch die Leistungsfähigkeit des Server begrenzt.

5.2.3 Verteilung der Nachrichten an alle Server

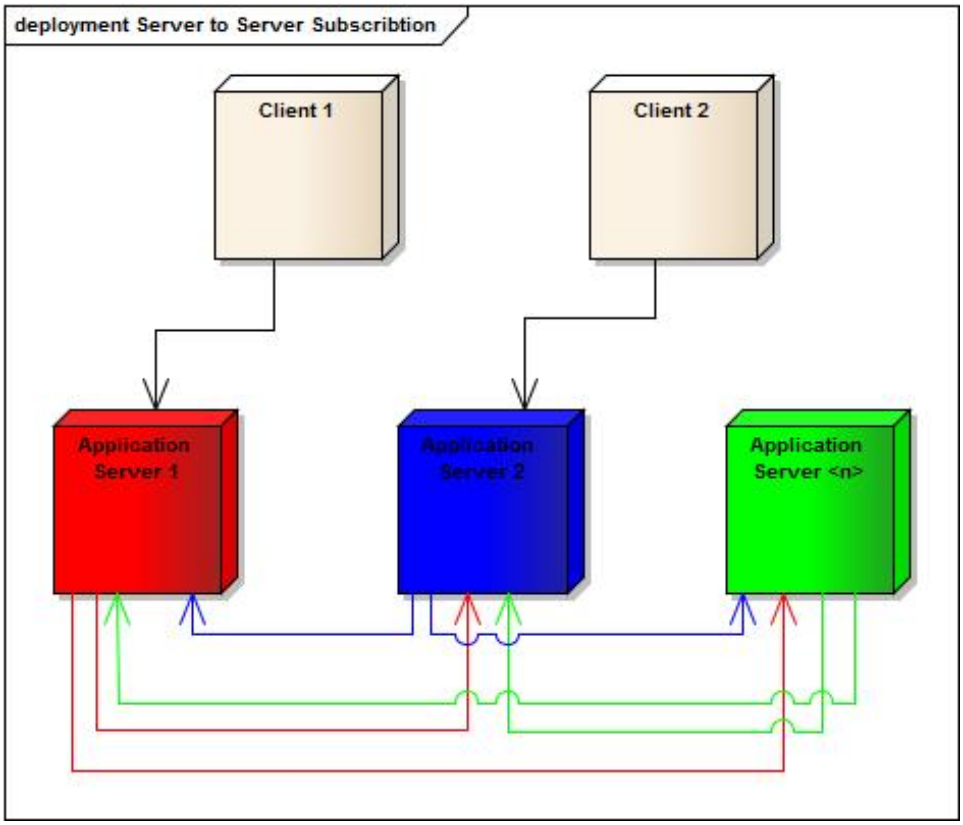

Abbildung 6: Verteilung der Nachrichten an alle Server

Bei diesem Verfahren werden die Nachrichten unter den Server verteilt, bevor sie ausgeliefert werden. Hierbei buchen die Server die Topics der anderen Server. Wodurch zusätzlicher Aufwand beim Filtern von Dubletten entsteht. Weiterhin könnte es hierbei nur unter Zuhilfenahme eines „Dirty Tricks“eine dynamische Lösung geben. Da das Buchen von Nachrichten im Jboss nur über Message Driven Beans möglich ist, diese verwenden allerdings Annotationen die zur Kompilierzeit vorliegen müssen. Daher müsste die Anwendung bei jeder Änderung der Serveranzahl neu ausgeliefert werden.

5.2.4 HornetQ in Jboss Clusterconfiguration

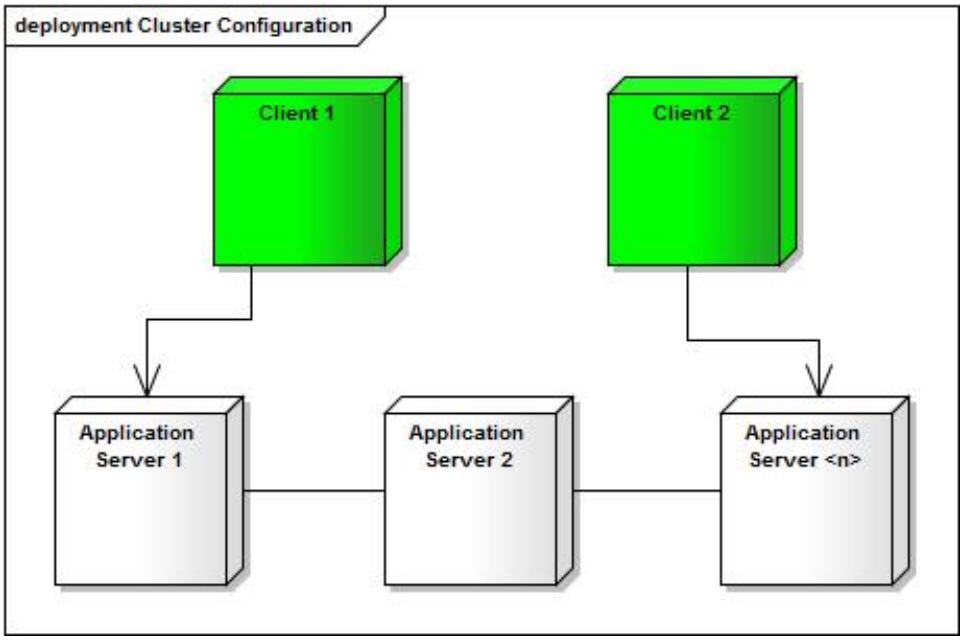

Abbildung 7: HornetQ in Jboss Clusterconfiguration

Für HornetQ Server wird in der HornetQ Dokumentation [1] eine Clustereinstellung beschrieben, wie sie in Abbildung 7 zu sehen ist. Dabei wird ausgenutzt das jeder Jboss Server auch als HornetQ Server verwendet werden kann. Diese Lösung wirft allerdings das Problem der Verteilung der Clients auf die Server auf. Dies lässt sich aber sehr einfach mittels Round-robin DNS[2] lösen. Da allerdings hier alle Nachrichten über alle Server verteilt werden, benötigt jeder Client nur die Verbindung zu einem Server. Es kann also jederzeit ein zusätzlicher Server zum Cluster hinzugefügt werden. Wird ein Server vom Cluster getrennt zu dem ein Client eine Verbindung hat so muss dieser sich an einem anderen Server anmelden und seine Daten aktualisieren. Da dieses Verfahren die Vorteile der anderen in sich vereint wurde es aus den gegeben ausgewählt.

