Integration von Anomalienerkennung in einem Kontrollzentrum für Softwarelandschaften

Bachelorarbeit

Kim Christian Mannstedt

29. März 2015

Christian-Albrechts-Universität zu Kiel
Institut für Informatik
Arbeitsgruppe Software Engineering

Betreut durch: Prof. Dr. Wilhelm Hasselbring
M. Sc. Florian Fittkau

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel,

Zusammenfassung

Diese Bachelorarbeit wird im Rahmen des Moduls „Abschlussprojekt - Software Engineering für parallele und verteilte Systeme“mit dem Thema „Kontrollzentrumintegration“geschrieben. Das Kontrollzentrum besteht dabei aus vier Ansichten: Die erste Ansicht ist die *Symptons View*, in der Anomalien erkannt werden, gefolgt von der *Diagnosis View*, welche sich mit der Ursachenerkennung beschäftigt. Anschließend folgt die *Planning View*, in der ein Plan entworfen wird, um die Anomalie zu beheben. Abschließend folgt die *Execution View*, in welcher der ausgearbeitete Plan umgesetzt wird.
Das Ziel der Bachelorarbeit besteht darin, eine Erkennung von Anomalien zu entwickeln. Es wird also die Datengrundlage für die *Symptons View* des Kontrollzentrums entwickelt. Unter einer Anomalie versteht man dabei das Fehlverhalten eines Programms oder einer Datenbank, was unter Umständen zu weiteren Fehlern, zum Beispiel in anderen Programmen, führt. Dabei unterscheidet man zwischen Punkt-, Kontext- und Kollektivanomalien.

Im weiteren Verlauf wird ein alternativer Vorhersagealgorithmus entwickelt, welcher durch eine von drei Gewichtungen die nächste Antwortzeit vorausberechnet. Dieser Algorithmus wird von der Anomalienerkennung genutzt, um eine auftretene Anomalie zu erkennen. Dieser Vorhersagealgorithmus wird evaluiert und mit den anderen Vorhersagealgorithmen der Anomalienerkennung hinsichtlich der Berechnung von Anomaliewerten verglichen. Die Ergebnisse dieser Evaluation sind, dass der alternative Algorithmus, wie die anderen Algorithmen der Anomalienerkennung, sowohl Punkt- als auch Kontextanomalien erkennt. Bei Kollektivanomalien kommt es auf die Dauer sowie der gewählten Gewichtung des alternativen Vorhersagealgorithmus an.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Diese Bachelorarbeit wird im Rahmen des Moduls „Abschlussprojekt - Software Engineering für parallele und verteilte Systeme“mit dem Thema „Kontrollzentrumintegration“geschrieben.

1.1. Motivation

Bei großen Softwarelandschaften ist das Auftreten von Anomalien keine Seltenheit. Damit der Benutzer die Gründe für Anomalien nicht selbst lokalisieren und anschließend lösen muss, wird im Rahmen des Abschlussprojektes ein semi-automatischen Kontrollzentrum entwickelt. Dieses Kontrollzentrum soll die Lokalisierung einer auftretenden Anomalie automatisch vornehmen. Anschließend soll das Kontrollzentrum dem Benutzer Vorschläge zur Bewältigung der Anomalien und eventuell auftretenden Kosten mittels eines Plans anzeigen. Der Benutzer kann dann diesen Plan vom Kontrollzentrum wie geplant umsetzen lassen, den Plan ändern oder diesen ablehnen.

1.2. Ziele

Im Folgenden wird auf die Ziele der Bachelorarbeit eingegangen. Dabei ist das Hauptziel die Entwicklung einer Erkennung von Anomalien für das Kontrollzentrum.

Z1: Anpassung von ΘPADx

ΘPADx [Frotscher 2013] wurde als Plug-In für das Monitoring Framework Kieker [*Kieker Website*; van Hoorn u. a. 2012] entwickelt. Daher muss ΘPADx so angepasst werden, dass es für die Verwendung als Plug-In für ExplorViz genutzt werden kann.

Z2: Implementierung und Evaluation eines alternativen Algorithmus

Im Rahmen dieser Bachelorarbeit soll ein alternativer Algorithmus für die Vorhersage des nächsten Messwertes bei der Antwortzeit, implementiert werden. Anschließend wird dieser Algorithmus mit den bestehenden Algorithmen verglichen und in speziellen Szenarien evaluiert.

1.3. Aufbau

In Kapitel 2 werden die benötigten Grundlagen und genutzten Technologien, die in dieser Bachelorarbeit zur Umsetzung der Ziele genutzt werden, kurz erläutert. Anschließend wird in Kapitel 3 ein Überblick über das Abschlussprojekt gegeben. In Kapitel 4 wird die Idee des alternativen Vorhersagealgorithmus erläutert und anschließend in Kapitel 5 die Implementierung. Das nächste Kapitel, Kapitel 6, bschreibt die Evaluierung des alternativen Vorhersagealgorithmus. Dort werden auch verschiedene Szenarien für den Algorithmus vorgestellt. Danach wird diese Bachelorarbeit zu anderen Arbeiten in Kapitel 7 abgegrenzt, bevor in Kapitel 8 ein Fazit sowie ein Ausblick gegeben wird.

Kapitel 2

Grundlagen und Technologien

Im Folgenden werden die benötigten Grundlagen und genutzen Technologien kurz erläutert.

2.1. Anomalienerkennung

Unter einer Anomalie versteht man das Fehlverhalten einer Applikation, was z.B. zu einer erhöhten Antwortzeit bei einer Datenbank führt. Diese Fehler aufzudecken ist die Aufgabe der Anomalienerkennung. Dabei ist es zum Beispiel für Unternehmen wichtig, Anomalien in den betrieblich genutzen Informationssystemen zu erkennen [Rohr u. a. 2007] und anschließend zu lokalisieren [Ehlers u. a. 2011]. Um festzustellen, ob eine Anomalie aufgetreten ist, werden die vorausberechneten Antwortzeiten und die tatsächlichen Antwortzeiten verglichen.
Dabei unterscheidet man drei Arten von Anomalien [Frotscher 2013, Seite 14]:

Punktanomalien Unter Punktanomalien versteht man einzelne Punkte in einer Datenreihe, die nicht normal sind. Diese passen also nicht zu den restlichen Punkten der Datenreihe. In Abbildung 2.1 ist eine solche Anomalie in orange dargestellt.

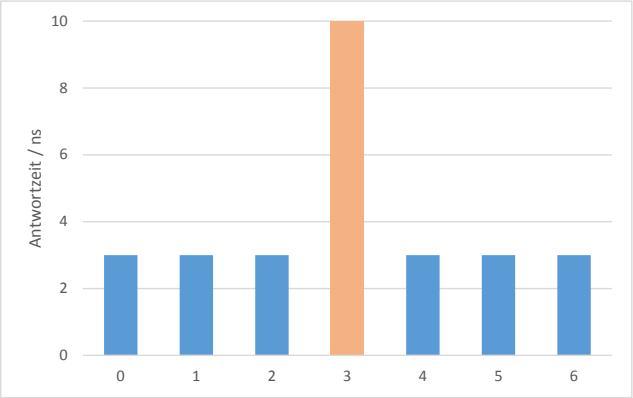

Abbildung 2.1. Visualisierung einer Punktanomalie

Kontextanomalien Eine Kontextanomalie ist eine Punktanomalie, die innerhalb eines bestimmten Kontexts, also z.B. im Vergleich zu den vorherigen Daten, als nicht normal eingestuft werden kann. Abbildung 2.2 zeigt eine Kontextanomalie. Dort ist ein Anstieg der Antwortzeit dargestellt. Zwischendurch fällt die Antwortzeit ab. Der Punkt in dem die Antwortzeit auf einmal deutlich geringer ist, ist die Kontextanomalie.

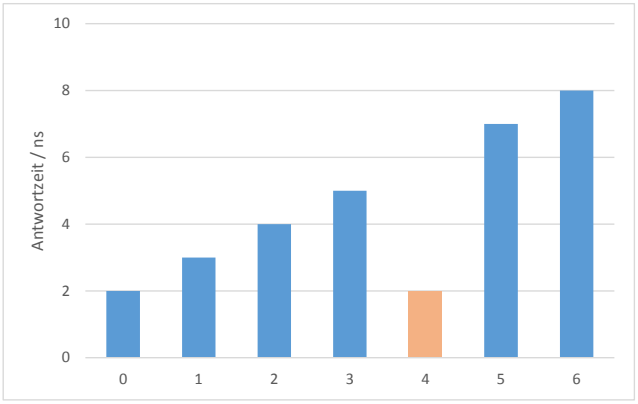

Abbildung 2.2. Visualisierung einer Kontextanomalie

Kollektivanomalien Diese Art der Anomalien ist sehr schwer zu entdecken, da sie nicht zu einem Zeitpunkt auftreten, sondern ein abnormales Verhalten der Antwortzeiten über einen längeren Zeitraum vorhanden ist. Die in Abbildung 2.3 orange dargestellten Säulen visualisieren eine Kollektivanomalie. Die Antwortzeiten zuvor und danach stellen ein normales Verhalten dar.

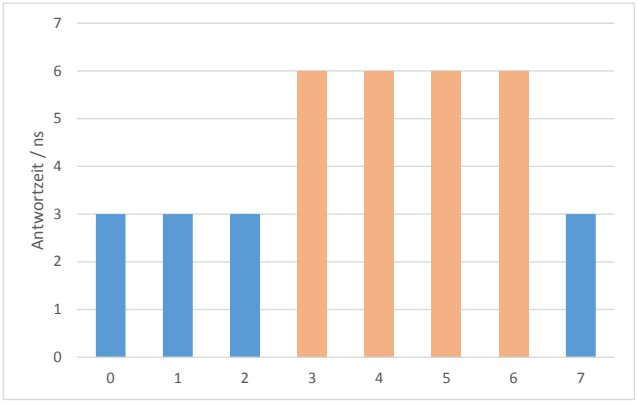

Abbildung 2.3. Visualisierung einer Kollektivanomalie

2.2. Anwortzeitvorhersage

Bei der Vorhersage soll mittels Algorithmen die nächste Anwortzeit vorausberechnet werden. Diese wird mithilfe der zuvor eingetretenen Antwortzeiten berechnet. Dabei gibt es verschiedene Strategien um den nächsten Wert zu berechnen. Die einfachste Variante besteht darin, den zuletzt eingetroffenen Wert als nächsten Wert zu nehmen. Eine andere Strategie ist es, aus einer gegebenen Anzahl von vorherigen Antwortzeiten den Durchschnittswert zu berechnen. Zusätzlich können die gegebenen Antwortzeiten noch gewichtet werden, indem zum Beispiel weiter in der Vergangenheit zurückliegende Antwortzeiten weniger stark für die Vorausberechnung berücksichtigt werden.

2.3. ExplorViz

ExplorViz [*ExplorViz Website*; Fittkau u. a. 2013] ist ein Tool zur interaktiven Visualisierung von großen Softwarelandschaften und bildet die Grundlage für das zu entwickelnde Kontrollzentrum, da dieses als Plug-In in ExplorViz eingebunden wird. Entwickelt wird ExplorViz von Florian Fittkau. Durch ExplorViz können Softwarelandschaften graphisch dargestellt werden, um die internen Zusammenhänge besser zu verstehen. Abbildung 2.4 zeigt die Visualisierung einer beispielhaften Softwarelandschaft in ExplorViz.

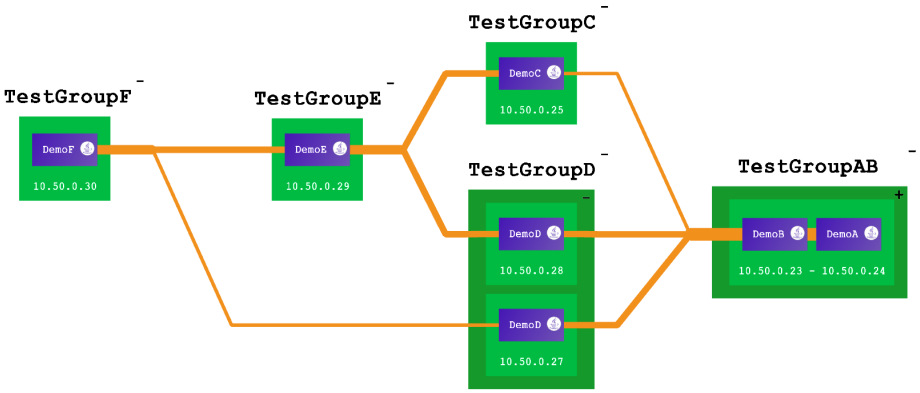

Abbildung 2.4. Visualisierung einer Softwarelandschaft in ExplorViz [*ExplorViz Website*]

Zusätzlich zu der Visualisierung der gesammten Softwarelandschaft kann der Benutzer tiefer in die einzelnen Applikationen hineingehen, wie in Abbildung 2.5 abgebildet. Dadurch kann der Benutzer die Abhängigkeiten besser nachvollziehen. In diesem Beispiel befindet sich der Benutzer in `DemoC` die zu der `NodeGroup` `TestGroupC` gehört. Die Applikation enthält `Packages`, in grün dargestellt, und `Classes`, die in blau dargestellt werden. Die orangen Linien stellen die Kommunikation zwischen zwei Klassen dar.

