

AutoGlobe: Automatische Administration von dienstbasierten Datenbankanwendungen

Daniel Gmach, Stefan Seltzsam, Martin Wimmer, Alfons Kemper

Technische Universität München – Lehrstuhl für Informatik III
85748 Garching bei München
(gmach|seltzsam|wimmerma|kemper)@in.tum.de

Abstract: Derzeit lässt sich ein Trend weg von monolithischen Systemen hin zu Service Oriented Architectures (SOAs) beobachten. Dieser Paradigmenwechsel erfordert neue Administrationstechniken, um die auf SOAs basierenden verteilten Datenbankanwendungen zuverlässig und kostengünstig betreiben zu können. Zu diesem Zweck entwickeln wir neue Selbstadministrationskonzepte. Die Grundlage hierfür bilden die Virtualisierung von Hardware und Diensten, sowie ein kontinuierliches Monitoring. Dadurch ist es möglich, die Verteilung der Dienste auf die zur Verfügung stehende Hardware durch statische und dynamische Allokationstechniken zu optimieren. Statische Allokationsalgorithmen liefern eine optimierte a priori Verteilung der Dienste auf die Hardware. Dazu werden Dienste mit komplementären Ressourcenanforderungen möglichst gemeinsam auf einem Rechner ausgeführt. Eine rein statische Optimierung kann allerdings nicht zeitnah auf unvorhersagbare Ereignisse, wie etwa Überlast- oder Fehlersituationen, reagieren. Deshalb setzen wir zusätzlich eine auf Fuzzy-Logik basierende Kontrollkomponente ein, die zur Laufzeit dynamisch Anpassungen der Dienstallokation vornimmt. Beispielsweise werden abgestürzte Dienste neu gestartet und Überlastsituationen durch Hinzunahme weiterer Instanzen oder den Umzug einer Instanz auf einen leistungsfähigeren Rechner behoben. Die vorgestellten Technologien stellen damit einen ersten Schritt in Richtung eines durchgängigen Quality of Service-Managements (QoS-Management) in einer derartigen Infrastruktur dar. AutoGlobe ist die prototypische Umsetzung der in diesem Beitrag beschriebenen Konzepte für eine adaptive Infrastruktur, die sich durch Selbstkonfiguration, Selbstoptimierung und eigenständige Fehlerbehebung auszeichnet.

1 Einleitung

Die Architektur der Systeme, auf denen Datenbankanwendungen wie Enterprise Resource Planning-Systeme (ERP) typischerweise ausgeführt werden, unterzieht sich derzeit einem drastischen Wandel. Waren bisher monolithische Softwarearchitekturen vorherrschend, geht der Trend heutzutage zu feingranularen Diensten, die in Service Oriented Architecture-Umgebungen ausgeführt werden. Beobachten lässt sich dieser Wandel etwa anhand der Entwicklung von SAP R/3 über mySAP hin zu SAP NetWeaver. Der Einsatz von SOAs hat allerdings tiefgreifende Auswirkungen auf Administrations- und Optimierungstechniken für Anwendungen. Deren feingranulare Zerlegung in eine Vielzahl interagierender Dienste erlaubt insbesondere den

Einsatz von Rechnerclustern und eine flexible Dienst-Rechner-Allokation. Durch die Virtualisierung der Dienste, d.h. der Einführung einer dynamischen Dienst-Rechner-Zuweisung, können Anpassungen dieser Allokation zur Laufzeit durchgeführt werden, um etwa lokale Leistungengpässe und Fehlerzustände, wie Rechner- oder Dienstaussfälle, zu kompensieren. Der Zuwachs an Flexibilität geht allerdings einher mit einer signifikanten Steigerung der Systemkomplexität, der nur durch eine weitgehende Automatisierung der Administration begegnet werden kann. Aus diesem Grund haben wir eine Kontrollkomponente zur kontinuierlichen Allokations- und damit Leistungsoptimierung von Anwendungen auf der Basis von SOAs entwickelt. Diese Softwarekontrollkomponente bildet den Kern unserer adaptiven Infrastruktur AutoGlobe, die sich durch Selbstkonfiguration, Selbstoptimierung und eigenständige Fehlerbehebung auszeichnet. Sie sorgt für eine zuverlässige Ausführung der Dienste und eine möglichst effiziente Auslastung der zur Verfügung stehenden Ressourcen. Dadurch wird zum einen die Komplexität des Gesamtsystems handhabbar. Zum anderen wird eine Reduktion der Gesamtbetriebskosten (Total Cost of Ownership, TCO) erreicht, da bislang zeitweise ungenutzte Ressourcen¹, wie sie typischerweise in unoptimierten Systemen auftreten, effizient genutzt werden. In vielen Fällen können dadurch andernfalls notwendige Neuanschaffungen entfallen, da mit der vorhandenen Hardware ein höherer Durchsatz erzielt werden kann.

Unser Ansatz zur Optimierung der Dienst-Rechner-Allokation basiert auf einer statischen und einer dynamischen Phase. Das Ergebnis der statischen Allokationsoptimierung, die auf der Grundlage von verdichteten historischen Lastdaten durchgeführt wird, ist eine längerfristig ausgelegte Verteilung der Dienste auf die zur Verfügung stehenden Rechner. Dazu werden Dienste mit komplementären Ressourcenanforderungen möglichst gemeinsam auf einem Rechner ausgeführt. Eine rein statische Optimierung kann allerdings nicht zeitnah auf unvorhersagbare Ereignisse, wie etwa Überlast- oder Fehlersituationen, reagieren. Deshalb setzen wir zusätzlich eine auf Fuzzy-Logik basierende Kontrollkomponente ein, die zur Laufzeit dynamisch Anpassungen der Allokation vornehmen kann. Zur Entscheidungsfindung stehen der Kontrollkomponente dabei die aktuellen Lastdaten der Dienste und Rechner zur Verfügung. So werden beispielsweise abgestürzte Dienste neu gestartet und Überlastsituationen durch Hinzunahme weiterer Instanzen auf bisher wenig belasteten Rechnern oder den Umzug einer Instanz auf einen leistungsfähigeren Rechner behoben. Wir benutzen einen Fuzzy Controller wegen seiner Robustheit, Anpassbarkeit und der intuitiven Spezifizierbarkeit seiner Regelbasis. Durch den Einsatz der beschriebenen Kontroll- und Monitoringmechanismen wird eine gleichmäßige Auslastung der Hardware sowie eine zuverlässige Ausführung der Dienste erreicht. Somit bilden sie die Grundlage für ein durchgängiges Quality of Service-Management [Wei99] für eine derartige Infrastruktur.

Die Anforderungen und Möglichkeiten von Diensten und Rechnern werden mittels einer deklarativen XML-Sprache beschrieben. Diese basiert auf einer frühen Version der XML-Sprache zur Beschreibung von Diensten und Rechnern der Projektgruppe

¹Ein Großteil der Rechner, auf denen betriebswirtschaftliche Anwendungen ausgeführt werden, haben erfahrungsgemäß eine geringere Durchschnittsauslastung als 40%.

Scheduling and Resource Management (siehe [GGFb]) des Global Grid Forums. Damit können z.B. die maximale und minimale Anzahl der Instanzen eines Dienstes festgelegt, die Leistung der Rechner untereinander in Beziehung gesetzt oder die Regeln des Fuzzy Controllers spezifiziert werden.

Dieser Beitrag ist wie folgt strukturiert: In Abschnitt 2 wird die Architektur von AutoGlobe vorgestellt, die auf unserer ServiceGlobe-Plattform zur rechnerunabhängigen Ausführung von Web Services basiert. In Abschnitt 3 beschreiben wir die dynamische und in Abschnitt 4 die statische Optimierung der Dienst-Rechner-Allokation. In Abschnitt 5 stellen wir verwandte Arbeiten vor, ehe wir in Abschnitt 6 eine Zusammenfassung und einen Ausblick liefern.

2 AutoGlobe: Eine selbstadministrierende IT-Infrastruktur

AutoGlobe setzt auf unserer Web Service-Plattform ServiceGlobe [KSK03] auf. ServiceGlobe ist vollständig in Java Version 2 implementiert und basiert auf Web Service-Standards [KL04, ACKM03] wie SOAP, UDDI und WSDL. ServiceGlobe unterstützt Standardfunktionalitäten einer Dienstplattform wie ein Transaktions- und ein Sicherheitssystem [SBK01] und zeichnet sich durch die Unterstützung von mobilem Code aus. Die Infrastruktur ermöglicht somit die Ausführung von Diensten auf beliebigen Rechnern zur Laufzeit. AutoGlobe ergänzt diesen Ansatz durch eine Kontrollkomponente zur zuverlässigen Ausführung von Diensten und zur automatischen Optimierung der Dienst-Rechner-Allokation.

