Philipps Universität Marburg

Fachbereich Mathematik und Informatik

Diplomarbeit

Cloud Monitoring mit Complex Event Processing

von

Bastian Hoßbach

im

Oktober 2011

Betreuer:

Prof. Dr. Bernhard Seeger

Zusammenfassung

Mit Hilfe von Cloud Computing können Infrastrukturen, Plattformen und Anwendungen flexibel, ortsungebunden sowie in beliebigem Umfang als Dienste im Web gemietet werden, anstatt sie durch Investitionen dauerhaft anschaffen zu müssen. Während in den letzten Jahren die hinter Cloud Computing stehenden Technologien aufgrund großer wirtschaftlicher Erfolge rasant weiterentwickelt wurden, existieren weiterhin eine Reihe vielseitiger und kritischer Risiken aus den Bereichen Leistungsfähigkeit, Sicherheit und Verfügbarkeit. Ein zentraler Grund für das Fortbestehen der Risiken ist, dass noch immer geeignete Werkzeuge zum Überwachen (Monitoring) der Dienste und ihrer Infrastrukturen fehlen. Complex Event Processing (CEP) ist eine weitere in der jüngeren Vergangenheit entstandene und erfolgreiche Technologie. Durch CEP können große und dynamische Datenmengen kontinuierlich und zeitnah durch viele parallel laufende Analysen detailliert ausgewertet werden. Da bei einer ganzheitlichen Überwachung von Cloud Computing sich schnell ändernde und massenweise Messdaten anfallen, die nur mit vielen und komplexen Überwachungsregeln zuverlässig untersucht und kontrolliert werden können, wird CEP zu einer idealen Basis für ein Cloud Monitoring. Trotz der guten Eignung von CEP für diese Aufgabe gibt es bei dem Verknüpfen beider Technologien Herausforderungen, die speziell angepasste Designs und Implementierungen verlangen. Diese Arbeit widmet sich daher der Entwicklung eines Cloud Monitorings auf der Grundlage von CEP. In realen Situationen konnte nachgewiesen werden, dass ein Cloud Monitoring mit CEP-Technologie ein leistungsstarkes Fundament bildet, auf dem individuelle Lösungen zur Beseitigung der Risiken von Cloud Computing entwickelt und implementiert werden können.

Danksagung

Hiermit möchte ich mich bei der AG Verteilte Systeme, geleitet von Herrn Prof. Dr. Bernd Freisleben, für viele hilfreiche Diskussionen zu den Themen Cloud Computing und Monitoring bedanken. Ein großes Dankeschön geht an die Herren Lars Baumgärtner, Pablo Graubner, Dr. Matthias Schmidt und Roland Schwarzkopf für die Implementierung der Sensoren und des Action Frameworks.

Meinen besonderen Dank spreche ich der gesamten AG Datenbanksysteme unter Leitung von Herrn Prof. Dr. Bernhard Seeger aus. Während ich an dieser Arbeit geschrieben habe, traf ich stets auf offene Ohren bei Fragen, neuen Ideen und Problemen.

Inhaltsverzeichnis

1 Einleitung

In diesem Kapitel wird in Abschnitt 1.1 die dieser Arbeit zugrunde liegende Thematik motiviert und es werden aktuelle Probleme in Abschnitt 1.2 sowie die Ziele dieser Arbeit in den Abschnitten 1.3 und 1.4 dargestellt. Das Kapitel schließt mit einem Überblick über den weiteren Aufbau der Arbeit in Abschnitt 1.5 ab.

1.1 Vision des Utility Computing

Seitdem vier Computer im Dezember 1969 erfolgreich zu dem ersten modernen Netzwerk *ARPANET* zusammengeschlossen wurden [Rob86], hat sich die bereits 1961 geäußerte Vision des Utility Computing [Fos+08] schrittweise ihrer vollständigen Realisierung genähert [Kle05]. Utility Computing bedeutet, dass die jederzeit mögliche, potentiell unbegrenzte und verbrauchsabhängig abgerechnete Nutzung von Informations- und Kommunikationstechnologie (IKT) zu einer für das alltägliche Leben notwendigen Selbstverständlichkeit werden wird. Diese Vision wird endgültig zur Realität, wenn IKT-Dienstleistungen den gleichen gesellschaftlichen Stellenwert wie Gas, Elektrizität, Telefonie und Wasser – die vier bereits etablierten *Utilities* – haben. Im Verlauf der letzten Jahrzehnte kam es durch den technologischen Fortschritt zu einer immer engeren Annäherung an das Utility Computing. Aufbauend auf vielen ausgereiften Technologien läutet das aktuelle Cloud Computing die nächste Phase in diesem evolutionären Prozess ein [Arm+09].

Mit Hilfe von Cloud Computing stehen beliebige Mittel der IKT wie Speicher, Laufzeitumgebungen oder fertige Anwendungen in einem Netzwerk[1] (z. B. dem Internet) als auf Web-Technologie basierende Dienste bereit. Für die Nutzung dieser Dienste wird nur ein Zugang zu dem Netzwerk benötigt. Die Abrechnung geschieht dabei nutzungsabhängig. Das bedeutet, es fallen nur für den Zeitraum, in dem die Dienste verwendet werden, Kosten an. Durch Virtualisierung werden technische Details vor den Nutzern verborgen sowie eine große Flexibilität, gute Auslastung der Infrastruktur und dynamische Skalierbarkeit der Dienste erreicht. Letzteres hat zur Folge, dass sich Dienste automatisch und in Echtzeit an veränderte Bedürfnisse eines einzelnen Nutzers anpassen können.

[1] Die Infrastruktur, die diese Dienste bereitstellt, wird vereinfachend als (die) *Cloud* bezeichnet. Diese Sprechweise wird auch im weiteren Verlauf der Arbeit verwendet werden.

Diese neue Art der Nutzung von IKT, die sowohl von Unternehmen als auch von Privatpersonen praktiziert wird, sorgt für mehr Flexibilität und Mobilität. So verwalten derzeit eine Vielzahl von Personen ihre privaten Daten (wie z.B. Fotos, Musik, E-Mails, Dokumente) in einer öffentlichen Cloud im Internet und müssen sich keine Gedanken mehr über Speicherknappheit machen. Weil der Zugriff über das Internet geschieht, sind die Daten jederzeit von jedem beliebigen Ort aus verfügbar und können dadurch zusätzlich mit anderen Personen ausgetauscht werden. Unternehmen hingegen nutzen Dienste in einer Cloud, um ihre Rechenzentren kostengünstiger zu gestalten. Diese können soweit rationalisiert werden, dass nur noch genügend Kapazitäten für das durchschnittliche Tagesgeschäft bereitstehen. Leistungsspitzen im Bedarf (z. B. zum Weihnachtsgeschäft bei einem Web-Shop) werden dadurch abgefangen, dass die kurzfristig zusätzlich benötigten Kapazitäten aus einer öffentlichen Cloud bezogen werden. Jüngere Unternehmen gehen teilweise noch einen Schritt weiter und verzichten vollständig auf den Betrieb eigener Rechenzentren und lagern ausnahmslos alle Daten und Anwendungen in einer öffentlichen Cloud.

Neben der Nutzung von Diensten in einer Cloud spielt auch das Anbieten eigener Dienste eine wichtige Rolle im Cloud Computing. Eine mögliche Variante ist zum Beispiel der Vertrieb von Anwendungen. Anstatt sie bei den Kunden zu installieren und einmalig abzurechnen, stehen sie in einer Cloud als jederzeit betriebsbereite Dienste zur Verfügung. Die Entwickler haben dadurch den Vorteil, dass sie nur noch eine einzige Installation pro Anwendung pflegen müssen. Die Kunden hingegen haben keinen Lizenzierungsaufwand mehr und zahlen nutzungsabhängig für den Einsatz der Anwendung (Mieten statt Kaufen).

1.2 Risiken von Cloud Computing

Neben einem enormen Potential[2] ist Cloud Computing auch mit Risiken verbunden. Die europäische Agentur für Informationssicherheit ENISA hat insgesamt 35 Risiken herausgearbeitet und empfiehlt daher die Nutzung von öffentlichen Clouds nur für unkritische Anwendungen und Daten sowie das permanente Bereithalten einer Ausstiegsstrategie aus dem Cloud Computing [CH09]. Zum einen entsteht durch die Nutzung von Clouds eine Abhängigkeit zu den Betreibern. Die Nutzer müssen darauf vertrauen, dass gebuchte Dienste wie vereinbart (Leistungsumfang, Verfügbarkeit, etc.) bereitstehen, da sie selbst wahrscheinlich nicht (mehr) über die nötigen Kapazitäten verfügen, um ihren Bedarf zu decken. Zum anderen gibt es Bedenken bei der Lagerung von Daten in einer Infrastruktur, in der die Besitzer der Daten keinen Einfluss mehr auf Speicher- und Sicherungsstrategien haben.

[2] Der Verband der dt. IKT-Industrie BITKOM erwartet für 2015 einen Umsatz von 8,2 Mrd. EUR durch Cloud Computing in Deutschland bei einem durchschn. Umsatzwachstum von 50 % pro Jahr [Bun10].

Laut aktuellen Umfragen nutzt nur jedes fünfte große Unternehmen die neue Cloud-Technologie [Str11]. Unter Beachtung der sich für Unternehmen durch Cloud Computing ergebenden (Kosten-)Vorteile und des einfachen Auslagerungsprozesses von Daten und Anwendungen in eine Cloud erscheint dieser Anteil gering. Informationsmanager von Unternehmen gaben in [Str11] die nachfolgenden Gründe gegen eine derzeitige Einführung von Cloud Computing in ihren Unternehmen an.

1.2.1 Gefährdung der Daten

Durch die Auslagerung von Daten in eine öffentliche Cloud sehen 77 % der Befragten sie dort mehr Gefahren ausgesetzt als in einem privaten Rechenzentrum. Da in einer öffentlichen Cloud die Daten zusammen mit denen anderer Nutzer aufbewahrt werden, ermöglicht jede Störung in der Isolation zwischen den Nutzern unautorisierte Zugriffe durch Fremde. Eine weitere Gefahr geht von den Mitarbeitern des Betreibers der Cloud aus, die aufgrund technischer Notwendigkeiten Zugriff auf die Daten haben und diesen Zugriff missbrauchen könnten. Zusätzlich ergibt sich ein Problem beim Schutz der Daten aus der öffentlichen und gut dokumentierten Schnittstelle. Wegen der Bereitstellung eines Web-basierenden Zugangs über das Internet sind die Daten prinzipiell für jeden Internetnutzer verfügbar. Eine große öffentliche Cloud mit den Daten vieler Unternehmen und Privatpersonen wird dadurch zu einer „Goldgrube“ für Datendiebe [Rai].

Um die Vorteile von Cloud Computing maximal ausnutzen zu können, müssen Daten ausschließlich in einer Cloud liegen. Lokale Duplikate würden dem Paradigma von Cloud Computing widersprechen, da sie das Rationalisierungspotential im unternehmensinternen Rechenzentrum erheblich senken. Daraus allerdings ergibt sich als ein weiteres Risiko, dass bei einem Datenverlust in der Cloud die Daten unwiederbringlich verloren sind.[3]

1.2.2 Kontrollverlust

Eng verbunden mit dem vorherigen Punkt ist die Sorge von 61 % der Befragten, dass bei der Nutzung von Diensten in einer externen Cloud die Kontrolle über die eigenen Daten fast gänzlich verloren geht. Alle Entscheidungen, wie z. B. Sicherungs- und Speicherstrategie, werden von dem Betreiber der Cloud getroffen, der unter Umständen andere Ziele verfolgt als seine Kunden. Weil die Infrastrukturen der großen öffentlichen Clouds rund um den Globus verteilt sind, ist in vielen Fällen sogar unklar, in welchem Land sich die Daten befinden.

[3] Dass dieses Szenario nicht nur theoretischen Überlegungen entspringt, zeigt zum Beispiel [CW].

Das Aufgeben der Entscheidungshoheit ist besonders problematisch für Unternehmen, da Speicherung und Verarbeitung von Daten einer ganzen Reihe an Regeln und Gesetzen unterworfen sind. Für deren Einhaltung sind z. B. in der Bundesrepublik Deutschland die dort ansässigen Unternehmen als rechtliche Inhaber der Daten selbst verantwortlich und nicht der Betreiber der Cloud, in der die Daten gespeichert oder verarbeitet werden.[4] Ohne die hier notwendige Kontrolle über die Daten ist es sehr schwierig bis unmöglich dem Staat und den eigenen Kunden gegenüber zu gewährleisten, dass alle geltenden Datenschutzbestimmungen korrekt umgesetzt sind. Dieses Problem wurde von 35 % der Befragten als Hindernis für den Einstieg in Cloud Computing angegeben.

1.2.3 Unzuverlässigkeit

Mehr als jeder zweite Befragte bezweifelt, dass der Betreiber einer öffentlichen Cloud die versprochenen Mindestleistungen jederzeit anbieten kann. Der Betreiber ist dafür verantwortlich, die Arbeitslast aller seiner Kunden so in seiner Infrastruktur aufzuteilen, dass jedem Kunden immer die ihm jeweils zugesicherten Kapazitäten zur Verfügung stehen. Dabei kann es allerdings zu vielfältigen Problemen kommen. Es ist daher gängige Praxis, dass sich Nutzer vor Leistungseinbrüchen vertraglich schützen. Ein solcher als Dienstgütevereinbarung (engl. *Service Level Agreement*, SLA) bezeichneter Vertrag ist rechtsverbindlich, hält die zu erbringenden Leistungen des Betreibers in Form von messbaren Werten exakt fest und definiert mögliche Sanktionen, die dem Betreiber bei einem Verstoß drohen [Off11]. Bei Nichteinhaltung einer oder mehrerer Vereinbarungen fallen üblicherweise keine oder weniger Kosten für den betroffenen Zeitraum an oder es erfolgt eine Gutschrift. Allerdings gibt es unzählige Anwendungen (z. B. aus den Bereichen Medizin oder Wertpapierhandel), bei denen ein Unterschreiten der minimal benötigten Leistung zum falschen Zeitpunkt katastrophale Folgen haben kann.

Ein gutes Drittel der Befragten zweifelt darüber hinaus die Verfügbarkeit einer Cloud gegenüber der eines klassischen Rechenzentrums an,[5] weil die Infrastruktur einer Cloud von sehr vielen Nutzern parallel in Anspruch genommen wird und durch das Verhalten anderer Nutzer größere Gefahren (wie beispielsweise Abstürze durch Überlastung) ausgehen können als von dem eigenen. Des Weiteren ist eine Cloud, die von vielen Nutzern verwendet wird, attraktiver für Angreifer mit der Absicht Schaden anzurichten als ein nur von einem einzigen Unternehmen genutztes Rechenzentrum, da bei einem vorsätzlich ausgelösten Totalausfall zahllose Unternehmen und Personen betroffen wären.

[4] § 11 Bundesdatenschutzgesetz

[5] Auch diese Überlegungen sind nicht nur reine Theorie, wie beispielsweise [FTD] zeigt.

1.2.4 Abhängigkeit

Aufgrund fehlender Standards im Cloud Computing hat jeder Betreiber einer öffentlichen Cloud seine eigenen Produkte und Schnittstellen entwickelt. Ein Wechsel der Cloud, weil beispielsweise ein anderer Betreiber über die Zeit attraktiver geworden ist, ist für Nutzer nur unter einem enormen Aufwand mit entsprechenden Kosten möglich. Neben der Anpassung an die neuen Schnittstellen verursacht auch ein Umzug an sich Kosten, weil in den meisten öffentlichen Clouds der Datentransport eine mit Kosten in Abhängigkeit von der Größe der Daten behaftete Dienstleistung ist. Die Hälfte aller Befragten gab diesen Punkt als Begründung für den Verzicht auf die Nutzung einer Cloud an.

Durch den erschwerten Wechsel ist ein Nutzer demnach langfristig an einen Betreiber gebunden und damit von ihm abhängig [Lea09b]. Bei 34 % der Befragten bestehen Vorbehalte gegen die wirtschaftliche Stabilität der Betreiber von öffentlichen Clouds, die im Lauf der langfristigen Bindung an sie einbrechen könnte. Die möglichen Folgen von finanziellen Problemen bei einem Betreiber reichen von massiven Einbußen in der Qualität der Dienste bis hin zu vollständigen Datenverlusten (weil der Betreiber z. B. plötzlich billigere Hardware einsetzt oder das technische Personal abgebaut hat).

1.3 Überwachungsbedarf

Die im letzten Abschnitt dargestellte Kritik an Cloud Computing lässt in einer gemeinsamen Schnittmenge erkennen, dass es unter anderem an Möglichkeiten zur Überwachung (engl. *Monitoring*) und Steuerung von Clouds mangelt. Dieser Eindruck wird von einer weiteren Umfrage, die weltweit unter 3.700 Unternehmen durchgeführt wurde, gestützt [Sym11]. Das Ergebnis dieser Umfrage, laut der über 75 % aller Großunternehmen Interesse an Cloud Computing haben, ist in Tabelle 1.1 wiedergeben. Dargestellt sind die vier meistgenannten Risiken sowie die genauen Anteile aller Befragten, die sie jeweils angegeben haben. Der Mangel an Werkzeugen zur Überwachung und Steuerung wurde von knapp zwei Dritteln aller Befragten genannt.

Risiko	**Anteil aller Befragten**
Unzuverlässigkeit	78 %
Unsicherheit	76 %
Leistungseinbußen	76 %
Mangel an Werkzeugen zur Überwachung und Steuerung	63 %

Tabelle 1.1: Risiken von Cloud Computing

1.4 Anforderungen an ein Cloud Monitoring

Erst auf der Grundlage einer leistungsstarken Überwachung, die genaue Einblicke in das Geschehen einer Cloud liefert, können viele der genannten Probleme und Risiken gelöst bzw. vermieden werden. Weil es sich bei Clouds um hochgradig komplexe und dynamische Systeme bestehend aus vielen Objekten mit sich ändernden Beziehungen zwischen ihnen handelt, ergeben sich besondere Anforderungen an ein Cloud Monitoring. Das Ziel dieser Arbeit ist die Entwicklung eines solchen Werkzeugs und deshalb sind die nachfolgend dargestellten Anforderungen zugleich Teilziele von ihr.

1.4.1 Analysen

Ein Cloud Monitoring darf sich nicht nur auf das Ermitteln und Speichern von Messwerten beschränken, sondern es müssen auch vielfältige und umfangreiche Möglichkeiten zum Untersuchen der gemessen Daten angeboten werden. Neben einer anpassbaren visuellen Darstellung der Messdaten ist ein einfaches Einbringen von individuellen und kontinuierlich ausgeführten Überwachungsregeln, die sowohl einzelne Objekte als auch die bestehenden Abhängigkeiten und Beziehungen zwischen ihnen untersuchen sowie kontrollieren können, notwendig. Zusätzlich sollte ein Cloud Monitoring Hilfestellungen geben, mit denen laufende Überwachungsregeln leichter evaluiert und weitere sinnvolle Regeln gefunden werden können.

1.4.2 Echtzeit

Aufgrund der hohen Dynamik von Clouds ist nur eine Überwachung in Echtzeit geeignet. Engpässe und Probleme sowie Chancen zur Leistungssteigerung oder Kostensenkung müssen so schnell wie möglich erkannt werden, um entsprechend darauf reagieren zu können. Nur wenn Ereignisse direkt bei ihrem Eintreten erkannt werden können, können die durch Probleme verursachten Schäden minimiert und die sich aus Chancen ergebenden Vorteile maximal ausgenutzt werden.

1.4.3 Erweiterbarkeit

Clouds unterliegen einem stetigen Wandel. Ständig werden neue Arten von Diensten eingebracht, wodurch sich völlig neue Beziehungen mit und zwischen vorhandenen Diensten ergeben können. Ein Cloud Monitoring muss einfach und schnell sowohl in die Lage versetzt werden können, einen neuen Dienst zu überwachen, als auch durch Hinzunahme zusätzlicher Analysen neue Zusammenhänge kennenlernen.

1.4.4 Ganzheitlichkeit

Jedes Überwachungssystem kann immer nur dann zuverlässig arbeiten, wenn es über alle Fakten genau Bescheid weiß. Beispielsweise wird bei einem durch Videokameras überwachten Raum, bei denen allerdings ein toter Winkel vorhanden ist, immer ein Restrisiko bleiben, das zu Unsicherheiten führt. Die installierten Kameras sind nur wenig mehr wert als überhaupt keine Kameras. Für Cloud Monitoring heißt das, dass jede einzelne Komponente in der physischen Infrastruktur und jeder einzelne ausgeführte Dienst überwacht werden muss. Zusätzlich müssen einem Cloud Monitoring alle bestehenden Zusammenhänge zwischen einzelnen Objekten und ganzen Objektgruppen bekannt sein.

1.4.5 Skalierbarkeit

Die Spannweite der Größe von Clouds ist enorm. Sie reicht von kleinen Infrastrukturen, auf denen nur wenige Dienste parallel ausgeführt werden, bis hin zu sehr großen Infrastrukturen mit mehreren tausend gleichzeitig laufenden Diensten. Ein konkretes Cloud Monitoring sollte zum Überwachen beider Extreme geeignet sein. Darüber hinaus muss es sich bei Bedarf komfortabel an eine veränderte Größe der Cloud anpassen lassen, weil sowohl die Anzahl von Nutzern als auch die Anzahl von Diensten mittel- und langfristig stark schwanken können (vor allem bei öffentlichen Clouds). Diese Art der (manuellen) Skalierbarkeit muss unbedingt erfüllt werden, aber sie entspricht noch nicht optimal dem Paradigma von Cloud Computing. Erst eine dynamische Skalierbarkeit durch ein selbstständiges Anpassen zur Laufzeit des Überwachungssystems an veränderte Größen ist robust und ökonomisch sinnvoll.

1.4.6 Steuerung

Das Steuern von überwachten Objekten ist zwar keine typische Aufgabe aus dem Bereich Monitoring, dennoch handelt es sich dabei um eine hinreichende Anforderung an Überwachungssysteme im Allgemeinen und an ein Cloud Monitoring im Speziellen. Die Größe und die Dynamik von Clouds führen bei einer soliden Überwachung zu vielen und schnell produzierten Ergebnissen, von denen ein großer Teil eine Reaktion erfordert. Menschen können die anfallenden Aufgaben weder zeitnah, noch im Hinblick auf Personalkosten vollständig erledigen. Aus diesen Gründen sollte ein Cloud Monitoring zusätzlich die von ihm produzierten Ergebnisse bewerten und entweder direkt oder indirekt darauf über Manipulationen in der überwachten Cloud reagieren können.