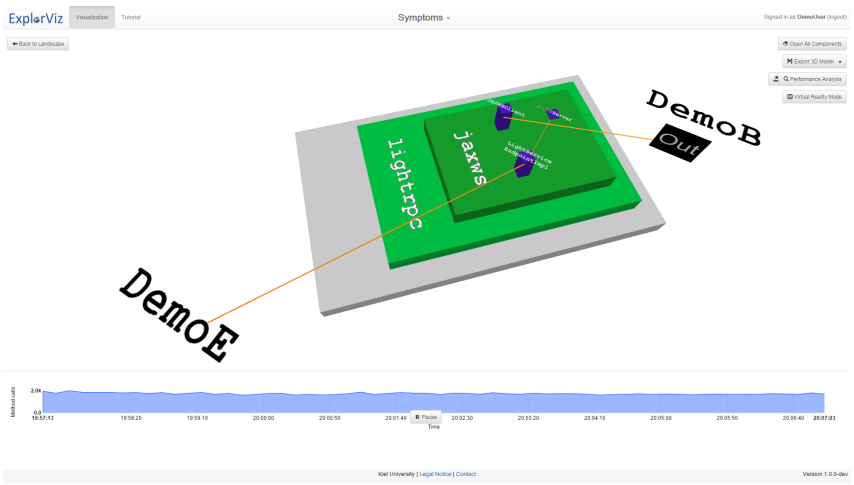

Abbildung 2.5. Visualisierung einer Applikation in ExplorViz [*ExplorViz Website*]

2.4. ΘPADx

ΘPADx [Frotscher 2013] ist eine Erweiterung von ΘPAD [Bielefeld 2012], welches von Tillmann Carlos Bielefeld im Rahmen seiner Diplomarbeit entwickelt wurde. ΘPADx wurde von Tom Frotscher entwickelt und kann eine übergebene Zeitreihe auf Anomalien untersuchen. Dafür wird der nächste Wert der Zeitreihe, durch einen Vorhersagealgorithmus, vorausberechnet und anschließend mit dem tatsächlich eingetretenen Wert verglichen. Mit diesen Informationen kann man die Anomaliewerte, ein Wert zwischen 0 und 1, berechnen und somit eine Einordnung der Anomalie nach ihrer Schwere vornehmen. Der Anomaliewert hängt dabei sehr stark von dem verwendeten Algorithmus zur Ermittlung des nächsten Wertes ab. Der Hauptunterschied zu ΘPAD ist, dass ΘPADx die drei Anomaliearten, dabei vor allem die Kontextanomalien mit einer höheren Genauigkeit erkennt. Abbildung 2.6 zeigt die Architektur von ΘPADx.

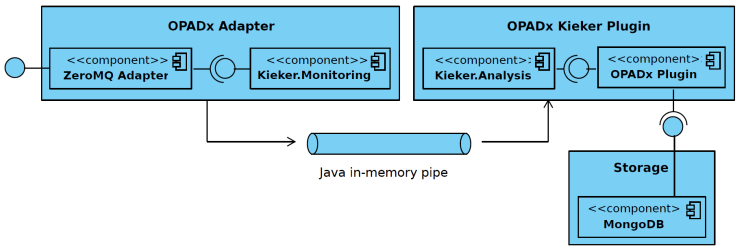

Abbildung 2.6. Architektur von ΘPADx [Frotscher 2013, S. 42]

2.5. OpenStack

OpenStack [*OpenStack Website*] ist eine Cloud Computing Software, welche mittels eines Rechenzentrums große Mengen von Rechen-, Speicher-, und Netzwerkressourcen verwaltet. OpenStack wird von Rackspace Cloud Computing und der NASA entwickelt und ist unter der Apache Lizenz als freie Software zugänglich. Mittlerweile wird das Projekt durch viele große Firmen der IT-Branche, darunter unter anderem Dell, IBM und Intel, unterstützt.

Kapitel 3

Projektmodul „Integration eines Kontrollzentrumkonzepts“

Dieses Kapitel erläutert zunächst die Idee des Projektes und gibt anschließend eine Übersicht über die vier Ansichten des Kontrollzentrums. Dabei liegt der Fokus auf der Anomalienerkennung. Bei den drei restlichen Ansichten Ursachenerkennung, Planungsphase und Ausführungsphase wird das Grundkonzept kurz erläutert.

3.1. Konzept

Die Idee des Kontrollzentrums besteht darin, dass die von ExplorViz visualisierte Softwarelandschaft auf Anomalien untersucht wird. Die Softwarelandschaft unterliegt dabei dem Landschaftsmodell in Abbildung 3.1. Dieses Modell teilt die Softwarelandschaft in mehrere Komponenten.
Anomalien sind dabei Fehlverhalten einer Applikation, was sich in erhöhten oder niedrigeren Antwortzeiten von Methoden wiederspiegelt. Wird eine Anomalie erkannt, so wird die Ursache der Anomalie vom Kontrollzentrum lokalisiert. Anschließend kann eine Strategie in Form eines Planes vom Kontrollzentrum entworfen werden, welche zur Auflösung der Anomalie führen soll. Dieser Plan kann dann vom Benutzer überprüft und gegebenfalls abgeändert werden, bevor der Plan vom Kontrollzentrum umgesetzt wird. Alternativ kann der Plan vom Benutzer auch abgelehnt werden.
Das Kontrollzentrum besteht aus insgesamt vier Ansichten. Jede dieser Ansicht hat eine bestimmte Aufgabe. Die erste Ansicht ist die `Symptons View`. In dieser Ansicht erfolgt die Anomalienerkennung. Die darauffolgende Ansicht ist die `Diagnosis View`, wo die Ursachenerkennung durchgeführt wird. In der dritten Ansicht, der `Planning View`, kann der Benutzer den Plan abändern und in der letzten Ansicht kann der Benutzer die Ausführung des Planes sehen. Diese Ansicht wird `Execution View` genannt.

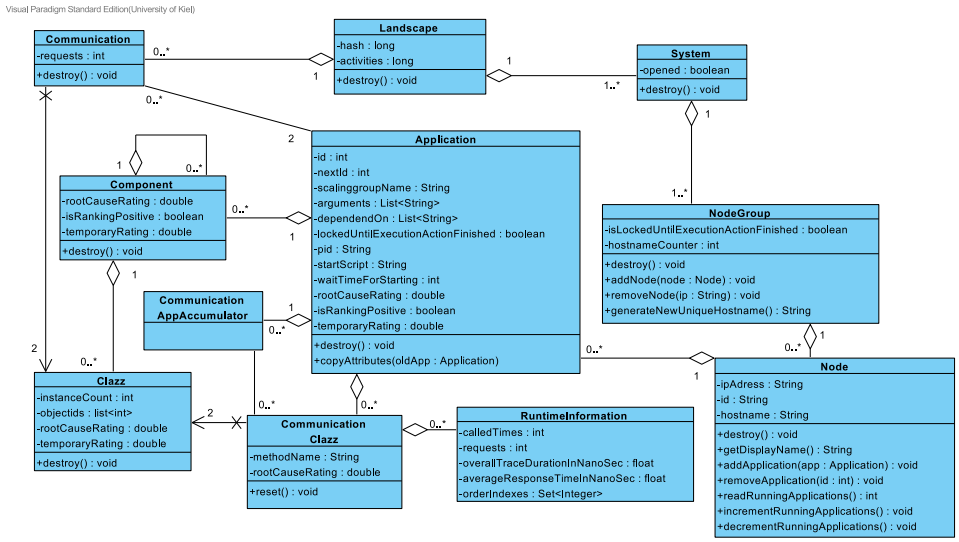

Abbildung 3.1. Klassendiagramm des Landschaftsmodells

3.2. Anomalienerkennung

In der Phase der Anomalienerkennung werden die Antwortzeiten aller Methoden, die sich in der Softwarelandschaft befinden mit vorherigen Antwortzeiten verglichen, um so die Anomaliewerte für diese Methode zu berechnen. Der Anomaliewert ist dabei ein normalisierter Wert zwischen -1 und 1 der angibt, wie schwerwiegend die Anomalie der Methode ist. Ein Wert von 0 bedeutet, dass es keine gefundene Anomalie gibt und die vorausberechnete Antwortzeit der tatsächlichen Antwortzeit entspricht. Je weiter der Wert an 1 oder -1 liegt, desto schwerwiegender ist die Anomalie. Der Unterschied zwischen einem positiven und einem negativen Wert liegt an der vorausberechneten und eingetroffenen Antwortzeit. Ist die vorausberechnete Antwortzeit höher als die tatsächlich eingetroffene Antwortzeit, so ist der Anomaliewert ein negativer Wert. Ist es umgekehrt, also die vorausberechnete Antwortzeit kleiner als die tatsächlich eingetroffene Antwortzeit, ist der Anomaliewert ein positiver Wert. Zur Berechnung der Vorhersage der Antwortzeit sind drei Algorithmen implementiert mit folgenden Konzepten:

NaiveForecaster Dieser Vorhersagealgorithmus nimmt die zuletzt tatsächlich eingetroffene Antwortzeit und gibt diese als Vorhersage für die nächste Antwortzeit zurück.

MovingAverageForecaster Der MovingAverageForecaster nimmt ein Fenster mit zuletzt tatsächlich eingetroffenen Antwortzeiten. Wie groß dieses Fenster ist, ist vom Benutzer konfigurierbar. Von den Zeiten in dem Fenster wird der Durchschnitt gebildet und als vorhergesagte Antwortzeit zurückgegeben.

WeightedForecaster Dem WeightedForecaster wird ebenfalls ein Fenster mit zuvor tatsächlich eingetroffenen Antwortzeiten übergeben. Diese Werte werden dann so gewichtet, dass weiter in der Vergangenheit zurückliegende Zeiten weniger berücksichtigt werden, als noch nicht so weit zurückliegende Zeiten. Anschließend wird von den gewichteten Zeiten der Durschnitt gebildet und diese Durchschnittszeit als vorhergesagte Antwortzeit zurückgegeben.

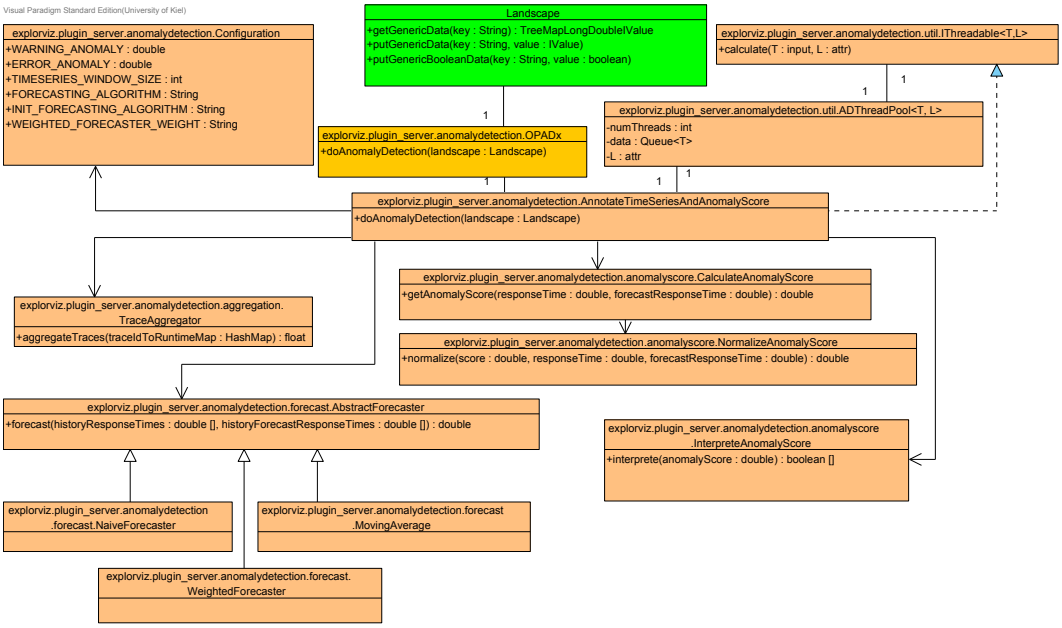

Abbildung 3.2. Klassendiagramm der Anomalienerkennung

In Abbildung 3.2 ist der Aufbau der Anomalienerkennung gezeigt. Klassen mit einem grünen Hintergrund bestanden schon vor dem Projekt, gelb hinterlegte Klassen wurden innerhalb des Projektes erweitert und alle Klassen mit einem orangen Hintergrund wurden neu implementiert. ExplorViz ruft die Methode `doAnomalyDetection(Landscape landscape)` der Klasse `OPADx` auf. Dort wird die Methode `doAnomalyDetection(Landscape landscape)` der Klasse `AnnotateTimeSeriesAndAnomalyScore` aufgerufen. Von dieser Klasse aus wird im Landschaftsmodell bis zur `CommunicationClazz` runtergegangen, um zu der Kommunikation der Klassen die jeweiligen Methoden zu kriegen. Diese werden dann auf die vorhandenden Threads des Servers aufgeteilt, um eine parallele Berechnung der Anomaliewerte zu ermöglichen. Zusätzlich werden in dieser Klasse alle berechneten Werte zusammengefasst. Somit ist die Klasse AnnotateTimeSeriesAndAnomalyScore die zentrale Klasse der Anomalienerkennung. Dort wird zunächst die Methode `aggregateTraces(HahsMap traceIdToRuntimeMap)` der Klasse `TraceAggregator` aufgerufen, um von den gesamten Antwortzeiten von jedem