Voraussetzung für dynamische Änderungen der Allokation ist die Virtualisierung von Diensten. Dazu wird jedem Dienst eine individuelle IP-Adresse zugewiesen, die immer an die physikalische Netzwerkkarte (engl. NIC) des ausführenden Rechners gebunden ist. Wird der Dienst von einem Rechner auf einen anderen umgezogen, so wird die IP-Adresse von der NIC des bisherigen Rechners gelöst und anschließend an die NIC des neuen Rechners gebunden.

AutoGlobe ist für den Einsatz auf heterogenen, verteilten und flexiblen Systemen mit zentraler Datenhaltung – bereitgestellt etwa über ein Network Attached Storage (NAS) oder ein Storage Area Network (SAN) – konzipiert. Bei dieser Art der Datenhaltung können Dienste auf beliebigen Rechnern ausgeführt werden, selbst wenn sie Zugriff auf persistente Daten benötigen, die nicht in einer Datenbank gespeichert sind. Blade-Server-Systeme sind moderne Vertreter dieser Systemarchitektur. Sie zeichnen sich durch im Vergleich zu traditionellen Mainframe-Architekturen geringere Anschaffungskosten und geringere Administrationskosten als bei herkömmlichen Clustern aus. Zudem können Rechen- und Speicherkapazität unabhängig voneinander flexibel an den jeweiligen Bedarf angepasst werden. Ersteres lässt sich durch die Anzahl der Blades, letzteres durch die Anzahl bzw. den Ausbau der SANs/NASs-Lösung steuern. Ein derartiges System dient als Testumgebung für AutoGlobe. Virtualisierte Dienste auf einem Blade-Server-System sind auch Grundlage kommerzieller Systeme, wie beispielsweise FlexFrame [Fle].

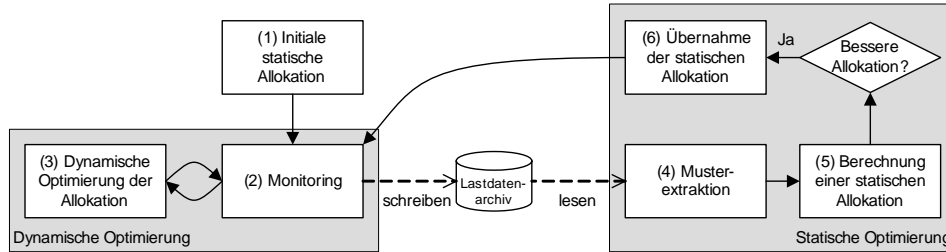


Abbildung 1: Zusammenspiel statischer und dynamischer Allokationsoptimierung

2.1 Statische und Dynamische Allokation

Abbildung 1 zeigt das grundlegende Zusammenspiel der statischen und der dynamischen Optimierung der Dienst-Rechner-Allokation. Die dynamische Optimierung sorgt für lokale Verbesserungen der Allokation während der Laufzeit, um beispielsweise Überlast- und Fehlersituationen zeitnah zu beheben. Die statische Optimierung wird eingesetzt, um eine globale, längerfristig ausgelegte neue Allokation zu berechnen, die dann wiederum als Grundlage für die dynamische Optimierung dient. Die Übernahme einer statischen Allokation bedingt eventuell eine aufwändigere Reallokation der Dienste und damit eine kurzfristige Unterbrechung der Ausführung von Teilen des Systems. Da dies aufgrund des Aufwands in der Regel nur in größeren Zeitabständen durchgeführt werden kann, wird die kontinuierliche dynamische Optimierung in Kombination dazu durchgeführt, um zeitnah auf unvorhersehbare Probleme reagieren zu können.

Die Inbetriebnahme des Systems erfolgt basierend auf einer initialen statischen Allokation (1). Diese kann von einem Administrator vorgegeben werden, oder auch automatisch bestimmt werden. Letzteres ist dann möglich, wenn der Ressourcenbedarf der auszuführenden Dienste beispielsweise aufgrund von Administrator- oder Entwicklervorgaben bekannt ist. Auf der Grundlage der aktuellen Allokation wird ein Monitoring der Dienste und Rechner durchgeführt (2). Die dabei gesammelten Informationen werden einerseits zur kontinuierlichen dynamischen Optimierung (3) verwendet. Andererseits werden die Daten in verdichteter Form im Lastdatenarchiv gespeichert und bilden die Grundlage für die statische Optimierung (4-6).

Die dynamische Optimierung (3) basiert im Wesentlichen auf einem Fuzzy Controller, der seine Entscheidungen regelbasiert trifft. Mit diesem Controller ist es möglich, Ausnahmesituationen, wie etwa Überlast- oder Fehlersituationen, zu erkennen und automatisch zu beheben.

Die statische Optimierung berücksichtigt zyklische Lastmuster der Dienste. Dazu werden die archivierten Lastdaten regelmäßig ausgewertet und charakteristische Lastmuster extrahiert (4). Basierend auf diesen Mustern werden Dienste mit komplementären Ressourcenanforderungen ermittelt und eine optimierte Allokation berechnet (5). Beispielsweise werden OLAP-Prozesse, die hauptsächlich nachts Last verursachen, mit überwiegend tagesaktiven OLTP-Diensten kombiniert. Die neu berechnete Dienst-Rechner-Allokation wird danach mit der bestehenden verglichen

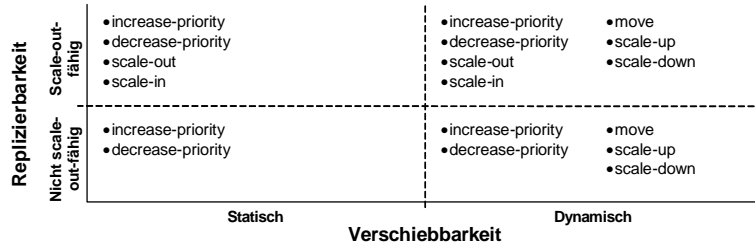


Abbildung 2: Einteilung der Dienste

und die zu erwartende Verbesserung des Systemverhaltens wie auch die entstehenden Kosten für ihre Umsetzung abgeschätzt. Bei geeignetem Kosten-Nutzen-Verhältnis wird die neu berechnete statische Allokation umgesetzt (6) und dient damit als neue Ausgangsbasis für die weitere kontinuierliche dynamische Optimierung.

2.2 Charakterisierung der Dienste

Dienste werden hinsichtlich ihrer Verschiebbarkeit und Replizierbarkeit (scale-out-Fähigkeit) klassifiziert. Je nach Kategorie lassen sich verschiedene adaptive Operationen (vgl. Tabelle 1) auf die Dienste anwenden (siehe Abbildung 2). Im Hinblick auf Verschiebbarkeit werden statische und dynamische Dienste unterschieden. Statische Dienste sind nach einer initialen Allokation einem Rechner fest zugeteilt, wohingegen dynamische Dienste einen Umzug auf einen anderen Rechner (move, scale-down und scale-up) zur Laufzeit unterstützen. Die Replizierbarkeit eines Dienstes gibt an, ob zur Laufzeit neue Instanzen erzeugt bzw. bereits laufende beendet werden können. Mehrinstanzfähige Dienste, deren Anzahl an Instanzen zur Laufzeit fest ist, werden als mehrere statische, nicht scale-out-fähige Dienste modelliert. Je mehr Aktionen ein Dienst unterstützt, desto flexibler kann der Fuzzy Controller in Bezug auf Ausnahmesituationen reagieren.

3 Dynamische Allokationsoptimierung

AutoGlobes Kontrollkomponente (Controller) analysiert kontinuierlich den Zustand des Systems basierend auf den Monitoringdaten und behebt Ausnahmesituationen

Aktion	Beschreibung
start	Starten eines Dienstes
stop	Stoppen eines Dienstes
scale-in	Stoppen einer Dienstinstanz
scale-out	Starten einer Dienstinstanz
scale-up	Umziehen einer Dienstinstanz auf einen stärkeren Rechner
scale-down	Umziehen einer Dienstinstanz auf einen schwächeren Rechner
move	Umziehen einer Dienstinstanz auf einen gleich starken Rechner
increase-priority	Erhöhen der Priorität eines Dienstes
reduce-priority	Senken der Priorität eines Dienstes

Tabelle 1: Übersicht über die Aktionen des Fuzzy Controllers

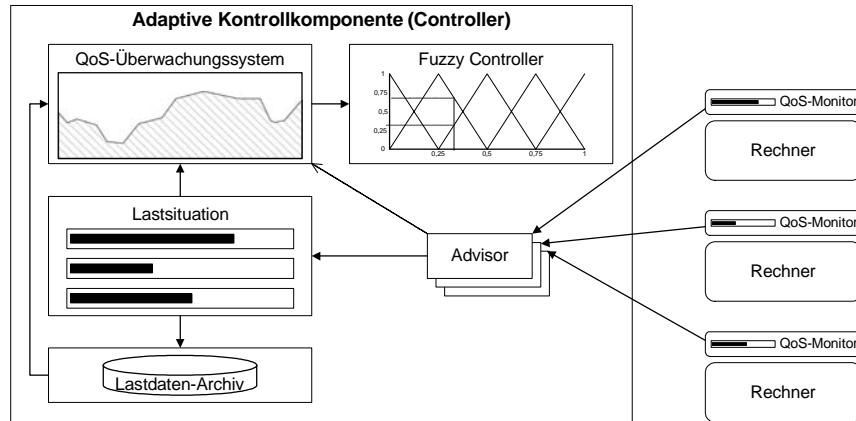


Abbildung 3: Architektur des Controllers

automatisch. Der Controller ist selbst ein Dienst und fügt sich damit nahtlos in die AutoGlobe-Architektur ein.