1.4.7 Vorhersagen

Oftmals zeichnen sich Probleme und Chancen in einer Cloud bereits ab, bevor sie vollständig entstanden sind und ihre Wirkungen entfalten. Damit eine Cloud optimal betrieben und genutzt werden kann, sollten von einem Cloud Monitoring erste Anzeichen für bestimmte Situationen sehr früh erkannt und (wahrscheinliche) zukünftige Ergebnisse bereits im Voraus produziert werden können. Mit Hilfe von zuverlässigen Vorhersagen können zum Beispiel durch Probleme verursachte Schäden nicht nur minimiert, sondern komplett vermieden werden.

1.5 Aufbau der Arbeit

In Kapitel 2 werden alle zum Verständnis der Arbeit notwendigen Begriffe, Konzepte, Methoden und Technologien vorgestellt. Auf diesen Grundlagen aufbauend werden in Kapitel 3 ein allgemeines, auf Datenströmen und CEP basierendes Design von Überwachungssystemen für Clouds und Implementierungsdetails sowie Anwendungsbeispiele des im Rahmen dieser Arbeit entwickelten Prototypen präsentiert. Mit den Ergebnissen aus Kapitel 3 steht eine solide Basis zum Überwachen und Steuern von Clouds bereit, die in vielfältiger Weise bezüglich der konkreten Risiken von Cloud Computing erweitert werden kann. Kapitel 4 demonstriert anhand einer dynamischen und proaktiven Lastbalancierung den Einsatz des Prototypen dieser Arbeit zur Unterstützung und Ermöglichung dringend benötigter Problemlösungen im Bereich Cloud Computing. In Kapitel 5 werden die Ergebnisse dieser Arbeit kompakt zusammengefasst und es wird ein umfassender Ausblick zu dem Thema Cloud Monitoring gegeben.

Zusammenfassung

Cloud Computing bedeutet dynamisches Bereitstellen von elastischen IKT-Diensten auf der Basis standardisierter Web-Technologien über ein Netzwerk zur Miete. Kritische Risiken in der Leistungsfähigkeit, Sicherheit und Verfügbarkeit der Dienste schließen viele Organisationen und Personen von Cloud Computing aus und stellen für bestehende Nutzer eine allgegenwärtige Gefahr dar. Zur Lösung von Problemen und zur Vermeidung der Risiken bedarf es leistungsfähiger Werkzeuge zum Überwachen und Steuern von Clouds. Die Anforderungen an ein Cloud Monitoring sind vielfältig und hoch.

2 Grundlagen

Dieses Kapitel erläutert alle zum Verständnis der weiteren Arbeit notwendigen Grundlagen. Nach einer ausführlichen Beschreibung von Cloud Computing in Abschnitt 2.1 folgt in Abschnitt 2.2 eine Einführung in die Verarbeitung von Datenströmen mit Datenstrommanagementsystemen. Darauf aufbauend wird in Abschnitt 2.3 Complex Event Processing als eine der wichtigsten Anwendungen dieser Systeme vorgestellt.

2.1 Cloud Computing

Der intuitiven Erklärung von Cloud Computing in Abschnitt 1.1 muss eine detailliertere Betrachtung folgen, um den zu überwachenden und steuernden Gegenstand präzise beschreiben zu können.

2.1.1 Definition

An dieser Stelle ist es zurzeit noch nicht möglich eine Standarddefinition wiederzugeben, da der Begriff Cloud Computing viele und zum Teil stark unterschiedliche Definitionen[1] – z. B. in [Arm+09], [McK09] und [MG09] – erfahren hat, von denen noch keine zum Standard erhoben wurde. Eine häufig zitierte und für den Zweck dieser Arbeit gut geeignete Definition stammt von dem für Standardisierungsprozesse in den Vereinigten Staaten verantwortlichen Institut NIST. In dieser Definition wird Cloud Computing anhand von drei Dienstklassen, vier Betriebsmodellen und fünf charakteristischen Eigenschaften beschrieben [MG09].

2.1.1.1 Dienstklassen

Die mit Hilfe von Cloud Computing angebotenen IKT-Dienste werden in disjunkten Dienstklassen zusammengefasst. Durch diese Verallgemeinerung sollen verschiedene Angebote leichter kategorisierbar und vergleichbar gemacht werden. Die einzelnen Dienstklassen sind hierarchisch in einer Schichtenarchitektur angeordnet und werden

[1] Allein in [Vaq+09] werden schon über 20 verschiedene Definitionen identifiziert.

deshalb auch als Schichten bezeichnet. Diese Hierarchie ist allerdings nicht streng, wie in Schichtenarchitekturen üblich. Das heißt, dass eine Schicht nicht nur auf die direkt unter ihr liegende Schicht, sondern auch auf alle niedrigeren Schichten zugreifen kann. Eine konkrete Umsetzung der Schichtenarchitektur muss nicht alle Schichten implementieren, sondern kann sich auf diejenigen beschränken, die tatsächlich benötigt werden. Manchmal ist es nicht möglich, einen Dienst eindeutig zu klassifizieren, weil eine Zuordnung zu mehreren Schichten denkbar ist. In solchen Fällen wird der Dienst der höchsten aller möglichen Schichten zugeordnet. Die NIST-Definition gibt insgesamt drei verschiedene Dienstklassen vor, die in Tabelle 2.1 zusammen mit jeweils stellvertretenden Diensten aufgeführt sind.

Dienstklasse	Typische IKT-Dienste
Infrastructure as a Service (IaaS)	Speicher, Maschinen, Netzwerke
Platform as a Service (PaaS)	Entwicklungs- und Laufzeitumgebungen
Software as a Service (SaaS)	Office-Anwendungen, CRM-Systeme

Tabelle 2.1: Dienstklassen

Infrastructure as a Service

Die IaaS-Schicht stellt die Basis in der Schichtenarchitektur dar. Zu ihr werden alle Dienste gezählt, die zum Aufbau von IKT-Infrastrukturen eingesetzt werden können. In Tabelle 2.1 sind als Beispiele Speicher, Maschinen und Netzwerke genannt. Ein herausragendes Merkmal von Diensten der IaaS-Schicht ist ihre Elastizität. Aufgrund von sehr feinen Granularitäten ist es möglich die nutzbaren (und damit auch die zu bezahlenden) Ressourcen exakt und dynamisch auf den Bedarf abzustimmen. Die Spannweite reicht dabei von sehr kleinen Infrastrukturen bestehend aus einer einzigen leichtgewichtigen Maschine bis hin zu einem kompletten Rechenzentrum mit vielen miteinander vernetzten Maschinen und mehreren Terabytes an Speicher. Zwischen diesen Extremen ist jede Zwischenstufe möglich. Da die Elastizität unter Verwendung von physischer Hardware weder dynamisch noch in Echtzeit möglich ist, werden die angebotenen Ressourcen virtualisiert zur Verfügung gestellt. Zusätzlich ergibt sich dadurch für die Nutzer eine einheitliche und von konkreter Hardware losgelöste Sicht auf ihre Ressourcen sowie für den Betreiber eine optimale Auslastung seiner Infrastruktur.

Platform as a Service

Auf die IaaS-Schicht setzt die PaaS-Schicht auf. Zu ihr zählen alle Dienste, die eine fertige Entwicklungs- oder Laufzeitumgebung für Anwendungen anbieten. Neben der Tatsache, dass sich der Konfigurations- und Administrationsaufwand für einen Nutzer auf minimalem Niveau befinden, sind die ständige und weltweite Verfügbar-

keit sowie eine nutzungsabhängige Abrechnung weitere Vorteile von Plattformdiensten in einer Cloud. Die Umgebungen sind meistens für einen bestimmten Zweck bestimmt und darauf optimal zugeschnitten. Entwicklungsumgebungen (z. B. das Django Framework [DF]) unterstützen dabei nur ganz bestimmte Programmiersprachen und beinhalten alle notwendigen Werkzeuge und Bibliotheken für die Entwicklung. Die Laufzeitumgebungen ermöglichen das Ausführen eigener (Web-)Anwendungen in einer Cloud. Alle angebotenen Umgebungen können sich in einer mit Hilfe von Diensten der IaaS-Schicht aufgebauten Infrastruktur befinden, wodurch sie ebenfalls dynamisch skalierbar werden.

Software as a Service

Den Abschluss in der NIST-Definition bildet die SaaS-Schicht. In ihr werden für die Endnutzer bestimmte Anwendungen als Dienste angeboten. Als Beispiele sind in Tabelle 2.1 Office-Anwendungen wie Text- oder Tabellenverarbeitung und Unternehmensanwendungen für die Verwaltung und Dokumentation von Kundenbeziehungen (engl. *Customer Relationship Management*, CRM) genannt. Sowohl der Anbieter einer Anwendung als auch die Nutzer haben nahezu keinen Administrationsaufwand mehr. Der Anbieter muss die Anwendung nur einmal installieren und einrichten. Das Gleiche gilt für Aktualisierungen. Die Nutzer haben dann jederzeit und von überall aus Zugriff auf die Anwendung, die nicht mehr aufwändig lizensiert werden muss, sondern unbürokratisch gemietet wird. Anwendungen können in einer aus der PaaS-Schicht stammenden Umgebung oder direkt in einer Infrastruktur aus der IaaS-Schicht ausgeführt werden, um die positiven Eigenschaften einer Cloud zu erben.

Zusammenspiel der Dienstklassen

Die drei Dienstklassen wurden in den vorhergehenden Absätzen weitestgehend isoliert voneinander betrachtet. Nachfolgend soll an einem realistischen Anwendungsfall das mögliche Zusammenspiel der Klassen demonstriert werden. Betrachtet wird der Entwicklungsprozess einer Anwendung über ihren gesamten Lebenszyklus hinweg in einer Cloud. Am Anfang wird von allen beteiligten Entwicklern eine zu ihren Bedürfnissen passende Entwicklungsumgebung gemeinsam benutzt. Nach Fertigstellung der Anwendung wird sie in einer Laufzeitumgebung, die ihre Ressourcen dynamisch aus der IaaS-Schicht bezieht, veröffentlicht. Abschließend wird die Anwendung als Dienst in der SaaS-Schicht auf der Basis eines verbrauchsabhängigen Kostenmodells zur Verfügung gestellt. Durch die Nutzung der IaaS-Schicht ist sichergestellt, dass die Anwendung stets über die richtigen Kapazitäten verfügt, um alle Nutzer optimal bedienen zu können und um die Betriebskosten für den Anbieter der Anwendung zu minimieren. Cloud Computing verändert damit nicht nur die Art, wie Ressourcen angeboten und genutzt werden, sondern auch wie Anwendungen entwickelt und vertrieben werden.

2.1.1.2 Betriebsmodelle

Bisher wurde implizit davon ausgegangen, dass es sich bei einer Cloud um eine öffentlich zugängliche Infrastruktur im Internet handelt. Neben den öffentlichen Clouds gibt es noch weitere Arten eine Cloud zu betreiben und zu nutzen.

Public Cloud
Das populärste und effizienteste Betriebsmodell stellt eine Public Cloud dar. Sie ist dadurch gekennzeichnet, dass Betreiber und Nutzer verschiedene Personen oder Organisationen sind. Die Infrastruktur befindet sich im Internet und ist jedem zugänglich. Durch einen sehr großen Nutzerkreis sind Dienste in einer Public Cloud aufgrund von Skaleneffekten in der Regel am kostengünstigsten im Vergleich zu den anderen Betriebsmodellen. Auf der Seite der Betreiber können Organisationen mit sehr großen Rechenzentren nicht selbst benötigte Kapazitäten an Dritte verkaufen, anstatt sie ungenutzt Kosten verursachen zu lassen. In Tabelle 2.2 sind eine Auswahl von Betreibern von zurzeit bekannten und großen Public Clouds zusammen mit den adressierten Schichten sowie Auszügen aus ihren Produktangeboten aufgelistet.

Betreiber	Schicht	Produkte
3tera [Ter]	IaaS	Maschinen, Speicher
	SaaS	CRM, Projektmanagement
Amazon [AWS; AEB]	IaaS	Maschinen, Netzwerke, Speicher
	PaaS	Laufzeitumgebungen
Bungee Connect [BC]	PaaS	Entwicklungs– und Laufzeitumgebungen
Dropbox [Droa]	IaaS	Speicher[2]
GoGrid [GG]	IaaS	Speicher
Google [Gg1; Gg2; Gg3; Gg4]	IaaS	Speicher
	PaaS	Laufzeitumgebungen
	SaaS	E-Mail, Kartographie, Office
Microsoft [WA; WL]	PaaS	Entwicklungs– und Laufzeitumgebungen
	SaaS	Office
Nirvanix [Nir]	IaaS	Speicher
Rackspace [Rac]	IaaS	Maschinen, Speicher
Salesforce [Sal]	PaaS	Entwicklungs– und Laufzeitumgebungen
	SaaS	CRM
Zoho [Zoh]	SaaS	CRM, Office, Projektmanagement

Tabelle 2.2: Auswahl von Betreibern einer Public Cloud

[2]Streng genommen bietet Dropbox keinen eigenen Speicher an, sondern baut auf den Speicherdienst von Amazon auf, um die eigenen Dienste wie z. B. Dateisynchronisation anbieten zu können [Drob].

Private Cloud

Eine Private Cloud stellt das genaue Gegenteil einer Public Cloud dar. Der Betreiber ist zugleich auch der einzige Nutzer. Sie befindet sich meistens in einem gut geschützten Intranet und ist nicht öffentlich zugänglich. Der Hauptgrund für den Einsatz einer Private Cloud sind Sicherheitsbedürfnisse, weil bei diesem Betriebsmodell die Daten und die Kontrolle darüber bei dem Besitzer verbleiben und Fremde nicht auf die Infrastruktur zugreifen können. Zum Beispiel könnte ein Unternehmen, das nicht auf sein eigenes Rechenzentrum verzichten möchte oder kann, dieses in eine Private Cloud umwandeln und dadurch seinen Abteilungen und Mitarbeitern zukünftig IKT-Dienstleistungen flexibel, dynamisch skalierbar und bedarfsgesteuert anbieten. Der Aufbau einer Private Cloud zur vollständigen oder ergänzenden Nutzung in einem privaten Rechenzentrum ist aufgrund vieler quelloffener Produkte, die teilweise die Schnittstellen von großen Public Clouds implementieren und dadurch besonders zum hybriden Einsatz geeignet sind, einfach und kostengünstig möglich [Bau+11a]. In Tabelle 2.3 sind eine Auswahl von verbreiteten und quelloffenen Produkten zusammen mit den Schichten, zu deren Aufbau sie gedacht sind, aufgelistet. In der Spalte „Kompatibel zu“ ist festgehalten, ob die Schnittstellen einer Public Cloud in dem Produkt implementiert sind.

Produkt	Schicht	Kompatibel zu
Apache Hadoop [AH]	PaaS	–
AppScale [AS]	PaaS	Google
CloudStack [CS]	IaaS	Amazon
Eucalyptus [Euc]	IaaS	Amazon
Nimbus [Nim]	IaaS	Amazon
OpenNebula [ON]	IaaS	Amazon
OpenStack [OS]	IaaS	Amazon
TyphoonAE [TAE]	PaaS	Google

Tabelle 2.3: Auswahl von quelloffenen Produkten zum Aufbau von Clouds

Community Cloud

Bei einer Community Cloud handelt es sich im Kern um eine Private Cloud, die allerdings nicht nur von einer einzigen Organisation oder Person betrieben und genutzt wird, sondern von mehreren gemeinsam. Beispielsweise könnten sich alle Hochschulen eines Bundeslandes eine gemeinsame Cloud teilen.

Hybrid Cloud

Eine Hybrid Cloud liegt dann vor, wenn Dienste aus Clouds mit unterschiedlichen Betriebsmodellen parallel genutzt werden. Es gibt mindestens zwei typische Situatio-

nen, die die Nutzung einer Hybrid Cloud nahelegen. Zum einen könnte die Anwesenheit von sehr sensiblen Daten die vollständige Nutzung einer Public oder Community Cloud verhindern, sodass zumindest für diese Daten eine Private Cloud betrieben werden muss. Zum anderen könnte die eigene Private oder Community Cloud nicht in der Lage sein, genügend Kapazitäten in Zeiten hoher Arbeitslast aufzubringen. Dann würden für diese Zeiträume die zusätzlich benötigten Kapazitäten aus einer Public Cloud bezogen werden. Besonders komfortabel ist die gleichzeitige Nutzung verschiedener Clouds, wenn diese identische Schnittstellen aufweisen (siehe dazu Tabelle 2.3).

2.1.1.3 Charakteristische Eigenschaften

Zum Abschluss der Definition können nun die fünf kennzeichnendsten Merkmale von Cloud Computing übersichtlich zusammengefasst werden.

Rasche Elastizität
Dienste in einer Cloud sind nahezu beliebig (d.h. in sehr feinen Granularitäten und ohne für den Nutzer spürbare Grenzen) skalierbar und passen sich schnell und automatisch dem Bedarf der Nutzer an.

Zugang über ein Netzwerk
Eine Cloud und ihre Dienste sind über ein Netzwerk erreichbar und setzen standardisierte Web-Technologien ein. Dadurch werden sie einem möglichst großen Nutzerkreis und ohne technische Barrieren zugänglich.

Selbstbedienung
Die Dienste stehen auf Anforderung der Nutzer zeitnah und automatisch bereit. Es gibt keine menschliche Interaktion mit dem Betreiber ihrer Cloud.

Pooling von Ressourcen
Innerhalb einer Cloud werden Ressourcen in großen Pools zusammengefasst und parallel von mehreren Nutzern verwendet, indem aus diesen Pools jedem Nutzer die von ihm benötigten Kapazitäten zur Verfügung gestellt und an seinen Bedarf angepasst werden.

Messbarkeit der Dienste
Die Nutzung von Diensten ist exakt messbar, da sie die Grundlage für die Abrechnung ist. Durch eine genaue Quantifizierung wird der Verbrauch sowohl für die Nutzer als auch für den Betreiber transparent. Des Weiteren sind Dienstgütevereinbarungen nur dann möglich, wenn die darin festgelegten Werte überprüfbar sind.

2.1.1.4 Randbemerkungen

Ergänzend zu der eben wiedergegebenen Definition werden einige Randbemerkungen in [MG09] gemacht, von denen einer im Hinblick auf die Entwicklung einer ganzheitlichen und flexiblen Überwachung besondere Beachtung geschenkt werden muss. Sie besagt, dass die drei vorgegebenen Dienstklassen nicht endgültig sind, sondern um weitere Klassen erweitert werden können. Wann immer sich Dienste nicht einer vorhandenen Klasse zuordnen lassen, ist es erlaubt, eine weitere Klasse als Schicht in die Architektur einzufügen. Beispiele für Dienste, die sich nicht auf natürliche Weise in eine der drei vorgegebenen Klassen einordnen lassen, sind zum Beispiel Computerspieldienste oder Druckdienste. In aktuellen Darstellungen des Schichtenmodells befindet sich bereits eine neue Dienstklasse über der SaaS-Schicht. Diese Klasse wurde als Human as a Service (HaaS) benannt und fasst alle Dienste zusammen, die zwar über eine Cloud bezogen werden, aber von Menschen erbracht werden müssen, weil Maschinen (noch) nicht in der Lage dazu sind (z. B. Erkennen von Personen auf Überwachungsbildern oder Entwerfen eines ansprechenden Designs).

Wegen der Unvollständigkeit der vorgegebenen Schichten hat sich eine Art Superklasse (Everything as a Service, XaaS) etabliert, von der neue Dienstklassen bei Bedarf abgeleitet werden können. Durch diese Modellierung werden die hohe Dynamik und Unvollständigkeit im Schichtenmodell deutlich. Dieser Punkt sollte unbedingt von einem Cloud Monitoring in Form von hoher Flexibilität und einfacher Erweiterbarkeit berücksichtigt werden.

2.1.1.5 Kritik

Obwohl die NIST-Definition schon sehr umfangreich ist, wird als Kritik häufig das Fehlen weiterer wichtiger Aspekte geäußert. Einige oft genannte Themen werden in [McI10] aufgelistet.[3] Dort wird bemängelt, dass der Sicherheit von Cloud Computing in der Definition zu wenig Beachtung geschenkt wird. Konkret angesprochen werden die Punkte Katastrophenmanagement, Sicherungsstrategien für gespeicherte Daten sowie Verfügbarkeit und Verlässlichkeit der Dienste. Dass alle diese Punkte auch eine hohe praktische Relevanz haben und damit berechtigt sind, in eine Definition mit aufgenommen zu werden, zeigen immer wieder Zwischenfälle in der Realität wie z. B. der dreitägige Ausfall [FTD] mit teilweisem Datenverlust [CW] einer der weltweit größten Public Clouds.

[3] Die Kritik an der NIST-Definition trifft auch auf die meisten anderen Definitionen zu.

2.1.2 Basistechnologien

Nach der Definition müssen für ein besseres Verständnis die wichtigsten Konzepte und Technologien vorgestellt werden, auf denen Cloud Computing basiert [Bau+11b]. Allen ist gemein, dass sie schon vor der Zeit von Cloud Computing existierten. Cloud Computing stellt damit streng genommen keine eigenständige und neue Technologie dar, sondern ist das Produkt einer neuartigen Kombination von bereits bekannten und etablierten Technologien.

2.1.2.1 Service-orientierte Architektur

Um die IKT in einem Unternehmen flexibler, schneller und kostengünstiger an Geschäftsprozesse anpassen zu können, wurde die Service-orientiere Architektur (engl. *Service Oriented Architecture*, SOA) als abstraktes Architekturmodell entwickelt [SN96]. In der SOA werden Geschäftsprozesse, die durch die IKT unterstützt werden sollen, in kleinere Komponenten, die von einer einzelnen Funktion[4] bis zu einem kompletten Geschäftsprozess jede Zwischengröße umfassen können, zerlegt. Diese Komponenten werden dann auf IKT-Dienste abgebildet und verteilt in dem Intranet des Unternehmens abgelegt und zur Verfügung gestellt. Der gesamte IKT-unterstütze Geschäftsprozess ergibt sich dann als Komposition von Diensten, die erst im Bedarfsfall dynamisch geladen und gebunden werden. Die Vorteile dieser Architektur sind die Wiederverwendbarkeit eines Dienstes in mehreren unterschiedlichen Geschäftsprozessen, der einfache, schnelle und kostengünstige Aufbau neuer IKT-gestützter Geschäftsprozesse durch Verwendung bereits vorhandener Dienste, die Optimierung bestehender Geschäftsprozesse durch Reorganisation der beteiligten Dienste (Business Process Redesign, BPR) sowie die Integration eines kompletten Geschäftsprozesses, der selbst als Dienst zur Verfügung steht, in andere Geschäftsprozesse [BL03].