[1] Referenz: [Hor15b]
[2] Round-robin Referenz: [rou15]

5.3 Apache Lucene

Als nächsten sollen die Lösungsansätze zur Verteilung der Indexfiles von Lucene betrachtet werden. Dazu soll zunächst Erwähnung finden, das es sich bei den Lucene Indexfiles nicht um Dateien im eigentlich Sinne handelt. Die Bezeichnung Indexfile ist also nur als Synonym für die eigentlichen Daten zu verstehen. Diese können im RAM vorgehalten oder in das Filesystem ausgelagert werden. Vom Prinzip her gibt es hier nur einen einzigen Lösungsansatz. Indem, bei einer Aktualisierung der Daten aus denen die Indexfiles erzeugt werden, diese als Veraltet markiert und nach einem festen Zeitintervall neu erzeugt werden. Für die Neuerzeugung der Indexfiles und deren Verteilung gibt es zwei, bei genauerer Betrachtung, drei mögliche Verfahren. Zwei dieser Verfahren sind sich sehr ähnlich, sie unterscheiden sich nur auf der Kommunikationsebene. Daher sollen sie zuerst behandelt werden. Bei diesen handelt es sich um ein Master/ Slave setting. Bei dem dritten Verfahren handelt es sich um verteilten Speicher.

5.3.1 Apache Lucene Master/ Salve

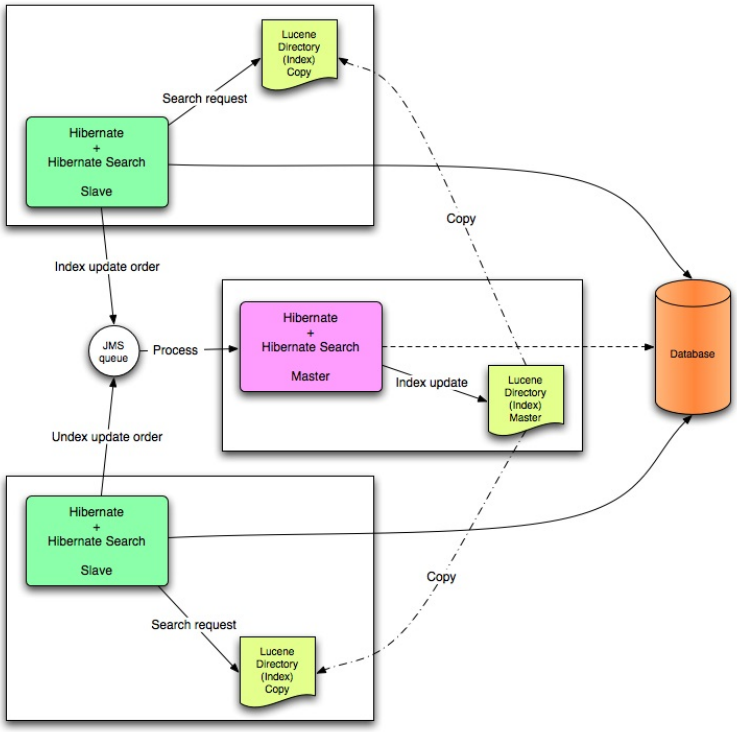

Abbildung 8: Master Slave für Apache Lucene [1]

Wie bereits erwähnt sind sich die beiden Verfahren sehr ähnlich, deswegen sollen sie hier gemeinsam vorgestellt werden. Hauptsächlich wird dabei zwischen Master und Slaves unterschieden. Der Master übernimmt dabei das ändern der Indexfiles und diese werden über eine Netzwerkressource allen Slaves zur Verfügung gestellt. Der grundsätzliche Unterschied dabei ist, wie bereits erwähnt, die Art der Übertragung der Meldung über Änderungen. Dazu gibt es einmal die Möglichkeit das die Slaves den Master mittels JMS Nachrichten über Änderungen Informieren oder, dass Master und Slave über Jgroups Kommunizieren. Da bei dieser Lösung die Kommunikation zwischen Master und Slave, nur in Richtung Master Implementiert wurde ergibt sich daraus das alle Slaves ihre Indexfiles nach einem festen Zeitintervall erneuern und nur der Master Änderungen vornimmt. Da der Zeitintervall vom Startzeitpunkt des Servers aus gerechnet wird, kann hierbei nicht garantiert

[1] [MSH15]

werden, das alle Änderungen auf allen Server gleichzeitig sichtbar sind.

5.3.2 Apache Lucene Replicated-Cache

Bei diesem Lösungsansatz wird eine weitere API verwendet der Infinispan Data-Grid. Dabei handelt es sich um einen Key-Value Store in dem die Indexfiles abgelegt werden. Dieser ist fester Bestandteil des Jboss EAP 6 und Kommuniziert über Jgroups. Diese Kommunikation findet allerdings auf einer anderen Ebene statt, dabei werden alle zum Cluster gehörenden Cache geändert. Was dazu führt das Änderungen die ein Server vornimmt sofort auf allen anderen Sichtbar sind. Da nur bei diesem Verfahren Garantiert werden kann, das alle Server zu gleichen Zeit die gleichen Suchergebnisse aufweisen, wurde dieses ausgewählt.