Aufruf einer Methode den Durchschnitt zu berechnen. Dieser Durchschnittswert dient als aktuelle Antwortzeit der CommunicationClazz. Anschließend wird die Methode `forecast(TreeMapLongDoubleIValue historyResponseTime, TreeMapLongDoubleIValue historyForecastResponseTime)` der abstrakten Klasse `AbstractForecaster` ausgewählt. In dieser Methode wird einer der drei Vorhersagealgorithmen ausgewählt. Welcher Algorithmus ausgewählt wird, ist vom Benutzer in der Klasse `Configuration` festgelegt. Jeder dieser Algorithmen hat eine Methode `forecast()`, mit den zur Berechnung benötigten Werten als Parametern, die in der `forecast`-Methode des `AbtractForecaster` aufgerufen wird.
Mit der vorausberechneten Antwortzeit und der tatsächlich eingetroffenen Antwortzeit kann der Anomaliewert berechnet werden. Dafür wird die Methode `getAnomalyScore(double responseTime, double forecastResponseTime)` der Klasse `CalculateAnomalyScore` aufgerufen. Berechnet wird der Anomaliewert, indem man von der tatsächlich eingetroffenen Antwortzeit die vorausberechnete Antwortzeit subtrahiert. Anschließend wird der Anomaliewert noch normalisiert, indem in der Methode `getAnomrmalyScore()` die Methode `normalize(double score, double responseTime, double forecastResponseTime)` der Klasse `NormalizeAnomalyScore` aufgerufen wird. In dieser Methode wird die AnomalyScore durch die Summe aus eingetroffener und vorhergesagter Antwortzeit dividiert, um einen Anomaliwert zwischen -1 und 1 zu erhalten. Dieser normalisierte Anomaliewert wird dann an die Klasse `CalculateAnomalyScore` zurückgegeben und von dort aus an die Klasse `AnnotateTimeSeriesAndAnomalyScore` zurückgegeben. Der normalisierte Anomaliewert kann nun von der Klasse `InterpreteAnomalyScore` in der Methode `interprete(double anomalyScore)` interpretiert werden, indem überprüft wird, ob der Anomaliewert größer, als der Wert einer Warning Anomaly oder Error Anomaly ist. Wie groß der Wert einer Warning bzw. Error Anomaly ist, ist in der Konfiguration festgelegt. Zurückgegeben wird ein boolsches Array mit zwei Werten, wobei der erste Wert für die `Warning-Anomaly` und der zweite Wert für die textttError-Anomaly steht. Anhand dieser Werte können die Flags für die beiden Anomalien gesetzt werden. Diese Flags werden von ExplorViz genutzt, um dem Benutzer die Anomalie in der Softwarelandschaft anzuzeigen.
Somit gibt es drei Fälle, die in der Softwarelandschaft angezeigt werden können. In Abbildung 3.3 sind diese drei Fälle dargestellt. Sind sowohl das `Warning-Anomaly`-Flag und das `Error-Anomaly`-Flag auf `false` gesetzt, so wird nur der Name der Klasse angezeigt. Ist das `Warning-Anomaly`-Flag auf `true` gesetzt, wird ein gelbes Warndreieck über dem Klassennamen angezeigt. Ein rotes Warndreieck wird angezeigt, sobald das `Error-Anomaly`-Flag auf `true` gesetzt wird.

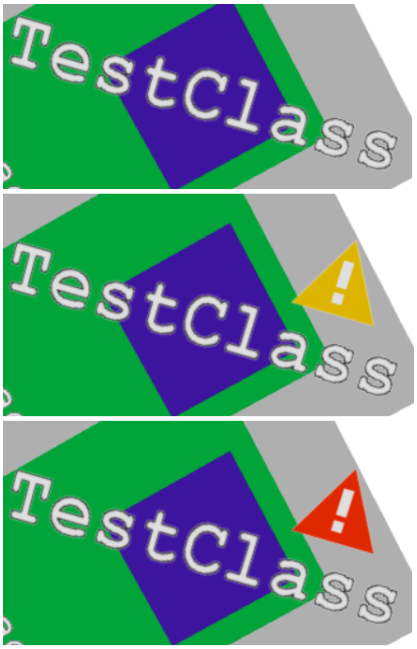

Abbildung 3.3. Visualisierung der gesetzen Flags

Diese Warndreiecke werden auch in allen `Packages` einer Applikation angezeigt, sofern eine Klasse existiert, in der eines der beiden Flags auf `true` gesetzt wurde. Zusätzlich kann sich der Benutzer die durchschnittliche Antwortzeit sowie die Anomaliewerte der letzten 10 Minuten darstellen lassen. Bei Klassen werden die tatsächlichen Werte genommen und bei Packages wird pro Zeitstempel der höchste Betrag aller Anomaliewerte der enthaltenen Klassen dargestellt. In Abbildung 3.4 ist eine solche Visualisierung dargestellt.

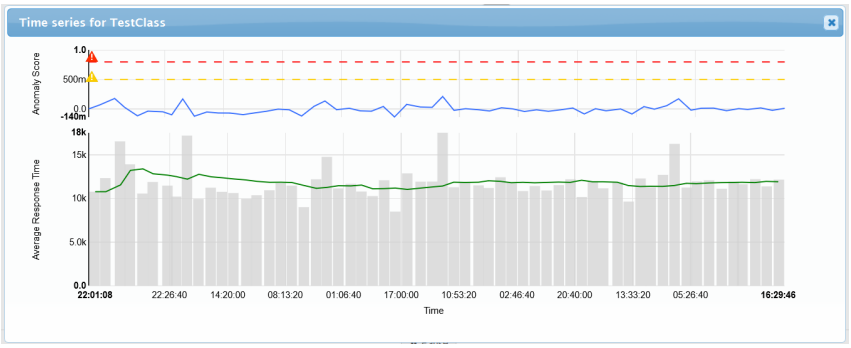

Abbildung 3.4. Visualisierung der durchschnittlichen Antwortzeit sowie der Anomaliewerte der letzten 10 Minuten

3.3. Ursachenerkennung

Bei der Ursachenerkennung soll mithilfe der zuvor berechneten Anomaliewerte der Methoden aus der Anomalienerkennung berechnet werden, wie wahrscheinlich es ist, dass eine Methode für die Anomalie verantwortlich ist. Ziel der Ursachenerkennung ist es, möglichst viele Methoden als Verursacher auszuschließen. Um dieses Ziel zu erreichen werden drei Algorithmen zur Verfügung gestellt:

Trivial Algorithmus Dieser Algorithmus bildet den Durchschnitt der Anomaliewerte einer Methode.

Simple Algorithmus Beim Simple Algorithmus hat jede Methode einen RanCorr-Wert. Ist der Anomaliewert einer aufrufenden Methode höher als der eigene, so wird der RanCorr-Wert der aufrufenden Methode erhöht, da es wahrscheinlich ist, dass die aufrufende Methode Ursache für die Anomalie ist. In dem Fall, dass der Anomaliewert einer aufgerufenen Methode höher als der Wert der eigenen Methode ist, wird der RanCorr-Wert der aufrufenden Methode gesenkt, da es wahrscheinlich ist, dass die aufgerufene Methode die Anomalie verursacht hat.

Advanced Algorithmus Der dritte Algorithmus betrachtet nicht nur die zuvor berechneten Anomaliewerte, sondern zusätzlich noch die architektonische Ebene. Dies bedeutet, dass die Anomaliewerte von benachbarten Objekten gewichtet werden. Diese Gewichtung ist umso höher, je dichter und öfter dieses Objekt aufgerufen wird.

Für eine bessere Visualisierung der Wahrscheinlichkeit einer Ursache, gibt es eine farbige Markierung der Objekte. Diese Markierung ist entweder grün, gelb oder rot, wobei grün für eine geringe Wahrscheinlichkeit und rot für eine hohe Wahrscheinlichkeit steht. In Abbildung 3.5 ist diese Markierung für eine geringe und eine hohe Wahrscheinlichkeit abgebildet.

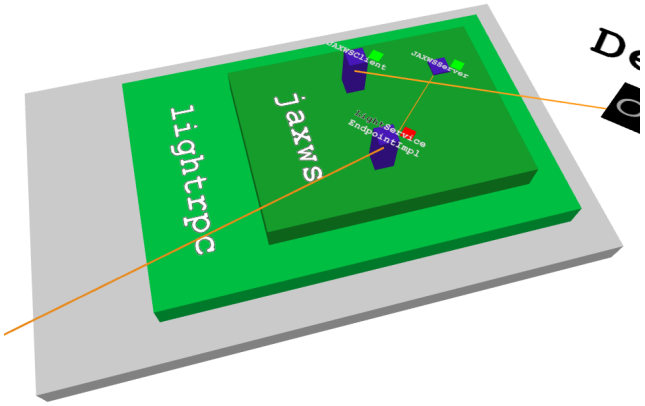

Abbildung 3.5. Visualisierung der Wahrscheinlichkeit

3.4. Planungsphase

Die Planungsphase ist die erste Phase des Kapazitätsmanagements und hat die Aufgabe, die berechneten Anomaliewerte der Anomalienerkennung und die berechneten Wahrscheinlichkeiten der Ursachenerkennung zu analysieren, um aufgrund dieser Analyse und der Softwarelandschaft einen Änderungsplan zu entwickeln. Der Benutzer kann anschließend den Plan annehmen, ändern oder ablehnen.
Um einen Plan zu erstellen, wird zunächst in der Softewarelandschaft nach Markierungen gesucht, die während der Ursachenerkennung gesetzt wurden. Diese Markierungen zeigen an, dass ein Knoten die Ursache einer Anomalie ist. Durch die Auswertung des Anomaliewertes einer Applikation eines solchen Knotens, kann eine Strategie zur Lösung der Anomalie erarbeitet werden. Alle Lösungsstrategien werden dann in einem Änderungsplan zusammengefasst, welcher dann dem Benutzer vorgeschlagen wird. Zu den Lösungsstrategien gehört unter anderem das Zuschalten von Knoten bei einer Überlast und Abschalten von Applikationen bei einer Unterlast. Diese Information wird durch den Anomaliewert gegeben. Durch ein Kontextmenü (Abbildung 3.6) kann der Benutzer in der `Planning View` den Plan abändern. In diesem Fall soll die Applikation `DemoE` migriert werden. Dort hat der Benutzer zusätzlich die Möglichkeit, eine Applikation zu migrieren oder neu zu starten sowie einen Knoten neu zu starten oder zu beenden.

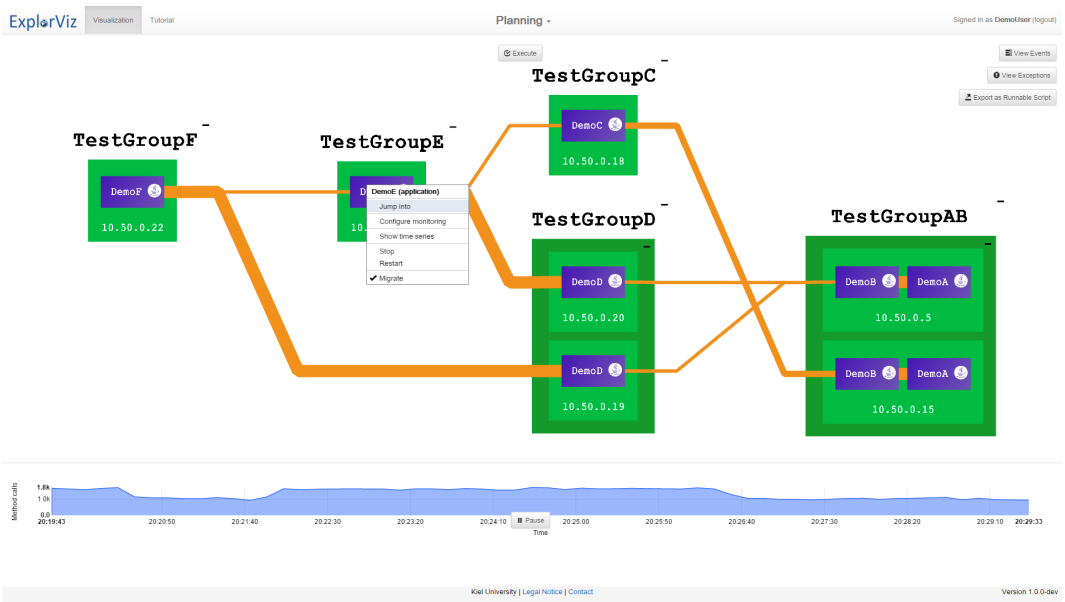

Abbildung 3.6. Kontextmenü in der Planungsphase

Der ausgearbeitete Plan kann in jeder Ansicht von ExplorViz angezeigt werden. Wird der Plan akzeptiert, gelangt der Benutzer in die Ausführungsphase. Bei manueller Änderung des Plans gelangt der Benutzer in die Planungsphase und beim Abbrechen des Plans bleibt der Benutzer in der aktuellen Phase. In Abbildung 3.7 wird ein Plan dargestellt. Dieser enthält die Information, in welcher Applikation eine Anomalie aufgetreten ist und was der Grund dafür ist. In Abbildung 3.7 ist es eine Überlast. Des Weiteren enthält der Plan einen Vorschlag, um die aufgetretene Anomalie zu beseitigen. In diesem Fall das Replizieren des Knotens, in der die Applikation enthalten ist. Schließlich wird dem Benutzer noch die Konsequenz des Planes angezeigt, sollte dieser so ausgeführt werden. In dem Beispielplan wäre dies die Verbesserung der Antwortzeit sowie eine Kostensteigerung von 5,- €.