3.1 Architektur des Controllers

Abbildung 3 zeigt die Architektur der Kontrollkomponente, die für die dynamische Allokationsoptimierung zuständig ist. Für jeden Rechner und jeden Dienst² ist jeweils ein dedizierter QoS-Monitor zuständig. Jeder Monitor ermittelt die relevanten Daten eines Rechners bzw. eines Dienstes und sendet diese an den zugehörigen Advisor. Überwacht werden beispielsweise die Auslastung eines Rechners und die Antwortzeiten eines Dienstes. Durch diese Daten hat der Controller stets einen aktuellen Überblick über die Lastsituation im System und kann die QoS-Parameter überwachen. Überschreitet ein überwachter Parameter (z.B. CPU-Last) eines Rechners bzw. eines Dienstes eine einstellbare Schranke, liegt eine Überlastsituation vor. Solche Überlast- und auch Fehlersituationen werden von den Advisors gemeldet und die entsprechenden Lastdaten werden dann für eine bestimmte Dauer vom QoS-Überwachungssystem beobachtet. Dadurch können kurze Lastspitzen, die in realen Systemen häufig auftreten, aber keine kritischen Situationen darstellen, ausgefiltert werden. Im Falle einer längerfristig andauernden Überlastsituation wird der Fuzzy Controller angestoßen. Dieser wählt basierend auf der aktuellen Lastsituation und seiner Regelbasis eine Aktion aus, um der Überlastsituation entgegen zu wirken. Falls beispielsweise eine CPU-Überlast eines Rechners festgestellt wird, kann der Controller Dienste von dem überlasteten Rechner zu anderen im Moment nicht ausgelasteten Rechnern umziehen. Auch bei Unterlastsituationen schreitet der Controller (bei entsprechender Regelbasis) ein. Da Unterlastsituationen konzeptuell wie Überlastsituationen behandelt werden, wird im Weiteren nicht näher darauf eingegangen. Fehlersituationen, wie beispielsweise der Absturz eines

²Abbildung 3 zeigt nur die Monitore und Advisors zur Überwachung der Rechner. Aus Übersichtlichkeitsgründen werden Dienste, die auf den Rechnern laufen, sowie die zugehörigen Monitore und Advisors weggelassen.

Dienstes, werden vom Controller z.B. durch einen Neustart behoben. Das Lastdatenarchiv speichert verdichtete historische Daten. Im Folgenden beschreiben wir die einzelnen Module genauer:

QoS-Monitore und Advisoren. Jeder Rechner und jeder Dienst innerhalb der AutoGlobe-Umgebung wird von einem dedizierten QoS-Monitor überwacht, der die gesammelten Daten an den zuständigen Advisor des Controllers schickt. Monitore sind spezielle Dienste zur Überwachung des Ressourcenverbrauchs von Rechnern bzw. zur Überwachung des Ressourcenverbrauchs und der Quality of Service-Parameter von Diensten.

QoS-Überwachungssystem. Diese Komponente wird eingesetzt, um tatsächliche Überlastsituationen von kurzen, harmlosen Lastspitzen zu unterscheiden. Dadurch werden Überreaktionen des Controllers verhindert und ein ruhiges, stabiles System erreicht. Falls ein Lastwert oder ein QoS-Parameter eine einstellbare Schranke überschreitet, erfolgt für einen festgelegten Zeitraum (*watchtime*) die Überwachung der entsprechenden Daten. Nur falls das arithmetische Mittel der Daten, die während dieses Beobachtungszeitraums erfasst werden, oberhalb der vorgegebenen Schranke liegt, wird die drohende QoS-Verletzung an den Fuzzy Controller gemeldet.

Fuzzy Controller. Der Fuzzy Controller ermittelt basierend auf einem Regelsatz und der aktuellen Lastsituation eine geeignete Aktion, um einer Überlastsituation entgegen zu wirken. Dazu wird die Anwendbarkeit aller möglichen Aktionen (siehe Tabelle 1) bestimmt und die beste ausgewählt. Falls die Aktion (z.B. move) einen Zielrechner zur Durchführung erfordert, bewertet der Fuzzy Controller die möglichen Rechner. Ausgeführt wird dann die Aktion mit der höchsten Anwendbarkeit mit dem geeignetsten Rechner als Zielrechner. Finden sich mehrere Aktionen bzw. Rechner mit demselben Wert wird zufällig aus diesen ausgewählt.

Lastdatenarchiv. Das Lastdatenarchiv speichert verdichtete Lastdaten bzw. Werte von QoS-Parametern. Diese Daten werden zur Berechnung der Durchschnittslast während der *watchtime* und als Eingabedaten für den Fuzzy Controller verwendet. Außerdem werden sie zur Bestimmung dienstspezifischer Lastmuster herangezogen, die die Grundlage für statische Allokationsoptimierungen darstellen.

3.2 Grundlagen eines Fuzzy Controllers

Fuzzy Controller sind spezielle Expertensysteme, die auf Fuzzy-Logik aufbauen [KY94]. Sie werden vor allem zur Regelung komplexer dynamischer Systeme verwendet, für die die Beschreibung des Systemverhaltens durch exakte mathematische Modelle schwierig oder gar unmöglich ist. Statt eines mathematischen Modells verwenden Fuzzy Controller das Wissen von menschlichen Experten in Form von linguistischen Variablen und Regeln. Eine Alternative zu Fuzzy Controllern stellen neuronale Netze dar. Die Trainingsphase derartiger Controller gestaltet sich aber sehr schwierig und man kann die erlernten Verhaltensweisen als Administrator nur sehr schwer beeinflussen.

Fuzzy-Logik ist die Theorie der unscharfen Mengen, die erstmals von Lotfi Zadeh

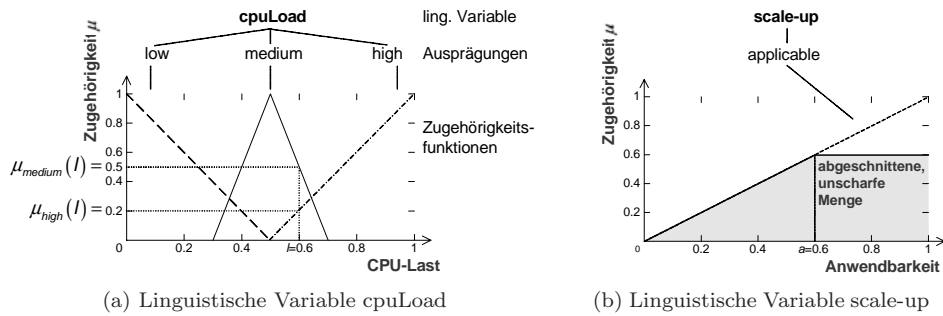


Abbildung 4: Zugehörigkeitsfunktionen für linguistische Variablen

im Jahr 1965 beschrieben wurde [Zad65]. Die Zugehörigkeit von Elementen zu einer unscharfen Menge liegt zwischen 0 und 1 und wird mit einer Zugehörigkeitsfunktion μ festgelegt. Sei X eine normale Menge, dann ist

$$A = \{(x, \mu_A(x)) \mid x \in X\} \text{ mit } \mu_A : X \rightarrow [0, 1]$$

eine Fuzzy-Menge bzw. unscharfe Menge auf X . Die Zugehörigkeitsfunktion μ_A weist jedem Element von X eine reelle Zahl in $[0, 1]$ zu. Eine größere Zahl bedeutet dabei einen höheren Zugehörigkeitsgrad.

Linguistische Variablen sind Variablen, deren Zustände unscharfe Mengen sind. Die Zustände repräsentieren Ausprägungen wie z.B. low, medium oder high. Eine linguistische Variable wird durch ihren Namen, eine Menge von Ausprägungen und eine Zugehörigkeitsfunktion pro Ausprägung definiert. Ein Beispiel hierfür ist die linguistische Variable cpuLoad, die in Abbildung 4(a) dargestellt ist. Die Abbildung zeigt die drei Ausprägungen low, medium und high und deren Zugehörigkeitsfunktionen.

Abbildung 5 zeigt die Architektur eines Fuzzy Controllers. Die adaptive Infrastruktur wird überwacht (1). Treten Ausnahmesituationen auf, stößt der Controller die Fuzzy-Auswertung an. In der Fuzzifizierungsphase (2) werden die Messdaten in unscharfe Mengen, die Eingabevariablen, überführt. Anschließend erfolgt mit ihnen die Auswertung der Regelbasis (3). In der abschließenden Defuzzifizierungsphase werden die ermittelten unscharfen Ergebnisse in einen Vektor bestehend aus scharfen Werten umgewandelt (4). Im Folgenden erklären wir die Arbeitsweise des Fuzzy Controllers anhand eines Beispiels zur Regulierung von Überlastsituationen.