6 Implementierung und Tests

In diesem Abschnitt werden die Implementations- und Konfigurationsdetails zu den Einzelnen ausgewählten Lösungen besprochen. Den Anfang bildet dabei die HornetQ in Jboss Clusterconfiguration, deren Implementationsdetails, der Aufbau der Testumgebung und das Testprotokoll. Darauf folgend die Implementation und das Testszenario der Ausgewählten Lösungen für Lucene.

6.1 Konfiguration HornetQ

Dieses Verfahren wird durch die Jboss Konfiguration Realisiert. Wie es im akquinet Blog: "CLUSTERING OF THE MESSAGING SUBSYSTEM HORNETQ IN JBOSS AS7 AND EAP 6"[1] beschrieben wird. Die dort beschriebene Möglichkeit wird für unsere Zwecke angepasst, dazu wird das zu verwendende conficfile des Jboss Servers ergänzt. In diesem Fall handelt es sich um die standalone.xml. Dazu müssen folgende Änderungen eingetragen werden.

[1] Referenz: [HW15]

Alle Änderungen erfolgen im messaging Abschnitt der standalone.xml. Dieser Bereich sollte etwa wie folgt aussehen.

```
<subsystem xmlns="urn:jboss:domain:messaging:1.3">
 <hornetq-server>
  <persistence-enabled>true</persistence-enabled>
  <!--   some other code   -->
 </hornetq-server>
</subsystem>
```

In diesem Abschnitt werden dann die Cluster Einstellungen ergänzt.

```
<subsystem xmlns="urn:jboss:domain:messaging:1.3">
 <hornetq-server>
  <clustered>true</clustered>
  <persistence-enabled>true</persistence-enabled>
  <cluster-user>cluster-user</cluster-user>
  <cluster-password>secred-password</cluster-password>
  <!--   some other code   -->
 </hornetq-server>
</subsystem>
```

Der cluster-user und das cluster-password müssen dann für alle Jboss Server, die zum selben Cluster gehören, gleich sein. Außerdem muss clustered auf true gesetzt werden.
Danach wird noch eine broadcast-group und discovery-group hinzugefügt. Diese werden benötigt damit sich die verschiedenen Server finden. Auch hierbei müssen alle Server die zum Cluster gehören die selben Einstellungen besitzen. Mit einer Ausnahme, der local-bind-address dieser Eintrag bezeichnet die Lokale IP-Adresse des Server an die er gebunden ist. Dazu wird folgender Code ergänzt.

```
<subsystem xmlns="urn:jboss:domain:messaging:1.3">
 <hornetq-server>
  <!--   some other code   -->

  <broadcast-groups>
   <broadcast-group name="bg-group1">
    <local-bind-address>192.168.133.128</local-bind-
       address>
    <local-bind-port>5432</local-bind-port>
    <group-address>231.7.7.7</group-address>
```

```
    <group-port>9876</group-port>
    <connector-ref>
     netty
    </connector-ref>
   </broadcast-group>
  </broadcast-groups>

  <discovery-groups>
   <discovery-group name="dg-group1">
    <socket-binding>messaging-group</socket-binding>
    <refresh-timeout>10000</refresh-timeout>
   </discovery-group>
  </discovery-groups>

  <!--   some other code  -->
 </hornetq-server>
</subsystem>
```

Danach folgen die Einstellungen für die Cluster Verbindung, durch diese wird bekannt gegeben, zu welchem Cluster der Server gehört.

```
<cluster-connections>
  <cluster-connection name="my-cluster">
    <address>jms</address>
    <connector-ref>netty</connector-ref>
    <reconnect-attempts>2</reconnect-attempts>
    <forward-when-no-consumers>
      true
    </forward-when-no-consumers>
    <discovery-group-ref discovery-group-name="dg-
        group1"/>
  </cluster-connection>
</cluster-connections>
```

Danach muss gegebenenfalls noch ein Eintrag in der Routing Tabelle des Betriebssystems ergänzt werden. Für Linux benutzen wir dafür:

```
route add -net 231.7.7.7 netmask 255.255.255.255 dev
   eth3
```

6.2 HornetQ in Jboss Clusterconfiguration, Tests

Dieser Abschnitt widmet sich den Testszenarien. Mithilfe dieser soll sichergestellt werden, dass sich das System wie erwartet verhält.

6.2.1 Testszenario 1: Verteilung der Nachrichten

Bei diesem Testszenario soll überprüft werden ob Nachrichten, die an einem Server geschickt werden, auf alle anderen verteilt werden.

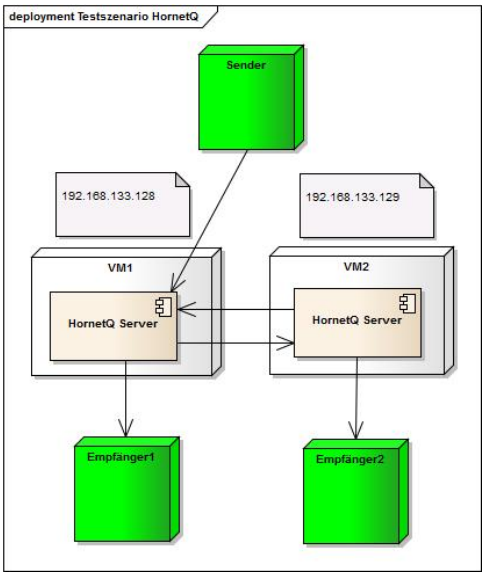

Abbildung 9: Testaufbau HornetQ 1

Testaufbau: Zwei identische Virtuelle Maschinen mit der selben Jboss Konfiguration. Zur Durchführung des Tests wird ein Sender[1] und zwei identische Empfänger[2] genutzt.