Weil die Dienste per Definition verteilt und heterogen (Implementierung in unterschiedlichen Programmiersprachen und Ausführung auf verschiedenen Plattformen) sind, muss die Kommunikation zwischen und mit ihnen auf der Grundlage von standardisierten Schnittstellen beruhen, damit sie nutzbar, kombinierbar und austauschbar werden. Die SOA kann auf verschiedene Arten umgesetzt werden [Kre08; WS04]. Die einfachste Variante basiert auf direkten Punkt-zu-Punkt Verbindungen (engl. *Peer-to-Peer*, P2P). Dabei sind entweder alle Dienste und ihre Adressen im voraus bekannt oder es gibt ein zentrales Verzeichnis, in dem nach passenden Diensten und Adressen gesucht werden kann. Um die Anzahl der physischen Verbindungen zu reduzieren, kann ein zentraler Vermittler verwendet werden. Für eine leichtere Integration der Dienste hat sich der Einsatz einer Middleware als zentraler Vermittler etabliert. Diese Middleware wird in der SOA als Enterprise Service Bus (ESB)

[4] In der Praxis wird häufig eine untere Schranke für die Granularität der Komponenten festgelegt.

[Cha04] bezeichnet und übernimmt neben der Vermittlung von Nachrichten noch weitere Aufgaben. Zu diesen zählen unter anderem die Konvertierung von abweichenden Nachrichtenformaten in einen einheitlichen Standard, die Bereitstellung des angesprochenen Verzeichnisses mit allen Diensten und ihren Adressen sowie die Komposition von Diensten. Ein weiterer Vorteil des ESB ist, dass Dienste und Anwendungen über unterschiedliche Kanäle und Protokolle miteinander kommunizieren können. In Abbildung 2.1 ist dieser Zusammenhang grafisch dargestellt. Mehrere Dienste und Anwendungen sind mit einem ESB über unterschiedliche Protokolle (FTP [RFCe], HTTP [RFCa], JMS [JSRc], SMTP [RFCb]) verbunden und können über ihn miteinander Nachrichten austauschen.

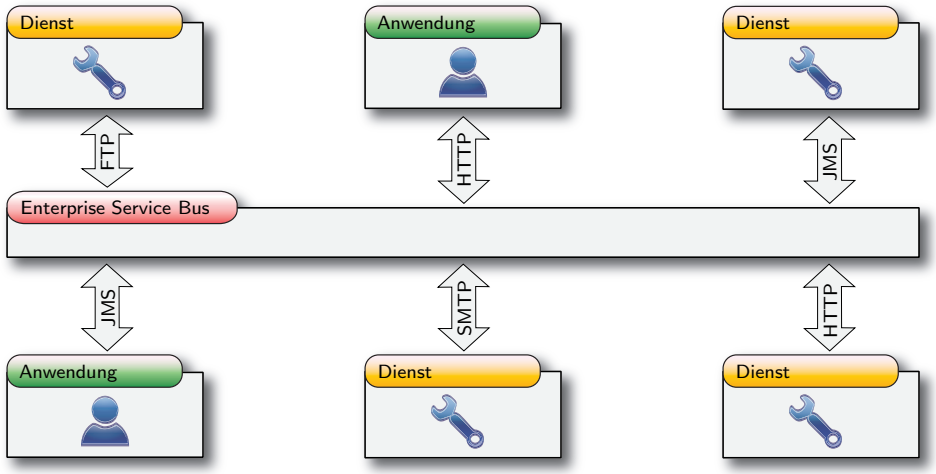

Abbildung 2.1: Enterprise Service Bus

Die Cloud Computing zugrunde liegende Architektur ist die SOA, allerdings ohne zwingende Orientierung der Dienste an Geschäftsprozessen. Alle anderen Merkmale der SOA finden sich in einer Cloud wieder: Die heterogenen IKT-Dienste sind der zentrale Baustein, die Dienste sind in einem Netzwerk verteilt, die Kommunikation zwischen und mit Diensten erfolgt über Nachrichten und die Dienste sind austauschbar, kombinierbar und wiederverwendbar.

2.1.2.2 Web Services

Die für Standards im Web verantwortliche Organistation W3C definiert einen Web Service [W3Ca] als eine durch einen URI eindeutig identifizierbare Anwendung, deren öffentliche Schnittstellen mit Hilfe von XML definiert und beschrieben sind. Seine Definition kann von anderen Anwendungen gefunden werden. Diese Anwendungen können dann mit dem Web Service über Internet-Protokolle XML-Nachrichten aus-

tauschen und auf die in seiner Definition festgelegten Art und Weise interagieren.

Verzeichnisdienst:	UDDI			
Schnittstellenbeschreibung:	WSDL			
Nachrichtenprotokoll:	SOAP			
Transportprotokoll:	HTTP	SMTP	FTP	...

Abbildung 2.2: Basis-Stack von Big Web Services

Web Services basieren auf einem Stapel von Standards (dem sogenannten Basis-Stack [KL04]), der in Abbildung 2.2 dargestellt ist. An der Spitze des Stapels befindet sich der Verzeichnisdienst UDDI (Universal Description, Discovery and Integration, [UDD]), über den Web Services und Metadaten über sie wie Eigenschaften, Voraussetzungen und Informationen über den Anbieter gefunden werden können. Bei der Nutzung von Web Services besteht der erste Schritt darin, im UDDI einen passenden Dienst und seine durch einen URI repräsentierte Adresse zu finden. Dieser Schritt kann übersprungen werden, wenn der Web Service und sein URI bereits bekannt sind. Weil sich UDDI nicht durchsetzen konnte [IWb], ist dieses Vorgehen mittlerweile üblich. Nachdem ein passender Web Service und seine Adresse gefunden wurden, muss ermittelt werden, wie der Web Service zu benutzen ist. Dazu sind seine Schnittstellen exakt mit der Sprache WSDL (Web Services Description Language [W3Cd]) in Form von XML beschrieben. Schließlich kann der Web Service über den Austausch von XML-Nachrichten mit SOAP (Simple Object Access Protocol [W3Cc])[5] verwendet werden. SOAP ist ein vollständiges Nachrichtenprotokoll für den Austausch von Daten und für Aufrufe von Prozeduren. Für den Transport der Nachrichten wird dabei auf ein standardisiertes Transportprotokoll des Internets zurückgegriffen. Beispielsweise können HTTP (Normalfall), SMTP (für Asynchronizität) oder FTP (für große Datenmengen) verwendet werden.

Neben dem Aufbau von Web Services durch WSDL/SOAP, den sogenannten Big Web Services, gibt es mit den RESTful Web Services einen zweiten populären und für Cloud Computing wichtigen Architekturstil für Web Services [RR07]. REST (Representational State Transfer, [Fie00]) baut direkt auf dem zustandslosen und dadurch effizienten HTTP auf und nutzt dessen generische Methoden `GET` für das Anfragen des aktuellen Zustands einer Ressource, `PUT` für das Anlegen neuer Ressourcen, `POST` für das Ändern des Zustands einer Ressource und `DELETE` für das Löschen einer Ressource [Bay02]. Die Schnittstelle der Dienste ist somit im Voraus klar definiert. Allerdings müssen sich die Nutzer und ein Dienst über die Kodierung der auszutauschenden Nachrichten einig sein, da bei REST kein Format festgelegt ist. Aber auch bei den RESTful Web Services ist XML das gängigste Nachrichtenformat.

[5] In [W3Cc] wird SOAP nicht mehr als Akronym, sondern als eigenständiger Begriff verwendet.

Die Unterschiede zwischen Big Web Services und RESTful Web Services sind groß und nicht immer ist es leicht für eine konkrete Anwendung die bessere der beiden Alternativen zu bestimmen. REST ist gut für eine schnelle und einfache Integration von Diensten geeignet und WSDL/SOAP bietet mehr Flexibilität und nimmt mehr Rücksicht auf die Güte von Diensten [PZL08]. Web Services eignen sich zur Umsetzung der SOA und im Fall von Cloud Computing sind die Dienste als Big oder RESTful Web Services im Stil der SOA umgesetzt. An dieser Stelle muss nachgetragen werden, dass die SOA als ein Architektur-Paradigma an keine konkreten Technologien gebunden ist. Die SOA kann auch durch andere Technologien als Web Services implementiert werden und umgekehrt ist nicht jede Ansammlung von Web Services eine Umsetzung der SOA [Nat03].

2.1.2.3 Virtualisierung

In allen Schichten einer Cloud werden Virtualisierungstechniken eingesetzt, um die Elastizität von Diensten und den Eindruck potentiell unbegrenzter Kapazitäten erreichen zu können. Durch Virtualisierung können physische Ressourcen simuliert, einzelne (physische oder virtuelle) Ressourcen auf mehrere virtuelle oder umgekehrt mehrere (physische oder virtuelle) Ressourcen auf eine virtuelle abgebildet werden. Die beiden Abbildungen können auch kombiniert werden (siehe dazu Pooling in Abschnitt 2.1.1.3). In einem ersten Schritt werden alle physischen Ressourcen in einer großen virtuellen Ressource zusammengefasst. Diese wird dann im Anschluss auf mehrere virtuelle Ressourcen aufgeteilt, die jeweils einer Wunschgröße entsprechen. Das Thema Virtualisierung und die Liste aller zurzeit verfügbaren Techniken sind sehr umfangreich. Im Folgenden wird daher für jede Schicht einer Cloud nur eine Auswahl der wichtigsten Virtualisierungstechniken vorgestellt [BM11; Bau+11b].

Speicher- und Netzwerkvirtualisierung
Für Cloud Computing sehr wichtig sind die Speicher- und die Netzwerkvirtualisierung. Bei der Speichervirtualisierung wird physischer Speicher logisch so zur Verfügung gestellt, dass er den Nutzerbedürfnissen optimal in Bezug auf Größe, Geschwindigkeit und Redundanz entspricht. Beispielsweise möchte ein Nutzer, der mehrere Terabytes an Daten speichern muss, sich weder um die Verwaltung und optimale Nutzung vieler Festplatten noch um die Sicherung seiner Daten selbst kümmern. Durch Speichervirtualisierung (z. B. mit Hilfe verteilter Dateisysteme) ist es möglich, viele einzelne Festplatten auf eine einzige logische abzubilden. Das Anlegen von Sicherungen sowie eine optimale Verteilung der Daten auf einzelne Festplatten geschieht vor den Nutzern verborgen im Hintergrund. Wächst oder sinkt der Bedarf an Speicherplatz, dann kann der logische Speicher dynamisch angepasst werden. Bei der Netzwerkvirtualisierung wird ein logisches lokales Netzwerk (engl. *Local Area Network*, LAN) zur Verfügung gestellt, das sich wie ein physisches LAN verhält. Es ist

in sich abgeschlossen und nach außen isoliert. Virtuelle Netzwerke entstehen entweder aus der Zerteilung eines einzelnen physischen Netzwerks oder aus dem Zusammenschluss mehrerer Teilnetze aus verschiedenen physischen Netzen. Durch die erste Variante kann eine Infrastruktur aufgeteilt werden und mit Hilfe der zweiten Variante können voneinander getrennte und möglicherweise auch räumlich weit auseinander liegende Netzwerke zusammengeschlossen werden.

Plattformvirtualisierung

Durch die Virtualisierung von ganzen Plattformen ist es möglich, ein einzelnes physisches Computersystem auf mehrere virtuelle aufzuteilen und diese parallel zu verwenden. Dazu wird jede Plattform auf einer eigenen virtuellen Maschine (engl. *Virtual Machine*, VM) ausgeführt. Um die Verwaltung aller virtuellen Maschinen auf einer physischen Maschine und um die Zuteilung der Ressourcen kümmert sich ein sogenannter Hypervisor. Der Hypervisor arbeitet entweder direkt auf der Hardware[6] (Para-Virtualisierung) oder er läuft als Anwendung[7] – dem sogenannten VM Monitor (VMM) – unter einem gängigen Betriebssystem (vollständige Virtualisierung). Abbildung 2.3 verdeutlicht die unterschiedlichen Konzepte.

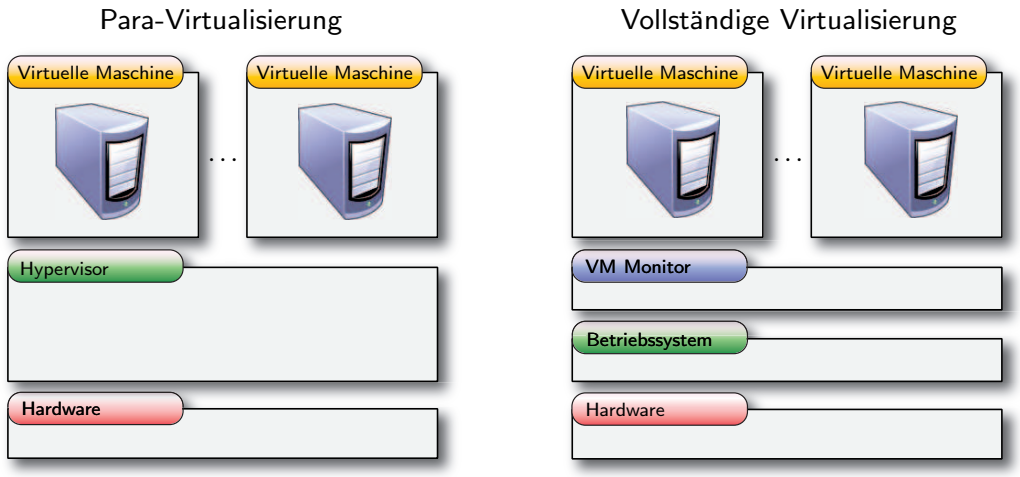

Abbildung 2.3: Plattformvirtualisierung

Bei einer vollständigen Virtualisierung wird den einzelnen virtuellen Maschinen jeweils ein komplettes und simuliertes Computersystem zur Verfügung gestellt. Da sich bestimmte Hardwarekomponenten nicht aufteilen lassen (beispielsweise eine Netzwerkkarte), kommt neben der Virtualisierung noch die Emulation von Hardware zum Einsatz. Ein populärer Hypervisor zur vollständigen Virtualisierung im

[6] Typ-1-Hypervisor
[7] Typ-2-Hypervisor

Linux-Umfeld ist das freie Produkt KVM[8] (Kernel-based Virtual Machine [KVM]). Das für Cloud Computing wichtigere Konzept zur Plattformvirtualisierung ist die Para-Virtualisierung. Hierbei werden Hardwarekomponenten weder simuliert noch emuliert. Stattdessen wird den virtuellen Maschinen eine API von dem Hypervisor, der ohne zusätzliches Betriebsystem auskommt, zur Verfügung gestellt, über die sie direkt auf die Hardware zugreifen können. Einer der am häufigsten verwendeten freien Hypervisor zur Para-Virtualisierung ist Xen [Xen]. Im kommerziellen Umfeld ist im Bereich der Plattformvirtuailiserung vor allem das Unternehmen VMware [VMW] zu nennen, das sowohl Produkte für vollständige Virtualisierung als auch für Para-Virtualisierung im Angebot hat.

Anwendungsvirtualisierung

Im Rahmen von Anwendungsvirtualisierung wird eine Anwendung auf einer VM ausgeführt. Alle von der Anwendung benötigten Ressourcen werden von der zugehörigen VM zur Verfügung gestellt. Die Vorteile von Anwendungsvirtualisierung sind Plattformunabhängigkeit und erhöhte Sicherheit, weil eine virtualisierte Anwendung keinen Schaden an dem System, unter dem ihre VM läuft, oder an anderen Anwendungen verursachen kann. Das bekannteste Beispiel aus dem Bereich Anwendungsvirtualisierung ist die JVM (Java Virtual Machine) [LY99]. Eine in einer Hochsprache geschriebene Anwendung (z. B. in Java oder in Scala) wird von einem entsprechenden Übersetzer in sogenannten Bytecode transformiert, der für die Ausführung auf einer JVM bestimmt ist.[9]

2.2 Datenstrommanagementsysteme

Datenstrommanagementsysteme (engl. *Data Stream Management Systems*) entstanden aus dem Bedürfnis heraus, sehr große und strömende Datenmengen kontinuierlich und in Echtzeit zu verarbeiten, da sich in den letzten Jahren aktive Datenquellen, wie z. B. Sensornetzwerke auf Basis von RFID-Technologie, aufgrund drastisch fallender Preise von entsprechender Hardware immer mehr etabliert haben. Traditionelle Datenbankmanagementsysteme stellten sich für diese Aufgabe als unbrauchbar heraus. Anfragen sind dort nur ein einmaliges Ereignis und müssen explizit von außen (z. B. von einem Nutzer oder einer Anwendung) angestoßen werden. Wegen der Größe der Datenmengen sind die Verwendung von Externspeicher und das Anlegen von Indexstrukturen über die Daten notwendig. Dadurch lassen sich trotz ausgereifter

[8] Neben vollständiger Virtualisierung ist mit KVM mittlerweile auch Para-Virtualisierung möglich.

[9] Moderne Implementierungen der JVM wandeln Teile des Bytecodes zur Leistungssteigerung durch einen Echtzeit-Übersetzer [Sug+00] zur Laufzeit in nativen Maschinencode um, anstatt ihn zu interpretieren. Diese Optimierungen geschehen allerdings außerhalb der Abstraktion, sodass die Sicht eines Programmierers auf die JVM davon unberührt bleibt.

Optimierungstechniken Anfragen nicht mehr in Echtzeit beantworten. Datenbankmanagementsysteme eignen sich deshalb nur für mittel- und langfristige Analysen auf historischen Daten. In Tabelle 2.4 (adaptiert aus [Krä07]) werden die wichtigsten Unterschiede zwischen einem Datenbankmanagementsystem (DBMS) und einem Datenstrommanagementsystem (DSMS) übersichtlich zusammengefasst.

Aspekt	DBMS	DSMS
Datenquellen	passiv, endlich, persistent im Externspeicher	aktiv, potentiell unbeschränkt, flüchtig im Hauptspeicher
Anfrageausführung	einmalig auf der gegenwärtigen Datenbasis	kontinuierlich bei jeder Änderung der Datenbasis
Anfrageresultat	exakt	exakt oder approximativ
Anfrageverarbeitung	Bedarfsgetrieben durch den Benutzer	Datengetrieben durch die Eingabedaten
Anfrageoptimierung	statisch vor der Anfrageausführung	statisch vor der ersten Anfrageausführung, dynamisch während der kontinuierlichen Anfrageausführung

Tabelle 2.4: Gegenüberstellung von DBMS und DSMS

In einem DSMS sind die Rollen von Anfragen und Daten im Vergleich zu einem DBMS vertauscht. Die Daten fließen kontinuierlich durch das System und werden von den dort dauerhaft gespeicherten Anfragen auf Ausgabedatenströme abgebildet. Nachdem ein Eingabedatum vollständig verarbeitet wurde, wird es verworfen. Auf diese Weise muss nicht auf langsamen Externspeicher zurückgegriffen werden, sondern die gesamte Verarbeitung der Daten kann mit Hilfe des schnellen Hauptspeichers geschehen. Den Anstoß zur Anfrageausführung geben dabei die Daten bei ihrem Eintreffen in ein DSMS. Bei der Eingabe kann es sich um potentiell unbeschränkte Datenströme handeln, weswegen bei bestimmten Anfragen immer nur Zwischenergebnisse ausgegeben werden können, die eine Approximation des echten Ergebnisses darstellen. Da die Anfragen dauerhaft im System verbleiben, reicht es nicht aus sie nur einmalig zum Zeitpunkt ihrer Erstellung zu optimieren. Eine ganze Reihe von möglichen Änderungen des Kontexts während ihrer Lebenszeit machen eine kontinuierliche Optimierung notwendig [Rie08]. Datenstrommanagementsysteme sollen Datenbankmanagementsysteme nicht ersetzen, sondern sie um die fehlende Möglichkeit zur Verarbeitung großer und strömender Datenmengen in Echtzeit ergänzen.

2.2.1 Datenstromalgebra

Um mit einem DSMS Datenströme verarbeiten zu können, müssen in ihm Operatoren zur Manipulation der Datenströme implementiert werden. Aus diesen physischen Operatoren ergibt sich dann die Semantik von Anfragen, die durch Kombinieren von physischen Operatoren entstehen. Eine solche Semantik basiert allerdings auf keiner mathematischen Grundlage, wodurch Beweise wie z. B. das Aufzeigen von Äquivalenzen unmöglich werden, ist von einem Nutzer nur schwer nachzuvollziehen und ist in vielen Fällen nicht deterministisch. Aus diesen Gründen sollten die Operatoren auf einer mathematischen Ebene definiert werden. Die physischen Operatoren müssen die von diesen sogenannten logischen Operatoren festgelegte Semantik implementieren. Zur Beschreibung der Semantik können dann ausschließlich die logischen Operatoren herangezogen werden, die in der Sprache der Mathematik das Verhalten der entsprechenden physischen Operatoren und damit von konkreten Implementierungen beschreiben, ohne näher auf diese eingehen zu müssen. Ein weiterer Vorteil ist, dass zu einem einzelnen logischen Operator unterschiedliche physische Operatoren implementiert werden können, die sich alle gleich verhalten und von denen ein Nutzer nichts wissen muss. Dadurch werden Implementierungen ohne die Semantik zu verändern austauschbar (physische Anfrageoptimierung).

Im Rahmen dieser Arbeit wird das Datenstrommanagementsystem *PIPES* (Public Infrastructure for Processing and Exploring Streams) [KS04] eingesetzt, das mit Hilfe der *XXL-Bibliothek* (eXtensible and fleXible Library) [BDS00; Ber+01; Cam+03; XXL] implementiert wurde. Für *PIPES* existiert eine eindeutig definierte Semantik, die in [KS05] und [Krä07] vorgestellt wurde und in Anlehnung an die sehr erfolgreiche erweiterte relationale Algebra [Cod70; DGK82] entstand. Weil alle Operatoren von *PIPES* als Eingabe einen oder mehrere Datenströme erhalten und als Resultat wieder einen Datenstrom liefern, erzeugen sie eine Algebra mit Datenströmen als Operanden. *PIPES* wird eine zentrale Rolle in dem Prototypen dieser Arbeit einnehmen und deshalb gibt es einen starken Zusammenhang zwischen der Ausdrucksstärke von *PIPES* und der Mächtigkeit des Prototypen. Die Ausdrucksstärke von *PIPES* wiederum ergibt sich aus den zur Verfügung stehenden logischen Operatoren, die in den nächsten Abschnitten vorgestellt werden.

2.2.1.1 Datenströme

Da Datenströme die Operanden einer Datenstromalgebra sind, müssen sie exakt definiert sein. Ein Datenstrom ist eine zeitlich geordnete, endliche oder unbeschränkte Folge von Daten.[10] Die einzelnen Daten in einem Datenstrom haben eine beliebige aber feste Struktur und können mit Metadaten angereichert sein (Zeitstempel, Vielfachheit,

[10] Mit Hilfe von Punktuationen können auch zeitlich ungeordnete Ströme verarbeitet werden [Tuc+03].

etc.). Daher werden die einzelnen Datenstromelemente auch als Tupel bezeichnet. Beispiele für mögliche Daten mit einer festen Struktur sind relationale Datensätze oder XML-Dokumente. Im Kontext von *PIPES* gibt es insgesamt drei verschiedene Definitionen von einem Datenstrom. Jeweils eine dieser Definitionen ist für die Datenströme bestimmt, die in ein DSMS fließen (Rohdatenströme), auf denen die physischen Datenstromoperatoren arbeiten (physische Datenströme) und auf denen die logischen Datenstromoperatoren definiert werden (logische Datenströme).