In der Fuzzifizierungsphase werden die scharfen Messwerte (hier die CPU-Last eines Rechners) in die entsprechenden linguistischen Eingabevariablen (cpuLoad) überführt, indem die Zugehörigkeitsgrade der einzelnen Ausprägungen mittels der Zugehörigkeitsfunktionen bestimmt werden. So ist beispielsweise ein Rechner mit der gemessenen CPU-Last $l = 0.6$ zu 0.5 mittel (medium) und zu 0.2 stark (high) ausgelastet (siehe Abbildung 4(a)).

In der Inferenzphase wird die Regelbasis basierend auf den unscharfen Eingabevariablen ausgewertet. Die Form der Regeln wird exemplarisch anhand zweier einfacher Beispielregeln erklärt. Dabei sind cpuLoad und performanceIndex Eingabe-

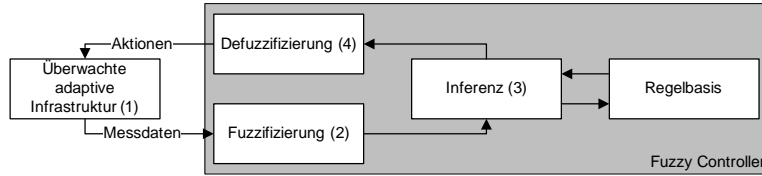


Abbildung 5: Architektur des Fuzzy Controllers [KY94]

scale-up und scale-out Ausgabevariablen. Der performanceIndex gibt die relative Leistungsfähigkeit von Rechnern an. Ein höherer Index bedeutet einen leistungsstärkeren Rechner.

IF (cpuLoad **IS** high **AND** (performanceIndex **IS** low **OR** performanceIndex **IS** medium)) **THEN** scale-up **IS** applicable

IF (cpuLoad **IS** high **AND** performanceIndex **IS** high) **THEN** scale-out **IS** applicable

Die erste Regel sagt aus, dass es sinnvoll ist, einen Dienst auf einen leistungsstärkeren Rechner umzuziehen, falls eine hohe Rechenauslastung auf dem ursprünglichen Rechner vorliegt und dessen performanceIndex low oder medium ist. Das Hinzufügen einer zusätzlichen Instanz ist der zweiten Regel zufolge dann sinnvoll, falls der Dienst ohnehin bereits auf einem sehr leistungsfähigen aber dennoch stark ausgelasteten Rechner läuft.

Konjunktionen von linguistischen Variablen in Vorbedingungen von Regeln werden mittels der Minimumfunktion verknüpft, Disjunktionen mit der Maximumfunktion. Als Beispiel sei die cpuLoad $l = 0.9$, dann sind die Zugehörigkeiten der Ausprägungen $\mu_{low} = 0$, $\mu_{medium} = 0$ und $\mu_{high} = 0.8$. Bei einem gegebenen performanceIndex $i = 5$ nehmen wir an, dass die Zugehörigkeitsgrade $\mu_{low} = 0$, $\mu_{medium} = 0.6$ und $\mu_{high} = 0.3$ sind. Dann ergibt sich für die Vorbedingung der ersten Regel $\min(0.8, \max(0.6, 0.3)) = 0.6$ und für die der zweiten $\min(0.8, 0.3) = 0.3$.

Während in der klassischen Logik die Schlussfolgerung wahr ist, wenn die Vorbedingung wahr ist, existieren für die Berechnung einer Fuzzy-Schlussfolgerung in der Literatur unterschiedliche Ansätze. Wir verwenden die bewährte Max-Min-Inferenzfunktion. Mit dieser Funktion wird die unscharfe Ergebnismenge in Höhe der Zugehörigkeit der Vorbedingung abgeschnitten. Umfasst die Regelbasis mehrere Regeln, die dieselbe Ausgabevariable betreffen, werden alle Ergebnisse mittels der Maximumfunktion verknüpft. Die daraus resultierenden, unscharfen Mengen sind das Ergebnis der Inferenzphase, siehe Abbildung 4(b).

In der Defuzzifizierungsphase wird aus den in der Inferenzphase ermittelten unscharfen Mengen ein Vektor scharfer Werte berechnet. Hierfür verwenden wir eine Maximummethode, die den linken Randpunkt des Maximumbereichs wählt. In unserem Beispiel ist dies der scharfe Wert $a = 0.6$, d.h. die Aktion scale-up ist zu 0.6 anwendbar (siehe Abbildung 4(b)). Analog ist die Aktion scale-out zu 0.3 anwendbar. Folglich wird der Controller die Aktion scale-up ausführen.

3.3 Fuzzy Controller zur Lastbalancierung

AutoGlobes Fuzzy Controller-Modul setzt sich aus zwei Fuzzy Controllern zusammen. Aufgabe des einen ist es, geeignete Aktionen bei Eintreten von Ausnahme-

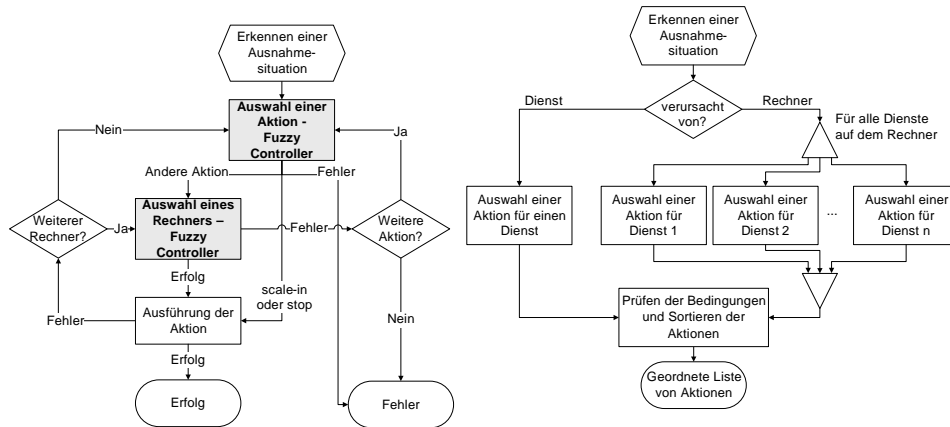


Abbildung 6: Ablaufdiagramm des Fuzzy Controller Moduls
Abbildung 7: Ablaufdiagramm für die Aktionsauswahl

situationen zu bestimmen. Falls die jeweilige Aktion einen Zielrechner erfordert, wird dieser mittels des zweiten Controllers ermittelt. Abbildung 6 zeigt die Interaktion der beiden Fuzzy Controller *Auswahl einer Aktion* und *Auswahl eines Rechners*. Nach dem Ausführen einer Aktion werden die davon betroffenen Dienste und Rechner für eine bestimmte Zeit von weiteren Aktionen ausgeschlossen (*gesperrt*). Dadurch wird ein Schwingen des Systems, d.h. ein wiederholtes Umziehen von Diensten, verhindert.

Die Regelsätze zur Steuerung des Fuzzy Controllers können von erfahrenen Administratoren für einzelne bzw. mehrere Dienste einmalig festgelegt werden. Diese vorkonfigurierten Regelsätze werden dann mit den Diensten ausgeliefert und können bei speziellen Wünschen individuell angepasst werden. Analog kann bei den linguistischen Variablen verfahren werden. Dadurch senkt sich der Administratoraufwand erheblich.

Ablauf der Aktionsauswahl: Im ersten Schritt werden die Eingabevariablen des Fuzzy Controllers initialisiert. Für die beobachteten Messwerte wie CPU-Last oder Antwortzeiten wird dazu das arithmetische Mittel über den Beobachtungszeitraum (*watchtime*) berechnet. Die weiteren Variablen werden mit entsprechenden Daten aus der Konfiguration des Systems (z.B. *performanceIndex*) belegt. Eine Übersicht über die Eingabevariablen ist in Tabelle 2 gegeben.

Der Controller unterscheidet, ob eine Ausnahmesituation durch einen Dienst (bzw.