Testablauf: Die Virtuellen Maschinen werden gestartet. Danach werden die Jboss-Server hochgefahren. Danach werden die Empfänger gestartet. Wobei sich Empfänger1 mit VM1 verbindet und Empfänger2 mit VM2. Darauf werden die zwei Testfälle gestartet.

- Testfall 1: Sender sendet Nachricht an VM1.

[1] siehe Anhang
[2] siehe Anhang

Erwartetes Ergebnis: Empfänger1 und Empfänger2 erhalten die Nachricht.

- Testfall 2: Sender sendet Nachricht an VM2.

  Erwartetes Ergebnis: Empfänger1 und Empfänger2 erhalten die Nachricht.

Testprotokoll 1:
Sender Protokoll:

```
D:\>java -jar Sender.jar 192.168.133.128
Mai 22, 2015 1:34:12 PM org.xnio.Xnio <clinit>
INFO: XNIO Version 3.0.7.GA-redhat-1
Mai 22, 2015 1:34:12 PM org.xnio.nio.NioXnio <clinit>
INFO: XNIO NIO Implementation Version 3.0.7.GA-redhat
   -1
Mai 22, 2015 1:34:12 PM org.jboss.remoting3.
   EndpointImpl
    <clinit>
INFO: JBoss Remoting version 3.2.16.GA-redhat-1
```

Empfänger1 Protokoll:

```
D:\>java -jar Empfaenger.jar 192.168.133.128
Mai 22, 2015 1:33:28 PM org.xnio.Xnio <clinit>
INFO: XNIO Version 3.0.7.GA-redhat-1
Mai 22, 2015 1:33:28 PM org.xnio.nio.NioXnio <clinit>
INFO: XNIO NIO Implementation Version 3.0.7.GA-redhat
   -1
Mai 22, 2015 1:33:28 PM org.jboss.remoting3.
   EndpointImpl
    <clinit>
INFO: JBoss Remoting version 3.2.16.GA-redhat-1
testnachricht 2
```

Empfänger2 Protokoll:

```
D:\>java -jar Empfaenger.jar 192.168.133.128
Mai 22, 2015 1:33:52 PM org.xnio.Xnio <clinit>
INFO: XNIO Version 3.0.7.GA-redhat-1
Mai 22, 2015 1:33:28 PM org.xnio.nio.NioXnio <clinit>
INFO: XNIO NIO Implementation Version 3.0.7.GA-redhat
   -1
```

```
Mai 22, 2015 1:33:28 PM org.jboss.remoting3.
   EndpointImpl
    <clinit>
INFO: JBoss Remoting version 3.2.16.GA-redhat-1
testnachricht 2
```

Testprotokoll 2:

Sender Protokoll:

```
D:\>java -jar Sender.jar 192.168.133.128
Mai 22, 2015 1:34:12 PM org.xnio.Xnio <clinit>
INFO: XNIO Version 3.0.7.GA-redhat-1
Mai 22, 2015 1:34:12 PM org.xnio.nio.NioXnio <clinit>
INFO: XNIO NIO Implementation Version 3.0.7.GA-redhat
   -1
Mai 22, 2015 1:34:12 PM org.jboss.remoting3.
   EndpointImpl
    <clinit>
INFO: JBoss Remoting version 3.2.16.GA-redhat-1
```

Empfänger1 Protokoll:

```
D:\>java -jar Empfaenger.jar 192.168.133.128
Mai 22, 2015 1:45:43 PM org.xnio.Xnio <clinit>
INFO: XNIO Version 3.0.7.GA-redhat-1
Mai 22, 2015 1:45:43 PM org.xnio.nio.NioXnio <clinit>
INFO: XNIO NIO Implementation Version 3.0.7.GA-redhat
   -1
Mai 22, 2015 1:45:43 PM org.jboss.remoting3.
   EndpointImpl
    <clinit>
INFO: JBoss Remoting version 3.2.16.GA-redhat-1
testnachricht 2
Mai 22, 2015 1:46:33 PM
org.jboss.naming.remote.protocol.v1.
   RemoteNamingStoreV1$
MessageReceiver handleEnd
ERROR: Channel end notification received,
closing channel Channel ID a156fb8c (outbound)
of Remoting connection 31690c95 to
/192.168.133.128:4447
```

Empfänger2 Protokoll:

```
D:\>java -jar Empfaenger.jar 192.168.133.129
Mai 22, 2015 1:46:24 PM org.xnio.Xnio <clinit>
INFO: XNIO Version 3.0.7.GA-redhat-1
Mai 22, 2015 1:46:24 PM org.xnio.nio.NioXnio <clinit>
INFO: XNIO NIO Implementation Version 3.0.7.GA-redhat
   -1
Mai 22, 2015 1:46:24 PM org.jboss.remoting3.
   EndpointImpl <clinit>
INFO: JBoss Remoting version 3.2.16.GA-redhat-1
testnachricht 2
```

Auswertung: Wie das Testszenario gezeigt hat ist die Verteilung der Topics innerhalb eines Cluster erfolgreich. Als nächstes muss daher das Verhalten im Fehlerfall überprüft werden.

6.2.2 Testszenario 2: Verteilung der Nachrichten, bei Ausfall eines Servers

Bei diesem Testszenario soll überprüft werden ob eine Nachricht von allen zum Cluster gehörenden Servern noch ausgeliefert wird. Nachdem der Server, auf dem sie erstellt wurde, ausgefallen ist. Es soll überprüft werden, ob auf allen Servern Kopien der Nachrichten vorliegen.