Rohdatenströme
Als Rohdatenströme werden die Eingabedatenströme in ein DSMS bezeichnet. Ein einzelnes Rohdatenstromelement (p, t) besteht aus einem Datum p, welches auch als Payload bezeichnet wird, und einem Zeitstempel t. Der Zeitstempel t stammt aus einer diskreten Zeitdomäne T mit Ordnungsrelation: $t \in \mathbb{T} = (T, \leq)$. *PIPES* kann keine ungeordneten Ströme verarbeiten, sodass die Elemente eines Rohdatenstroms aufsteigend nach ihren Zeitstempeln geordnet vorliegen müssen. Der Payload p stammt aus einer beliebigen Menge Ω von homogen strukturierten Daten: $p \in \Omega$.

Physische Datenströme
Von den physischen Operatoren werden die sogenannten physischen Datenströme verarbeitet. Neben einem Payload $p \in \Omega$ haben die Elemente eines physischen Datenstroms ein halboffenes Zeitintervall $[t_{start}, t_{ende})$ mit $t_{start}, t_{ende} \in \mathbb{T}$ und sind primär nach t_{start} und sekundär nach t_{ende} geordnet. Das Intervall gibt dabei den Zeitraum an, in dem der Payload gültig ist. Ein Rohdatenstrom wird in einen physischen Datenstrom transformiert, indem das Gültigkeitsintervall der transformierten Tupel auf die kleinste darstellbare Zeiteinheit – ein sogenanntes Chronon – gesetzt wird. Jedes Tupel (p, t) des Rohdatenstroms wird auf ein Tupel $(p, [t, t + 1))$ des physischen Datenstroms abgebildet.

Logische Datenströme
Weil sich weder die Rohdatenströme noch die physischen Datenströme gut dazu eignen, eine Semantik zu definieren, wurden als dritte Art die logischen Datenströme eingeführt. Die Tupel eines logischen Datenstroms sind Tripel (p, t, n), wobei $p \in \Omega$ ein Payload und $t \in \mathbb{T}$ ein Zeitstempel (wie bei Rohdatenströmen) ist. Da bei einem Rohdatenstrom Duplikate nicht ausgeschlossen wurden, werden bei einem logischen Datenstrom alle Tupel mit identischem Payload p und Zeitstempel t auf ein einziges Tupel (p, t, n) abgebildet, bei dem $n \in \mathbb{N}$ die Vielfachheit der identischen Paare (p, t) anzeigt. Ein logischer Datenstrom lässt sich also gewinnen, indem für jeden Zeitpunkt jeweils alle identischen Payloads gezählt werden. Dieses Verfahren lässt sich bei Rohdatenströmen direkt anwenden. Bei physischen Datenströmen muss zuvor jedes Zeitintervall in alle in ihm enthaltenen einzelnen Zeitpunkte zerlegt werden, um so zu jedem Zeitpunkt im Intervall ein Tupel eines Rohdatenstroms erstellen zu können.

2.2.1.2 Schnappschuss-Reduzierbarkeit

Die (erweiterte) relationale Algebra hat sich über die letzten Jahrzehnte hinweg zu einem fundamentalen Konzept der Informatik entwickelt und dient daher vielfach als Referenz für neuere Arbeiten im Bereich der Anfrageverarbeitung. Ein Operator einer temporal-relationalen Algebra – wie z. B. ein Datenstromoperator auf Datenströmen mit relationalen Datensätzen als Payloads – wird als Schnappschuss-reduzierbar zu einem Operator der (erweiterten) relationalen Algebra bezeichnet, wenn bei Betrachtung von Schnappschüssen die identischen Eigenschaften vorliegen, wie bei dem Operator aus der (erweiterten) relationalen Algebra [Sno87]. Ein Schnappschuss eines logischen Datenstroms zu einem festen Zeitpunkt ist eine Multimenge, die aus allen Payloads besteht, die zu diesem Zeitpunkt gültig sind. Zu einem Datenstrom S aus der Menge aller logischen Datenströme $\mathbb{S}_\Omega$ über Ω wird ein Schnappschuss zum Zeitpunkt $t \in \mathbb{T}$ mit Hilfe der Funktion τ aus Definition 2.1 erzeugt.

$$\tau : \mathbb{S}_\Omega \times \mathbb{T} \to 2^{\Omega \times \mathbb{N}}, \text{ mit } \tau_t(S) := \left\{ \; (p, n) \mid (p, t', n) \in S \; \wedge \; t' = t \; \right\} \tag{2.1}$$

In Abbildung 2.4 wird der Zusammenhang zwischen einem relationalen Operator op_{rel} und einem temporal-relationalen Operator op_{temp} in Form eines kommutativen Diagramms verdeutlicht.

$$\begin{array}{ccc} S_1, \ldots, S_n & \xrightarrow{\tau_t} & R_1, \ldots, R_n \\ \Big\downarrow{\scriptstyle op_{temp}} & & \Big\downarrow{\scriptstyle op_{rel}} \\ S' & \xrightarrow[\tau_t]{} & R' \end{array}$$

Abbildung 2.4: Schnappschuss-Reduzierbarkeit

Der Operator op_{temp} ist genau dann Schnappschuss-reduzierbar zu dem Operator op_{rel}, wenn für alle Zeitpunkte t der Schnappschuss seiner Ausgabe das gleiche Resultat liefert, als wenn auf den Schnappschüssen seiner Eingaben der Operator op_{rel} angewandt worden wäre. Für beliebige logische Datenströme $S_1, \ldots, S_n$ mit relationalen Payloads als Eingabe muss die Gleichung 2.2 erfüllt werden.

$$\forall t \in \mathbb{T}.\ \tau_t \circ op_{temp} = op_{rel} \circ \tau_t \tag{2.2}$$

Der Vorteil einer Schnappschuss-reduzierbaren Algebra liegt darin, dass zum einen die Semantik den meisten Nutzern vertraut vorkommen wird, da die (erweiterte) relationale Algebra weit verbreitet und gut verstanden ist und dass zum anderen alle Äquivalenzen, die in der (erweiterten) relationalen Algebra gelten, automatisch auch in der Schnappschuss-reduzierbaren Algebra gelten [SJS01; Sli01].

2.2.1.3 Logische Datenstromoperatoren

Auf der Grundlage von logischen Datenströmen wird im Folgenden zu jedem Operator aus der erweiterten relationalen Algebra ausschließlich der Sortierung ein zu ihm Schnappschuss-reduzierbarer Operator zur Verarbeitung von Datenströmen definiert. Am Ende jeder Definition wird der jeweilige Ausdruck im SQL-Dialekt von *PIPES* angegeben, mit dem der Operator verwendet werden kann. Weil die Operatoren Schnappschuss-reduzierbar sind, gelten alle bekannten Äquivalenzen aus der erweiterten relationalen Algebra auch in der Datenstromalgebra. Abkürzungen wie die Verbünde (engl. *Joins*), die Maximum-Vereinigung, der Schnitt oder die strikte Differenz werden daher nicht extra aufgeführt, sondern können exakt auf die Weise wie in der erweiterten relationalen Algebra gebildet werden.

Selektion
Sei S ein logischer Datenstrom aus der Menge aller logischen Datenströme $\mathbb{S}_\Omega$ über einer beliebigen aber festen Menge Ω von homogen strukturierten Daten und b ein beliebiges Prädikat (Boolesche Funktion) aus $\mathbb{P}$, der Menge aller Prädikate. Bei einer Selektion σ auf S sollen nur diejenigen Daten $p \in \Omega$ an die Ausgabe weitergeleitet werden, die bestimmten Eigenschaften beschrieben durch b genügen. Die Struktur der Daten wird dabei nicht verändert, sodass auch der Ausgabedatenstrom aus der Menge $\mathbb{S}_\Omega$ stammt.

$$\sigma : \mathbb{S}_\Omega \times \mathbb{P} \to \mathbb{S}_\Omega$$
$$\sigma_b(S) := \{\ (p,t,n) \in S \mid b(p)\ \}$$

```
SELECT *
FROM   S
WHERE  b;
```

PIPES-SQL 2.1: Selektion mit Prädikat b auf Datenstrom S

Verallgemeinerte Projektion
Mit Hilfe der verallgemeinerten Projektion μ wird jedes Eingabedatum $p \in \Omega$ aus einem logischen Eingabedatenstrom $S \in \mathbb{S}_\Omega$ durch eine Abbildungsfunktion f aus der Menge aller Abbildungsfunktionen $\mathbb{F}$ zu exakt einem Ausgabedatum $f(p) = p' \in \Omega'$ transformiert. Weil sich die Vielfachheiten durch Anwendung von f ändern können, müssen diese in der Ausgabe neu berechnet werden. Darüber hinaus können die Ergebnisse von f eine andere Struktur als ihre Argumente haben.

$$\mu : \mathbb{S}_\Omega \times \mathbb{F} \to \mathbb{S}_{\Omega'}$$

$$\mu_f(S) := \Big\{ (p', t, n') \mid \exists X \subseteq S.\ X \neq \emptyset \wedge X = \big\{(p,t,n) \in S \mid f(p) = p'\big\} \wedge n' = \sum_{(p,t,n)\in X} n \Big\}$$

```
SELECT f
FROM   S;
```

PIPES-SQL 2.2: Projektion mit Abbildungsfunktion f auf Datenstrom S

Vereinigung

Um zwei logische Datenströme S_1 und S_2 über der gleichen Domäne Ω zusammenfließen zu lassen, wird die Vereinigung $\cup$ benötigt. Die Domäne der Ausgabedaten entspricht der Domäne Ω der Eingabedaten. Jedes Tupel aus den beiden Eingaben kommt genau einmal in der Ausgabe vor. Bei identischen Paaren (p, t) in beiden Eingaben muss die Vielfachheit in der Ausgabe durch Addition der entsprechenden Vielfachheiten aus den Eingaben angepasst werden.

$$\cup : \mathbb{S}_\Omega \times \mathbb{S}_\Omega \to \mathbb{S}_\Omega$$

$$\cup(S_1, S_2) := \left\{ (p, t, n_1 + n_2) \;\middle|\; \bigwedge_{i\in\{1,2\}} ((p,t,n_i) \in S_i \vee n_i = 0) \wedge \sum_{i\in\{1,2\}} n_i > 0 \right\}$$

```
SELECT * FROM S1
UNION
SELECT * FROM S2;
```

PIPES-SQL 2.3: Vereinigung der Datenströme S_1 und S_2

Kartesisches Produkt

Durch das kartesische Produkt $\otimes$ zwischen zwei beliebigen logischen Datenströmen $S_1 \in \mathbb{S}_\Omega$ und $S_2 \in \mathbb{S}_{\Omega'}$ wird jedes Datum aus einer der Eingaben mit allen Daten, die den gleichen Zeitstempel haben, aus der anderen Eingabe kombiniert und umgekehrt. Die Ausgabedaten haben eine neue Domäne Ω'', die durch Kombination der

Domänen Ω und Ω' der Eingaben entsteht. Der Operator $\oplus$ ist für das Kombinieren von zwei Daten $p_1 \in \Omega$ und $p_2 \in \Omega'$ verantwortlich. Es ist offensichtlich, dass die Vielfachheit eines Ausgabedatums dem Produkt der Vielfachheiten der entsprechenden Eingabedaten entspricht.

$$\otimes : \mathbb{S}_\Omega \times \mathbb{S}_{\Omega'} \to \mathbb{S}_{\Omega''}$$
$$\otimes(S_1, S_2) := \{\ (\oplus(p_1, p_2), t, n_1 \cdot n_2) \mid (p_1, t, n_1) \in S_1 \ \wedge\ (p_2, t, n_2) \in S_2\ \}$$

```
SELECT *
FROM   S1, S2;
```

PIPES-SQL 2.4: Kartesisches Produkt der Datenströme S_1 und S_2

Duplikateliminierung

Die Duplikateliminierung δ setzt die Vielfachheit jedes Datums $p \in \Omega$ aus einem logischen Eingabedatenstrom $S \in \mathbb{S}_\Omega$ auf 1. Dadurch werden mehrfache identische Daten zu einem Zeitpunkt entfernt.

$$\delta : \mathbb{S}_\Omega \to \mathbb{S}_\Omega$$
$$\delta(S) := \{\ (p, t, 1) \mid \exists n \in \mathbb{N}.\ (p, t, n) \in S\}$$

```
SELECT DISTINCT *
FROM            S;
```

PIPES-SQL 2.5: Duplikateliminierung auf Datenstrom S

Differenz

Der Ausgabedatenstrom, der durch die Differenz $-$ zwischen zwei logischen Datenströmen S_1 und S_2 über der gleichen Domäne Ω entsteht, enthält zu jedem Zeitpunkt alle Daten der ersten Eingabe S_1 abzüglich der Daten der zweiten Eingabe S_2. Für jedes Vorkommen eines Datums in der zweiten Eingabe S_2 wird die Vielfachheit des entsprechenden Datums der ersten Eingabe S_1 dekrementiert und durch Null nach unten beschränkt.

$$- : \mathbb{S}_\Omega \times \mathbb{S}_\Omega \to \mathbb{S}_\Omega$$

$$-(S_1, S_2) := \Big\{ (p,t,n) \;\Big|\; (\exists n_1, n_2 \in \mathbb{N}.\, n_1 > n_2 \wedge (p,t,n_1) \in S_1 \wedge (p,t,n_2) \in S_2 \wedge n = n_1 - n_2) \vee ((p,t,n) \in S_1 \wedge \nexists n_2 \in \mathbb{N}.\, (p,t,n_2) \in S_2) \Big\}$$

```
SELECT * FROM S1
EXCEPT
SELECT * FROM S2;
```

PIPES-SQL 2.6: Differenz zwischen den Datenströmen S_1 und S_2

Gruppierung

Bei der Gruppierung γ wird ein logischer Datenstrom S über Ω mit Hilfe einer Gruppierungsfunktion $g : \Omega \to \{1, 2, 3, \ldots, k\}$ aus der Menge aller Gruppierungsfunktionen $\mathbb{G}$ vollständig in k disjunkte Ausgabedatenströme $S_1, \ldots, S_k$ über Ω zerlegt. Jedes Datum aus der Eingabe wird genau einem logischen Ausgabedatenstrom zugeordnet. In *PIPES* sind keine beliebigen Gruppierungsfunktionen möglich, sondern es wird immer nach einer Teilmenge X der Struktur der Daten gruppiert. Alle Daten, die in X übereinstimmen, werden auf dieselbe Gruppe abgebildet.

$$\gamma : \mathbb{S}_\Omega \times \mathbb{G} \to \overbrace{\mathbb{S}_\Omega \times \ldots \times \mathbb{S}_\Omega}^{k \text{ mal}}$$

$$\gamma_g(S) := (S_1, \ldots, S_k), \text{ mit } S_i := \{ (p,t,n) \in S \mid g(p) = i \}$$

```
SELECT   X' ⊆ X
FROM     S
GROUP BY X;
```

PIPES-SQL 2.7: Gruppierung nach X auf Datenstrom S

Um aus dem Ergebnis der Gruppierung einen einzelnen Teildatenstrom extrahieren zu können, wird in Gleichung 2.3 die aus der Mathematik bekannte kanonische Projektion π verwendet.

$$\pi_j(\gamma_g(S)) = S_j \tag{2.3}$$

Aggregation
Mit Hilfe der Aggregation α können für einen Zeitpunkt t alle Daten aus einem logischen Eingabedatenstrom S über Ω zu einem einzigen Datum p' mit eigener Domäne Ω' verdichtet werden. Dazu wird eine Aggregationsfunktion $a : 2^{\Omega \times \mathbb{N}} \rightarrow \Omega'$ aus der Menge aller Aggregationsfunktionen $\mathbb{A}$ verwendet, die als Eingabe eine Menge von Paaren $(p \in \Omega, n \in \mathbb{N})$ erhält und daraus das Zieldatum p' berechnet. Um die Aggregationsfunktion auf S anwenden zu können, muss zuvor der Schnappschuss zu einem Zeitpunkt t erstellt werden.

$$\alpha : \mathbb{S}_{\Omega} \times \mathbb{A} \rightarrow \mathbb{S}_{\Omega'}$$
$$\alpha_a(S) := \left\{ (p', t, 1,) \mid \tau_t(S) \neq \emptyset \wedge p' = a(\tau_t(S)) \right\}$$

```
SELECT a
FROM   S;
```

PIPES-SQL 2.8: Aggregation mit Aggregat a auf Datenstrom S

Bis auf die Sortierung, die wegen der Annahme zeitlich geordneter Ein- und Ausgabedatenströme nicht benötigt wird, steht mit den Operatoren von *PIPES* eine mächtige Algebra zur Verarbeitung von Datenströmen zur Verfügung, die für relationalen Daten Schnappschuss-reduzierbar zur erweiterten relationalen Algebra ist.

2.2.1.4 Fensteroperatoren

In der Praxis kommt es bei einigen der vorgestellten Operatoren bei einer direkten Anwendung auf unbeschränkte Datenströme zu Problemen. Zu diesen Operatoren zählen alle diejenigen, die erst dann eine Ausgabe produzieren können, wenn alle Eingaben vollständig vorliegen (z. B. die Differenz). Bis dahin kommt es zu einer dauerhaften Blockierung der Ausgabe. Darüber hinaus werden die Eingabedatenströme ab einem bestimmten Zeitpunkt mehr Daten geliefert haben, als im Hauptspeicher abgelegt werden können. Um das Problem der Blockierung zu vermeiden und um eine Verarbeitung ausschließlich im Hauptspeicher zu ermöglichen, wird mit Hilfe von Fensteroperatoren immer nur ein endlicher Ausschnitt der Eingabedatenströme betrachtet. Neben diesen technischen Notwendigkeiten gibt es noch eine praktische Motivation für ihren Einsatz [GÖ03]. Die Daten von jüngeren Eingabetupeln haben meistens eine deutlich höhere Relevanz als die von älteren. Mit einem Fensteroperator ist es möglich, nur die jüngsten und damit relevantesten Daten zu verarbeiten.

Gleitendes Zeitfenster
Das gleitende Zeitfenster ω_w^{time} legt ein Zeitfenster der Größe w Zeiteinheiten auf einen logischen Datenstrom $S \in \mathbb{S}_\Omega$. Zu jedem Eingabetupel (p, t, n) aus S wird dazu das Zeitintervall $[t, t + w)$ berechnet und für jeden Zeitpunkt innerhalb dieses Intervalls wird das entsprechende Datum p durch Einfügen in den Ausgabedatenstrom auf gültig gesetzt.

$$\omega^{time} : \mathbb{S}_\Omega \times \mathbb{T} \to \mathbb{S}_\Omega$$

$$\omega_w^{time}(S) := \left\{ (p, t', n') \mid \exists X \subseteq S.X = \left\{(p, t, n) \in S \middle| \max(t' - w + 1, 0) \leq t \leq t'\right\} \right.$$
$$\left. \wedge\ X \neq \emptyset\ \wedge\ n' = \sum_{(p,t,n)\in X} n \right\}$$

```
SELECT *
FROM   S WINDOW(RANGE w);
```

PIPES-SQL 2.9: Gleitendes Zeitfenster der Dauer w auf Datenstrom S

Gleitendes Zählfenster
Während bei dem gleitenden Zeitfenster die Anzahl der Tupel zu einem festen Zeitpunkt im Ausgabedatenstrom durch die Zeit und damit variabel bestimmt ist, ist die Anzahl der Ausgabetupel bei dem gleitenden Zählenster konstant und wird von dem Nutzer vorgegeben. Das gleitende Zählfenster ω_N^{count} stellt für einen Datenstrom $S \in \mathbb{S}_\Omega$ zu jedem Zeitpunkt die jeweils $N \in \mathbb{N}$ jüngsten Tupel aus S bereit.

$$\omega^{count} : \mathbb{S}_\Omega \times \mathbb{N} \to \mathbb{S}_\Omega$$

$$\omega_N^{count}(S) := \left\{ (p, t', n') \mid \exists X \subseteq S.X = \{(p, t, n) \in S \mid \right.$$
$$\max(\kappa(t', S) - N + 1, 1) \leq \kappa(t, S) \leq \kappa(t', S)\}\ \wedge\ X \neq \emptyset$$
$$\left. \wedge\, n' = \sum_{(p,t,n)\in X} n \right\}, \text{mit } \kappa(v, S) := \left|\{(p, u, n) \in S | u \leq v\}\right|$$

```
SELECT *
FROM   S WINDOW(ROWS N);
```

PIPES-SQL 2.10: Gleitendes Zählfenster der Größe N auf Datenstrom S

Partitionierendes Zählfenster

Bei dem partitionierendem Zählfenster handelt es sich um eine Kombination von Gruppierungsoperator und gleitendem Zählfenster. Ein Eingabedatenstrom $S \in \mathbb{S}_\Omega$ wird in einem ersten Schritt durch eine Gruppierungsfunktion $g \in \mathbb{G}$ in $k \in \mathbb{N}$ verschiedene Ausgabedatenströme partitioniert. Auf jeden dieser Teildatenströme wird dann jeweils ein gleitendes Zählfenster mit Parameter $N \in \mathbb{N}$ gelegt. Die Vereinigung aller Ausgabedatenströme aus diesen gleitenden Zählfenstern ergibt den endgültigen Ausgabedatenstrom des partitionierenden Zählfensters. In *PIPES* ist das partitionierende Zählfenster durch einen eigenen SQL-Ausdruck repräsentiert. Wie zuvor bei der Gruppierung kann keine allgemeine Gruppierungsfunktion verwendet werden.

$$\omega^{partition} : \mathbb{S}_\Omega \times \mathbb{G} \times \mathbb{N} \to \mathbb{S}_\Omega$$

$$\omega^{partition}_{g,N}(S) := \bigcup_{i=1}^{k} \omega^{count}_{N}(\pi_i(\gamma_g(S)))$$

```
SELECT *
FROM   S WINDOW(PARTITION BY X ROWS N);
```

PIPES-SQL 2.11: Partitionierendes Zählfenster mit Partitionierung nach X und Größe N auf Datenstrom S

2.2.2 Erweiterungen

Obwohl die Datenstromalgebra ein mächtiges Werkzeug zur Verarbeitung von Datenströmen darstellt, wurde bereits sehr früh in der Datenstromgemeinschaft erkannt, dass es eine Reihe von interessanten Anfragen gibt, die sich damit nicht formulieren lassen [GÖ03]. Aus diesem Grund wurden in den letzten Jahren Erweiterungen wie iterative Anfragen [CGM09], Mustererkennung [Agr+08] und benutzerdefinierte Aggregate [Bai+06] entwickelt, die mittlerweile zum Standard eines modernen DSMS gehören. Weil diese Erweiterungen in enger Verbindung mit Complex Event Processing stehen, wird auf sie erst in Abschnitt 2.3.3, nachdem der Bedarf an ihnen motiviert wurde, näher eingegangen.