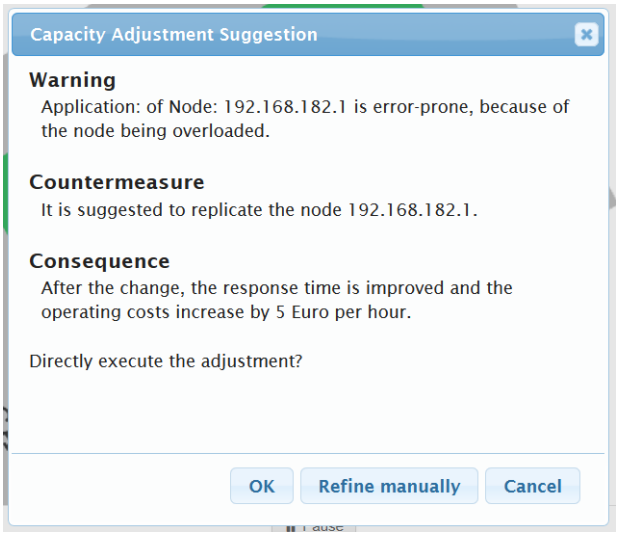

Abbildung 3.7. Plan, welcher dem Benutzer vorgeschlagen wird

3.5. Ausführungsphase

Diese Phase ist der zweite Teil des Kapazitätsmanagements und hat zur Aufgabe, den in der ersten Phase erstellten Änderungsplan umzusetzen, nachdem dieser vom Benutzer genehmigt oder abgeändert wurde. Wird der vom Kontrollzentrum vorgeschlagene Plan abgelehnt, kommt diese Phase nicht zum Einsatz. Sollte bei der Ausführung eines Plans ein Fehler auftreten, so wird der Zustand vor Beginn der Planausführung durch eine Kompensation wiederhergestellt. Dabei wird zu jeder Aktion, die in dem Plan enthalten ist, die gegensätzliche Aktion ausgeführt. In Abbildung 3.8 ist eine Tabelle mit allen Aktionen und den dazugehörigen Gegenaktionen abgebildet. Wobei der Neustart von Knoten und Applikationen keine Gegenaktion hat. Die Gegenaktion der Terminierung eines Knoten

Applikationen	
Aktion	Gegenaktion
Neustart	-
Terminierung	Neustart
Migration	Migration
Knoten	
Aktion	Gegenaktion
Neustart	-
Start	Terminierung
Terminierung	Start
Replizierung	Terminierung

Abbildung 3.8. Kompensation von Applikationen und Knoten

oder einer Applikation ist das Starten des terminierten Knotens oder der terminierten Applikation. Wird eine Applikation migriert, so ist die Gegenaktion des Migrierens ein weiteres Migrieren der Applikation auf den ursprünglichen Knoten. Sieht der Plan das Starten eines Knotens vor, so ist die Gegenaktion die Terminierung des Knotens. Ebenso ist es bei der Replizierung, also dem duplizieren eines Knotens. Dabei wird dort das Duplikat des Knotens terminiert.
In Abbildung 3.9 ist die `Execution View` während der Umsetzung eines Plans abgebildet. Der Plan beinhaltet die Migration der Applikation `DemoE`, welche sich auf dem Knoten 10.50.0.21 befindet. In der Abbildung ist die Applikation bereits auf dem neuen Knoten kopiert und die Kommunikation mit der originalen Applikation auf dem Knoten 10.50.0.21 ist bereits beendet. Dies erkennt man an den dünnen orangen Linien, die an der Applikation ankommen. Im nächsten Schritt wird die Kommunikation mit der Kopie der Applikation auf dem neuen Knoten 10.50.0.15 gestartet.

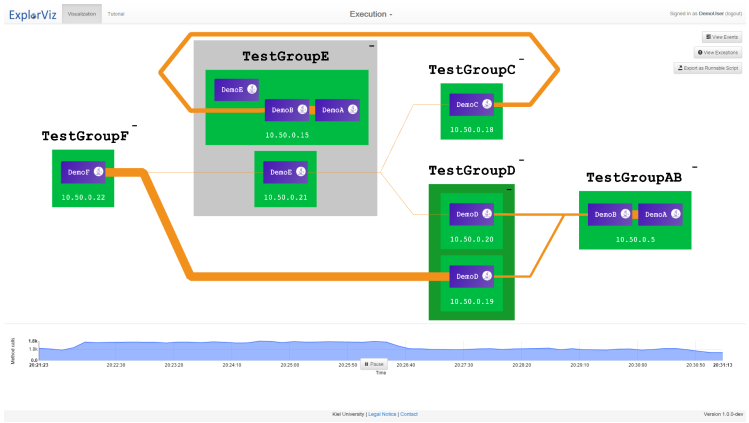

Abbildung 3.9. Ausführungsphase während der Verabeitung eines Plans

Kapitel 4

Alternativer Vorhersagealgorithmus

Im Folgenden wird das Konzept eines alternativen Vorhersagealgorithmus erläutert. Der alternative Algorithmus ist ein gewichteter Algorithmus und wir nennen ihn deshalb WeightedForecaster. Dieser Algorithmus greift das Prinzip des MovingAverageForecasters auf und erweitert dieses um eine Gewichtung der einzelnen Antwortzeiten.

4.1. Konzept

Die Idee des alternativen Vorhersagealgorithmus ist es, die einzelnen Antwortzeiten, welche der Algorithmus als Eingabe erhält, zu gewichten und anschließend die nächste Antwortzeit vorherzusagen. Die übergebenen Antwortzeiten werden Fenster genannt und die Größe des übergebenen Fensters wird vorher vom Benutzer in einer Konfiguration festgelegt. Der Algorithmus multipliziert jeden dieser Zeitwerte in dem Fenster mit einem vorher berechneten Gewicht. Dabei ist der erste Wert des Fensters die Antwortzeit der Methode, die am weitesten in der Vergangenheit zurückliegt und der letzte Wert die Antwortzeit, welche bei der letzten Messung ermittelt wurde. Da in einer Softwarelandschaft unterschiedliche Arten von Applikationen enthalten sein können, soll der Benutzer eine von drei möglichen Gewichtungen auswählen können, um die Antwortzeit vorherzusagen:

LOGARITHMIC Ist diese Gewichtung vom Benutzer ausgewählt worden, so wird eine logarithmische Gewichtung zur Basis 10 vorgenommen. Die Formel für das Gewicht eines Wertes lautet $1 + log(i + 1)$, wobei i den Index des Wertes in dem Fenster beginnend mit 0 darstellt. Um eine Berechnung des Logarithmus von 0 zu vermeiden, wird i um 1 erhöht. Zusätzlich wird zu dem Ergebnis der Logarithmusberechnung 1 addiert, um später eine Multiplikation der Antwortzeit mit 0 zu vermeiden. Der erste Wert des übergebenen Fenster wird also mit 1 multipliziert und bei jeder weiteren Zeit ändert sich das Gewicht durch den Logarithmus von $i + 1$ addiert mit 1.
In Abbildung 4.1 ist ein Beispiel für eine logarithmische Gewichtung gegeben. Zu erkennen ist eine Zeitreihe mit 15 Zeitstempeln. Die Gewichtung liegt zwischen 1 und ungefähr $2,18$. Für die angegebene Zeitreihe wird mit der logarithmischen Gewichtung ein Vorhersagewert von ungefähr $6,397$ ns berechnet. Dieser Wert berechnet sich durch die Division mit Gleichung 4.1 als Dividend und Gleichung 4.2 als Divisor.

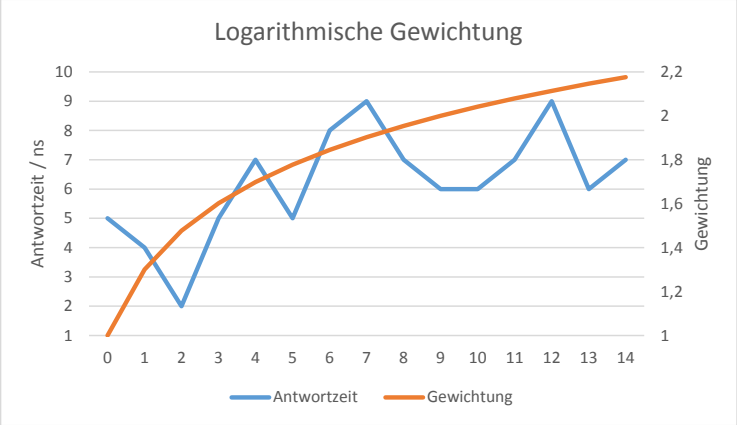

Abbildung 4.1. Beispiel einer logarithmischen Gewichtung

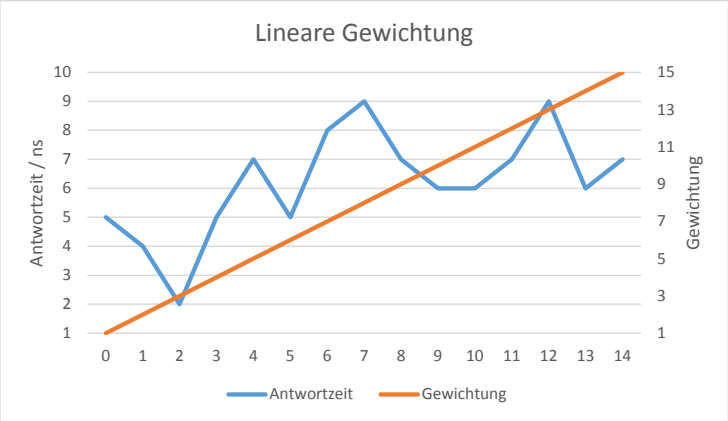

Abbildung 4.2. Beispiel einer linearen Gewichtung

LINEARLY Bei dieser Gewichtung handelt es sich um eine lineare Gewichtung. Sie beginnt bei 1 und bei jedem weiteren Wert wird das Gewicht um 1 erhöht, also ist das Gewicht für jeden Wert $i + 1$, wobei i den Index des Wertes in dem Fenster beginnend mit 0 darstellt. Um eine Multiplikation der vergangenen Antwortzeit mit 0 zu verhindern, wird das Gewicht zum Index i der Wert 1 addiert.
Abbildung 4.2 zeigt eine lineare Gewichtung einer Zeitreihe mit 15 Zeitstempeln. Die Gewichtung beträgt zu Beginn 1 und steigt dann kontinuierlich mit jedem weiteren Zeitstempel um 1 an, sodass die Gewichtung beim letzten Zeitstempel 15 beträgt. Durch die Berechnung mit Gleichung 4.1 und Gleichung 4.2 ergibt sich eine vorausberechnete Antwortzeit von ungefähr 6,758 ns.

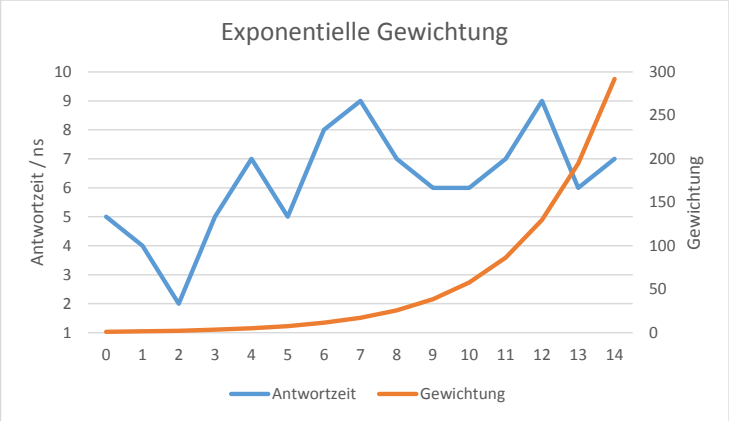

Abbildung 4.3. Beispiel einer exponentiellen Gewichtung

EXPONENTIALLY Bei dieser Gewichtung haben die am weitesten in der Vergangenheit zurückliegenden Zeitstempel den geringsten Einfluss auf die berechnete Antwortzeit beachtet. Es handelt sich bei dieser Gewichtung um eine exponentielle Gewichtung zur Basis 1,5. Der Exponent beginnt mit 0 und wird bei jedem weiteren Wert des Fensters um 1 erhöht. Daraus ergibt sich ein Gewicht von $1,5^i$ für jeden Wert, wobei auch hier i den Index des Wertes in dem Fenster beginnend mit 0 darstellt.
Das Diagramm in Abbildung 4.3 zeigt eine beispielhafte exponentielle Gewichtung. Die Gewichtung beginnt mit 1 und steigt dann exponentiell an bis eine Gewichtung von ungefähr 291,9 für den letzten Zeitstempel im Zeitfenster erreicht ist. Die aus den Daten des Diagramms vorausberechnete Antwortzeit durch Gleichung 4.1 und Gleichung 4.2 beträgt somit ungefähr $6,971$ ns.