Variable	Description
<i>cpuLoad</i>	CPU-Last auf dem Rechner (Durchschnittslast aller CPUs)
<i>memLoad</i>	benötigter Hauptspeicher des Rechners
<i>performanceIndex</i>	Leistungsfähigkeit des Rechners
<i>instanceLoad</i>	Durchschnittliche Last aller Instanzen des Rechners
<i>serviceLoad</i>	Durchschnittliche Last aller Instanzen des Dienstes
<i>instancesOnServer</i>	Anzahl der Instanzen auf dem Rechner
<i>instancesOfService</i>	Anzahl der Instanzen des Dienstes

Tabelle 2: Eingabevariablen für den Fuzzy Controller zur Auswahl einer Aktion

```
<ruleBase name="serviceOverloaded">
  <rule>
    <condition>cpuLoad is high and
      (performanceIndex is low or performanceIndex is medium)</condition>
    <action>scale-up is applicable</action>
  </rule>
  <rule>
    <condition>cpuLoad is high and performanceIndex is high</condition>
    <action>scale-out is applicable</action>
  </rule>
  ...
</ruleBase>
```

Abbildung 8: Auszug der Regelbasis für den Auslöser *service-is-overloaded*

eine Dienstinstanz) oder durch einen Rechner ausgelöst wird (siehe Abbildung 7). Falls ein Dienst der Auslöser ist, entscheidet der Controller anhand von Informationen über die betroffene Instanz, allen weiteren Instanzen des Dienstes und dem Rechner auf dem die betroffene Instanz ausgeführt wird. Andere Dienste, die auf demselben Rechner laufen, werden nicht berücksichtigt. Ist ein Rechner der Auslöser, so werden Informationen über alle Dienste auf dem entsprechenden Rechner verwendet.

Da die Auswahl der Aktion von der auslösenden Situation abhängt, verwendet unser Controller verschiedene Regelbasen für die verschiedenen Auslöser. Wir unterscheiden derzeit zwischen folgenden Auslösern: Über- und Unterlast einer Dienstes, bzw. einer Dienstinstanz (*service-is-overloaded* und *service-is-idle*), sowie Über- und Unterlast eines Rechners (*host-is-overloaded* und *host-is-idle*). Durch Hinzufügen neuer Regelbasen können wichtige Dienste individuell behandelt werden. Dies kann beispielsweise genutzt werden, um geschäftskritische Datenbankdienste vorzugsweise auf leistungsstarken Rechnern auszuführen. Eine Regelbasis setzt sich aus mehreren Regeln zusammen, von denen jede aus einer Vorbedingung und einer Schlussfolgerung besteht. Abbildung 8 zeigt die Regeln aus Abschnitt 3.2 als Teil der Regelbasis für den Auslöser *service-is-overloaded* in unserer XML-Notation. Änderungen an Regeln und Regelbasen können zur Laufzeit vorgenommen werden.

Der Fuzzy Controller zur Auswahl einer Aktion wertet die zutreffende Regelbasis aus und ermittelt scharfe Werte für die Ausgabevariablen. Diese entsprechen den in Tabelle 1 aufgelisteten Aktionen. Von diesen Aktionen kann der Controller allerdings nur diejenigen ausführen, die keine der von Administratoren festgelegten Bedingungen verletzen. Beispielsweise läuft eine traditionelle Datenbank normalerweise exklusiv auf einem Rechner. Aktionen, die derartige Bedingungen verletzen, werden als nicht anwendbar gekennzeichnet und im weiteren Ablauf ignoriert. Alle anderen Aktionen sind mit der Anwendbarkeit laut Regelbasis annotiert. Wenn ein Rechner der Auslöser war, führen wir den Fuzzy Controller für jeden Dienst auf dem Rechner aus und sammeln alle möglichen Aktionen. Die Aktion mit der höchsten Anwendbarkeit wird, sofern sie oberhalb einer vom Administrator vorgegebenen Schranke liegt, ausgeführt.

Ablauf der Rechnerauswahl: Für die Aktionen scale-out, scale-up, scale-down,

Variable	Beschreibung
cpuLoad	CPU-Last auf dem Rechner (Durchschnittslast über alle CPUs)
memLoad	benötigter Hauptspeicher des Rechners
instancesOnServer	Anzahl der Instanzen auf dem Rechner
performanceIndex	Leistungsfähigkeit des Rechners
numberOfCpus	Anzahl der CPUs des Rechners
cpuClock	Geschwindigkeit des Prozessors
cpuCache	Cache-Größe der CPUs
memory	Größe des Hauptspeichers auf dem Rechner
swapSpace	Größe des Swap Speichers
tempSpace	Größe des verfügbaren, temporären Plattenspeichers

Tabelle 3: Eingabevariablen für den Fuzzy Controller zur Auswahl eines Rechners

move und start müssen jeweils Zielrechner bestimmt werden. Dazu werden alle nicht-gesperrten, für den Dienst geeigneten Rechner mittels des zweiten Fuzzy Controllers bewertet. Tabelle 3 zeigt die linguistischen Eingabevariablen des Fuzzy Controllers zur Auswahl eines Rechners.

Da die Auswahl eines Rechners von der durchzuführenden Aktion abhängt, stehen hier ebenfalls verschiedene Regelbasen für die unterschiedlichen Aktionen zur Verfügung. Anhand der passenden Regelbasis wird ermittelt, wie gut sich ein Rechner als Zielrechner eignet. Nach der Defuzzifizierung wird der geeignetste Rechner ausgewählt.

Umsetzen der Entscheidung des Controllers: Der Controller bietet einen vollautomatischen und einen halbautomatischen Modus. Im ersten Fall werden Aktionen ohne Rückfrage ausgeführt, im zweiten Fall ist eine Bestätigung durch den Administrator erforderlich. Tritt bei der Ausführung einer Aktion ein Fehler auf, wird der nächstbeste Rechner als Zielrechner gewählt. Steht kein weiterer Rechner mehr zur Verfügung, wird die nächstbeste Aktion gewählt, bis keine Aktion mehr zur Verfügung steht. Ist das der Fall, wird der Administrator über die graphische Oberfläche des Controllers darüber informiert. Die GUI zeigt außerdem das überwachte System übersichtlich an und ermöglicht es, Aktionen manuell auszulösen.

3.4 Experimentelle Ergebnisse

Zur Validierung der Leistungsfähigkeit unserer prototypischen Implementierung haben wir ein umfangreiches Simulationssystem entwickelt, das eine typische ERP-Installation simuliert. Erste Studien zeigen, dass durch den Einsatz der dynamischen Optimierung unserer adaptiven Infrastruktur bis zu 35% mehr Benutzeranfragen auf derselben Hardware bearbeitet werden können. Eine detailliertere Beschreibung des Simulationssystems und der erzielten Ergebnisse steht unter [Aut] zur Verfügung. Weitere Studien zeigen, dass unsere Kombination von dynamischer und statischer Optimierung weitere Leistungssteigerungen ermöglichen.

4 Statische Allokation

Viele betriebswirtschaftliche Datenbankanwendungen weisen ein regelmäßig wiederkehrendes Verhalten auf, was den Verbrauch an Ressourcen wie CPU oder Arbeitsspeicher, die Netzwerkauslastung oder auch die Anzahl angemeldeter Benut-

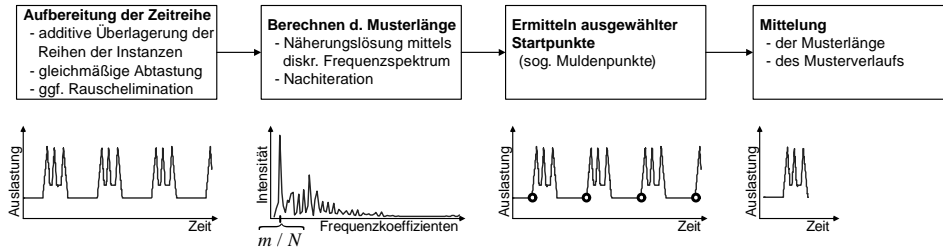


Abbildung 9: Extraktion von Lastmustern

zer und Antwortzeiten betrifft. Kennzeichnend für einen Human Resources (HR)-Dienst ist etwa die Last, die durch die Gehaltsabrechnung am Ende eines jeden Monats entsteht. Es treten also zyklisch wiederkehrende Phasen auf, zu denen Dienste stark ausgelastet sind. Andererseits gibt es auch Phasen, während derer Dienste kaum benötigt werden. Kennt man das Verhalten der Dienste, bietet es sich an, diejenigen mit komplementärer Charakteristik auf denselben und solche bei denen sich die Zeiten mit starker Auslastung überschneiden auf unterschiedlichen Rechnern auszuführen. Dadurch lassen sich kritische Überlastsituationen vermeiden. In diesem Abschnitt stellen wir Verfahren zur Extraktion von Lastmustern und zur Berechnung statischer Allokationen vor.