2.3 Complex Event Processing

Complex Event Processing (CEP) [Luc02; Luc07; Luc08] beschreibt Begriffe, Konzepte und Methoden, mit deren Hilfe hauptsächlich komplexe Ereignisse (engl. *Complex Events*) automatisch in einer Menge von einfachen Ereignissen erkannt werden können. Ein komplexes Ereignis ist die Kombination von mehreren einfachen Ereignissen und repräsentiert abgeleitetes und höherwertiges Wissen. Unter einem beliebigen einfachen oder komplexen Ereignis ist eine nicht näher definierte und maschinell verarbeitbare Information aus der realen Welt zu verstehen, die zusätzlich mit einer weiteren Dimension (Sequenznummer, Zeitstempel, etc.) angereichert ist, über die sich eine eindeutige Reihenfolge von zusammengehörigen Ereignissen herstellen lässt. Neben dieser funktionalen Seite von CEP werden weitere und nichtfunktionale Anforderungen an konkrete Systeme gestellt. Es müssen sehr große und dynamische Mengen von Ereignissen verarbeitet werden können und das Erkennen von komplexen Ereignissen muss in Echtzeit geschehen. Diesen Anforderungen und der verlangten zusätzlichen Dimension, mit der sich immer eine eindeutige Ordnung herstellen lässt, liegt die Annahme zugrunde, dass Ereignisse in einer festen zeitlichen Abfolge entstehen und schnell fließende Ereignisströme bilden. An einem Beispiel aus der Gebäudeautomation sollen die Begriffe und die Arbeitsweise von CEP verdeutlicht werden.

Beispiel
Ein Haus ist mit zwei unterschiedlichen Typen von Sensoren ausgestattet, die jeweils im gesamten Gebäude verteilt angebracht sind. Die Sensoren des ersten Typs messen die Temperatur und geben diesen Wert zusammen mit ihrer Position und der aktuellen Zeit in periodischen Abständen über einen Ereignisstrom an ein CEP-System weiter. Die Sensoren des anderen Typs können Rauch erkennen. Sie senden regelmäßig den aktuellen Status („Rauch“ oder „Kein Rauch“) zusammen mit ihrer Position und der aktuellen Zeit über einen anderen Ereignisstrom an dasselbe CEP-System wie die Temperatursensoren. Die einzelnen Daten, die kontinuierlich in dem CEP-System eintreffen, sind einfache Ereignisse mit einem niedrigen Informationsgehalt. Mit dem Ziel ein Feuer zu erkennen, sucht das CEP-System permanent nach folgendem komplexen Ereignis: „Ein Rauchsensor meldet Rauch und alle Temperatursensoren im Umkreis von 30 Metern zu diesem Rauchsensor melden zur gleichen Zeit einen Anstieg der Temperatur.“ Die Forderung, dass komplexe Ereignisse zeitnah erkannt werden müssen, wird an einem möglichen Feuer besonders verständlich.

2.3.1 Ereignisanfragesprache

Die zu erkennenden komplexen Ereignisse werden mittels einer Ereignisanfragesprache spezifiziert. Die Mindestanforderungen an eine solche Anfragesprache lassen sich durch vier wesentliche Grundoperationen beschreiben [EB09].

Filterung

In CEP-Anwendungen geht es um die Verarbeitung von Ereignismengen, von denen häufig nur ein kleinerer Teil interessant ist. Mit der Filterung ist es möglich, aus der Masse an Eingabeereignissen nur die relevanten Ereignisse herauszusuchen. Beispielsweise sind in einem Ereignisstrom, der von Rauchsensoren produziert wird, nur die Ereignisse von Interesse, die ein mögliches Feuer aufzeigen und deshalb weiterverarbeitet werden sollten. Die meiste Zeit hingegen werden sich die Rauchmelder jedoch stabil verhalten und nur irrelevante Ereignisse produzieren, deren Verarbeitung keine neuen Erkenntnisse liefern wird.

Korrelation

Das Verknüpfen von einfacheren Ereignissen zu einem komplexeren Ereignis ist essentiell für ein CEP-System und wird als Korrelation von Ereignissen bezeichnet. Besonders mächtig ist diese Operation, wenn unterschiedliche Typen von Ereignissen miteinander korreliert werden. In dem einführenden Beispiel werden Ereignisse mit zwei verschiedenen Typen (Temperaturereignis, Rauchereignis) immer dann miteinander korreliert, wenn die sie erzeugenden Sensoren räumlich eng beieinander liegen.

Temporaler Verbund

Mit dem temporalen Verbund können komplexe Ereignisse auf der Grundlage von Reihenfolge oder Zeitdauer aus einfachen Ereignissen entstehen. Zum Beispiel kann ein kontinuierliches Ansteigen der Temperatur nur dann ermittelt werden, wenn mehrere zeitlich aufeinanderfolgende Ereignisse eines Temperatursensors zusammen und unter Beachtung ihrer Reihenfolge betrachtet werden.

Akkumulation

Unter der Akkumulation werden sowohl die Aggregation als auch die Negation verstanden. Bei der Aggregation wird eine Menge von Ereignissen übersichtlich und kompakt auf ein einziges Ereignis (bspw. Durchschnittstemperatur aller Temperatursensoren in einem Gebäude oder Durchschnittstemperatur in den letzten fünf Stunden eines einzelnen Temperatursensors) reduziert. Mit Hilfe der Negation lässt sich das Fehlen eines Ereignisses in einem Ereignisstrom erkennen. Wenn ein erwartetes Ereignis (z.B. Feueralarm nachdem ein Feuer erkannt und gemeldet wurde) nicht eintritt, dann kann dies auf ein Problem hindeuten.

2.3.2 CEP-Anwendungen

Trotz Unterschiede zwischen verschiedenen verfügbaren CEP-Systemen, die meistens in der eingesetzten Technik zur Verarbeitung der Ereignisströme, der Anfragesprache oder der Semantik der Anfragesprache bestehen, folgen alle dem gleichen Anwendungsprinzip. Abbildung 2.5 stellt die Arbeitsweise und die Referenzarchitektur [Lea09a] einer allgemeinen CEP-Anwendung vereinfacht dar.

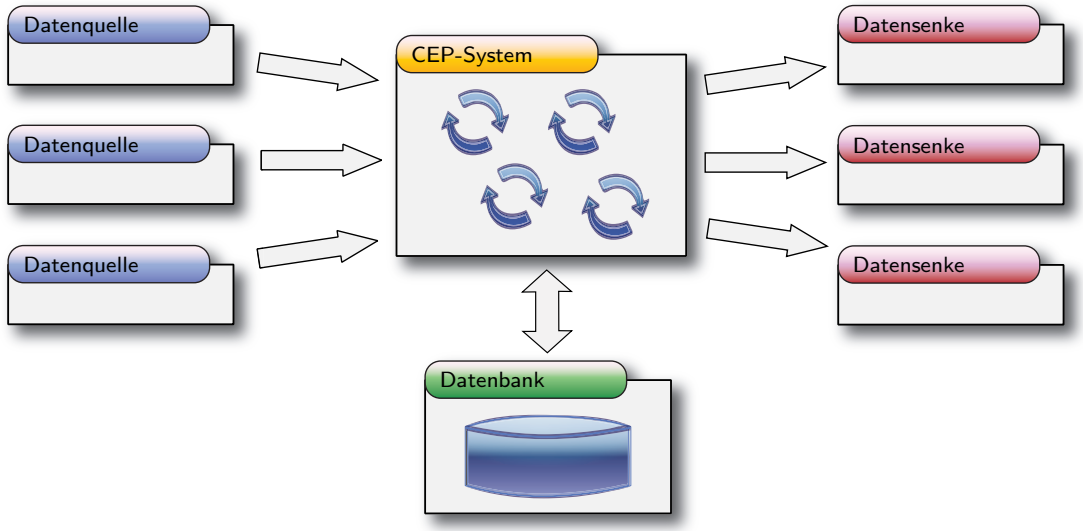

Abbildung 2.5: Vereinfachte Architektur einer CEP-Anwendung

Datenquellen
Alle Datenquellen, die relevante Ereignisse generieren können, müssen an das CEP-System angeschlossen werden. In der Regel handelt es sich dabei um viele und heterogene Quellen (Datenbanken über JDBC [JSRa], Sensoren über TCP/IP-Sockets [RFCd; RFCc], RSS-Datenfeeds [RSS] über HTTP, etc.). Mit Hilfe von Konvertern (im CEP-Jargon Adapter genannt) werden die Verbindungen hergestellt und die Elemente der Ereignisströme in das interne Format des jeweiligen CEP-Systems umgewandelt.

CEP-System
Innerhalb eines CEP-Systems sind alle Ereignisanfragen dauerhaft abgelegt. Sie werden auf den Ereignisströmen von den Datenquellen kontinuierlich ausgewertet und die von ihnen zeitnah erkannten komplexen Ereignisse werden an die Datensenken weitergegeben.

Datenbank
Der Einsatz einer Datenbank ist nicht zwingend für CEP-Anwendungen notwenig. Allerdings müssen manche Anfragen auf Kontextwissen zurückgreifen, um ausgewertet werden zu können. Beispielsweise könnten die Gebäudesensoren aus dem Beispiel nicht selbst ihre Positionen kennen, sondern nur eindeutige Identifikationsnummern besitzen. In der Datenbank müsste dann zu jeder Identifikationsnummer die jeweilige Position des Sensors gespeichert werden, auf die bei der Auswertung der Anfragen zurückgegriffen werden kann.

Datensenken
Komplexe Ereignisse, die von den Anfragen im CEP-System erkannt werden, müssen zur weiteren Verarbeitung an entsprechende Datensenken weitergeleitet werden. Gängige Datensenken sind beispielsweise Cockpits, die die wichtigsten Ergebnisse übersichtlich und in Echtzeit grafisch darstellen, Nachrichtendienste, die kritische Ergebnisse in Form von E-Mail oder SMS an betroffene Entscheidungsträger senden, oder Data Warehouses, in denen alle Ergebnisse langfristig gespeichert werden. Wie bei den Datenquellen werden für die Datensenken Adapter benötigt, die das CEP-System mit unterschiedlichen Datensenken verbinden und die Ergebnisse entsprechend konvertieren.

2.3.3 CEP-Systeme

Über das Verhältnis zwischen CEP-Systemen und Datenstrommanagementsystemen herrscht zurzeit noch keine vollkommene Klarheit [CM11]. Selbst untereinander sind aktuelle CEP-Systeme nahezu unvergleichbar, weil es zwischen je zwei Systemen signifikante Unterschiede gibt [Luc08; Bot+10] und ein Austausch nur unter größten Anstrengungen möglich ist (siehe dazu Abschnitt 3.3.2). Dieser Arbeit liegt das Verständnis zugrunde, dass Datenstrommanagementsysteme als CEP-Systeme eingesetzt werden können und CEP somit eine Anwendung von ihnen ist. Damit wird nicht ausgeschlossen, dass es auch andere Technologien gibt, auf denen CEP-Systeme basieren können.

Ein großer Unterschied zwischen beiden Systemarten war in der Vergangenheit, dass ein DSMS nicht in der Lage war Muster (und damit viele potentielle komplexe Ereignisse) zu erkennen. Diese Schwäche ist mittlerweile in allen modernen Datenstrommanagementsystemen durch die Einführung von mindestens einer der in Abschnitt 2.2.2 angesprochenen Erweiterungen beseitigt worden. In vielen dieser Systeme ist eine Mustererkennung (engl. *Pattern Matching*) nativ implementiert, weil dadurch eine sehr effiziente Ausführung erreicht werden kann [Agr+08]. Mit Hilfe von iterativen Anfragen lassen sich Rekursionen realisieren. In [CGM09] wird gezeigt, wie unter Ausnutzung von iterativen Anfragen Muster erkannt werden können. Durch

benutzerdefinierte Aggregate (engl. *User Defined Aggregates*, UDAs) lassen sich neben den fest implementierten Aggregationsfunktionen (Durchschnitt, Maximum, Summe, etc.) beliebige weitere (monotone) Funktionen in ein DSMS bringen [Bai+06]. Da eine Anfragesprache durch die Erweiterung von UDAs Turing-vollständig wird [LWZ04], lassen sich auch damit Muster erkennen. In *PIPES* sind sowohl UDAs als auch eine native Mustererkennung implementiert (Anfrage 3.10 verwendet die fest eingebaute Mustererkennung und zeigt die zugehörige Syntax und Semantik in *PIPES*). Im Folgenden wird der Nachweis erbracht, dass moderne Datenstrommanagementsysteme wie *PIPES* als CEP-Systeme eingesetzt werden können.

Ereignisströme sind spezielle Datenströme
In Abschnitt 2.2.1.1 wurde ein (Roh-)Datenstrom als Folge von Paaren (p, t) bestehend aus einem beliebigen Payload p und einem Zeitstempel t definiert. Ein Ereignisstrom hingegen wurde als Folge von Paaren (e, s) bestehend aus einem Ereignis e und einer Ordnungsinformation s definiert. Weil es sich bei einem Ereignis um einen speziellen Typ von Payload handelt, können Ereignisströme mit Zeitstempeln als Ordnungsinformationen als Spezialfall von Datenströmen angesehen werden. Falls eine andere Ordnungsdimension (z. B. Sequenznummern) eingesetzt wird, dann muss diese als zusätzliches Attribut dem Payload zugeschrieben werden. Als Zeitstempel kann der Zeitpunkt verwendet werden, an dem ein Ereignis in einem DSMS eingetroffen ist (Systemzeit). Eine direkte Verarbeitung von Ereignisströmen durch ein DSMS ist folglich immer möglich.

SQL mit Mustererkennung genügt den Anforderungen von CEP
Zu klären ist die Frage, ob sich die vier Grundoperationen von CEP (siehe Abschnitt 2.3.1) mit der Anfragesprache des eingesetzten DSMS ausdrücken lassen. Dazu werden für den weiteren Verlauf Anfragesprachen auf der Basis von SQL mit Mustererkennung als Erweiterung betrachtet. Die Filterung von CEP entspricht exakt der Selektion von SQL. Bei der Korrelation von CEP handelt es sich um die Bildung des kartesischen Produkts in SQL. Weil bei einer Korrelation allerdings nicht jede mögliche Kombination zwischen Ereignismengen gebildet werden soll, sondern nur Kombinationen von Ereignissen, die über bestimmte Beziehungen miteinander verbunden sind, muss nach der Bildung des kartesischen Produkts noch eine Selektion auf der Grundlage der Beziehungen stattfinden. Damit lässt sich die Korrelation durch den allgemeinen Verbund in SQL ausdrücken. Der temporale Verbund von CEP kann nicht durch die Grundoperatoren von SQL ausgedrückt werden. Hierzu wird die Mustererkennung, die auf zeitlich geordneten Ereignisströmen kontinuierlich nach einem spezifizierten Muster sucht, benötigt. Die Aggregation von CEP entspricht dem gleichbenannten Operator von SQL. Negationen von CEP können wieder durch die Mustererkennung ausgedrückt werden, indem die entsprechenden negativen Muster (ohne die erwarteten Ereignisse) spezifiziert werden.

Zusammenfassung

Die Dienste im Cloud Computing lassen sich in verschiedene und übereinander gestapelte Schichten, zwischen denen Abhängigkeiten und Beziehungen bestehen können, einteilen. Die hinter einer Cloud stehende physische Infrastruktur kann auf verschiedene Arten betrieben werden. Eine Cloud wird wesentlich durch Pooling von Ressourcen, dynamischer Skalierbarkeit, Selbstbedienung, einem netzwerkbasierten Zugang und messbaren Dienstqualitäten gekennzeichnet. Clouds implementieren die Service-orientierte Architektur durch den Einsatz von Web Services. Unterschiedliche Virtualisierungstechniken kommen in allen Schichten einer Cloud zum Einsatz. Datenstrommanagementsysteme erlauben das kontinuierliche Verarbeiten von großen und strömenden Datenmengen in Echtzeit. Eine Möglichkeit zur Formulierung von Datenstromanfragen ist die deklarative Datenbankanfragesprache SQL, deren Syntax, Semantik und Aussdrucksstärke auf Datenströme übertragen werden können. Moderne Datenstrommanegementsysteme haben zusätzlich zu ihrer Kernanfragesprache eine Reihe von Erweiterungen erfahren, mit denen unter anderem das Erkennen von Mustern möglich wird. Complex Event Processing ist eine Technologie für das automatische und zeitnahe Erkennen komplexer Ereignisse in einer großen und dynamischen Menge von einfachen Ereignissen. Als CEP-Systeme können Moderne Datenstrommanagementsysteme eingesetzt werden.

3 Design und Implementierung

Nach einem Überblick über existierende Monitoring-Produkte und ihrer kritischen Überprüfung auf Tauglichkeit für Cloud Computing in Abschnitt 3.1 wird eine allgemeine Architektur für Systeme zum Überwachen von Clouds in Abschnitt 3.2 vorgeschlagen. Anschließend wird in Abschnitt 3.3 eine Implementierung dieser Architektur vorgestellt.

3.1 Verwandte Arbeiten

Bevor der Vorschlag einer Architektur und Implementierungsdetails des Prototypen *CEP4Cloud* (Complex Event Processing For Cloud Computing) vorgestellt werden, wird für eine Auswahl von derzeit erhältlichen Monitoring-Produkten ihre Erfüllung der in Abschnitt 1.4 entwickelten Anforderungen an ein Cloud Monitoring diskutiert. Alle Produkte, die nur zur Überwachung Hersteller-spezifischer Dienste dienen (wie z. B. *Amazon CloudWatch* [ACW], welches ausschließlich für die Dienste von Amazon genutzt werden kann), werden dabei ausgelassen.

3.1.1 Cloudkick

Mit dem Dienst *Cloudkick* [CK; Gol] aus der SaaS-Schicht ist es möglich, virtuelle Maschinen aus unterschiedlichen Public Clouds und eigene physische oder virtuelle Maschinen gemeinsam in Echtzeit zu überwachen. Es werden die gängigsten Metriken der IaaS-Schicht unterstützt, die grafisch aufbereitet über einen Web-Browser abrufbar sind. *Cloudkick* kann über benutzerdefinierte Skripte um neue Metriken erweitert werden. Die API unterstützt aber nur Infrastrukturen, wodurch eine ganzheitliche Überwachung unmöglich wird. Wenn zuvor festgelegte Grenzwerte einer Metrik über- oder unterschritten werden, dann kann eine automatische Benachrichtigung per E-Mail oder SMS ausgelöst werden. Das Definieren von komplexeren Situationen oder Manipulationen an den überwachten Maschinen sind mit *Cloudkick* nicht möglich. Als elastischer Dienst kann sich *Cloudkick* allen Veränderungen in der Anzahl der überwachten Maschinen und Metriken anpassen.

3.1.2 Ganglia

Ganglia [Gan] ist ein vollständig quelloffenes Produkt zum Überwachen von Maschinen in einem Cluster oder Grid und kann über eine angebotene API oder durch direkter Veränderung des frei verfügbaren Quelltexts erweitert werden. Es kann laut seinen Entwicklern maximal 2.000 Maschinen gleichzeitig in Echtzeit überwachen und ist damit nicht beliebig skalierbar. Die gemessenen Werte werden von *Ganglia* parallel grafisch als dynamische Web-Dokumente verfügbar gemacht und langfristig in einer Textdatei gespeichert, um sie nachträglich analysieren zu können. Es gibt weder eigene Analyse-Werkzeuge noch ist eine Zusammenarbeit mit einer Datenbank vorgesehen.

3.1.3 Hyperic

Das quelloffene Produkt *Hyperic* [Hyp; Sac09] ist mit unzähligen Metriken für Maschinen und für eine breite Palette an Applikations-Servern, Betriebssystemen, Datenbanken, Middleware, Web-Servern und vielen anderen Plattformen und Anwendungen ausgestattet. Zusammen mit dem frei verfügbaren Quelltext und einer komfortableren API ist *Hyperic* damit für ganzheitliche Überwachungen ausgelegt. Es wird in einer freien und in einer kommerziellen Version angeboten und ist in der Lage, alle genannten Objekte bis zu einer bestimmten Grenze in Echtzeit zu überwachen. Die Messwerte können in verschiedenen visuellen Darstellung in einem Web-Browser angezeigt und dynamisch aktualisiert werden. *Hyperic* bietet in der Visualisierungskomponente den Nutzern die Möglichkeit an, unterschiedliche Metriken gemeinsam grafisch darzustellen, um auf diese Weise Zusammenhänge zwischen ihnen erkennen zu können. Zusätzlich können von den Nutzern einfache Situationen definiert werden, bei deren Eintreten eine Benachrichtigung oder eine Aktion (z. B. Neustarten einer Maschine) erfolgen sollen. Für exakte, komplexe und automatische Analysen sind weder die gemeinsame Darstellung von Metriken noch das Definieren einfacher Situationen ausreichend. Ein besonderes Alleinstellungsmerkmal von *Hyperic* ist die Fähigkeit, durch Auslesen aller laufenden Prozesse auf einem System alle unterstützten Metriken und Objekte (siehe Anfang des Absatzes) automatisch zu erkennen und deren Überwachung zu starten. Eine einzelne Instanz von *Hyperic* kann laut Hersteller maximal eine Millionen Datensätze pro Minute von bis zu 2.000 verschiedenen Maschinen verarbeiten. Daraus ergeben sich weniger als 16.700 Datensätze pro Sekunde, wodurch weder eine beliebige Skalierbareit noch eine Verarbeitung größerer Datenmengen in Echtzeit möglich sind.

3.1.4 Nagios, Icinga und Opsview

Nagios [Nag] ist sowohl ein freies und quelloffenes als auch ein kommerzielles Produkt zum vollständigen Überwachen von Infrastrukturen. Es ist in der Lage Netzwerkgeräte, einzelne Maschinen und einzelne Netzwerkdienste (wie z. B. LDAP-Server, FTP-Server, DHCP-Server, etc.) auf Maschinen zu überwachen. Eine Installation kann verteilt vorgenommen werden und ist damit skalierbar und durch Einführung von Redundanz zusätzlich ausfallsicherer. Neue sogenannte Überwachungs- und Anzeigemodule können entweder hinzugekauft oder selbst entwickelt werden. Eine große Auswahl fertiger Module wird auch kostenlos von vielen anderen Nutzern zur Verfügung gestellt. *Nagios* benutzt die gemessenen Werte als Entscheidungsgrundlage, um Benachrichtigungen auszulösen. Bei Eintreten von nur sehr eingeschränkt definierbaren abnormalen Situationen wird automatisch eine Benachrichtigung (z. B. in Form einer E-Mail oder SMS an die Administratoren) verschickt. Komplexere Situationen können nicht definiert werden und ein Eingreifen in das beobachtete Geschehen ist ebenfalls nicht möglich. Neben den Benachrichtigungen werden die erfassten Daten zusätzlich als Web-Dokumente grafisch aufbereitet angezeigt. Es können allerdings keine kontinuierlichen Analysen eingebracht werden. Bei der Messung und der Darstellung von Daten geht es bei *Nagios* hauptsächlich darum, Einblicke in die jüngste Vergangenheit zu gewinnen. Für eine zuverlässige Überwachung in Echtzeit, die zwar von den Entwicklern als Eigenschaft von *Nagios* angegeben wird, kann es allerdings nicht eingesetzt werden [Pat10].