Nachdem für die jeweilige Variante die gewichteten Antwortzeiten berechnet wurden, kann für das Fenster der Durchschnittswert berechnet werden. Dies geschieht durch die Division der aufsummierten, gewichteten Antwortzeiten (Gleichung 4.1) und der aufsummierten Gewichte (Gleichung 4.2). Diese beiden Werte werden durch folgende Formeln berechnet:

$$weightedResponseTimes = \sum_{0}^{i} t_i * w_{t_i} \tag{4.1}$$

$$sumOfWeights = \sum_{0}^{i} w_{t_i} \tag{4.2}$$

Dabei steht t_i für die Antwortzeit in dem Fenster an der Stelle i und w_{t_i} für das durch eine der drei Varianten berechnete Gewicht der Antwortzeit t_i in dem übergebenen Fenster.
Das Ergebnis dieser Division ist eine vorhergesagte Antwortzeit, welche zur Berechnung des Anomaliewertes genutzt wird.

4.2. Erkennung von Anomaliearten

Bei Anomalien unterscheidet man drei Arten. Diese sind Punktanomalien, Kontextanomalien und Kollektivanomalien und wurden in Abschnitt 2.1 näher beschrieben. Damit es zu einer Anomalie kommt, bedarf es einer signifikanten Änderung in der Antwortzeit. Durch den alternativen Vorhersagealgorithmus erkennt die Anomalienerkennung Punktanomalien und Kontextanomalien. Bei den Kollektivanomalien wird der Beginn erkannt. Je länger die Kollektivanomalie dauert, desto geringer ist die Wahrscheinlichkeit, dass die Anomalie auch als solche erkannt wird. Es kommt somit auf die Dauer einer Kollektivanomalie an, wie groß das betrachtete Fenster ist und welche Gewichtung ausgewählt ist.

Kapitel 5

Implementierung des alternativen Vorhersagealgorithmus

In diesem Kapitel wird genauer auf die Implementierung des alternativen Vorhersagealgorithmus eingegangen. Der WeightedForecaster wurde in der Programmiersprache Java in der Version 1.7 implementiert.

5.1. Initialisierung

Der Algorithmus bekommt einen Parameter übergeben, wie in Listing 5.1 abgebildet. Dieser Parameter sind die vergangenen, tatsächlich eingetretenen Antwortzeiten in Form einer `TreeMapLongDoubleIValue`. In dieser `Map` sind zum einen die Antwortzeiten als `Double`-Werte gespeichert sowie die dazugehörigen Zeitstempel als `Long`-Werte, um einen zeitlichen Bezug herstellen zu können. Um besser mit den Antwortzeiten arbeiten zu können, werden diese in eine `ArrayList` geschrieben, wobei die Antwortzeit mit dem kleinsten Zeitstempel den Index 0 bekommt. Alle weiteren Antwortzeiten werden in aufsteigender Reihenfolge bezüglich des Zeitstempels in die `ArrayList` geschrieben. In Listing 5.1 werden anschließend einige Variablen, die zur Berechnung der vorhergesagten Antwortzeit genutzt werden, initialisiert. Die erste Variable heißt `size` und enthält die Größe der `ArrayList`. Die nächste Variable, die initialisiert wird, heißt `weightedResponseTimes`. Diese Variable enthält die Summe der gewichteten historischen Antwortzeiten. Die letzte Variable heißt `sumOfWeights` und enthält später die Summe der Gewichte. Beide Variablen werden mit 0 initialisiert und sind `Double`-Werte. Welche Berechnung als nächstes ausgeführt wird, hängt von der Konfiguration ab. Diese legt fest, welche Gewichtung zur Berechnung der vorhergesagten Antwortzeit angewendet wird. Dabei ist der Ablauf der Berechnung in allen drei Fällen identisch.

```
1 public static double forecast(TreeMapLongDoubleIValue historyResponseTimes) {
2     ArrayList<Double> historyResponseTimesValues = new ArrayList<>(
3                 historyResponseTimes.values());
4     int size = historyResponseTimesValues.size();
5     double weightedResponseTimes = 0;
6     double sumOfWeights = 0;
```

Listing 5.1. Initialisierung wichtiger Variablen

```
1 if (Configuration.WEIGHTED_FORECASTER_WEIGHT.equals("LOGARITHMIC")) {
2     for (int i = 0; i < size; i++) {
3         double weight = 1 + Math.log10(i + 1);
4         weightedResponseTimes += historyResponseTimesValues.get(i) * weight;
5         sumOfWeights += weight;
6     }
7     return weightedResponseTimes / sumOfWeights;
8 }
```

Listing 5.2. Berechnung der Antwortzeit durch logarithmische Gewichtung

Zunächst wird für jede vergangene Antwortzeit das Gewicht berechnet, anschließend die gewichtete Antwortzeit, welche zu dem Wert der Variable `weightedResponseTimes` addiert wird. Zuletzt wird zu dem bestehenden Wert der Variable `sumOfWeights` das berechnete Gewicht addiert. Anschließend kann mit den aufsummierten gewichteten Antwortzeiten und der Summe aller Gewichte die vorhergesagte Antwortzeit berechnet werden. Dies geschieht, indem die beiden Variablen dividiert werden.
Um die vom Benutzer gewünschte Gewichtung zur Berechnung zu verwenden, wird zunächst aus der Klasse `Configuration` die zu verwendende Gewichtung abgefragt und nacheinander mit den drei möglichen Gewichtungen verglichen. Für den Fall, dass eine nicht existierende Gewichtung in der Konfiguration angegeben wurde, wird eine `FalseWeightInConfigurationException` geworfen.

5.2. Logarithmische Gewichtung

Listing 5.2 zeigt die Berechnung der vorhergesagten Antwortzeit durch die logarithmische Gewichtung. Zur Berechnung des Logarithmus wird dabei die von Java zur Verfügung gestellte Klasse `Math` benutzt. Es wird dabei der Logarithmus zur Basis 10 genutzt. Nachdem die Antwortzeit berechnet wurde, wird dieser zurückgegeben.

5.3. Lineare Gewichtung

Wurde nicht die logarithmische Gewichtung ausgewählt, so wird nach der linearen Gewichtung gefragt, deren Implementierung in Listing 5.3 dargestellt ist. Auch hier wird die berechnete Antwortzeit nach der Berechnung zurückgegeben, für den Fall, dass diese Gewichtung in der Konfiguration so festgelegt wurde.

5.4. Exponentielle Gewichtung

Nachdem die beiden vorherigen Gewichtungen nicht in der Konfiguration festgesetzt wurden, wird durch eine weitere Anweisung nach der dritten Variante gefragt.

```
else if (Configuration.WEIGHTED_FORECASTER_WEIGHT.equals("LINEARLY")) {
    for (int i = 0; i < size; i++) {
        double weight = i + 1;
        weightedResponseTimes += historyResponseTimesValues.get(i) * weight;
        sumOfWeights += weight;
    }
    return weightedResponseTimes / sumOfWeights;
}
```

Listing 5.3. Berechnung der Antwortzeit durch lineare Gewichtung

```
else if (Configuration.WEIGHTED_FORECASTER_WEIGHT.equals("EXPONENTIALLY")) {
    for (int i = 0; i < size; i++) {
        double weight = Math.pow(1.5, i);
        weightedResponseTimes += historyResponseTimesValues.get(i) * weight;
        sumOfWeights += weight;
    }
    return weightedResponseTimes / sumOfWeights;
}
```

Listing 5.4. Berechnung der Antwortzeit durch exponentielle Gewichtung

Die Implementierung dafür ist in Listing 5.4 abgebildet. Für die Berechnung der Gewichtung wird die Exponentialfunktion `pow()` der Klasse `Math` verwendet. Als Basis für die Exponentialfunktion wird der Wert $1,5$ verwendet. Als Exponent wird der Index i der Antwortzeit, für welche die Gewichtung angewendet werden soll, verwendet. Nachdem alle Antwortzeiten des übergebenen Fensters gewichtet wurden, wird die nächste Antwortzeit vorausberechnet und zurückgegeben.

Kapitel 6

Evaluierung

In diesem Kapitel wird der alternative Algorithmus evaluiert. Dazu werden zunächst die Ziele der Evaluierung festgelegt sowie die Methodik und der Aufbau erläutert. Anschließend werden drei Szenarien beschrieben, welche dann ausgewertet und bewertet werden.

6.1. Ziele

Das primäre Ziel der Evaluation ist die Überprüfung des alternativen Vorhersagealgorithmus auf die Erkennung der drei Anomaliearten. Zusätzlich soll eine Untersuchung zu der Erkennung im Vergleich zu den beiden anderen, in der Anomalienerkennung enthaltenen Vorhersagealgorithmen, erfolgen.

6.2. Methodik

Für die Ausführung der fünf Szenarien wird dem WeightedForecaster ein Fenster mit einer Größe von 15 übergeben. Dieses Fenster enthält 15 Wertepaare aus je einem Zeitstempel und einer dazugehörigen Antwortzeit. Die Zeitstempel beginnen mit 0 und erhöhen sich pro Zeitstempel um 1. Die Werte für die Antwortzeiten wurden durch einen Zufallszahlengenerator generiert, wobei die Antwortzeiten des Zeitfensters der Punktanomalien und Kollektivanomalie im Bereich zwischen 500 Nanosekunden (ns) und 650 ns liegen und die Antwortzeiten für die Kollektivanomalie zwischen 500 ns und 1200 ns. Anschließend wurden die Antwortzeiten der Kontextanomalie aufsteigend bzw. absteigend sortiert. Die tatsächlich eingetroffene Antwortzeit wurde ebenfalls durch einen Zufallszahlengenerator generiert. Als Ergebnis gibt der Algorithmus einen Wert zurück, welcher die vorausberechnete Antwortzeit in Nanosekunden darstellt. Dies gilt auch für die beiden anderen Algorithmen.
Um die vorausberechneten Antwortzeiten am Ende analysieren zu können, werden mit den vorausberechneten Antwortzeiten und einer weiteren Antwortzeit, welche die tatsächlich eingetroffene Antwortzeit darstellt, Anomaliewerte berechnet, welche anschließend normalisiert werden. Dabei wird immer eine kleinere Antwortzeit und eine größere Antwortzeit gewählt, um sowohl die Erkennung einer Unterlast, als auch einer Überlast überprüfen zu können. Weiter werden die berechneten normalisierten Anomaliewerte auf drei Stellen

nach dem Komma gerundet. Aus Darstellungsgründen werden die von den Algorithmen vorhergesagten Antwortzeiten in den entsprechenden Abbildungen dieser Werte auf sieben Stellen nach dem Komma gerundet.
Für die Feststellung, ab wann ein Anomaliewert eine Anomalie darstellt, wird für den Schwellwert einer Warnung ein Wert von $\pm 0,5$ und für den Schwellwert eines Fehlers ein Wert von $\pm 0,7$ verwendet.

6.3. Aufbau

Der alternative Vorhersagealgorithmus sowie die beiden anderen Algorithmen, wurden auf einem PC mit Windows 8.1 in der 64-bit Variante, einem Intel Core i7-4500U und 16 GB RAM getestet.

6.4. Szenarien

Im Folgenden werden die verschiedenen Szenarien für den alternativen Vorhersagealgorithmus vorgestellt. Dabei wird in jedem der Szenarien jede Gewichtung des Weighted-Forecasters benutzt. Zusätzlich werden die beiden anderen Algorithmen mit dem selben Zeitfenster eine Antwortzeit vorausberechnen, um Vergleichswerte zu erhalten. Für die Punktanomalie und Kontextanomalie gibt es jeweils zwei Szenarien. Im ersten wird die Anomalie durch eine Unterlast ausgelöst, im zweiten durch eine Überlast. Für die Kollektivanomalie existiert ein Szenario. In diesem wird die Anomalie durch eine Überlast ausgelöst.

Punktanomalie durch Unterlast

In diesem Szenario werden die drei Algorithmen auf das Erkennen eine Punktanomalie überprüft. Die Anomalie wird dafür durch eine Unterlast ausgelöst. In Abbildung 6.1 ist dieses Fenster in Form eines Diagramms abgebildet. Die Antwortzeiten dieses Fensters liegen alle zwischen 514 ns und 649 ns und es gibt innerhalb des Fensters keine Anomalien, da die einzelnen Veränderungen zwischen den Zeitstempeln nicht ausreichend sind.

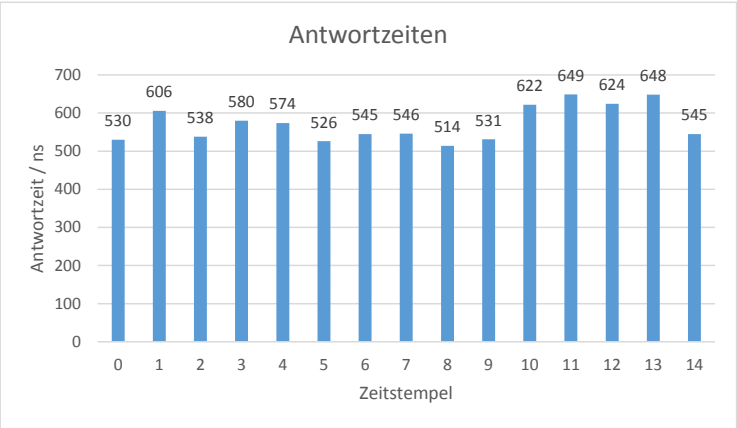

Abbildung 6.1. Übergebenes Zeitfenster für die Algorithmen vor einer Unterlast

Punktanomalie durch Überlast

Im zweiten Szenario zur Punktanomalie wird die Anomalie durch eine Überlast ausgelöst. Als Eingabe zur Berechnung der vorausgesagten Antwortzeit erhalten alle Algorithmen das in Abbildung 6.2 abgebildete Diagramm, welches das Fenster mit den vergangenen Antwortzeiten enthält. Diese Antwortzeiten liegen zwischen 512 ns und 642 ns. Da es keine signifikanten Änderungen zwischen den einzelnen Zeitstempeln gibt, enthält dieses Fenster keine Anomalien.