4.1 Analyse der Lastdaten

Zur Laufzeit werden für jede Dienstinstanz der Ressourcenverbrauch bzw. die QoS-Parameter überwacht und gleichzeitig in ein Archiv geschrieben. Die gesammelten Daten werden ausgewertet, um dienstspezifische, zyklisch wiederkehrende Lastmuster zu extrahieren. Auf welcher Art von Daten die Analyse durchgeführt wird, ist für die algorithmische Vorgehensweise ohne Bedeutung, weshalb im Weiteren allgemeine Zeitreihen betrachtet werden. Die Zeitreihe eines Dienstes erhält man durch additive Überlagerung der Lastdaten der einzelnen Instanzen. Für die Analyse wird das klassische Komponentenmodell (siehe [SS01]) der Zeitreihenanalyse verwendet. Danach wird eine Reihe $(x_t)_{1 \leq t \leq N}$ modelliert als $x_t = m_t + k_t + s_t + u_t$ mit $t \in \{1, \dots, N\}$. Der Trend m_t ist eine monotone Funktion, die eine mittel- bis langfristige Auf- oder Abwärtsbewegung modelliert. Die Restkomponente u_t beschreibt Störeinflüsse, die die Messdaten überlagern und in einer Vorverarbeitungsstufe eliminiert werden müssen. Die zyklische Komponente setzt sich aus der Überlagerung einer Konjunkturkomponente k_t , die längere, nicht notwendig regelmäßige Schwankungen repräsentiert, und der Saison s_t , die regelmäßige und relativ unveränderte Schwankungen modelliert, zusammen.

Abbildung 9 zeigt die Schritte zur Extraktion eines Lastmusters. Zuerst muss die Periode T , d.h. die Dauer eines Zyklus, ermittelt werden. Eine gängige Methode hierfür ist die Auswertung des Stichprobenspektrums. Die Reihe (x_t) , bzw. deren kontinuierliche Fortsetzung, lässt sich in eine äquivalente Darstellung von überlagerten harmonischen Schwingungen überführen. Die Funktion $I(\lambda)$ mit

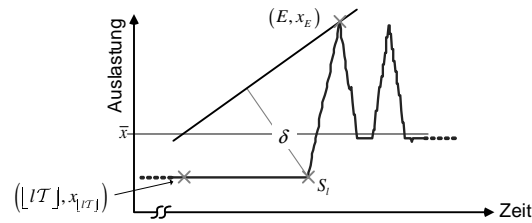


Abbildung 10: Bestimmen ausgewählter Startpunkte („Muldenpunkte“)

$$I(\lambda) = N \cdot \left[C(\lambda)^2 + S(\lambda)^2 \right], \text{ mit } \lambda \in \mathbb{R} \text{ und}$$

$$C(\lambda) = \frac{1}{N} \sum_{1 \leq t \leq N} (x_t - \bar{x}) \cdot \cos 2\pi\lambda t, \quad S(\lambda) = \frac{1}{N} \sum_{1 \leq t \leq N} (x_t - \bar{x}) \cdot \sin 2\pi\lambda t$$

gibt an, mit welcher Intensität die harmonische Schwingung der Frequenz λ in der um den Mittelwert \bar{x} bereinigten Reihe auftritt. Unter der bisher gemachten Annahme, dass ein Dienst ein charakteristisches Lastmuster aufweist, gilt, dass die Intensitätsfunktion an der Stelle $1/T$ ein globales Maximum annimmt. Die rechenintensive Auswertung von $I(\lambda)$ kann umgangen werden, indem man zuerst an den Stellen i/N mit $i \in \{1, \dots, N/2\}$ auswertet und die Stelle m/N , an der das Maximum auftritt, bestimmt. Da $I(\lambda)$ sich mittels der Fouriertransformierten von $(x_t - \bar{x})$ berechnen lässt, ist dies, verwendet man die diskrete FFT (Fast Fourier Transformation), sehr effizient. Durch eine nachfolgende iterative Auswertung von $I(\lambda)$ in der Umgebung von m/N lässt sich die Periodendauer \mathcal{T} gut approximieren.

Anschließend lässt sich bereits ein Ausschnitt der Zeitreihe als Musterapproximation extrahieren. Um ein Muster durch Mittelung der auftretenden Zyklen zu bestimmen, müssen die einzelnen Wiederholungen zuverlässig extrahiert werden. Andernfalls führt ein einfaches Mitteln über Ausschnitte der Länge \mathcal{T} aufgrund von Rechenungenauigkeiten und Störungen von (x_t) im Allgemeinen zu gestörten Daten. Benötigt werden markante Startpunkte für den Beginn eines jeden Zyklus.

Um den Startpunkt des l -ten Zyklus zu finden, geht man wie in Abbildung 10 skizziert vor. Ausgehend von $x_{[lT]}$, dem Messpunkt dessen Index am nächsten an $l \cdot \mathcal{T}$ und $\leq l \cdot \mathcal{T}$ ist, bestimmt man das darauf folgende lokale Extremum x_E . Für den Fall, dass $x_{[lT]} \leq \bar{x}$ gilt, ist dies ein lokales Maximum, andernfalls ein lokales Minimum. Anschließend bestimmt man im Intervall $[[lT], E]$ den Punkt S_l mit maximalem Abstand δ zur Geraden, die durch $([lT], \bar{x})$ und (E, x_E) festgelegt ist. Mit dieser Vorgehensweise rastet die Suche anschaulich gesprochen in einem „Muldenpunkt“ (S_l) ein, der als Startpunkt für den Zyklus verwendet wird. Der Vorteil dieses Verfahrens ist, dass keinerlei Parametrisierung der Musterextraktion erforderlich ist. Hat man die Startpunkte bestimmt (vgl. Abbildung 9, Schritt 3), so mittelt man abschließend über die Wiederholungen. Voraussetzung für eine Extraktion ist, dass mindestens zwei Zyklen in der ursprünglichen Zeitreihe auftreten. Um eine Mittelung durchzuführen, bedarf es mindestens vier Wiederholungen, da aufgrund der Verschiebung der Startpunkte und um unvollständige Zyklen auszuschließen, jeweils das erste und letzte Vorkommen nicht berücksichtigt werden.

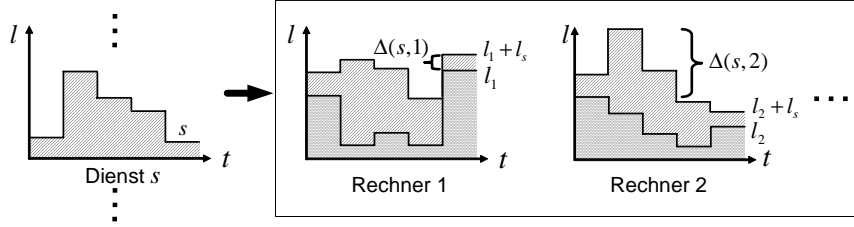


Abbildung 11: "Greedy"-Auswahl eines Rechners

4.2 Statische Allokation der Datenbankdienste auf Rechner

Basierend auf extrahierten Lastmustern aus Abschnitt 4.1 lässt sich eine statische Allokation berechnen. Die Berechnung erfolgt dabei für eine einstellbare Zeitspanne der Dauer T , die in äquidistante Abschnitte unterteilt wird. Für diesen Zeitraum wollen wir eine Allokation finden, bei der erstens die Rechner einen ausgegogenen Lastverlauf haben und zweitens der Fuzzy Controller so wenig Aktionen wie möglich durchführen muss. Dazu verwenden wir unter anderem eine Greedy-Allokationsheuristik. Diese allokiert Dienste mit komplementären Ressourcenbedürfnissen auf dieselben Rechner. Zwei Dienste weisen beispielsweise dann komplementäre Ressourcenbedürfnisse auf, wenn der eine vor allem tagsüber Last erzeugt während der andere vorwiegend nachts arbeitet.

Zu Beginn der Greedy-Heuristik wird eine Sortierung der Dienste bezüglich Klassifizierung (Replizierbarkeit und Verteilbarkeit, vgl. Abbildung 2), mittlerem Ressourcenbedarf und Priorität erstellt. So werden statische, nicht scale-out-fähige Dienste zuerst eingeplant. Hardwareressourcen sollten bei der statischen Allokationsberechnung nicht vollständig eingeplant werden. Damit die Dienstausführung sicher gestellt werden kann, muss auf jedem Rechner ein Ressourcenpuffer vorhanden sein. Die *Einplanungsgrenze*, welche von den Administratoren festgelegt wird, gibt an bis zu welcher relativen Rechnerauslastung Dienste eingeplant werden. Die Auslastung eines Rechners oberhalb dieser Grenze wird als Überlastsituation angesehen.

Um die Funktionsweise der Heuristik zu zeigen, nehmen wir an, dass bereits einige Dienste zugewiesen sind. In einem nächsten Schritt soll der Dienst s allokiert werden. Dessen Ressourcenbedarf ist durch den Vektor $(l_s^{(t)})_{1 \leq t \leq T}$ beschrieben. $l_s^{(t)}$ bezeichnet den Ressourcenbedarf des Dienstes s zum Zeitpunkt t . Entsprechend steht $l_c^{(t)}$ für die bereits eingeplante Last auf dem Rechner c zur Zeit t . $l_s^{(t)}$ und $l_c^{(t)}$ stellen von einer spezifischen Rechenleistung unabhängige Werte dar.