Wegen Unzufriedenheit über den langsamen Entwicklungsprozess und den monolithischen Ansatz von *Nagios* entstand als Abspaltung das quelloffene Produkt *Icinga* [Ici]. Es wird als konsequent modularisiertes System entwickelt und bietet eine vollständige Kompatibilität zu der API von *Nagios*. Ein Unterschied ist, dass es nur eine freie Variante von *Incinga* gibt und diese einen vergleichbaren Funktionsumfang wie die kommerzielle Version von *Nagios* hat. Ein weiteres Konkurrenzprodukt zu *Nagios* ist *Opsview* [OV]. Dabei handelt es sich im Kern um *Nagios*, welches um weitere quelloffene Produkte und eigene Module ergänzt wurde. Dadurch wird standardmäßig ein erweiterter Funktionsumfang (mehr Metriken, leistungsstärkere Darstellungen und Überwachung von Betriebssystemen) sowie Kompatibilität zu den Schnittstellen der Public Cloud von Amazon erreicht.

Sowohl *Incinga* als auch *Opsview* sind ihrem Vorbild *Nagios* sehr ähnlich und unterscheiden sich von ihm hauptsächlich in Detailfragen. Deshalb gelten die Kritikpunkte für einen Einsatz als Cloud Monitoring von *Nagios* auch für seine beiden Derivate. Von allen drei Produkten scheint jedoch die Entwicklung von *Opsview* derzeit am stärksten in Richtung eines Überwachungssystems für Clouds zu gehen.

3.1.5 RevealCloud

Das Monitoring-Werkzeug *RevealCloud* [RC; IWa] ist eines der wenigen verfügbaren Produkte, die explizit für Cloud Computing entwickelt wurden. Das Hauptaugenmerk von *RevealCloud* liegt auf einer Überwachung von virtuellen Maschinen in Echtzeit. Die Leistungsfähigkeit der Überwachung wird durch eine beliebige Skalierbarkeit sichergestellt, da *RevealCloud* als Dienst der SaaS-Schicht angeboten und bei dem Hersteller in einer Cloud ausgeführt wird. Die verfügbaren Metriken in *RevealCloud* beschränken sich nur auf die wichtigsten Leistungsindikatoren von Maschinen und alle gemessenen Werte sind über das Web visualisiert abrufbar. Eine Anpassung der Visualisierung, die wie die Metriken fest vorgegeben ist, an individuelle Nutzerwünsche ist aber nicht möglich. Die Möglichkeiten für tiefergehende visuelle Analysen und Alarme sind dadurch sehr stark eingeschränkt. Bei einem Über- oder Unterschreiten frei definierbarer Grenzwerte kommt es zu einer Benachrichtigung. Der größte Nachteil von *RevealCloud* liegt darin, dass der Quelltext nicht offenliegt und die Ausführung bei dem Hersteller stattfindet. Deshalb ist es nicht möglich das Produkt zu erweitern (z. B. um zusätzliche Metriken), wodurch keinerlei Flexibilität gegeben ist. Plattformen oder Anwendungen werden von *RevealCloud* nicht überwacht und die kleine Auswahl an Metriken für Infrastrukturen reicht nicht aus, um eine Cloud ganzheitlich zu überwachen.

3.1.6 WatchMouse

WatchMouse [WM] ist zwar kein vollwertiges Monitoring-Produkt, dennoch wird es wegen seiner Einzigartigkeit und Nützlichkeit für Cloud Computing vorgestellt. Es ist für eine Überwachung der Verfügbarkeit und Leistungsfähigkeit von Diensten im Web gedacht. *WatchMouse* selbst wird als Dienst der SaaS-Schicht angeboten und ist dadurch beliebig skalierbar. Eine einmal aktivierte Instanz von *WatchMouse* überprüft kontinuierlich das Antwortverhalten der zu überwachenden Dienste im Web und kann auf diese Weise Ausfälle, Leistungseinbrüche und Fehlfunktionen erkennen und über eine Vielzahl an Nachrichtenkanälen den Anbietern melden. Die physische Infrastruktur von *WatchMouse* ist rund um den Globus verteilt, wodurch nicht nur die Verfügbarkeit von einem bestimmten Punkt aus, sondern eine weltweite Verfügbarkeit kontrolliert werden kann. Weil es sich bei *WatchMouse* um ein innovatives Nischenprodukt handelt, kann mit ihm keine ganzheitliche Überwachung einer Cloud realisiert werden. Allerdings sind die meisten Dienste in einer Cloud als Web Services implementiert und über das Internet erreichbar, sodass eine kontinuierliche, umfangreiche und skalierbare Überwachung der Verfügbarkeit und Unversehrtheit aller Dienste ermöglicht wird.

3.1.7 Zenoss

Ein weiteres quelloffenes Monitoring-Produkt für einzelne Maschinen und ganze Netzwerke ist *Zenoss* [Zen]. Es wird in einer freien und in einer kommerziellen Version angeboten. Die gesammelten Daten werden in Web-Dokumenten visuell dargestellt und zusätzlich in einem sogenannten Logbuch (einfache Textdatei) dauerhaft gespeichert. Dort können sie nachträglich und manuell mit externen Werkzeugen, mit denen *Zenoss* allerdings keinerlei Zusammenarbeit vorsieht, analysiert werden. In seinem Funktionsumfang ist *Zenoss* mit *Nagios* vergleichbar und um erweitert werden zu können, ist *Zenoss* durch eine vollständige Implementierung der API von *Nagios* in der Lage, dessen Module direkt zu verwenden [Lin09]. Eine weitere Besonderheit von *Zenoss* ist, dass Trends durch Extrapolation abgeleitet werden und dadurch visualisierte Rohdaten um die Darstellung ihrer erwarteten zukünftigen Entwicklungen erweitert werden können. Davon unabhängig ist die Kritik von *Nagios* auf *Zenoss* übertragbar.

3.1.8 Diskussion

Alle vorgestellten Monitoring-Produkte folgen dem Prinzip, dass Messwerte an einem zentralen Punkt gesammelt werden, um sie von dort aus für menschliche Nutzer visuell über eine Web-Schnittstelle verfügbar zu machen. Ein Teil der Systeme persistiert zusätzlich alle gewonnen Daten für eine nachträgliche Analyse mit externen Werkzeugen. Auch wenn viele der Produkte vor der Zeit von Cloud Computing entstanden sind, lassen sich die meisten aufgrund ihrer Quelloffenheit oder APIs nahezu beliebig erweitern, um der meist schon sehr umfangreichen Menge an verfügbaren Metriken weitere hinzuzufügen. Immer steht allerdings die Darstellung der Daten im Vordergrund. Eine Komponente zum Erzeugen von Reaktionen auf bestimmte Messwerte und Ereignisse fehlt meistens oder sie beschränkt sich auf das Versenden von Mitteilungen. Möglichkeiten zum Einbringen von einfachen oder komplexen Überwachungsregeln, die kontinuierlich ausgeführt werden, fehlen ebenso. Zusammenfassend lässt sich feststellen, dass moderne Werkzeuge zum Überwachen von IKT eine Vielzahl an fertigen Metriken mitbringen, sich beliebig um weitere erweitern lassen und sehr mächtige und flexible Visualisierungen anbieten. Die nur sehr schwach ausgeprägten Möglichkeiten zur Reaktion auf bestimmte Messwerte und fehlende Komponenten zur automatischen Analyse in Echtzeit der Daten werden der Dynamik und Komplexität von Cloud Computing allerdings nicht gerecht. Derzeit sind keine Produkte verfügbar, die eine Cloud ganzheitlich, insbesondere unter Berücksichtigung von Zusammenhängen zwischen unterschiedlichen Objekten oder im Zeitverlauf eines Objekts, überwachen und steuern können. Zusätzlich sind der Skalierbarkeit und Echtzeit aller Produkte mit Ausnahme der drei vorgestellten SaaS-Dienste klare Grenzen gesetzt.

Die meisten Anforderungen aus Abschnitt 1.4 sind Punkte, für die CEP entweder explizit entwickelt wurde oder die unter Ausnutzung von CEP erfüllt werden können. Darüber hinaus macht die Fähigkeit, eine Vielzahl komplexer Analysen kontinuierlich und in Echtzeit auf großen Datenmengen ausführen zu können, CEP zu einem idealen Kandidaten für Cloud Monitoring. Im Folgenden sollen kurz zwei Beispiele gegeben werden, die stellvertretend für die erreichbaren Größen von aktuellen und zukünftigen Systemen stehen und die von einem Überwachungssystem unterstützt werden müssen. Zum einen ist ein Verbund aus 200.000 Festplatten zu nennen, der insgesamt 120 Petabyte Speicherplatz zur Verfügung stellt [HO]. Zum anderen haben die NASA und Rackspace ein quelloffenes Projekt zum Aufbau von Cloud-Infrastrukturen gestartet [OS]. Dieses soll Clouds ermöglichen, die aus bis zu einer Millionen physischer und 60 Millionen virtueller Maschinen bestehen können [Mor]. Solche großen Systeme werden meistens dafür verwendet, eine Vielzahl von Nutzern parallel zu bedienen, um damit Skaleneffekte optimal auszunutzen. Jeder Nutzer in einer Cloud hat allerdings pro gebuchten Dienst einen oder mehrere individuelle SLAs. Jeder einzelne SLA muss von mindestens einer maßgeschneiderten Regel überwacht werden. In dieser für Clouds typischen Situation fallen sehr große Mengen von Messwerten an, auf denen unzählige Analysen kontinuierlich und zeitnah ausgewertet werden müssen. Nachfolgend werden zum Abschluss zwei weitere und essentielle Vorteile eines SQL-basierenden CEP-Systems, wie z. B. *PIPES*, für ein Cloud Monitoring erläutert.

Optimierung von Anfragen
Aufgrund der deklarativen Formulierung von Anfragen ist es möglich, sie zu optimieren. Dadurch können viele Anfragen auf großen Datenmengen gleichzeitig und in Echtzeit ausgewertet werden sowie gemeinsame Teilanfragen nur einmal ausgewertet und ihre Ergebnisse gemeinsam verwenden werden. Neue Überwachungsregeln können in Form von Anfragen zur Laufzeit in ein CEP-System gebracht und bestehende Anfragen entfernt oder verändert werden. Alle Änderungen zur Laufzeit haben keinen negativen Einfluss auf die Leistungsfähigkeit, da die Anfrageoptimierung kontinuierlich geschehen kann [Rie08]. Durch die in Abschnitt 2.2.1.3 vorgestellte Übertragung der Semantik von SQL auf Datenströme können viele bereits vorhandene Optimierungstechniken von Datenbanken adaptiert werden.

Extraktion von verborgenem Wissen
In den letzten Jahren wurde viel Aufwand in die Entwicklung effizienter Algorithmen gesteckt, mit denen kontinuierlich aus strömenden Datenmengen nicht direkt sichtbares Wissen zum Vorschein gebracht werden kann. Sehr gut erforscht sind z. B. das Entdecken von Mustern, Anomalien und Perioden, das Berechnen der am häufigsten vorkommenden Objektmengen, das Klassifizieren von Daten sowie das Entdecken oder Vorhersagen von Trends und Evolutionen. Mit einem CEP-System kann die Implementierung solcher Algorithmen vollständig oder unterstützend geschehen.

3.2 Architektur

In diesem und im nächsten Abschnitt wird eine Monitoring-Lösung auf der Basis von Datenströmen und CEP explizit für das Überwachen von Clouds entwickelt. Deshalb berücksichtigt das Design die funktionalen Anforderungen aus Abschnitt 1.4 und die sich aus der Praxis ergebenden zusätzlichen Anforderungen in Bezug auf die Anzahlen von Daten und Überwachungsregeln. Die Hauptziele sind eine komfortable und flexible Erweiterbarkeit sowie eine möglichst gute Performanz bei schonendem Umgang mit allen Ressourcen (insbesondere des im Cloud Computing wertvollen Netzwerks). In Abbildung 3.1 ist der Vorschlag dieser Arbeit für eine allgemeine Architektur, die im Folgenden anhand ihrer einzelnen Komponenten detailliert vorgestellt werden wird, von Überwachungssystemen für Clouds dargestellt. Die Architektur besteht aufgrund einer notwendigen Skalierbarkeit und der teilweise enormen Größe der zu überwachenden Clouds aus verteilten Komponenten, die jeweils über eigene Ressourcen verfügen und die selbst wiederum verteilt betrieben werden können.

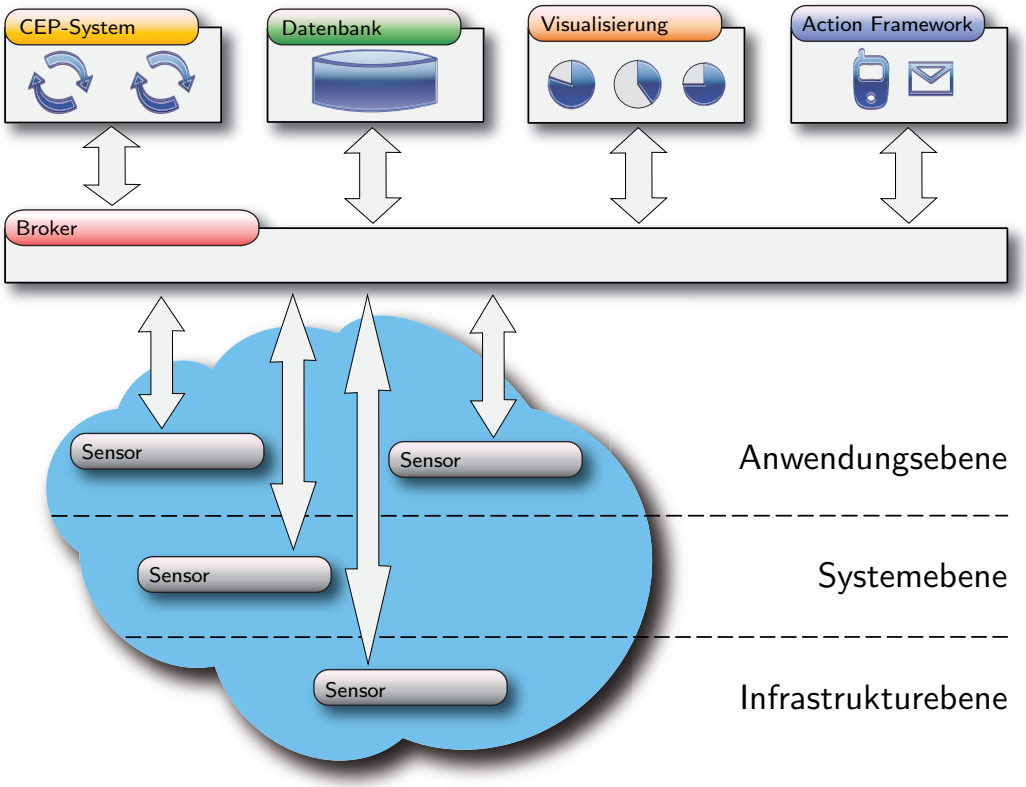

Abbildung 3.1: Architektur von Überwachungssystemen für Clouds

3.2.1 Broker

Der Broker ist dafür verantwortlich, dass die verteilten Komponenten über Nachrichten miteinander kommunizieren können. Er nimmt die Nachricht einer einzelnen Komponente entgegen und verteilt sie an alle Empfänger. Der Broker muss gewährleisten, dass die Nachrichten zuverlässig und schnell zugestellt werden. Die Zuverlässigkeit ist notwendig, damit es zu keinem Informationsverlust kommt. Von dem schnellen Nachrichtentransport hängt ab, ob der Echtzeitcharakter gewahrt werden kann. Alle Nachrichten sind zusätzlich mit dem Zeitstempel ihrer Entstehung ausgestattet. Damit genügen sie den Anforderungen von Ereignissen im Kontext von CEP.

3.2.1.1 Ereignis-gesteuerte Architektur

Der gesamten Kommunikation liegt die Ereignis-gesteuerte Architektur (engl. *Event Driven Architecture*, EDA) [Mic06] zugrunde. Die wichtigsten Eigenschaften der EDA sind ein effizienter, nichtlinearer Nachrichtentransport von mehreren Sendern an mehrere Empfänger gleichzeitig (nur die tatsächlichen Empfänger bekommen die Nachrichten gezielt zugestellt, wodurch eine unnötige Belastung der Kommunikation, wie z. B. bei einem Broadcast, vermieden wird) und eine Fokussierung auf Ereignisse. Letzteres bedeutet, dass Ereignisse (z. B. Zustandsänderungen oder neue Daten in einer der Komponenten) diejenigen Elemente sind, die (automatisch) Nachrichten generieren und sie ohne die Empfänger zu nennen versenden und dadurch Aktionen in anderen Komponenten auslösen können. Häufig wird die EDA als Erweiterung oder Verbesserung der SOA angesehen, da verteilte Komponenten nicht mehr lose gekoppelt sind, sondern vollständig losgelöst voneinander agieren [Mic06; Mar06].

Der Broker implementiert die EDA durch das sogenannte Publish/Subscribe-Paradigma [Mar06]. Hierbei melden die Empfänger beim Broker im Vorfeld an, welche Nachrichten sie empfangen möchten. Ein Sender hingegen veröffentlicht eine Nachricht ohne explizit die Empfänger anzugeben. Über den Broker wird dann die Nachricht an alle an ihn angeschlossenen Komponenten, die von ihm als Empfänger der Nachricht identifiziert wurden, ausgeliefert. Die Zuordnung einer Nachricht zu Empfängern kann bei Publish/Subscribe über mehrere unterschiedliche Filterkriterien (Inhalt, verwendeter Nachrichtenkanal, Nachrichtenverfasser, etc.) hergestellt werden [Eug+03]. Für den Broker ist die einfache Variante der Filterung über verschiedene Nachrichtenkanäle ausreichend. Eine Komponente sendet ihre Nachrichten in bestimmte, aus einer Menge von vielen möglichen, Kanäle und alle Komponenten, die mit diesen Kanälen verbunden sind, bekommen die Nachricht ohne weitere Filterungen zugestellt.

Beispiel
Ein Ereignis-gesteuertes Nachrichtensystem auf der Basis von Publish/Subscribe sind Staumeldungen im Rundfunk. Eine Rundfunkanstalt (Sender) gibt zu bestimmten Zeiten (z. B. planmäßig alle 30 Minuten und außerplanmäßig bei besonderen Situationen) Meldungen und Empfehlungen (Nachricht) zu Staus (Ereignisse) mittels ihrer Sendefrequenz bekannt (Publish). Personen, die sich für Staumeldungen interessieren (Empfänger), müssen dazu angemeldet sein, was im vorliegenden Fall heißt, dass das Radio auf die Frequenz des Senders eingestellt ist (Subscribe). Anschließend entscheiden die Empfänger selbstständig, ob und wie auf die Nachrichten zu reagieren ist (z. B. Wechseln der Route).

3.2.2 Sensoren

Die zu überwachenden Metriken werden periodisch von Sensoren, die in allen Schichten der Cloud sowie der Infrastruktur angebracht sind, gemessen und in Form einer Nachricht an den Broker geschickt. Für jeden zu überwachenden Objekt-Typ gibt es genau einen Sensor-Typ, der auf allen Instanzen ausgeführt wird und relevante Metriken dieser Objekte misst. Weil im Allgemeinen viele verschiedene Objekte einer Instanz parallel überwacht werden sollen, werden alle Sensor-Typen zu einem Agenten, der das gemeinsame Starten, Ausführen, Beenden und Kommunizieren mit dem Broker übernimmt, zusammengefasst. Zum Starten der Überwachung einer kompletten Instanz in der Cloud muss dann nur noch der Agent ausgeführt werden, um alle Sensoren zu starten. Die Installation der Sensoren beschränkt sich damit auf das Verteilen des Agenten auf alle Instanzen in der zu überwachenden Cloud.

3.2.2.1 Datenströme

Für jeden Sensor-Typ gibt es im Broker jeweils einen eigenständigen Nachrichtenkanal, in den die in Nachrichten verpackten Messwerte kontinuierlich als Datenstrom gesendet werden. Der zugehörige Kanal ist für jeden Typ von Sensor somit eindeutig bestimmt und für jeden Empfänger ist leicht ersichtlich, welche Daten sich in den einzelnen Kanälen befinden. Jeder Sensor-Typ erzeugt dadurch einen separaten Datenstrom. In Abbildung 3.2 werden die Zusammenhänge exemplarisch anhand der Sensor-Typen zur Überwachung der Objekt-Typen „CPU“, „Prozesse“ und „JVM Heap-Speicher“ dargestellt.

In dem Beispiel existieren beliebig viele Maschinen in der überwachten Cloud, auf denen der Verbrauch der CPU von jeweils einem Sensor des gleichen Typs gemessen wird. Die gemessenen Werte werden als Nachrichten in einem von allen Sensoren geteilten Datenstrom „CPU“ über den Broker den anderen Komponenten zur Verfügung gestellt. Auf jeder der Maschinen befindet sich eine beliebige Anzahl von Systemen.