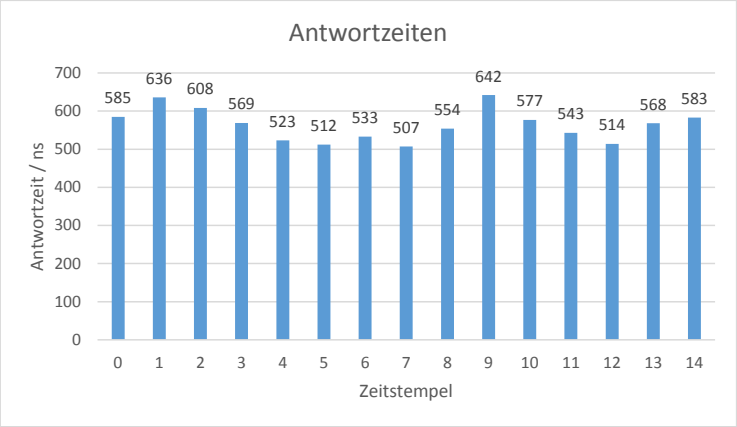

Abbildung 6.2. Übergebenes Zeitfenster für die Algorithmen vor einer Überlast

Kontextanomalie durch Unterlast

Im ersten Szenario der Kontextanomalie wird diese durch eine Unterlast ausgelöst. Die Grundlage zur Berechnung stellt das in Abbildung 6.3 gegebene Zeitfenster. Dieses, den Algorithmen übergebene Fenster, enthält Antwortzeiten die einen durchgängigen Anstieg von 509 ns bis auf 1144 ns verzeichnen. Da die Differenz zwischen benachbarten Antwortzeiten nicht relevant genug sind, enthält das Zeitfenster keine Anomalien.

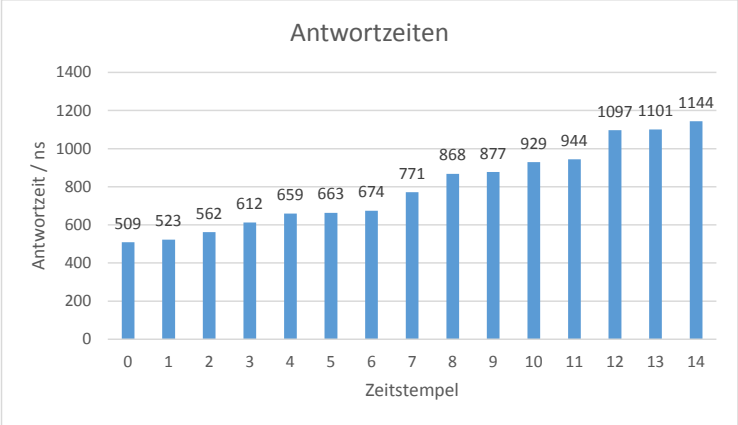

Abbildung 6.3. Übergebenes Zeitfenster für die Algorithmen vor einer Unterlast

Kontextanomalie durch Überlast

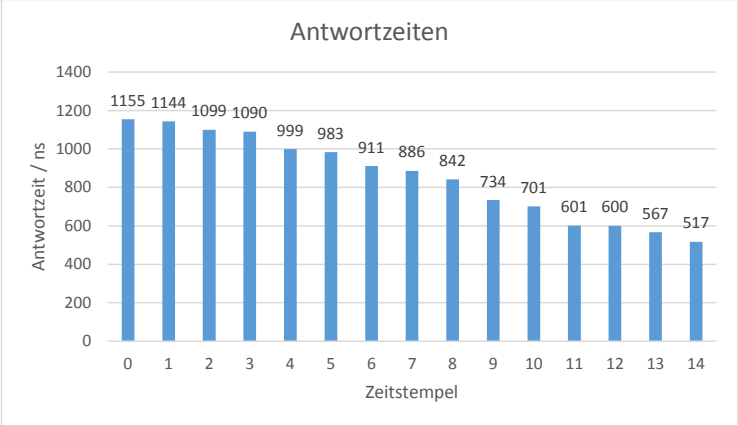

Abbildung 6.4. Übergebenes Zeitfenster für die Algorithmen vor einer Überlast

Um das Erkennen einer Kontextanomalie durch Überlast zu überprüfen, wird das Zeitfenster in Abbildung 6.4 als Eingabe für die Algorithmen verwendet. Die in diesem Fenster enthaltenen Antwortzeiten sinken durchgängig und in unregelmäßigen Abständen von 1155 ns auf 517 ns ab. Da es keine wesentlichen Änderungen in der Antwortzeit zwischen benachbarten Zeitstempeln gibt, existieren in diesem Zeitfenster keine Anomalien.

Kollektivanomalie

Das Szenario einer Kollektivanomalie wird für eine Anomalie durch Unterlast durchgeführt. Da eine Kollektivanomalie nicht nur eine Dauer von einem Zeitstempel hat, werden mehrere Berechnungen nacheinander durchgeführt, wobei die tatsächlich eingetretene Antwortzeit aus der letzten Berechnung dem Zeitfenster als aktuellste Antwortzeit hinzugefügt wird und die älteste Antwortzeit aus dem Fenster entfernt wird. Das aktualisierte Fenster wird anschließend den Algorithmen zur Berechnung der vorausgesagten Antwortzeit übergeben. Die Dauer der Kontextanomalie beträgt fünf Zeitstempel.
Das Diagramm in Abbildung 6.5 zeigt die vergangenen Antwortzeiten vor dem Beginn der Kollektivanomalie. Die Antwortzeiten liegen zwischen 504 ns und 600 ns und werden den drei Algorithmen als anomaliefreies Fenster zu Beginn der ersten Berechnung übergeben.

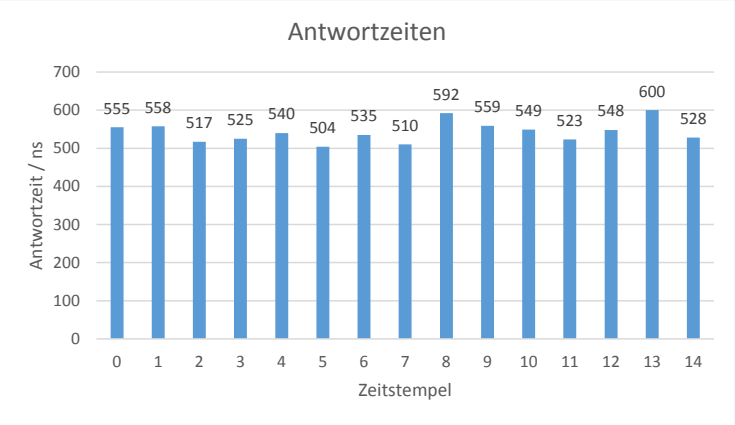

Abbildung 6.5. Übergebenes Zeitfenster für die Algorithmen

6.5. Ergebnisse

Im Folgenden Abschnitt werden die Ergebnisse der Eingaben aus den Szenarien aus dem vorherigen Abschnitt dargestellt.

Punktanomalie durch Unterlast

Abbildung 6.6 zeigt die vorausberechneten Antwortzeiten, welche mit dem Zeitfenster aus Abbildung 6.2 ermittelt wurden. Der WeightedForecaster berechnet mit der logarithmischen Gewichtung eine Antwortzeit von ungefähr 574,52 ns, mit der linearen Gewichtung eine Antwortzeit von 581,75 ns und mit der exponentiellen Gewichtung eine Antwortzeit von annähernd 593,75 ns. Im Vergleich dazu hat der NaiveForecaster eine Antwortzeit von 545 ns vorausberechnet und der MovingAverageForecaster eine Antwortzeit von circa $571,8\overline{6}$ ns.
Durch diese vorausberechneten Antwortzeiten sowie der tatsächlichen Antwortzeit von 134 ns, errechnet die Anomalienerkennung folgende, in Abbildung 6.6 abgebildete, Anomaliewerte: Mit dem Ergebnis des WeightedForecaster durch die logarithmische Gewichtung einen Wert von -0,622, durch die lineare Gewichtung einen Wert von -0,626 und durch die exponentielle Gewichtung einen Wert von -0,632. Durch das Ergebnis des NaiveForecasters ergibt sich ein Anomaliewert von -0,605 und mit dem MovingAverageForecaster als Vorhersagealgorithmus in der Anomalienerkennung ergibt sich ein Anomaliewert von -0,620.

Unterlast			
Algorithmus	vorhergesagte Antwortzeit (ns)	tatsächliche Antwortzeit (ns)	normalisierter Anomaliewert
WeightedForecaster (logarithmisch)	574,5162252	134	-0,622
WeightedForecaster (linear)	581,7500000		-0,626
WeightedForecaster (exponentiell)	593,7504106		-0,632
NaiveForecaster	545,0000000		-0,605
MovingAverageForecaster	571,8666667		-0,620

Abbildung 6.6. Ausgabe der Algorithmen während der Punktanomalie

Punktanomalie durch Überlast

Durch das Fenster aus Abbildung 6.2 ergeben sich die in Abbildung 6.7 abgebildeten vorausberechneten Antwortzeiten. Der alternative Vorhersagealgorithmus berechnet eine Antwortzeit von circa 561,18 ns mit der logarithmischen Gewichtung, eine Antwortzeit von annähernd 558,99 ns und mit der exponentiellen Gewichtung eine Antwortzeit von ungefähr 563,8 ns. Der NaiveForecaster berechnet eine Antwortzeit von 583 ns und der

MovingAverageForecaster eine Antwortzeit von 563,6 ns.
Die Anomalienerkennung berechnet mit der vorhergesagten Antwortzeit des WeightedForecasters und mit der tatsächlichen Antwortzeit von 2365 ns einen Anomaliewert von 0,616 bei Verwendung der logarithmischen Gewichtung, mit der linearen Gewichtung einen Anomaliewert von 0,618 und durch die exponentielle Gewichtung berechnet die Anomalienerkennung einen Anomaliewert von 0,615. Bei der vorausberechneten Antwortzeit des NaiveForecasters ergibt sich ein Anomaliewert von 0,604 und bei der Anwendung der Antwortzeit des MovingAverageForecasters berechnet sich ein Anomaliewert von 0,615.

Überlast			
Algorithmus	vorhergesagte Antwortzeit (ns)	tatsächliche Antwortzeit (ns)	normalisierter Anomaliewert
WeightedForecaster (logarithmisch)	561,1769197	2365	0,616
WeightedForecaster (linear)	558,9916667		0,618
WeightedForecaster (exponentiell)	563,8039988		0,615
NaiveForecaster	583,0000000		0,604
MovingAverageForecaster	563,6000000		0,615

Abbildung 6.7. Ausgabe der Algorithmen während der Punktanomalie

Kontextanomalie durch Unterlast

Unterlast			
Algorithmus	vorhergesagte Antwortzeit (ns)	tatsächliche Antwortzeit (ns)	normalisierter Anomaliewert
WeightedForecaster (logarithmisch)	829,3942974	200	-0,611
WeightedForecaster (linear)	906,7666667		-0,639
WeightedForecaster (exponentiell)	1047,8574086		-0,679
NaiveForecaster	1144,0000000		-0,702
MovingAverageForecaster	795,5333333		-0,598

Abbildung 6.8. Ausgabe der Algorithmen während der Kontextanomalie

Aufgrund der Daten des Fensters aus Abbildung 6.3 wurden die vorausberechneten Antwortzeiten in Abbildung 6.8 berechnet. Der WeightedForecaster berechnet eine Antwortzeit von 829,3942974 ns (logarithmische Gewichtung), $906,\overline{76}$ ns (lineare Gewichtung) beziehungsweise 1047,8574086 ns (exponentielle Gewichtung)voraus, während der NaiveForecaster eine Antwortzeit von 1144 ns und der MovingAverageForecaster eine Antwortzeit von $795,\overline{53}$ ns voraus berechnet. Die tatsächlich eingetretene Antwortzeit beträgt 200 ns.

Durch diese und die vorausberechneten Antwortzeiten des WeightedForecasters ergeben sich nach der Berechnung der Anomalienerkennung normalisierte Anomaliewerte von -0,611 mit der logarithmischen Gewichtung, -0,639 mit der linearen Gewichtung und -0,679 mit der exponentiellen Gewichtung. Bei der Verwendung des NaiveForecasters ergibt sich ein Anomaliewert von -0,702, während die Anomalienerkennung unter der Verwendung des MovingAverageForecasters einen Anomaliewert von -0,598 errechnet.