Allokation von nicht scale-out-fähigen Diensten: Betrachten wir zunächst den Fall, dass s ein nicht scale-out-fähiger Dienst ist. Dann wird s vollständig auf dem Rechner eingeplant, auf dem die Maximallast durch Hinzunahme von s am geringsten ansteigt und die nach der Einplanung unterhalb der vorgegebenen Einplanungsgrenze liegt. Der absolute Lastanstieg durch Einplanung von s auf c wird durch Formel (1) abgeschätzt.

$$\Delta(s, c) = \max_{1 \leq t \leq T} \left\{ l_c^{(t)} + l_s^{(t)} \right\} - \max_{1 \leq t \leq T} \left\{ l_c^{(t)} \right\} \quad (1)$$

Hierbei bezeichnet $l_c^{(t)}$ die Last vor der Einplanung des Dienstes s zum Zeitpunkt t . Falls jeder Rechner eine höhere Maximallast als die Einplanungsgrenze aufweist, wird derjenige mit der geringsten Maximallast ausgewählt. Abbildung 11 verdeutlicht den Iterationsschritt exemplarisch anhand zweier Rechner. So fällt der Anstieg der Maximallast auf Rechner 1 geringer aus als auf Rechner 2.

Allokation von scale-out-fähigen Diensten: Die Allokation von scale-out-fähigen Diensten bietet mehr Freiheitsgrade, da man diese in mehrere Instanzen aufteilen und somit auf mehrere Rechner verteilen kann. Dazu führen wir Variablen $x_{s,c}$ ein. $x_{s,c}$ beschreibt den Anteil der auf Rechner c ausgeführten Instanz des Dienstes s am gesamten Ressourcenbedarf von s . Im Folgenden wird $x_{s,c}$ auch als Instanzgröße bezeichnet. Sie nimmt Werte zwischen 0 (keine Instanz auf dem Rechner) und 1 (kompletter Dienst auf dem Rechner) an. Die statische Allokationsoptimierung arbeitet in Kombination mit dem Dispatcher, der Benutzeranfragen proportional zu den vorgegebenen Instanzgrößen auf die Instanzen eines Dienstes verteilt.

Bei der Allokation von scale-out-fähigen Diensten können verschiedene Strategien verwendet werden:

Verbot von Überlastsituationen: Dienste dürfen Rechner höchstens bis zur Einplanungsgrenze auslasten.

Begrenzter Zeitraum: Dienste dürfen Überlastsituationen auf Rechnern für begrenzte Zeit hervorrufen.

Maximale Überlast: Dienste dürfen Rechner bis zu einer festgelegten Lastgrenze oberhalb der Einplanungsgrenze belasten.

Begrenzte Durchschnittslast: Die durchschnittliche Auslastung eines Rechners darf die Einplanungsgrenze nicht überschreiten.

Die Wahl der Strategie hängt von der Verschiebbarkeit der Dienste ab. Bei statischen Diensten eignet sich die erste Strategie. Überlastsituationen, die von dynamischen Diensten erzeugt werden, können durch die dynamische Kontrollkomponente behandelt werden. Für diese Klasse von Diensten eignen sich daher die übrigen Strategien.

Abbildungen 12(a) und 12(b) verdeutlichen die Strategie *Verbot von Überlastsituationen* anhand eines Beispiels. Falls ein scale-out-fähiger Dienst s nicht vollständig auf einem Rechner eingeplant werden kann (siehe Abbildung 12(a)), wird s aufgeteilt. Dazu wird die maximal einplanbare Instanzgröße bestimmt, die den Rechner nicht überlastet (siehe Abbildung 12(b)).

Um den Einfluss von Instanzgrößen zu berücksichtigen wird Formel (1) erweitert zu:

$$\Delta(s, c) = \frac{\max_{1 \leq t \leq T} \{l_c^{(t)} + x_{s,c} \cdot l_s^{(t)}\} - \max_{1 \leq t \leq T} \{l_c^{(t)}\}}{x_{s,c}} \quad (2)$$

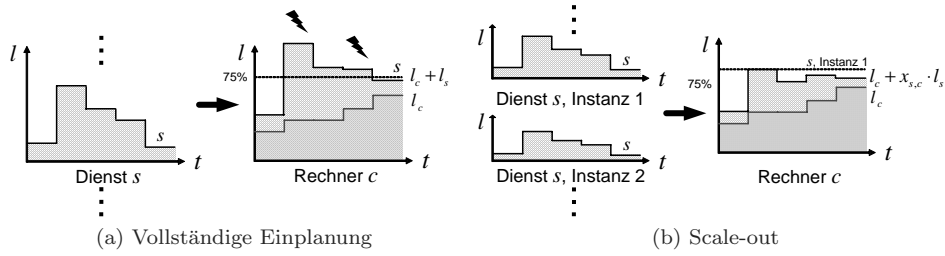


Abbildung 12: Einplanen eines scale-out-fähigen Dienstes

Die Auswahl des Rechners, auf den eine Instanz von s sich am besten einplanen lässt, erfolgt anhand der Kostenfunktion (3). Die Aufteilung von s in viele, relativ kleine Instanzen wird verhindert. Dies wird zum einen durch die Neudefinition von $\Delta(s, c)$ berücksichtigt und zum anderen erfolgt eine Bestrafung einer Aufteilung durch den zweiten Term der Kostenfunktion (3) – je kleiner (angegeben durch $0 < x_{s,c} \leq 1$) eine einzuplanende Instanz ist, desto größer sind die durch diesen Term modellierten Kosten. Der Einfluss lässt sich anhand der Gewichte α und β variieren.

$$\Theta(s, c) = \alpha \cdot \Delta(s, c) + \frac{\beta}{x_{s,c}} \quad (3)$$

Verwendet man eine der anderen Strategien, so muss in die Kostenfunktion eine Bestrafung der voraussichtlichen Überlastsituationen mit aufgenommen werden. Zwar sind Überlastsituationen prinzipiell erlaubt, dennoch müssen sie nach Möglichkeit vermieden werden. Die Heuristik wählt in jedem Schritt den Rechner \hat{c} aus, für den die Kostenabschätzung (3) den geringsten Wert annimmt. Auf diesen wird eine Instanz der Größe $x_{s,\hat{c}}$ des Dienstes s eingeplant. In den folgenden Iterationsschritten werden die restlichen Anteile von s auf anderen Rechnern eingeplant. Die Greedy-Heuristik bricht ab, sobald alle Dienste vollständig eingeplant sind, d.h. eine statische Allokation berechnet wurde.

4.3 Übernahme der statischen Allokation

Eine neu berechnete Allokation kann entweder automatisch oder manuell übernommen werden. Vor der automatischen Übernahme prüft das System, ob sie der aktuellen Allokation vorzuziehen ist. In den Vergleich beider Allokationen fließen Kosten für die dazu notwendige Reallokation der Dienste ein. Der Vorteil der neuen, alternativen Allokation gegenüber der aktuellen, wird anhand einer Auswertung von (zu erwartenden) Über- und Unterlastsituationen der verwendeten Rechner bewertet. Bei manueller Änderung der Allokation dient diese Bewertung den Administratoren als Entscheidungshilfe.

5 Verwandte Arbeiten

In [WMHZ02] wird der Nutzen automatischer Tuningkonzepte für Datenbanksysteme hervorgehoben. Demzufolge soll eine entsprechende Überwachungskomponente eingesetzt werden, die folgende drei Phasen abdeckt: Beobachtung, Vorhersage und

Reaktion. [MLR03] stellt den Anfrageoptimierer von IBM DB2 vor, der sich durch die eigenständige Optimierung auszeichnet, indem z.B. zugrunde liegende Kostenmodelle und erstellte Statistiken ohne Benutzerinteraktion angepasst werden. Merkmale einer adaptiven Infrastruktur, wie Selbstkonfiguration, Selbstoptimierung und eigenständige Fehlerbehebung, sind zunehmend in aktuellen bzw. zukünftigen Versionen kommerzieller Datenbanksysteme umgesetzt. Neue Administrations- und Managementansätze von IBM DB2, Microsoft SQL Server und Oracle 10g wurden in [CDL04] vorgestellt. Bei diesen Selbstadministrierungs-Konzepten der DBMS-Hersteller wird allerdings das Datenbanksystem isoliert betrachtet. Dies reicht aber nicht aus um den Endnutzern ein durchgängiges Quality-of-Service-Management der datenbankbasierten Dienste zu gewährleisten. Hierzu muss man das Gesamtsystem, also Datenbankanwendungen inklusive den darunter liegenden Datenbanken, optimieren. Hier setzt unser Projekt an, in dem wir eine adaptive Infrastruktur für dienstbasierte, heterogene Systemlandschaften entwickeln.