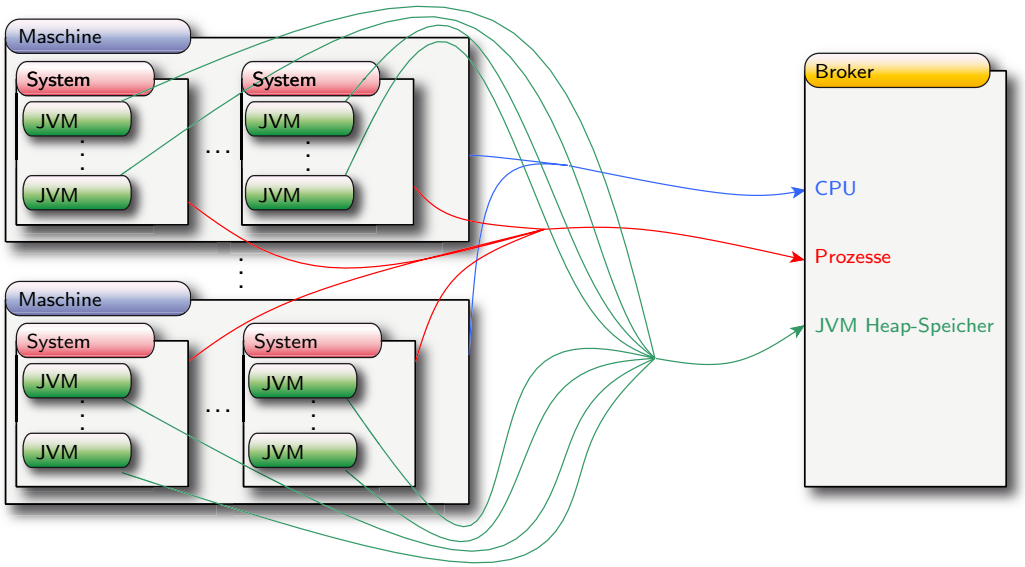

Abbildung 3.2: Logische Sicht auf die Datenströme

Alle aktiven Prozesse und ihre Kenndaten werden pro System von jeweils einer Instanz des entsprechenden Sensor-Typs ermittelt und als Nachrichten über einen gemeinsamen Datenstrom „Prozesse“ veröffentlicht. Auf jedem System wiederum können mehrere JVMs ausgeführt werden, deren Heap-Speicher pro System von jeweils einem Sensor ausgelesen werden. Die produzierten Nachrichten münden auch hier in einem gemeinsam genutzten Datenstrom „JVM Heap-Speicher“.

3.2.3 CEP-System

Die Nachrichtenkanäle innerhalb des Brokers können als unbeschränkte Folgen von Nachrichten interpretiert werden und sind daher Datenströme. Um diese mit dem angeschlossenen CEP-System direkt verarbeiten zu können, muss sichergestellt werden, dass alle Nachrichten eines Nachrichtenkanals homogen strukturiert sind (z. B. relationale Tupel oder XML-Dokumente mit jeweils beliebigem, aber festem Schema) und dass jede Nachricht den Zeitpunkt ihrer Entstehung beinhaltet. Beide Bedingungen müssen nur von denjenigen Datenströmen erfüllt werden, die auch mit CEP analysiert werden sollen. Primär ist das CEP-System für die Auswertung der Sensorströme verantwortlich, die bereits in der geforderten Form vorliegen. Die von dem CEP-System erzeugten Ergebnisse können wieder als Datenströme über den Broker allen anderen Komponenten zur Verfügung gestellt werden.

3.2.3.1 Datenmodellierung

Um mit CEP sinnvoll Ereignisse korrelieren zu können sowie um Zusammenhänge zwischen einzelnen Objekten erkennen und kontrollieren zu können, müssen die einfachen Ereignisse, die von den Sensoren produziert werden, um Metadaten angereichert werden. Im Fall von Cloud Computing ist es besonders wichtig, unterschiedliche Metriken und Metriken aus verschiedenen Ebenen miteinander in Beziehung setzen zu können. Damit zum Beispiel die Arbeitslast einer einzelnen physischen Maschine detaillierter aufgeschlüsselt werden kann, werden alle Prozesse benötigt, die auf dieser Maschine laufen. Wenn nun auf der Maschine mehrere VMs mit jeweils eigenem Betriebssystem aktiv sind, dann wird nicht nur eine Zuordnung von einem Prozess zu seinem Betriebsystem, sondern auch eine Zuordnung zu der Maschine benötigt. Ein geeignetes Instrument, um die Beziehungen in einer Cloud abzubilden, ist die Vergabe von eindeutigen Schlüsseln an jedes existierende Objekt unter Beachtung von Abhängigkeiten zu anderen Objekten (z. B. ein Prozess zu seinem System). Das Prinzip der Schlüsselvergabe ist exemplarisch in Form eines ER-Diagramms [Che76] in Abbildung 3.3 dargestellt.

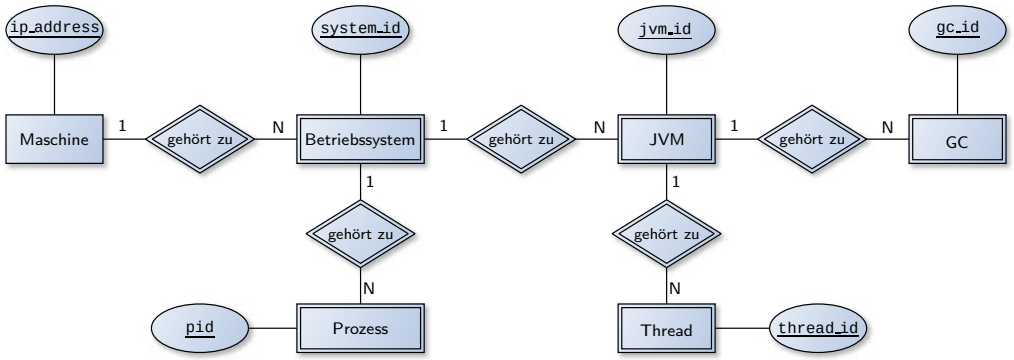

Abbildung 3.3: Prinzipieller Aufbau des Datenmodells

Eine physische oder virtuelle Maschine kann eindeutig anhand ihrer IP-Adresse identifiziert werden. Alle Metriken, die auf einer Maschine überwacht werden, erhalten als zusätzliches Attribut die IP-Adresse. Für Betriebssysteme muss ein synthetischer Schlüssel generiert werden, der pro Maschine eindeutig ist, da ein Betriebsystem von einer Maschine abhängig ist (schwache Entität). Ein Betriebssystem kann damit durch seinen eigenen Schlüssel im Kontext einer Maschine und zusätzlich mit der IP-Adresse der zugehörigen Maschine innerhalb einer Cloud eindeutig identifiziert werden. Für JVMs muss ebenfalls ein künstlicher Schlüssel generiert werden, der pro Betriebssystem maximal einmal vergeben werden darf, oder es wird auf die vom Betriebsystem vergebene Prozess-ID zurückgegriffen. Eine JVM hängt schwach von dem Betriebssystem, unter dem sie läuft, ab. Weil es sich bei einer JVM um einen

speziellen Prozess handelt, gelten die eben gemachten Aussagen allgemein für alle Prozesse eines Betriebssystems. Einige Prozesse sollten aber gesondert und nicht nur als einfacher Prozess betrachtet werden, wenn sie z. B. einen Dienst in der Cloud darstellen oder zur Ausführung eines Dienstes notwendig sind. In dem vorliegenden Beispiel wurde der JVM als Möglichkeit zur Anwendungsvirtualisierung besondere Beachtung geschenkt. Weitere Prozesse, die auf die gleiche Weise detaillierter betrachtet werden sollten, sind unter anderem Prozesse von Applikations-Servern, Datenbanken, Mail-Servern, Middleware und Web-Servern. Weil jeder Prozess Objekte enthalten kann, die zu seiner vollständigen Überwachung mit berücksichtigt werden müssen, können weitere schwache Entitäten entstehen, die sich in dem Datenmodell widerspiegeln müssen. In dem skizzierten ER-Diagramm wurde diese nächste Stufe der Hierarchie für die Speicherbereinigungen (engl. *Garbage Collections*, GCs) und die Threads einer JVM modelliert. Für eine GC kann der Name des Algorithmus, den sie implementiert, als Schlüssel verwendet werden, da dieser pro JVM, von der eine GC schwach abhängt, eindeutig ist. Bei den von einer JVM abhängigen Threads können als Schlüssel die eindeutig vergebenen Thread-IDs wiederverwendet werden. Die Entitäten sowie ihre vollständigen Schlüssel sind in Tabelle 3.1 zusammengefasst.

Entität	Schlüssel	Abkürzung
Maschine	(`ip_address`)	
Betriebssystem	(`ip_address`, `system_id`)	`system_key`
Prozess	(`ip_address`, `system_id`, `pid`)	`process_key`
JVM	(`ip_address`, `system_id`, `jvm_id`)	`jvm_key`
Thread	(`ip_address`, `system_id`, `jvm_id`, `thread_id`)	`thread_key`
GC	(`ip_address`, `system_id`, `jvm_id`, `gc_id`)	`gc_key`

Tabelle 3.1: Entitäten und ihre Schlüssel

Gemäß den Regeln der ER-Modellierung ergibt sich der vollständige Schlüssel einer schwachen Entität aus seinen eigenen Schlüsselattributen und dem Schlüssel der starken Entität, von der er abhängt. Weil die Schlüssel mit der Anzahl der Stufen der Hierarchie in ihrer Größe linear anwachsen, wurden zusammengesetzte Schlüssel abgekürzt, um Anfragen in dieser Arbeit lesbarer zu gestalten. Für die Generierung von künstlichen Schlüsseln ist der jeweilige Agent verantwortlich. Natürliche Schlüssel können von den Agenten vor Ort ausgelesen werden. Zusätzlich reichern sie die Messwerte in jeder Nachricht vor dem Verschicken um die benötigten Schlüssel an. Es sei noch einmal betont, dass nur der prinzipielle Aufbau beschrieben wurde und kein vollständiges Datenmodell für Cloud Computing dargestellt ist. In einer Cloud existieren viele weitere Entitäten (z. B. komplette Infrastrukturen und Nutzer von Diensten) und Beziehungen, die ein produktives Überwachungssystem berücksichtigen muss.

3.2.4 Action Framework

Das Action Framework dient der automatischen und schnellen Steuerung von einzelnen Objekten (z. B. Migrieren einer VM) in einer Cloud sowie dem Versenden von Benachrichtigungen (z. B. E-Mail oder SMS an einen verantwortlichen Administrator) bei besonderen Ereignissen, die einer menschlichen Entscheidung bedürfen. Jede Aktion wird durch ein eigenständiges Modul im Action Framework repräsentiert und muss einfach zu integrieren sein. Eine Aktion ist einer Anfrage im CEP-System sehr ähnlich. Sie ist nach dem Starten kontinuierlich aktiv und kann zur Laufzeit beendet, gelöscht oder verändert werden. Das Anstoßen von Aktionen ist in zwei unterschiedlichen Varianten denkbar.

In einer einfachen Version des Action Frameworks gibt es innerhalb des Brokers einen separaten Nachrichtenkanal, auf dem nur das Action Framework Nachrichten empfängt. Alle anderen Komponenten können in diesen Kanal ihre Anweisungen mit allen benötigten Informationen als Nachricht (z. B. „Verschiebe VM **X** von Maschine **Y** nach Maschine **Z**“) einfügen, um eine Aktion zu veranlassen. Das Ausführen einer Aktion ist damit nichts anderes als ein entfernter Prozeduraufruf. In einer erweiterten Version kann dem Action Framework deutlich mehr Autonomie eingeräumt werden. Eine erste Stufe dazu ist das selbstständige Ermitteln von fehlenden Parametern. Bei Eintreffen einer unvollständigen Anweisung (z. B. „Verschiebe VM **?** von Maschine **Y** nach Maschine **?**“) müssen als erstes die fehlenden Parameter (in dem Beispiel eine geeignete VM von Maschine **Y** sowie eine Zielmaschine mit ausreichend freien Kapazitäten) berechnet werden, um die Aktion ausführen zu können. Eine nochmalige Erhöhung der Autonomie kann erreicht werden, wenn das Action Framework die Ausführung von Aktionen selbstständig anstoßen kann. Aktionen könnten sowohl zeitlich gesteuert als auch aufgrund eigener Entscheidungen ausgeführt werden. Letzteres ist problemlos möglich, weil über den Broker alle Sensorströme und alle Ergebnisse des CEP-Systems direkt verfügbar sind. In Kapitel 4 wird anhand des erweiterten Action Frameworks gezeigt, wie es mit anderen Komponenten zusammenarbeiten kann, um eine dynamische und proaktive Lastbalancierung in Echtzeit für die gesamte überwachte Cloud zu realisieren.

Da die Folgen einer Aktion meistens in einzelnen Objekten der Cloud wirken sollen, ist eine Kooperation mit den Agenten unabdingbar. Viele Befehle können nur vor Ort ausgeführt werden und benötigen bestimmte Privilegien dazu. Weil sich die Agenten bereits in direkter Nähe zu den entsprechenden Objekten befinden, müssen diese nur noch um die Fähigkeit Befehle auszuführen erweitert sowie mit den minimal notwendigen Rechten ausgestattet werden. Die Kommunikation zwischen Action Framework und Agenten kann einfach über extra angelegte Nachrichtenkanäle geschehen.

3.2.5 Datenbank

Der Einsatz einer Datenbank ist maßgeblich für die Effektivität einzelner Komponenten der Architektur sowie für eine einfache Konfiguration und Ausführung des Gesamtsystems erforderlich. Die Sensor-Agenten benötigen eine persistente und aktuelle Datenbasis, um unabhängig voneinander die erforderlichen künstlichen Schlüssel an Objekte der Cloud vergeben zu können. Beim Generieren von neuen Schlüsseln kann in der Datenbank nachgesehen werden, welche Schlüssel bereits vergeben wurden und deshalb nicht verwendet werden dürfen. Die daraufhin generierten Schlüssel werden dann ebenfalls in der Datenbank vermerkt, um eine wiederholte Vergabe zu sperren. Darüber hinaus kann es sinnvoll sein, regelmäßig Stichproben der einzelnen Sensorströme dauerhaft festzuhalten. Auch hierfür ist eine Datenbank der ideale Ort zum effizienten Speichern und Abfragen der Proben. Über die Synergien zwischen einem CEP-System und einer Datenbank wurde bereits in Abschnitt 2.3.2, in dem die Datenbank als ein integraler Bestandteil von CEP-Anwendungen vorgestellt wurde, diskutiert. Das Action Framework benötigt für viele Aktionen Kontextwissen, um diese sinnvoll ausführen zu können. Beispielsweise müssen Benachrichtigungen an diejenigen Personen versendet werden, die erstens in der Lage sind die richtigen Entscheidungen zu treffen und die zweitens möglichst schnell reagieren können. In einer Datenbank können alle in Frage kommenden Personen mit ihren Kontaktdaten (E-Mail Adresse, Mobilfunknummer, etc.), Kompetenzen und Arbeitszeiten gepflegt werden. Benachrichtigungen können so gezielt an die richtigen Empfänger erfolgen. Eine besonders wichtige Funktion der Datenbank, auf die in dieser Arbeit nicht detailliert eingegangen werden kann, ist eine global benötigte Benutzerverwaltung. Einzelne Anfragen, Aktionen und Visualisierungen müssen Nutzern zugeordnet werden, die diese verändern, löschen oder verwenden dürfen. Für einzelne Datenströme muss geklärt werden, welche Benutzer darauf zugreifen dürfen und welche ausgeschlossen sind, da sie sensible Informationen beinhalten können. Das Gleiche gilt für das CEP-System und das Action Framework. Für beide Komponenten muss festgelegt werden, welche Nutzer das Recht erhalten, neue Anfragen bzw. Aktionen in das Überwachungssystem zu bringen. Sowohl die Authentifizierung als auch die Autorisierung sollten global für das gesamte System (anstatt redundant für jede einzelne Komponente) umgesetzt werden, um Konflikte zu vermeiden und um den Administrationsaufwand zu minimieren.

Die skizzierten Anwendungsfälle stellen nur eine kleine Auswahl der Möglichkeiten, die durch eine Datenbank ermöglicht werden, dar. In Kapitel 4 wird die vorhandene Datenbank intensiv in die Problemlösung mit einbezogen. Dort ist sie hauptsächliche eine sinnvolle Ergänzung zu dem Broker, über den synchron kommuniziert werden kann, indem sie den Komponenten zusätzlich ein asynchrones Austauschen von (großen) Datenmengen sowie ein effizientes Speichern und zielgerichtetes Abfragen der Daten ermöglicht.

3.2.6 Visualisierung

Für schnelle und umfassende Einblicke in das aktuelle Geschehen einer Cloud sind Darstellungen der Daten in Form von Diagrammen wesentlich besser dem menschlichen Wahrnehmungssystem angepasst als Ausdrucke der Werte in Form von blanken Zahlen. Die zurzeit vorteilhafteste Möglichkeit zur Visualisierung ist das Bereitstellen von grafisch aufbereiteten Web-Dokumenten über ein Netzwerk (alle in Abschnitt 3.1 vorgestellten Produkte nutzen diese Art der Visualisierung), weil dadurch keine zusätzliche Anwendung per Hand auf den Endgeräten installiert werden muss und die Darstellungen von jedem beliebigen Zugangspunkt zu dem Netzwerk verfügbar sind. Bei der Erzeugung von Visualisierungen müssen zwei Dinge besonders beachtet werden. Zum einen müssen die als Datenströme vorliegenden Daten in Echtzeit dargestellt und aktualisiert werden und zum anderen muss den Nutzern die Möglichkeit geboten werden, umfangreich auf die Visualisierungen Einfluss nehmen zu können.

Um zügige Darstellungen und Aktualisierungen zu erreichen, ist es nicht zweckmäßig die Grafiken von einem zentralen Web-Server erstellen zu lassen und dann an die Endgeräte zu verteilen. Jeder Nutzer hat seine individuelle Darstellungsform und schon bei bereits wenigen Nutzern sind die Rechenkapazitäten des Web-Servers schnell ausgeschöpft. Zusätzlich wird das Netzwerk durch die hohen Speichergrößen von Grafiken unnötig belastet. Da mittlerweile alle in Frage kommenden Endgeräte über mehr als ausreichend Rechenkapazitäten verfügen, genügt es nur die darzustellenden Daten an die Endgeräte zu verteilen und die Grafiken dort zu generieren. Dazu muss bei dem ersten Aufruf der Web-Dokumente die darin eingebettete Anwendung an das Endgerät übertragen werden. Geeignete Technologien sind beispielsweise Flash [Fla] bzw. Silverlight [SL] für ansprechende Grafiken und Animationen oder AJAX (Asynchronous JavaScript and XML) [W3Cb] bzw. Java-Applets [JA] für einen schonenden Umgang mit Ressourcen (z. B. auf mobilen Endgeräten wie Smartphones).

Weil die Art der Darstellung sehr stark die daraus visuell ableitbaren Informationen beeinflusst, muss sie von den Nutzern selbst bestimmt werden können. Während z. B. die Darstellung des jeweils aktuellsten Wertes in einem Tachodiagramm mit fester Skala sehr gut die gegenwärtige Belastung bzw. Ausnutzung vermittelt, kann in einem Liniendiagramm die zeitliche Entwicklung eines Wertes abgelesen werden. Viele Diagrammarten sind über Parameter definiert (z. B. Skala in einem Tachodigramm oder Zeitintervall in einem Liniendiagramm), die ebenfalls von den Nutzern beeinflussbar sein sollten. Darüber hinaus werden nur die wenigsten Nutzer an einer Darstellung aller verfügbaren Werte interessiert sein. Deshalb müssen von den Nutzern die darzustellenden Werte selbst ausgewählt werden können, um übersichtliche Cockpits zu ermöglichen.

3.3 CEP4Cloud

Für eine Implementierung des im vorhergehenden Abschnitt beschriebenen Designs müssen die einzelnen Komponenten bereitgestellt und mit dem Broker verbunden werden. Um ein plattformunabhängiges Gesamtsystem zu erhalten, wurden mit Java umgesetzte Produkte verwendet und eigene Komponenten ebenfalls in Java geschrieben. Bei den meisten benötigten Komponenten kann jeweils auf eine umfangreiche Auswahl an freien und kommerziellen Produkten zurückgegriffen werden. Weil während der Zusammenarbeit die Komponenten untereinander keine Bindungen eingehen, kann nach der Implementierung jede einzelne von ihnen unter geringem Anpassungsaufwand jederzeit ausgetauscht werden.

3.3.1 Broker

Wegen der Entscheidung für Java als Entwicklungssprache sollte als Broker ein unmittelbar zu Java kompatibles Produkt eingesetzt werden. Für den Nachrichtenaustausch von verteilten Java-Komponenten ist JMS aus der API von Java ein weit verbreiteter Standard. Damit lässt sich auch das benötigte Publish/Subscribe-Paradigma direkt umsetzen. Bei JMS handelt es sich allerdings nicht um einen einsatzbereiten Nachrichtenvermittlungsdienst, sondern nur um eine Sammlung von Schnittstellen. Um tatsächlich Nachrichten austauschen zu können, muss ein JMS-Server eingesetzt werden, der die Vermittlung von Nachrichten in einem Netzwerk übernimmt. Für *CEP4Cloud* kommt der beliebte und frei erhältliche JMS-Server *HornetQ* [HQ] als Broker zum Einsatz. Innerhalb des JMS-Servers muss für jeden Datenstrom ein eigener Nachrichtenkanal – ein sogenanntes JMS-Topic – angelegt werden. Mit Hilfe von JMS-Topics kann, im Gegensatz zu den ebenfalls verfügbaren Nachrichtenschlangen, das Publish/Subscribe-Paradigma umgesetzt werden. Die anderen Komponenten von *CEP4Cloud* können sich bei einzelnen JMS-Topics anmelden und dann über sie Nachrichten veröffentlichen und empfangen. Um den Komponenten die JMS-Topics möglichst einfach zugänglich zu machen, kann parallel zu dem JMS-Server ein JNDI-Server [JND] eingesetzt werden. Von ihm können sowohl die Konfiguration von JMS als auch ein JMS-Topic anhand seines Namens direkt als Java-Objekte geladen werden. Weil die Schnittstellen fest von der Java-API vorgegeben sind, kann der verwendete JMS-Server ohne Änderungen problemlos gegen jeden anderen ausgetauscht werden. Dadurch ergibt sich der Vorteil, dass *CEP4Cloud* ohne Modifikationen in Umgebungen mit vorhandenem ESB integriert werden kann. Falls in einer Umgebung, in der *CEP4Cloud* verwendet werden soll, bereits ein ESB vorhanden ist, dann kann dieser sämtliche Aufgaben des Brokers übernehmen, da alle modernen ESBs JMS unterstützen und die Implementierung der EDA mit Hilfe von Publish/Subscribe ermöglichen [Mar06].

3.3.2 CEP-System

Wie bereits mehrfach angedeutet, kommt als CEP-System das in Java entwickelte Datenstrommanagementsystem *PIPES* mit der in Abschnitt 2.2 vorgestellten Ausdrucksstärke, Syntax und Semantik zum Einsatz. Weil es nur eine einzige Datenquelle und Datensenke gibt (jeweils den Broker über JMS) ist das Anbinden von einem CEP-System besonders komfortabel. Es wird lediglich ein Adapter benötigt, der Nachrichten über JMS-Topics entgegennimmt und in ein für das CEP-System verarbeitbares Format umwandelt und umgekehrt Ergebnisse von dem CEP-System in das vereinbarte Nachrichtenformat bringt und über JMS-Topics versendet. Um das CEP-System gegen ein anderes auszutauschen, muss nur der Adapter entsprechend modifiziert werden. Aufgrund der weiten Verbreitung von JMS werden passende Adapter meistens mit ausgeliefert, sodass diese direkt verwendet werden können.