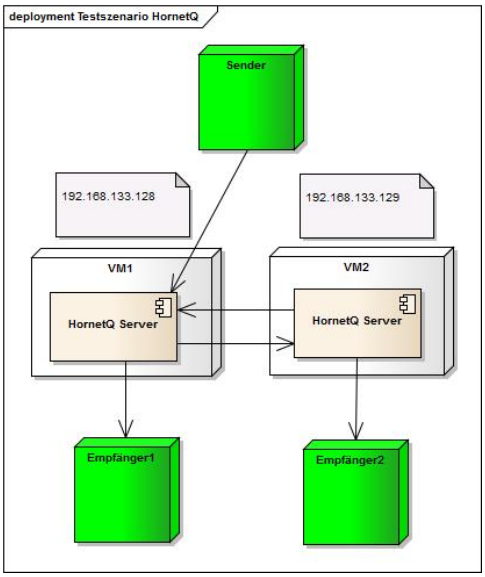

Abbildung 10: Testaufbau HornetQ 2

Testaufbau: Zwei identische Virtuelle Maschinen mit der selben Jboss Konfiguration. Zur Durchführung des Tests wird ein Sender[1] und zwei identische EmpfängerT2[2] genutzt.

Testablauf: Die Virtuellen Maschinen werden gestartet und die Jboss-Server hochgefahren. Danach meldet sich je ein EmpfängerT2 an jedem Server an. Wobei der EmpfängerT2 hier das Topic bucht, aber noch nicht in den Empfangsmodus schaltet. Danach wird Mithilfe des Senders an Server2 (192.168.133.129) eine Nachricht übermittelt. Dieser wird dann Hard beendet (durch kill -9 PID). Darauf werden die Empfänger in den Empfangsmodus geschaltet.

[1] siehe Anhang
[2] siehe Anhang

Erwartetes Ergebnis: Der Empfänger an Server1 erhält die Nachricht, der Empfänger an Server2 erzeugt eine Fehlermeldung und beendet sich.

Testprotokoll 2:

Sender Protokoll:

```
D:\>java -jar Sender.jar 192.168.133.129
Mai 23, 2015 12:21:14 PM org.xnio.Xnio <clinit>
INFO: XNIO Version 3.0.7.GA-redhat-1
Mai 23, 2015 12:21:14 PM org.xnio.nio.NioXnio <clinit>
INFO: XNIO NIO Implementation Version 3.0.7.GA-redhat
   -1
Mai 23, 2015 12:21:14 PM org.jboss.remoting3.
EndpointImpl <clinit>
INFO: JBoss Remoting version 3.2.16.GA-redhat-1
```

Empfänger1 Protokoll:

```
D:\>java -jar EmpfaengerT2.jar 192.168.133.128
Mai 23, 2015 12:20:23 PM org.xnio.Xnio <clinit>
INFO: XNIO Version 3.0.7.GA-redhat-1
Mai 23, 2015 12:20:23 PM org.xnio.nio.NioXnio <clinit>
INFO: XNIO NIO Implementation Version 3.0.7.GA-redhat
   -1
Mai 23, 2015 12:20:23 PM org.jboss.remoting3.
EndpointImpl <clinit>
INFO: JBoss Remoting version 3.2.16.GA-redhat-1
Ferbindung Erfolgreich, beliebige Taste zum vortfahren
   .

testnachricht 2
Mai 23, 2015 12:22:02 PM org.jboss.naming.remote
.protocol.v1.RemoteNamingStoreV1
$MessageReceiver handleEnd
ERROR: Channel end notification received,
closing channel Channel ID 9b4f6603 (o
utbound) of Remoting connection 47ffa2ec to null
```

Empfänger2 Protokoll:

```
D:\>java -jar EmpfaengerT2.jar 192.168.133.129
Mai 23, 2015 12:20:21 PM org.xnio.Xnio <clinit>
INFO: XNIO Version 3.0.7.GA-redhat-1
Mai 23, 2015 12:20:21 PM org.xnio.nio.NioXnio <clinit>
```

```
INFO: XNIO NIO Implementation Version 3.0.7.GA-redhat
   -1
Mai 23, 2015 12:20:21 PM org.jboss.remoting3.
EndpointImpl <clinit>
INFO: JBoss Remoting version 3.2.16.GA-redhat-1
Ferbindung Erfolgreich, beliebige Taste zum vortfahren
   .
Mai 23, 2015 12:21:47 PM org.jboss.naming.remote.
protocol.v1.RemoteNamingStoreV1
$MessageReceiver handleEnd
ERROR: Channel end notification received,
closing channel Channel ID b417afbe (outbound)
of Remoting connection 245215e7 to /192.168.133.129
   :4447

javax.jms.IllegalStateException: HQ119017:
Consumer is closed
        at org.hornetq.core.client.impl.
           ClientConsumerImpl.checkClosed
           (ClientConsumerImpl.java:1013)
        at org.hornetq.core.client.impl.
          ClientConsumerImpl.receive
          (ClientConsumerImpl.java:189)
        at org.hornetq.core.client.impl.
          ClientConsumerImpl.receive
          (ClientConsumerImpl.java:394)
        at org.hornetq.jms.client.
           HornetQMessageConsumer.
          getMessage
          (HornetQMessageConsumer.java:220)
        at org.hornetq.jms.client.
           HornetQMessageConsumer.
          receive(HornetQMessageConsumer.java:129)
        at jms.test.EmpfaengerT2.
          main(EmpfaengerT2.java:52)
Caused by: HornetQException[errorType=OBJECT_CLOSED
  message=HQ119017: Consumer is closed]
        ... 6 more
```

Auswertung: Durch dieses Testszenario wurde Nachgewiesen, das die Topics über alle Server verteilt werden. Daher ist, auch im Falle des Verlustes eines der im Cluster befindlichen Server, die Verteilung der Topics innerhalb des Clusters gewährleistet.