Die Grid-Infrastruktur stellt Techniken für die Integration verteilter Datenbanken und allgemeiner Ressourcen bereit. Mit der Definition von Monitoring- und Tuning-Konzepten für Grid-Anwendungen befasst sich die Performance Working Group des Global Grid Forums [GGFa]. In [Bou01] werden Terminologie und Konzepte von Lastbalancierung vorgestellt und auf die Komplexität der Lastverteilung für Rechnerlandschaften eingegangen. Methoden der Lastbalancierung für parallele Datenbanksysteme werden in [RM95, RM96] untersucht. [FA04] präsentiert ein neues Konzept zur dynamischen Anpassung einer dienstbasierten Software-Architektur, um zur Laufzeit z.B. auf Ressourcenengpässe zu reagieren. In [ADZ00] wird eine Methode zur Ressourcenverteilung in einem Rechnercluster vorgestellt, welche auf der individuellen Ressourcenverteilung der einzelnen Rechner basiert. [RBSS02] befasst sich mit der Skalierung von DB-Anwendungen durch Replizierung und behandelt diesbezüglich Aspekte der Kohärenz (Aktualität) der Daten.

Mit Autonomia [DHX⁺03] und Automate [ABL⁺03] existieren Forschungsprojekte, die Aspekte einer adaptiven Infrastruktur, wie Selbstoptimierung, und eigenständige Fehlerbehebung realisieren. In dem verteilt aufgebauten System AutoMate teilen die überwachten Dienste dem System Informationen wie beispielsweise ihren Ressourcenbedarf und aktuellen Zustand mit. Dies erfordert, im Gegensatz zu AutoGlobe, die Anpassung der Dienste an das System. Ferner unterscheidet sich AutoGlobe von Autonomia und AutoMate hinsichtlich der auf einen Fuzzy Controller basierenden Kontrollkomponente. Dadurch lässt sich die kontinuierliche Überwachung und die Reaktion auf Ausnahmesituationen flexibel konfigurieren und administrieren.

6 Zusammenfassung und Ausblick

In diesem Beitrag haben wir eine adaptive Infrastruktur für dienstbasierte Datenbankanwendungen vorgestellt, die die zunehmende Komplexität derartiger Systeme handhabbar macht und eine zuverlässige und kostengünstige Bereitstellung der Dienste ermöglicht. Unser Ansatz basiert auf einer Kombination von statischer und dynamischer Optimierung der Dienst-Rechner-Allokation. Das Ergebnis

der statischen Optimierung ist eine längerfristig ausgelegte Verteilung der Dienste auf die verfügbare Hardware. Zur Laufzeit werden Ausnahmesituationen automatisch erkannt und durch einen auf Fuzzy-Logik basierenden Controller behoben. Da bei einer guten statischen Dienst-Rechner-Allokation zur Laufzeit weniger Ausnahmesituationen auftreten, führt die Kombination von statischer und dynamischer Allokation zu einem ausgeglicheneren und ruhigeren Systemverhalten.

Die Konzepte zur statischen und dynamischen Optimierung wurden als Erweiterung des ServiceGlobe-Systems im Rahmen des AutoGlobe-Projekts prototypisch realisiert. Anhand umfangreicher Simulationsstudien [Aut] wurde die Leistungsfähigkeit der Infrastruktur gezeigt. Das bislang größte System, auf dem der Prototyp eingesetzt wurde, um eine vollständige ERP-Installation zu überwachen, ist ein Blade-System mit insgesamt 160 Prozessoren.

Die Forschungen im Rahmen von AutoGlobe werden in folgende Richtungen weitergeführt: Zum einen wird untersucht, wie Lastmuster, die bei der statischen Optimierung gewonnen werden, auch zur Verbesserung der dynamischen Optimierung beitragen können. Darüber hinaus werden Dispatcher, die Anfragen auf die verschiedenen Instanzen eines Dienstes verteilen, stärker mit der statischen und der dynamischen Kontrollkomponente verzahnt. Letztendlich soll AutoGlobe die Grundlage für ein umfassendes QoS-Management bilden, das Verfügbarkeit (availability), Leistungsfähigkeit (performability) und Vorhersagbarkeit (predictability) der Datenbankdienste umfasst.

Danksagung

Wir bedanken uns bei den Herren Wolfgang Becker, Ingo Bohn, Thorsten Dräger und Daniel Scheibli von der SAP Adaptive Computing Infrastructure Abteilung für ihre Kooperation. Außerdem danken wir den Herren Stefan Aulbach, Tobias Brandl, Michael Denk, Stefan Krompaß und Thomas Schelchshorn für die Hilfe bei der Implementierung des Systems.

Literatur

- [ABL⁺03] M. Agarwal, V. Bhat, H. Liu, V. Matossan, V. Putty, C. Schmidt, G. Zhang, L. Zhen, M. Parashar, B. Khargharia und S. Hariri. AutoMate: Enabling Autonomic Applications on the Grid. In *Proceedings of the International Workshop on Active Middleware Services (AMS)*, Seiten 48–59, Seattle, WA, USA, Juni 2003.
- [ACKM03] G. Alonso, F. Casati, H. Kuno und V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer Verlag, September 2003.
- [ADZ00] M. Aron, P. Druschel und W. Zwaenepoel. Cluster Reserves: A Mechanism for Resource Management in Cluster-based Network Servers. In *Measurement and Modeling of Computer Systems*, Seiten 90–101, 2000.
- [Aut] AutoGlobe Simulation Studies. <http://www-db.in.tum.de/research/projects/AutoGlobe>.
- [Bou01] T. Bourke. *Server Load Balancing*. O'Reilly & Associates, Sebastopol, CA, USA, 2001.

- [CDL04] S. Chaudhuri, B. Dageville und G. Lohman. Self-Managing Technology in Database Management Systems. Tutorial at the International Conference on Very Large Data Bases (VLDB), Toronto, Canada, September 2004.
- [DHX⁺03] X. Dong, S. Hariri, L. Xue, H. Chen, M. Zhang, S. Pavuluri und S. Rao. Autonomia: An Autonomic Computing Environment. In *Proceedings of the International Performance Computing and Communications Conference (IPCCC)*, Seiten 61–68, Phoenix, AZ, USA, April 2003.
- [FA04] P. Falcarin und G. Alonso. Software Architecture Evolution through Dynamic AOP. In *Proceedings of the First European Workshop on Software Architecture (EWSA)*, Jgg. 3047, Seiten 57–73, St. Andrews Scotland, 2004.
- [Fle] FlexFrame für mySAP Business Suite. <http://www.fujitsu-siemens.de/rl/mobility/flexframe.html>.
- [GGFa] Grid Monitoring Architecture Working Group of the Global Grid Forum (GGF). <http://www-didc.lbl.gov/GGF-PERF/GMA-WG/>.
- [GGFb] Scheduling and Resource Management Area of the Global Grid Forum (GGF). <https://forge.gridforum.org/projects/srm/>.
- [KL04] D. Kossmann und F. Leymann. Web Services. In *Informatik-Spektrum*, Jgg. 27(2), Seiten 117–128, 2004.
- [KSK03] M. Keidl, S. Seltzsam und A. Kemper. ServiceGlobe: Flexible and Reliable Web Services on the Internet (Poster Presentation). In *Proceedings of the International World Wide Web Conference (WWW)*, Budapest, Hungary, Mai 2003.
- [KY94] G. J. Klir und B. Yuan. *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice Hall, Upper Saddle River, NJ, USA, 1994.
- [MLR03] V. Markl, G. M. Lohman und V. Raman. LEO: An Autonomic Query Optimizer for DB2. *IBM Systems Journal*, 42(1):98–106, 2003.
- [RBSS02] U. Röhm, K. Böhm, H.-J. Schek und H. Schuldt. FAS – a Freshness-Sensitive Coordination Middleware for a Cluster of OLAP Components. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, Seiten 754–765, Hong Kong, China, August 2002. Morgan Kaufmann Publishers.
- [RM95] E. Rahm und R. Marek. Dynamic Multi-Resource Load Balancing in Parallel Database Systems. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, Seiten 395–406, 1995.
- [RM96] E. Rahm und R. Marek. Dynamic Load Balancing in Parallel Database Systems. In *Proceedings of EURO-PAR*, Jgg. 1123 of *Lecture Notes in Computer Science (LNCS)*, Seiten 37–52, Lyon, 1996.
- [SBK01] S. Seltzsam, S. Börzsönyi und A. Kemper. Security for Distributed E-Service Composition. In *Proceedings of the International Workshop on Technologies for E-Services (TES)*, Jgg. 2193 of *Lecture Notes in Computer Science (LNCS)*, Seiten 147–162, Rome, Italy, September 2001.
- [SS01] Rainer Schlittgen und Bernd Streitberg. *Zeitreihenanalyse*. R. Oldenbourg Verlag, 9. Auflage, 2001.
- [Wei99] G. Weikum. Towards Guaranteed Quality and Dependability of Information Services. In Alejandro P. Buchmann, Hrsg., *8th GI Fachtagung: Datenbanksysteme in Büro, Technik und Wissenschaft*, Seiten 379–409, Freiburg, Germany, 1999. Springer Verlag.
- [WMHZ02] G. Weikum, A. Mönkeberg, C. Hasse und P. Zaback. Self-tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, Seiten 20–31, Hong Kong, China, August 2002.
- [Zad65] L. A. Zadeh. Fuzzy Sets. *Information and Control*, 8(3):338–353, 1965.