Kontextanomalie durch Überlast

Für das zweite Szenario der Kontextanomalie berechnet der WeightedForecaster mit der logarithmischen Gewichtung eine Antwortzeit von 819,7788145 ns voraus, während mit der linearen Gewichtung eine Antwortzeit von $739,9\overline{3}$ ns vorausberechnet wird. Bei Verwendung der exponentiellen Gewichtung berechnet der WeightedForecaster einen Wert von 604,7112193 ns. Die Anomalienerkennung berechnet mit der tatsächlichen Antwortzeit von 3440 ns und der vorhergesagten Antwortzeit des WeightedForecasters unter der Verwendung der logarithmischen Gewichtung einen normalisierten Anomaliewert von 0,615. Unter der Verwendung der linearen Gewichtung wird ein Wert von 0,646 berechnet und mit der exponentiellen Gewichtung ergibt sich ein normalisierter Anomaliewert von 0,701. Mit der vorausberechneten Antwortzeit des NaiveForecasters ergibt sich ein Anomaliewert von 0,739 und mit der Antwortzeit des MovingAverageForecasters berechnet die Anomalienerkennung einen normalisierten Wert von 0,602. Die Ergebnisse der Berechnung der Antwortzeiten sowie der normalisierten Anomaliewerte sind in Abbildung 6.9 dargestellt.

Überlast			
Algorithmus	vorhergesagte Antwortzeit (ns)	tatsächliche Antwortzeit (ns)	normalisierter Anomaliewert
WeightedForecaster (logarithmisch)	819,7788145	3440	0,615
WeightedForecaster (linear)	739,9333333		0,646
WeightedForecaster (exponentiell)	604,7112193		0,701
NaiveForecaster	517,0000000		0,739
MovingAverageForecaster	855,2666667		0,602

Abbildung 6.9. Ausgabe der Algorithmen während der Kontextanomalie

Kollektivanomalie

Die in Abbildung 6.10 abgebildete Tabelle gibt eine Übersicht über die vorausgesagten Antwortzeiten, die tatsächlichen Antwortzeiten sowie die Anomaliewerte über die Dauer von fünf Zeitstempeln der Kollektivanomalie sowie einen weiteren Zeitstempel, nachdem die Anomalie vorbei ist. Die Kollektivanomalie beginnt mit einer Antwortzeit von 134 ns. Daraus ergeben sich mit den vorausberechneten Antwortzeiten, welche zwischen 528 ns

und 549,75515 ns liegen, Anomaliewerte im Bereich von -0,595 und -0,608. Für die Berechnung des nächsten Anomaliewertes wird die tatsächlich eingetroffene Antwortzeit aus dem ersten Schritt als aktuellste Antwortzeit hinzugefügt. Dadurch ergeben sich vorhergesagte Antwortzeiten von 124 ns bis $514,1\overline{3}$ ns im zweiten Schritt. Die tatsächliche Antwortzeit beträgt 156 ns, wodurch sich Anomaliewerte in einem Bereich von -0,534 bis 0,114 ergeben. Die tatsächliche Antwortzeit von 156 ns ist nun die aktuellste Antwortzeit des betrachteten Zeitfensters und die Antwortzeit zum Zeitstempel 1 ist nicht mehr in dem Zeitfenster enthalten. Aufgrund dieser Änderung ergeben sich neue Werte: Die vorhergesagten Antwortzeiten liegen zwischen 156 ns und $487,\overline{3}$. Die tatsächliche Antwortzeit beträgt 123 ns, wodurch die Anomaliewerte zwischen -0,597 und -0,118 liegen.
Im vierten Schritt ist 123 ns die aktuellste Antwortzeit im Zeitfenster und die Antwortzeit mit dem Zeitstempel 2 ist nicht mehr in dem Zeitfenster enthalten. Die vorhergesagten Antwortzeiten liegen in diesem Schritt zwischen 123 ns und $461,0\overline{6}$ ns. Mit der tatsächlichen Antwortzeit von 134 ns ergeben sich Anomaliewerte zwischen -0,527 und 0,075 für diesen Schritt.
Der fünfte Schritt ist der letzte Zeitstempel zu dem die Kontextanomalie noch auftritt. Die aktuelle Antwortzeit beträgt 134 ns und die vorausberechneten Antwortzeiten der Algorithmen liegen zwischen 143 ns und 435,6 ns. Mit diesen Werten ergeben sich Anomaliewerte von -0,529 bis 0,032.
Einen Zeitstempel nach der Kollektivanomalie, also Schritt 6 in Abbildung 6.10, ist die tatsächliche Antwortzeit mit 578 ns wieder in dem Bereich, wie es vor dem Auftreten der Anomalie war. Die Algorithmen sagen Antwortzeiten für diesen Schritt von 134 ns bis $408,5\overline{3}$ ns voraus. Dadurch berechnet die Anomalienerkennung Anomaliewerte im Bereich von 0,172 bis 0,624.

Schritt 1			
Algorithmus	vorhergesagte Antwortzeit (ns)	tatsächliche Antwortzeit (ns)	normalisierter Anomaliewert
WeightedForecaster (logarithmisch)	543,4036625	124	-0,629
WeightedForecaster (linear)	546,2333333		-0,630
WeightedForecaster (exponentiell)	550,7810646		-0,632
NaiveForecaster	528,0000000		-0,620
MovingAverageForecaster	542,8666667		-0,628
Schritt 2			
Algorithmus	vorhergesagte Antwortzeit (ns)	tatsächliche Antwortzeit (ns)	normalisierter Anomaliewert
WeightedForecaster (logarithmisch)	509,4784734	156	-0,531
WeightedForecaster (linear)	493,8750000		-0,520
WeightedForecaster (exponentiell)	408,1918732		-0,447
NaiveForecaster	124,0000000		0,114
MovingAverageForecaster	514,1333333		-0,534
Schritt 3			
Algorithmus	vorhergesagte Antwortzeit (ns)	tatsächliche Antwortzeit (ns)	normalisierter Anomaliewert
WeightedForecaster (logarithmisch)	478,8184079	123	-0,591
WeightedForecaster (linear)	449,1083333		-0,570
WeightedForecaster (exponentiell)	323,8212049		-0,449
NaiveForecaster	156,0000000		-0,118
MovingAverageForecaster	487,3333333		-0,597
Schritt 4			
Algorithmus	vorhergesagte Antwortzeit (ns)	tatsächliche Antwortzeit (ns)	normalisierter Anomaliewert
WeightedForecaster (logarithmisch)	447,3206477	143	-0,516
WeightedForecaster (linear)	403,5666667		-0,477
WeightedForecaster (exponentiell)	256,5801963		-0,284
NaiveForecaster	123,0000000		0,075
MovingAverageForecaster	461,0666667		-0,527
Schritt 5			
Algorithmus	vorhergesagte Antwortzeit (ns)	tatsächliche Antwortzeit (ns)	normalisierter Anomaliewert
WeightedForecaster (logarithmisch)	417,5542601	134	-0,514
WeightedForecaster (linear)	363,8083333		-0,462
WeightedForecaster (exponentiell)	218,4286795		-0,240
NaiveForecaster	143,0000000		-0,032
MovingAverageForecaster	435,6000000		-0,529
Schritt 6			
Algorithmus	vorhergesagte Antwortzeit (ns)	tatsächliche Antwortzeit (ns)	normalisierter Anomaliewert
WeightedForecaster (logarithmisch)	387,2236259	578	0,198
WeightedForecaster (linear)	326,1083333		0,279
WeightedForecaster (exponentiell)	189,9760238		0,505
NaiveForecaster	134,0000000		0,624
MovingAverageForecaster	408,5333333		0,172

Abbildung 6.10. Ausgabe der Algorithmen während der Kollektivanomalie

6.6. Diskussion

In diesem Abschnitt werden die Ergebnisse der Szenarien aus dem vorherigen Abschnitt bewertet. Für die Bewertung werden die berechneten Anomaliewerte der Anomalienerkennung unter Mithilfe der verschiedenen Algorithmen miteinander verglichen.

Punktanomalie durch Unterlast

Die Punktanomalie in ersten Szenario wurde durch eine Unterlast ausgelöst. Die Anomalienerkennung hat mit jedem der Algorithmen die Anomalie erkannt. Das negative Vorzeichen der normalisierten Anomaliewerte sagt aus, dass es sich um eine Unterlast handelt. Die Anomalienerkennung stuft jede Anomalie als Warnung ein. Dabei wurde durch die Vorhersage des NaiveForecasters die Anomalie mit -0,605 am schwächsten eingestuft und durch die Vorhersage des WeightedForecasters mittels der exponentiellen Gewichtung die Anomalie mit -0,632 am stärksten eingestuft. Mit der logarithmischen Gewichtung liegt der Anomaliewert durch den WeightedForecaster mit -0,622 annähernd an dem Anomaliewert von -0,620, welcher mithilfe des MovingAverageForecasters berechnet wurde.

Punktanomalie durch Überlast

Im zweiten Szenario zur Punktanomalie wurde diese durch eine Überlast ausgelöst. Dies bedeutet, dass die Anomaliewerte allesamt positiv sein müssen. Da dies der Fall ist, wie in Abbildung 6.7 abgebildet, hat die Anomalienerkennung durch die Algorithmen erkannt, dass es eine Überlast gibt. Die Anomalienerkennung stuft die berechneten Anomaliewerte anschließend als Warnung ein, da die Anomaliewerte zwischen 0,5 und 0,7 liegen. Somit hat jeder Algorithmus die Überlast erkannt. Dabei hat die Anomalienerkennung mithilfe des NaiveForecasters den niedrigsten Anomaliewert von 0,604 berechnet und mit dem WeightedForecaster in der linearen Gewichtung mit 0,618 den höchsten Anomaliewert berechnet.

Kontextanomalie durch Unterlast

Im ersten Szenario der Kontextanomalie wurde den Algorithmen ein Fenster mit ansteigenden Antwortzeiten übergeben, um eine Anomalie durch Unterlast hervorzurufen. Die Ergebnisse dieser Unterlast aus Abbildung 6.8 zeigen, dass die Anomalienerkennung die aufgetretene Unterlast bei der Berechnung mit dem Ergebnis des WeightedForecaster und des MovingAverageForecaster als Warnung einstuft. Durch die Vorausberechnung des NaiveForecasters wird ein Anomaliewert von -0,702 berechnet, was durch die Anomalienerkennung als Fehler eingestuft wird. Der niedrigste Anomaliewert von -0,598 wurde durch die vorausberechnete Antwortzeit des MovingAverageForecasters berechnet. Die durch die Daten des WeightedForecasters berechneten Anomaliewerte liegen im Mittelfeld, wobei durch die logarithmische Gewichtung der niedrigste Anomaliewert von -0,611 im

direkten Vergleich der drei Gewichtungen berechnet wurde und durch die exponentielle Gewichtung der höchste Wert von -0,679.

Kontextanomalie durch Überlast

Das zweite Szenario zur Kontextanomalie hatte die Ergebnisse, die in der Tabelle in Abbildung 6.9 dargestellt sind, zur Folge. Dabei wird die Anomalie dreimal als Warnung und zweimal als Fehler eingestuft. Der Grund der Anomalie ist in allen Fällen eine Überlast. Durch die vorausberechnete Antwortzeit des MovingAverageForecasters ergibt sich ein normalisierter Anomaliewert von 0,602. Dieser Wert ist der niedrigste für dieses Szenario. Die Ergebnisse durch den WeightedForecaster liegen mit Anomaliewerten von 0,615 (logarithmische Gewichtung), 0,646 (lineare Gewichtung) und 0,701 (exponentielle Gewichtung) im Mittelfeld, wobei durch die exponentielle Gewichtung die Anomalie als Fehler eingestuft wird und bei den beiden anderen Gewichtungen als Warnung. Der höchste Anomaliewert berechnet die Anomalienerkennung bei der Verwendung der Antwortzeit des NaiveForecasters. Diese beträgt 0,739 und ist damit auch als Fehler eingestuft worden.

Kollektivanomalie

In Abbildung 6.11 sind die Anomaliewerte während der Kollektivanomalie aus Abbildung 6.10 dargestellt. Zu erkennen ist, dass die Anomalienerkennung durch die vorausberechneten Antwortzeiten der Algorithmen den Beginn der Kollektivanomalie erkennt, und dass es sich um eine Unterlast handelt. In jedem Fall stuft die Anomalienerkennung die Anomalie zu Beginn als Warnung ein.
Beim nächsten Zeitpunkt führt die Verwendung des NaiveForecasters sowie des WeightedForecasters mit der exponentiellen Gewichtung, dazu, dass die Kollektivanomalie nicht mehr als Warnung eingestuft wird, da der Anomaliewert die -0,5 überschritten hat. Mit dem MovingAverageForecaster sowie dem WeightedForecaster mit der logarithmischen und linearen Gewichtung, wird die Anomalie weiterhin als Warnung eingestuft. Dies ist auch zum nächsten Zeitpunkt so. Bei Zeitstempel 18 liegt der Anomaliewert mit -0,477 durch die Berechnung mittels des WeightedForecasters unter Verwendung der linearen Gewichtung über der Marke von -0,5, wodurch die Anomalie nicht mehr als solche eingestuft wird. Bei der Verwendung der logarithmischen Gewichtung des WeightedForecasters und des MovingAverageForecasters liegen die Anomaliewerte mit -0,516 und -0,527 noch im Bereich einer als Warnung eingestuften Anomalie. Zum letzten Zeitpunkt der Kollektivanomalie ordnet die Anomalienerkennung unter Verwendung des WeightedForecasters (logarithmische Gewichtung) und des MovingAverageForecasters die Anomalie weiterhin als Warnung ein, während unter der Verwendung aller anderen Algorithmen die Anomalie weiter nicht erkannt wird. Einen Zeitstempel weiter, zu der Zeit als die Anomalie vorüber ist, erkennt die Anomalienerkennung eine Anomalie, wo es keine mehr gibt. Dies geschieht unter der Verwendung des NaiveForecasters sowie des WeightedForecasters mit der exponentiellen Gewichtung.