6.3 Apache Lucene Infinispan, Implementation

Dieses Verfahren wird durch die Erweiterung der Jboss Konfiguration Realisiert. Dazu wird folgender Code innerhalb der Standalone.xml ergänzt. Unter dem Abschnitt extensions:

```
<extensions>
  <extension module="org.jboss.as.clustering.
     infinispan"/>
  <extension module="org.jboss.as.clustering.jgroups"/
     >
</extensions>
```

Was die entsprechende Module innerhalb des Jboss aktiviert.
Danach erweitern wir die Subsystemeinstellungen von Infinispan beziehungsweise ergänzen diese, falls sie nicht vorhanden sind.

```
<subsystem xmlns="urn:jboss:domain:infinispan:1.4">
  <cache-container name="LuceneLokalitaeten" aliases="
     Lokalitaeten"
      default-cache="default"
      jndi-name="java:jboss/infinispan/Lokalitaeten"
         start="EAGER">
    <transport lock-timeout="60000"/>
    <replicated-cache name="LuceneIndexesMetadata"
       mode="SYNC"
      start="LAZY" batching="true">
    <locking isolation="REPEATABLE_READ"/>
    </replicated-cache>
    <replicated-cache name="LuceneIndexesData" mode="
       SYNC"
      start="LAZY" batching="true">
    <locking isolation="REPEATABLE_READ"/>
    </replicated-cache>
    <replicated-cache name="LuceneIndexesLocking" mode
       ="SYNC"
      start="LAZY" batching="true">
    <locking isolation="REPEATABLE_READ"/>
    </replicated-cache>
  </cache-container>
</subsystem>
```

Um die Verteilung des Lucene Caches zu Gewährleisten benötigen wir einen Cache-Container, auf den Mithilfe von jndi zugegriffen werden kann. Außerdem benötigen wir drei Replicated-Caches in denen die Indexfiles die Lucene erzeugt abgelegt werden. Die Namen dieser sind dabei durch Lucene vorgegeben.
Auch wird noch das Jgroup-Subsystem ergänzt.

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.1"
  default-stack="udp">
  <stack name="udp">
    <transport type="UDP" socket-binding="jgroups-udp"
       />
    <protocol type="PING"/>
    <protocol type="MERGE3"/>
    <protocol type="FD_SOCK" socket-binding="jgroups-
       udp-fd"/>
    <protocol type="FD"/>
    <protocol type="VERIFY_SUSPECT"/>
    <protocol type="pbcast.NAKACK"/>
    <protocol type="UNICAST2"/>
    <protocol type="pbcast.STABLE"/>
    <protocol type="pbcast.GMS"/>
    <protocol type="UFC"/>
    <protocol type="MFC"/>
    <protocol type="FRAG2"/>
    <protocol type="RSVP"/>
  </stack>
</subsystem>
```

Weiterhin müssen noch die Socket-binding Einträge für Jgroups ergänzt werden.

```
<socket-binding-group
  name="standard-sockets" default-interface="public"
  port-offset="${jboss.socket.binding.port-offset:0}">
    <!-- Some other code -->
    <socket-binding name="jgroups-udp"
      port="55200" multicast-address=
      "${jboss.default.multicast.address:230.0.0.4}"
      multicast-port="45688"/>
    <socket-binding name="jgroups-udp-fd" port="54200"
       />
    <!-- Some other code -->
    </socket-binding-group>
```

Als nächstes müssen noch die Einstellungen für Lucene angepasst werden. Dies kann entweder Programmmatisch erfolgen oder durch anpassen der persistens.xml im Projekt. In diesem Fall wird die Anpassung via persistens.xml vorgenommen. Dazu ist folgender Code zu ergänzen.

```
<properties>
  <!-- Some other code -->
  <property
    name="hibernate.search.default.directory_provider"
    value="infinispan" />
  <property
    name="hibernate.search.infinispan.
       cachemanager_jndiname"
    value="java:jboss/infinispan/Lokalitaeten" />
  <property
    name="hibernate.search.infinispan.data_cachename"
    value="LuceneIndexesData" />
  <property
    name="hibernate.search.infinispan.
       locking_cachename"
    value="LuceneIndexesLocking" />
  <property
    name="hibernate.search.infinispan.
       metadata_cachename"
    value="LuceneIndexesMetadata" />
  <property
    name="hibernate.search.default.indexwriter.
       ram_buffer_size"
    value="128" />
  <property
    name="hibernate.connection.release_mode"
    value="after_statement" />
  <property
    name="hibernate.search.lucene_version"
    value="LUCENE_CURRENT" />
    <!-- Some other code -->
</properties>
```

Danach muss gegeben falls noch ein Eintrag in der Routing Tabelle des Betriebssystems ergänzt werden. Für Linux benutzen wir dafür:

```
route add -net 230.0.0.4 netmask 255.255.255.255 dev
   eth3
```

6.4 Apache Lucene, Infinispan Testszenarien

6.4.1 Testszenario 1: Clusterweiter zugriff auf Index-Files

Bei diesem Testszenario soll überprüft werden ob alle Knoten des Clusters zugriff auf die Indexfiles haben. Hier werden die zwei ausgewählten Lösungen gegenüber gestellt.

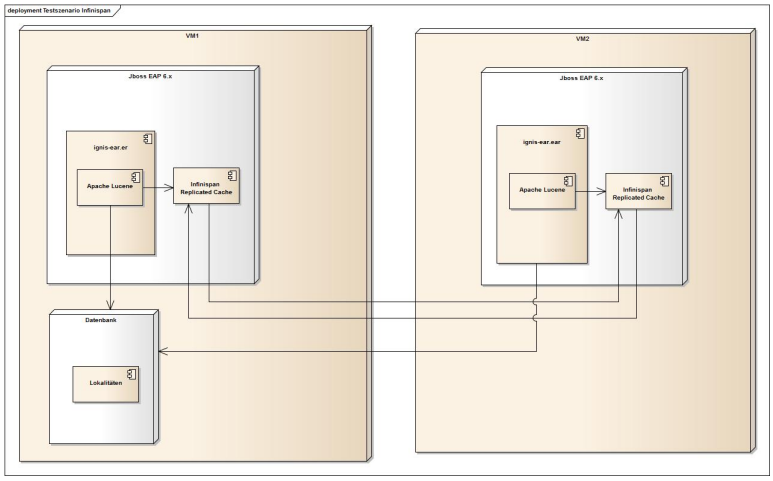

Abbildung 11: Testaufbau Infinispan

Testaufbau: Zwei Virtuelle Maschinen, auf beiden befindet sich je ein Jboss der entsprechend auf den Infinispan Replicated Cache konfiguriert wurde. Auf einer der beiden VMs ist zusätzlich eine Oracle Datenbank mit einem Testdatenbestand installiert.

Testablauf: Die Jboss Server werden gestartet, danach wird auf beiden die Anwendung IGNIS Plus Deployed. Nachdem Lucene die Indexfiles für die Lokalitäten in den Cache geladen hat wird auf einem der beiden Jboss Server eine Änderung vorgenommen und Überprüft ob diese auf dem anderen sichtbar ist.

Erwartetes Ergebiss: Nach einer Zeit t sind alle Änderungen auf beiden Jboss Servern sichtbar.

Auswertung: Der Jboss Server auf VM1 erzeugt Timeoutexceptions. Deswegen wurde die Timeoutzeit von einer Minute auf eine Stunde zu Test-

zwecken erhöht. Danach werden die Indexfiles vollständig erzeugt allerdings führt das zuschalten des Jboss Servers auf VM2 zu einem vollständigen System freeze von VM1.

7 Fazit

Ziel der Arbeit war es, je ein optimales Verfahren für die Verteilung der JMS Topics und ein optimales Verfahren für die Synchronisierung der Änderungen an den Lucene Indexfiles zu finden. Dazu wurden in Kapitel 5 Auswahl der Lösungsverfahren vier mögliche Verfahren für die JMS Topics und zwei mögliche Verfahren für Lucene vorgestellt.
Für die Synchronisierung der JMS Topics wurde das Verfahren zum Clustering der in Jboss integrierten HornetQ Server ausgewählt. Danach wurde dieses Verfahren in einer der Minimalkonfiguration entsprechenden Umgebung Implementiert. Für die Testszenarien wurden zwei verschiedene Programme angefertigt, ein Sender und ein Empfänger.
Für die Verteilung der Lucene Indexfiles wurden die Replicated Caches des in Jboss Integrierten Infinispan Data Grids ausgewählt. Danach wurde dieses Verfahren Implementiert und einem Minimaltest unterzogen. Durch diesen hat sich gezeigt, dass das ausgewählte Verfahren in seinem jetztigen Status nicht Funktionsfähig ist. Leider konnte bis zum Abgabetermin der Arbeit nicht mehr geklärt werden, wodurch das Fehlverhalten verursacht wird.
Unter Berücksichtigung der Ergebnisse dieser Arbeit steht der Nutzung des für die Verteilung der JMS Topics ausgewählten Verfahrens nichts im Wege.

Literatur

[Apa15] *Apache Lucene Core*. https://lucene.apache.org/core/. Version: 05 2015

[AVI15] ; ISO (Veranst.): *ISO/IEC 25010:2011(en)*. https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en. Version: 05 2015

[Con15] Consulting, Sopra S.: *Firmen internes Wiki*. 05 2015

[Fri15] Friedrichsen, Uwe: Resilience. In: *JAVA MAGAZIN 5.15* (2015)

[Hoc15] *High availability*. http://en.wikipedia.org/wiki/High_availability. Version: 05 2015

[Hor15a] *HornetQ*. http://hornetq.jboss.org. Version: 05 2015

[Hor15b] *HornetQ: Chapter 38. Clusters*. http://docs.jboss.org/hornetq/2.2.5.Final/user-manual/en/html/clusters.html. Version: 05 2015

[HW15] Heinz Wilming, Immanuel S.: *CLUSTERING OF THE MESSAGING SUBSYSTEM HORNETQ IN JBOSS AS7 AND EAP 6*. Akqinet. http://blog.akquinet.de/2012/11/24/clustering-of-the-messaging-subsystem-hornetq-in-jboss-as7-and-eap-6/. Version: 05 2015

[jax15] *JAX-WS*. https://jax-ws.java.net. Version: 05 2015

[jbo15a] *Jboss Dokumentation*. http://jbossas.jboss.org/docs/. Version: 05 2015. – 6.x Dokumentation did not exist anymore

[Jbo15b] *JbossDeveloper*. http://www.jboss.org. Version: 05 2015

[JSR15] ; java Community Process (Veranst.): *JSR 914: JavaTM Message Service (JMS) API*. https://www.jcp.org/en/jsr/detail?id=914. Version: 05 2015

[MSH15] ; jboss.org (Veranst.): *Hibernate: Chapter 3. Configuration*. https://docs.jboss.org/hibernate/search/4.

4/reference/en-US/html/search-configuration.html#search-configuration-directory. Version: 05 2015

[rou15] ; Wikipedia (Veranst.): *Round-robin DNS*. http://en.wikipedia.org/wiki/Round-robin_DNS. Version: 05 2015

[Ska15] *Skalierbarkeit*. de.wikipedia.org/wiki/Skalierbarkeit. Version: 05 2015

[TS13] Tino Stelter, Thomas K.: Gesamtarchitekturdokument fuer Ignis Plus. (2013)

[W3C15] *W3C Soap 1.2*. http://www.w3.org/TR/soap12/. Version: 05 2015

[Wik15] *SOAP Wikipedia*. http://de.wikipedia.org/wiki/SOAP. Version: 05 2015

8 Anhang

8.1 Sender.java

```
package jms.test;

import java.util.Hashtable;

import javax.jms.MapMessage;
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueReceiver;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.jms.Topic;
import javax.jms.TopicConnection;
import javax.jms.TopicConnectionFactory;
import javax.jms.TopicPublisher;
import javax.jms.TopicSession;
import javax.jms.TopicSubscriber;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NameNotFoundException;

public class Sender {

  public static void main(String[] args) {
    Topic queue = null;
    TopicPublisher receiver = null;
    try {
      Hashtable<String, String> env = new Hashtable();
      env.put(Context.INITIAL_CONTEXT_FACTORY,
          "org.jboss.naming.remote.client.
             InitialContextFactory");
      env.put(Context.PROVIDER_URL, "remote://" + args
         [0] + ":4447");
      env.put(Context.SECURITY_PRINCIPAL, "user");
```

```
        env.put(Context.SECURITY_CREDENTIALS, "password"
            );
        env.put(Context.URL_PKG_PREFIXES, "org.jboss.
            naming.remote.client");
        Context context = new InitialContext(env);
        TopicConnectionFactory fac = (
            TopicConnectionFactory) context
             .lookup("jms/RemoteConnectionFactory");
        TopicConnection connect = fac.
            createTopicConnection("user",
             "password");
        TopicSession session = connect.
            createTopicSession(false,
             Session.AUTO_ACKNOWLEDGE);
        try {
          queue = (Topic) context.lookup("ignis/jms/
              EventTopic");
        } catch (NameNotFoundException e) {
          e.printStackTrace();
        }
        receiver = session.createPublisher(queue);
        connect.start();
        TextMessage msg = session.createTextMessage();
        msg.setText("testnachricht 2");
        receiver.publish(msg);
        connect.close();
      } catch (Exception e) {
        e.printStackTrace();
      }
    }

}
```

8.2 Empfaenger.java

```
package jms.test;
import java.util.Hashtable;

import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueReceiver;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.jms.Topic;
import javax.jms.TopicConnection;
import javax.jms.TopicConnectionFactory;
import javax.jms.TopicSession;
import javax.jms.TopicSubscriber;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NameNotFoundException;

public class Empfaenger {

  public static void main(String[] args) {
    Topic queue = null;
    TopicSubscriber receiver = null;
    try {
      Hashtable<String, String> env = new Hashtable();
      env.put(Context.INITIAL_CONTEXT_FACTORY, "org.
          jboss.naming.remote.client.
          InitialContextFactory");
      env.put(Context.PROVIDER_URL, "remote://" + args
          [0] + ":4447");
      env.put(Context.SECURITY_PRINCIPAL, "user");
      env.put(Context.SECURITY_CREDENTIALS, "password"
          );
      env.put(Context.URL_PKG_PREFIXES, "org.jboss.
          naming.remote.client");
      Context context = new InitialContext(env);
```

```
      TopicConnectionFactory fac = (
         TopicConnectionFactory) context
          .lookup("jms/RemoteConnectionFactory");
      TopicConnection connect = fac.
         createTopicConnection("user", "password");
      TopicSession session = connect.
         createTopicSession(false,
          Session.AUTO_ACKNOWLEDGE);
      try {
        queue = (Topic) context
            .lookup("ignis/jms/EventTopic");
      } catch (NameNotFoundException e) {
        e.printStackTrace();
      }
      receiver = session.createSubscriber(queue);
      connect.start();
      TextMessage msg = (TextMessage) receiver.receive
         (10000000);
      System.out.println(msg.getText());
      connect.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

}
```

8.3 EmpfaengerT2.java

```
package jms.test;
import java.util.Hashtable;

import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueReceiver;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.jms.Topic;
import javax.jms.TopicConnection;
import javax.jms.TopicConnectionFactory;
import javax.jms.TopicSession;
import javax.jms.TopicSubscriber;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NameNotFoundException;

public class EmpfaengerT2 {

  public static void main(String[] args) {
    Topic queue = null;
    TopicSubscriber receiver = null;
    try {
      Hashtable<String, String> env = new Hashtable();
      env.put(Context.INITIAL_CONTEXT_FACTORY, "org.
          jboss.naming.remote.client.
          InitialContextFactory");
      env.put(Context.PROVIDER_URL, "remote://" + args
          [0] + ":4447");
      env.put(Context.SECURITY_PRINCIPAL, "user");
      env.put(Context.SECURITY_CREDENTIALS, "password"
          );
      env.put(Context.URL_PKG_PREFIXES, "org.jboss.
          naming.remote.client");
      Context context = new InitialContext(env);
```

```
      TopicConnectionFactory fac = (
         TopicConnectionFactory) context
          .lookup("jms/RemoteConnectionFactory");
      TopicConnection connect = fac.
         createTopicConnection("user", "password");
      TopicSession session = connect.
         createTopicSession(false,
          Session.AUTO_ACKNOWLEDGE);
      try {
        queue = (Topic) context
            .lookup("ignis/jms/EventTopic");
      } catch (NameNotFoundException e) {
        e.printStackTrace();
      }
      receiver = session.createSubscriber(queue);
      connect.start();
      try {
        System.out.println("Ferbindung Erfolgreich,
           beliebige Taste zum vortfahren.");
        System.in.read();
      } catch (Exception e) {
        // TODO: handle exception
      }
      TextMessage msg = (TextMessage) receiver.receive
         (10000000);
      System.out.println(msg.getText());
      connect.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

}
```