Werden die anderen Algorithmen genutzt, wird keine vermeintliche Anomalie erkannt. Somit erkennen zwar alle Algorithmen den Beginn der Kollektivanomalie, jedoch nur zwei Algorithmen, der MovingAverageForecaster und der WeightedForecaster mit der logarithmischen Gewichtung, erkennen die Anomalie die gesamte Zeit über. Des Weiteren wird nach dem Ende der Kollektivanomalie durch die Verwendung des NaiveForecasters sowie des WeightedForecasters mit der exponentiellen Gewichtung irrtümlich eine Anomalie erkannt.

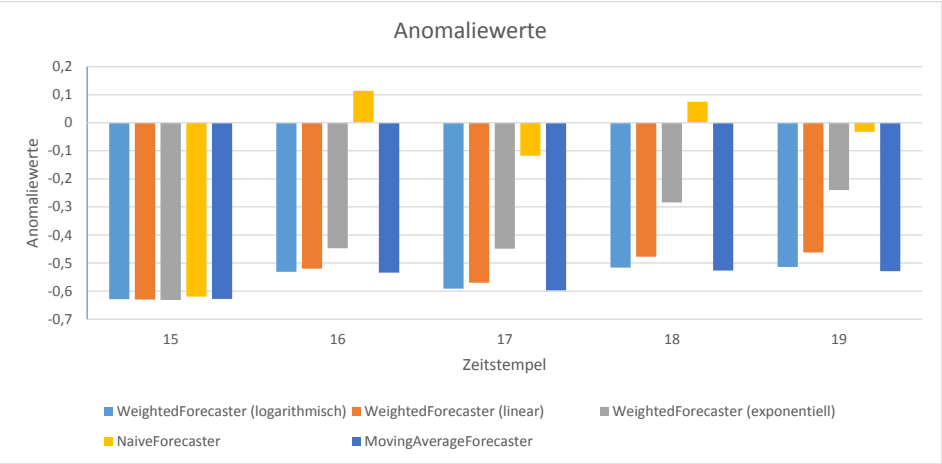

Abbildung 6.11. Anomaliewerte während der Kollektivanomalie

6.7. Einschränkung der Validität

Durch eine längere Dauer einer Kollektivanomalie, bei gleichbleibender Größe des Zeitfensters, würde das Ende der Kollektivanomalie auch nicht mehr mit der logarithmischen Gewichtung erkannt werden. Des Weiteren würde ein ständiger Wechsel von kleinen und großen Antwortzeiten dazu führen, dass am Anfang diese als Anomalie eingestuft werden würde, mit längerer Dauer dieses Wechsels jedoch nicht mehr.

Kapitel 7

Verwandte Arbeiten

Im Folgenden wird die in dieser Bachelorarbeit implementierte Anomalienerkennung zu anderen Arbeiten abgegrenzt.

7.1. ΘPADx

In der Masterarbeit über ΘPAD [Frotscher 2013] wird die Erkennung von Anomalien durch die Analyse von Antwortzeiten bereits thematisiert. Jedoch wurde dieses Tool nicht für ExplorViz entwickelt, sondern für das Monitoring-Framework Kieker. Des Weiteren erfolgt die Berechnung der Anomaliewerte auf unterschiedliche Arten: Während bei ΘPADx der Anomaliewert durch den euklidischen Abstand berechnet wird, benutzt die in dieser Bachelorarbeit implementierte Anomalienerkennung zur Berechnung der Anomaliewerte, die Subtraktion von tatsächlicher und vorausberechneter Antwortzeit. Weiter wird bei der Normalisierung in ΘPADx der Anomaliewert auf das Intervall von 0 und 1 skaliert und bei der Normalisierung für das Kontrollzentrum wird der Anomaliewert auf einen Wert zwischen -1 und 1 skaliert. Somit kann allein anhand eines Anomaliewertes bei ΘPADx nicht erkannt werden, ob es sich um eine Unterlast oder Überlast handelt.

7.2. ΘPAD

ΘPAD [Bielefeld 2012] wurde im Rahmen einer Diplomarbeit von Tillmann Carlos Bielefeld als Plug-In für das Monitoring-Framework Kieker entwickelt. Neben der Tatsache, dass die Anomaliewerte durch den euklidischen Abstand berechnet werden und anschließend auf das Intervall von 0 bis 1 normalisiert werden, wurden die Vorhersagealgorithmen nicht in Java, sondern in R programmiert. Die Programmiersprache wurde für statistisches Rechnen entwickelt. Zudem wurden andere Algorithmen zur Vorhersage implementiert. Dazu gehört unter anderem der SeasonForecaster, der die Antwortzeit vor 24 Stunden als Vorhersagewert ausgibt.

7.3. Analyse des Zeitverhaltens

Die in der Dissertation [Rohr 2014] von Matthias Rohr vorgestellte Anomalienerkennung verwendet den Ansatz, dass Anomalien Objekte sind, die ungewöhnlich sind oder in gewisser Weise im Widerspruch zu den meisten Objekten eines Datensatzes stehen [Rohr 2014, S. 26]. Dabei werden drei verschiedene Anomaliearten unterschieden [Rohr 2014, S. 26]: Atypische Ausführungen, fehlerhafte Ausführungen sowie Ausführungen aus einer anderen Klasse. Die Anomalienerkennung besteht in der Arbeit von Rohr aus drei Schritten. Zunächst werden Datensätze vorbereitet. Anschließend werden Anomaliealgorithmen initialisiert und zuletzt werden Anomaliewerte berechnet. Die Anomalienerkennung analysiert hierbei die tatsächlich eingetroffenen Antwortzeiten, ohne dabei eine Antwortzeit vorauszuberechnen. Die Algorithmen dienen zur Berechnung der Anomaliewerte unter der Verwendung von Dichte und Abstand von Punkten, um so Ausreißer, also Anomalien, zu lokalisieren.

Kapitel 8

Fazit und Ausblick

8.1. Fazit

Im Folgenden wird ein Fazit zu der Anomalienerkennung und zu dem alternativen Vorhersagealgorithmus gegeben.

Anomalienerkennung

Die Anomalienerkennung für das Kontrollzentrum erkennt Punkt-, Kontext- sowie Kollektivanomalien anhand von Antwortzeiten, die beim Aufruf einer Methode entstehen. Dabei wird ein Anomaliewert für jede aufgerufene Methode mithilfe der tatsächlich eingetroffenen Antwortzeit und einer vorausberechneten Antwortzeit ermittelt. Die Vorausberechnung einer Antwortzeit erfolgt durch die Anwendung eines von insgesamt drei Vorhersagealgorithmen. Dieser Algorithmus berechnet einen Vorhersagewert aufgrund von vergangenen Antwortzeiten. Nachdem der Anomaliewert für eine Methode berechnet wurde, wird dieser normalisiert, um einen Wert im Intervall zwischen -1 und 1 zu erhalten. Anschließend können durch den normalisierten Anomaliewert die entsprechenden Flags für eine Visualisierung einer Warnung oder eines Fehlers gesetzt werden.

Alternativer Vorhersagealgorithmus

Der alternative Vorhersagealgorithmus, WeightedForecaster genannt, ist ein Algorithmus, der aus vergangenen Antwortzeiten die nächste Antwortzeit vorausberechnet. Dieser Algorithmus wird in der Anomalienerkennung zur Vorausberechnung der nächsten Antwortzeit genutzt und stellt somit einen elementaren Bestandteil der Anomalienerkennung dar. Die Vorausberechnung geschieht über ein gewichtetes arithmetisches Mittel von vergangenen Antwortzeiten. Die Besonderheit an diesem Vorhersagealgorithmus ist die Auswahl von drei verschiedenen Gewichtungen. Zur Verfügung stehen eine logarithmische, lineare und exponentielle Gewichtung. Somit kann der Benutzer die Stärke der Gewichtung je nach Applikation, die ExplorViz überwacht, selbst festlegen und den Gegebenheiten anpassen. Die Evaluierung des WeightedForecasters hat gezeigt, dass durch den Algorithmus mit allen Gewichtungen, Punkt- und Kontextanomalien erkannt werden. Bei einer Kollektivanomalie kommt es sowohl auf die Dauer der Anomalie, als auch auf die gewählte Gewichtung an. Zwar wird durch jede der Gewichtungen der Beginn der Anomalie erkannt,

jedoch wird mit der logarithmischen Gewichtung die Anomalie über eine längere Dauer hinweg erkannt, als wie es mit den beiden anderen Gewichtungen der Fall ist. Aufgrund der exponentiellen Gewichtung wird, nach dem die Anomalie vorüber ist, vermeintlich eine weitere Anomalie erkannt.

8.2. Ausblick

Im Folgenden wird ein Ausblick gegeben, in welcher Form man die Anomalienerkennung und den alternativen Vorhersagealgorithmus verbessern und erweitern kann.

Anomalienerkennung

Um die Anomalienerkennung des Kontrollzentrums zu verbessern, kann für den Anomaliewert einer Applikation anstatt des Maximums der Anomaliewerte der Methoden dieser Applikation der Durchschnitt der Anomaliewerte der Methoden berechnet werden.
Des Weiteren können, um eine höhere Korrektheit der Vorausberechnung von Antwortzeiten zu ermöglichen, weitere Algorithmen, wie zum Beispiel der ARIMA-Algorithmus, implementiert werden. Ein weiterer Algorithmus, den man in die Anomalienerkennung integrieren kann, ist einer, der die Antwortzeiten der letzten 24 Stunden, der letzten Woche Tage oder des letzten Monats analysiert, um so eine Antwortzeit vorauszuberechnen.

Alternativer Vorhersagealgorithmus

Für eine bessere Vorausberechnung der Antwortzeit kann dem WeightedForecaster bei der Berechnung der gewichteten Antwortzeit aus dem übergebenen Fenster, eine negative Gewichtung gegeben werden, wenn zu dieser Zeit die tatsächlich eingetroffene Antwortzeit kleiner ist, als die vorausberechnete Antwortzeit. Somit würden bei dem Algorithmus neben den vergangen Antwortzeiten auch die zu diesen Antwortzeiten passenden vergangenen, vorausberechneten Antwortzeiten in die Berechnung des Vorhersagewertes mit eingehen. Die Auswahl der Gewichte kann man so erweitern, dass der Benutzer die Gewichtung für jede Methode individuell anpassen kann und eigene Formeln für die Gewichtung in der Konfiguration hinterlegen kann.

Literaturverzeichnis

[Bielefeld 2012] T. C. Bielefeld. Online Performance Anomaly Detection. Diplomarbeit. Kiel University, 2012. (Siehe Seiten 6 und 41)

[Ehlers u. a. 2011] J. Ehlers, A. van Hoorn, J. Waller und W. Hasselbring. Self-adaptive software system monitoring for performance anomaly localization. In: *Proceedings of the 8th IEEE/ACM International Conference on Autonomic Computing (ICAC '11)*. ACM, 2011, Seiten 197–200. (Siehe Seite 3)

[*ExplorViz Website*] ExplorViz. ExplorViz Website. Letzter Zugriff am 18.03.2014. URL: http://www.explorviz.net. (Siehe Seiten 5, 6)

[Fittkau u. a. 2013] F. Fittkau, J. Waller, C. Wulf und W. Hasselbring. Live Trace Visualization for Comprehending Large Software Landscapes: The ExplorViz Approach. *Proceedings of the 1st IEEE International Working Conference on Software Visualization (VISSOFT 2013)* (2013). (Siehe Seite 5)

[Frotscher 2013] T. Frotscher. Architecture-Based Multivariate Anomaly Detection for Software Systems. Masterarbeit. Kiel University, 2013. (Siehe Seiten 1, 3, 6 und 41)

[*Kieker Website*] Kieker. Kieker Website. Letzter Zugriff am 29.11.2014. URL: http://kieker-monitoring.net/. (Siehe Seite 1)

[*OpenStack Website*] OpenStack. OpenStack Website. Letzter Zugriff am 29.11.2014. URL: http://www.openstack.org. (Siehe Seite 7)

[Rohr 2014] M. Rohr. Workload-sensitive Timing Behavior Analysis for Fault Localization in Software Systems. Dissertation. Kiel University, 2014. (Siehe Seite 42)

[Rohr u. a. 2007] M. Rohr, S. Giesecke und W. Hasselbring. Timing behavior anomaly detection in enterprise information systems. In: *Proceedings of Ninth International Conference on Enterprise Information Systems (ICEIS'07)*. INSTICC Press, 2007, Seiten 494–497. (Siehe Seite 3)

[Van Hoorn u. a. 2012] A. van Hoorn, J. Waller und W. Hasselbring. Kieker: a framework for application performance monitoring and dynamic software analysis. In: *Proceedings of the 3rd joint ACM/SPEC International Conference on Performance Engineering (ICPE 2012)*. ACM, 2012, Seiten 247–248. (Siehe Seite 1)

Daten auf DVD

Folgende Daten befinden sich auf der angehängten DVD:

- Diese Arbeit als PDF-Dokument
- Das Kontrollzentrum als ausführbares Java-Projekt