Obwohl das Ersetzen technisch ohne nennenswerten Aufwand möglich ist, gibt es eine Reihe von Hindernissen und Risiken, die bei einem tatsächlichen Austausch in einem produktiven System mit vielen vorhandenen Anfragen auftreten werden sowie zu einem deutlich höheren Aufwand führen und nicht beabsichtigte Ergebnisse des neuen CEP-Systems zur Folge haben können. Der Grund dafür ist, dass derzeit weder CEP-Systeme noch Anfragesprachen standardisiert sind. Neben an SQL angelehnten Sprachen gibt es CEP-Systeme, in denen Anfragen in völlig anderen Sprachen formuliert werden. Jede vorhandene Anfrage muss somit bei einem Wechsel neu erstellt werden. Aber auch wenn von einem SQL-basierendem CEP-System zu einem anderen auf SQL basierenden gewechselt wird und die Syntax beider Systeme ähnlich ist, muss dennoch jede einzelne Anfrage intensiv überprüft werden, da jedes CEP-System eine einzigartige Semantik besitzt und identische Anfragen auf verschiedenen Systemen meistens zu verschiedenen Ergebnissen führen. Weil die Unterschiede häufig nur von kleinen Details ausgemacht werden, ist das Finden äquivalenter Anfragen eine mühsame und fehleranfällige Aufgabe, die dennoch gewissenhaft ausgeführt werden muss, da die Ergebnisse z. B. die Grundlage für automatische Aktionen in der Cloud sind. In [Bot+10] finden sich Beispiele für kleine, aber weitreichende Unterschiede zwischen verschiedenen CEP-Systemen sowie ein erster Ansatz zur allgemeinen Beschreibung ihres Verhaltens. Bevor nicht das Verhalten von unterschiedlichen CEP-Systemen exakt beschrieben und vorhergesagt werden kann, können keine Automatismen zum Portieren von Anfragen entwickelt werden. Die ersten Arbeiten in diesem Bereich fangen allerdings gerade erst an und bei vielen kommerziellen Systemen wird ihre Semantik als Geschäftsgeheimnis zurückgehalten oder ist sogar den Entwicklern nicht exakt bekannt (z. B. weil es nur eine Implementierung der Operatoren und keine mathematische Beschreibung von ihnen gibt), wodurch schnelle Erfolge unwahrscheinlich werden.

3.3.3 Datenbank

Für die anzuschließende Datenbank ergibt sich die gleiche Situation wie zuvor bei dem CEP-System. Es wird zu jedem Datenbankmanagementsystem nur ein passender Adapter benötigt, der Anfragen über JMS-Topics entgegennimmt und die Ergebnisse wieder als Nachrichten über JMS-Topics versendet. Da Java zum Einsatz kommt, kann ein einziger Adapter auf der Basis von JDBC für alle gängigen Datenbankmanagementsysteme verwendet werden. Dadurch müssen Anfragen nicht wie bei dem CEP-System nach einem Austausch umgeschrieben werden. Auch das Problem unterschiedlicher Semantiken ergibt sich bei der Datenbank nicht, weil SQL schon sehr lange international standardisiert ist und sich alle Hersteller an diesen Standard halten.[1] Auf die Verwendung von Hersteller-spezifischen Erweiterungen sollte allerdings verzichtet werden, weil sie zum einen nicht von JDBC unterstützt werden und zum anderen den Wechsel zwischen verschiedenen Systemen erschweren.

3.3.4 Visualisierung

Datenströme werden von *CEP4Cloud* über einen Web-Server auf zwei verschiedene Arten als Web-Dokumente angeboten. Für optisch ansprechende und animierte Cockpits wird das auf Flash-Technologie beruhende Werkzeug *MashZone* [MZ] eingesetzt. Es ist sehr benutzerfreundlich und flexibel. In *MashZone* können die Nutzer ihre Cockpits nach Belieben schnell und komfortabel über eine grafische Benutzeroberfläche selbst aufbauen. Dazu wird sukzessive aus einer umfangreichen Menge vorgegebener Diagrammarten eine ausgewählt und mit einem oder mehreren Datenströmen verbunden. Das Resultat ist ein fertiges Echtzeit-Diagramm, das innerhalb eines Cockpits frei positioniert und skaliert werden kann sowie sich optisch dem eigenen Geschmack anpassen lässt.

Die Nachteile von *MashZone* sind ein recht großer Ressourcenverbrauch durch Flash und Probleme bei der Aktualisierung, wenn die Daten schnell strömen. Aus diesen Gründen werden als Alternative Cockpits in Form von Java-Applets angeboten. Die Echtzeit-Diagramme entstehen dabei mit Hilfe der Bibliothek *JFreeChart* [JFC] als Komposition vieler einzelner Teile (wie z. B. Hintergründe, Legenden, Skalen oder Zeiger) in Form von Java-Objekten und können dadurch nahezu beliebige Ausprägungen annehmen. Der Nachteil von diesem Ansatz ist, dass sowohl die Kompostion als auch die Positionierung und Skalierung der fertigen Diagramme in Java-Code beschrieben werden müssen. Sollen nicht nur vorgefertigte Cockpits angeboten werden, sondern auch benutzerdefinierte, dann muss zusätzlich die fehlende Benutzeroberfläche implementiert werden, die aus Nutzereingaben den entsprechenden Java-Code generiert und in das Java-Applet einbindet.

[1] Der aktuelle Standard ist [Int09].

3.3.5 Überwachungsebenen

Für alle bisher betrachteten Komponenten konnten fertig implementierte Lösungen eingesetzt werden. Bei den Sensoren ist hingegen mehr Implementierungsaufwand notwendig. Jeder zu überwachender Objekt-Typ benötigt einen eigenen Sensor-Typ in Java. Die Sensor-Typen folgen alle dem gleichen Prinzip. Sie messen zu bestimmten Zeitpunkten oder auf Anforderung ihres Agenten den Wert einer Metrik und geben ihn an ihren Agenten weiter. Der Agent reichert den gemessenen Wert noch um den Zeitpunkt der Messung und dem Schlüssel des überwachten Objekts an und sendet alle Informationen als ein Ereignis in den zu diesem Sensor-Typ zugehörigen Nachrichtenkanal. Dieses Vorgehen erzeugt das in Abbildung 3.2 dargestellte Layout der Datenströme und hat den Nachteil, dass für jede einzelne Messung eine Nachricht erzeugt wird. Das folgende Beispiel verdeutlicht das Problem.

Beispiel
Angenommen es wird eine Infrastruktur bestehend aus 10 Maschinen überwacht. Auf jeder Maschine werden 10 Systeme ausgeführt und unter jedem System laufen 100 Prozesse. Ein Prozess in dieser Infrastruktur soll einmal pro Sekunde ausgelesen werden. Die Überwachung dieser kleinen Infrastruktur würde damit zu $10 \cdot 10 \cdot 100 = 10\,000$ Nachrichten pro Sekunde führen, die nur aufgrund der Prozesse entstehen.

Neben den Prozessen gibt es noch eine Vielzahl weiterer Objekt-Typen in einer Cloud, die ebenfalls in einer großen Anzahl vorkommen und deshalb viele Nachrichten erzeugen (z. B. Threads). Anstatt zur Stabilität und Leistungsfähigkeit einer Cloud beizutragen, würde eine solche Implementierung die Probleme sogar noch verschlimmern, indem es die für Cloud Computing sehr wichtigen Netzwerkkapazitäten selbst aufbraucht. Das Problem wird in *CEP4Cloud* durch ein zweistufiges Verfahren gelöst. Zuerst wird die Anzahl der Nachrichten durch sogenannte Bündelungen drastisch reduziert. Anschließend wird jede einzelne Nachricht vor dem Versenden komprimiert, um die Größe der Nachrichten zu verringern. Die Netzwerkbelastung wird dadurch minimiert. Um die logische Sicht auf die Datenströme für das Überwachungssystem und seine Komponenten wiederherzustellen, werden nach dem Empfang der Nachrichten diese in umgekehrten Reihenfolge zuerst dekomprimiert und die rekonstruierten Bündel wieder in einzelne Ereignisse zerlegt. Ein geeignetes Kompressionsverfahren muss für jede Metrik gesondert ausgewählt werden. Für Prozesse bietet sich beispielsweise eine Differenzkodierung an, bei der nur Veränderungen mitgeteilt werden. Mögliche Veränderungen können sein, dass ein neuer Prozess entstanden ist (alle Informationen müssen gesendet werden), dass ein bestehender Prozess beendet wurde (nur die Prozess-ID muss gesendet werden) oder dass sich eine Kennzahl von einem bestehenden Prozesses verändert hat (die Differenz zu dem letzten übermittelten Wert muss gesendet werden). Die Bündelungen geschehen in *CEP4Cloud* auf zwei verschiedene Arten, die im Folgenden beschrieben werden.

Bündelung von Ereignissen

Einige Sensor-Typen sind für das Auslesen vieler parallel existierender Objekte desselben Typs (z. B. Prozesse auf einem System) verantwortlich. Anstatt jedes dieser Objekte getrennt auszulesen und als Ereignis zu versenden, können alle Objekte gleichzeitig ausgelesen und die Werte in Form einer Tabelle angeordnet werden. Die resultierende Tabelle (z. B. Prozesstabelle) kann als eine einzige Nachricht versendet werden. Bei allen Sensor-Typen, bei denen eine Bündelung von Ereignissen möglich ist, reduziert sich die Anzahl der Nachrichten bei N vorhandenen Objekten von N auf 1.

Bündelung von Sensorströmen

Manche Sensor-Typen können mit anderen Sensor-Typen synchronisiert werden. Zum Beispiel ist es problemlos möglich, auf einer Maschine die Metriken zu CPU, Netzwerk und Speicher gleichzeitig zu messen und in einer einzigen Nachricht zu versenden. Für N miteinander synchronisierbare Sensor-Typen ist damit, im Gegensatz zu N Nachrichten ohne Synchronisation, nur noch eine Nachricht notwenig. Da allerdings jeder Sensor-Typ einen eigenen Datenstrom verwendet, muss für die Bündel ein gesonderter Nachrichtenkanal angelegt werden. Nach dem Eintreffen am Überwachungssystem muss ein Bündel aufgelöst werden und die einzelnen darin befindlichen Ereignisse müssen auf die richtigen Datenströme aufgeteilt werden.

Eine weitere Möglichkeit, um die Anzahl der Nachrichten pro Zeiteinheit zu reduzieren, ist das Verringern der zeitlichen Frequenz, mit der ein konkreter Sensor Ereignisse erzeugt. Die Zeitintervalle müssen so gewählt werden, dass zum einen die einzelnen Komponenten von *CEP4Cloud* noch zuverlässig arbeiten können und zum anderen nicht zu viele überflüssige Nachrichten erzeugt werden. Ein erster Ansatz zur Auswahl der richtigen Frequenzen können die in SLAs direkt festgelegten oder daraus ableitbaren Werte sein. Während ein Unterschreiten dieser Werte eine Überwachung eines SLAs unmöglich macht, führt ein Überschreiten zu übermäßig vielen Nachrichten. Auch die Eigenschaften der überwachten Objekt-Typen sollten einen Einfluss auf die Wahl der Frequenzen haben. Threads, die innerhalb weniger Nanosekunden ihren Zustand ändern können, müssen deutlich häufiger ausgelesen werden als eine Festplatte, dessen Werte sich nur langsam verändern.

In *CEP4Cloud* werden in Anlehnung an die drei Schichten der NIST-Definition (siehe Abschnitt 2.1.1.1) drei Ebenen festgelegt. Für jede Ebene werden nachfolgend die Implementierung ihrer Sensor-Typen sowie die dadurch verfügbaren Metriken, die jederzeit durch neue Sensor-Typen erweitert werden können, besprochen. Die Besprechung endet jeweils mit der Präsentation von exemplarischen Anfragen und möglichen Reaktionen auf ihre Ergebnisse. Die Aktionen im Action Framework von *CEP4Cloud* sind kontinuierlich ausgeführte Java-Anwendungen, die die vorhandenen JMS-Topics selbstständig auswerten. Für Manipulationen an Objekten wird der Agent, der über ein eigenes JMS-Topic seine Befehle erhält, entsprechend modifiziert.

3.3.5.1 Infrastrukturebene

Auf der Infrastrukturebene sollen sowohl virtualisierte Infrastrukturen als auch die physische Infrastruktur einer Cloud vollständig überwacht werden. Im Cloud Computing ist es dabei, wie bei jedem anderen verteilten System, wichtig über jedes einzelne Glied eines Verbundes detailliert informiert zu sein, weshalb *CEP4Cloud* einzelne physische und virtuelle Maschinen beobachtet. Aufgrund der Plattformunabhängigkeit und der Isolation durch eine VM gibt es aus Java heraus keine Möglichkeiten, die Leistungsdaten einer Maschine direkt abzufragen. Ein Sensor-Typ auf dieser Ebene muss daher für jede in Frage kommende Plattform, auf der er als Instanz ausgeführt werden soll, jeweils eine native Implementierung bereithalten. Eine Java-Methode kann dann über JNI (Java Native Interface) auf die passende native Implementierung zugreifen, um die entsprechende Messung durchzuführen. Die nativen Implementierungen müssen allerdings nicht extra vorgenommen werden, weil viele existierende Produkte zum Überwachen von Infrastrukturen, wie z.B. *Ganglia* (Abschnitt 3.1.2) oder *Nagios* (Abschnitt 3.1.4), unzählige und für alle gängigen Plattformen ausgelegte Implementierungen bereits beinhalten und sie einfach wiederverwendet werden können. Sofern keine exotischen Metriken benötigt werden, kann die sehr kleine und quelloffene Java-Bibliothek *Sigar*[2] (System Information Gatherer and Reporter) [Sig] anstatt eines vollständigen Monitoring-Produkts verwendet werden. *Sigar* ist eine Sammlung von nativen Implementierungen gängiger Metriken für verschiedene Plattformen und bietet über fertig vorliegende JNI-Methoden direkten Zugriff auf sie. Die Auswahl der korrekten nativen Implementierung wird selbständig von *Sigar* getroffen. Auf der Infrastrukturebene von *CEP4Cloud* kommt *Sigar* wegen der einfachen Integration als fertige Java-Bibliothek zum Einsatz.

Metriken

In Tabelle 3.2 sind alle in *CEP4Cloud* verfügbaren Metriken der Infrastrukturebene aufgelistet. Weil die jeweils aktuellen Werte zu CPU, Festplatte, Netzwerk und Speicher gleichzeitig ermittelt werden können, sind die vier dafür verantwortlichen Sensor-Typen miteinander synchronisiert und ihre Datenströme gebündelt. Alle Messwerte werden in einer einzigen Nachricht über den für dieses Bündel angelegten Datenstrom `MachineSensor` zum Broker geschickt. Dort wird jede Nachricht in vier Teile zerlegt und je ein Teil den eigentlich beabsichtigten Datenströmen `CPUSensor` für den Sensor-Typ der CPU, `DiskSensor` für den Sensor-Typ der Festplatte, `MemorySensor` für den Sensor-Typ des Hauptspeichers und `NetworkSensor` für den Sensor-Typ des Netzwerks zugeführt. Das Aufteilen des Bündels kann effizient von dem CEP-System erledigt werden, indem vier Anfragen, die jeweils nur aus einer Projektion bestehen, den Datenstrom des Bündels auf die vier korrekten Datenströme abbilden.

[2] *Sigar* ist ein leichtgewichtiges Nebenprodukt von Hyperic (Abschnitt 3.1.3) und wird dort ebenfalls zum Auslesen der Leistungsdaten einzelner Maschinen verwendet.

Bündel	Datenstrom	Metrik
MachineSensor	CPUSensor	Anzahl der Kerne
		Taktrate in MHz
		Von Systemprozessen genutzte Kapazität in %
		Von Benutzerprozessen genutzte Kapazität in %
		Ungenutzte Kapazität in %
	DiskSensor	Total verfügbarer Externspeicher in TB
		Freier Externspeicher in TB
	MemorySensor	Total verfügbarer Hauptspeicher in GB
		Freier Hauptspeicher in GB
	NetworkSensor	Anzahl empfangener Pakete pro Sekunde
		Anzahl gesendeter Pakete pro Sekunde
		Anzahl empfangener Bytes pro Sekunde
		Anzahl gesendeter Bytes pro Sekunde

Tabelle 3.2: Metriken der Infrastrukturebene von CEP4Cloud

Anfragen

Auf der Basis der zur Verfügung stehenden Metriken lassen sich auf der Infrastrukturebene die Leistung der einzelnen Maschinen kontinuierlich überwachen, neue Metriken ableiten sowie Engpässe und Anomalien erkennen. In Anfrage 3.1 werden durch eine einfache Selektion alle Maschinen, die eindeutig durch ihren Schlüssel `ip_address` identifiziert sind, ermittelt, bei denen nur noch weniger als 20 % des verfügbaren Externspeichers (`disk_total`) ungenutzt sind (`disk_free`).

```
SELECT ip_address
FROM   DiskSensor
WHERE  disk_free < 0.2 * disk_total;
```

Anfrage 3.1: Maschinen mit ausgelastetem Externspeicher ermitteln

Anfrage 3.2 zeigt, wie mit Hilfe der verallgemeinerten Projektion Berechnungen kontinuierlich durchgeführt werden können und dadurch indirekt neue Metriken verfügbar werden. Zu jeder Maschine (`ip_address`) wird auf der Basis der empfangenen bzw. der gesendeten Bytes pro Sekunde (`bytes_in` bzw. `bytes_out`) und der Anzahl der empfangenen bzw. gesendeten Netzwerkpakete pro Sekunde (`pakets_in` bzw. `pakets_out`) die durchschnittliche Paketgröße der empfangen und gesendeten Daten pro Sekunde berechnet.

```
SELECT ip_address,
       bytes_in / pakets_in,
       bytes_out / pakets_out
FROM   NetworkSensor;
```

Anfrage 3.2: Durchschnittliche Größe der Netzwerkpakete berechnen

Eine komplexere Analyse des Netzwerkverkehrs ist in Anfrage 3.3 dargestellt. Für jede Maschine (`ip_address`) wird eine eigene Gruppe erstellt. Innerhalb dieser Gruppe werden jeweils der Durchschnitt (**`avg`**) und die Standardabweichung (**`stddev`**) für die eintreffenden und ausgehenden Datengrößen berechnet. Die Grundlage zur Berechnung dieser Aggregate sind alle in der letzten Minute gemessenen Werte. Eine Ausgabe wird nur dann produziert, wenn eine der Standardabweichungen um mehr als ein Drittel von dem zugehörigen Durchschnittswert abweicht. Die **`HAVING`**-Klausel ist dabei nur die Kurzschreibweise für eine Selektion nach der Gruppierung. Alle Maschinen mit unregelmäßigem Netzwerkverkehr werden dadurch gemeldet.

```
SELECT   ip_address,
         avg(bytes_in),  stddev(bytes_in),
         avg(bytes_out), stddev(bytes_out)
FROM     NetworkSensor WINDOW(RANGE 1 MINUTE)
GROUP BY ip_address
HAVING   stddev(bytes_in)  * 3 > avg(bytes_in)
OR       stddev(bytes_out) * 3 > avg(bytes_out);
```

Anfrage 3.3: Starke Schwankungen im Netzwerkverkehr erkennen

Aktionen

Während bei Anfrage 3.2 eine neue Metrik abgeleitet wird, werden bei den beiden anderen Anfragen Maschinen gefiltert, die bestimmte Eigenschaften (zu Ende neigender Externspeicher, unruhige Kommunikation) aufweisen. Durch Aktionen im Action Framework kann vielfältig darauf reagiert werden. Auf Ergebnisse von Anfrage 3.1 kann entweder eine Benachrichtigung versendet werden oder automatisch eine Vergrößerung des Externspeichers veranlasst werden. Die Schwankungen im Netzwerkverkehr bei den durch Anfrage 3.3 gemeldeten Maschinen können unterschiedliche Ursachen haben. Als erste Reaktion könnten die laufenden Prozesse auf der Maschine analysiert werden oder jedes einzelne Netzwerkpaket auf schadhaften Inhalt überprüft werden. Viele Anfragen wie z. B. Anfrage 3.1 melden eine besondere Situation nicht nur einmalig, sondern solange sie besteht bei jedem Eintreffen neuer Werte. Aus diesem Grund muss für jede Aktion eine zeitliche Sperre (Timeout) definiert werden, damit sie bei identischen Eingaben nicht permanent ausgeführt wird.

3.3.5.2 Systemebene

Die überwachten Objekte auf der Systemebene von *CEP4Cloud* sind einzelne Betriebssysteminstanzen, die anhand der auf ihnen laufenden Prozesse kontrolliert werden. Bei der Implementierung des für Prozesse verantwortlichen Sensor-Typs ergibt sich eine ähnliche Situation wie auf der Infrastrukturebene. Java hat keinen direkten Zugriff auf das Betriebssystem und unterschiedliche Betriebssysteme haben verschiedene Schnittstellen, über die Daten zu Prozessen abgerufen werden können. Daher muss wieder über JNI auf eine für das jeweilige Betriebssystem zugeschnittene native Implementierung zurückgegriffen werden.

Metriken
Auf der Systemebene von *CEP4Cloud* werden die Werte zu den in Tabelle 3.3 aufgelisteten Metriken für jeden laufenden Prozess ermittelt. Da in der Regel sehr viele Prozesse gleichzeitig ausgeführt werden, werden alle Prozesse gemeinsam ausgelesen und die resultierenden Ereignisse in einer Nachricht gebündelt und über den extra angelegten Datenstrom `SystemSensor` verschickt. Innerhalb von *CEP4Cloud* müssen alle Nachrichten aus diesem Datenstrom wieder in einzelne Ereignisse zerlegt und separat in den dafür vorgesehenen Datenstrom `ProcessSensor` eingefügt werden.

Bündel	Datenstrom	Metrik
SystemSensor {	ProcessSensor {	ID
		Name
		Besitzer
		CPU-Verbrauch in %
		Speicherverbrauch in %
		Priorität

Tabelle 3.3: Metriken der Systemebene von CEP4Cloud

Anfragen
Mit den Informationen auf der Systemebene können detailliertere Einblicke gewonnen werden, als auf der Infrastrukturebene. Während auf der Infrastrukturebene die Gesamtbelastung der einzelnen Ressourcen überwacht wird, kann auf der Systemebene der anteilige Verbrauch der einzelnen laufenden Systeme ermittelt werden, wie in Anfrage 3.4. In dieser Anfrage wird für jedes einzelne System (`system_key`) in der überwachten Cloud der Gesamtverbrauch (**`sum`**) an CPU (`process_cpu`) und Hauptspeicher (`process_memory`) sowie die durchschnittliche Priorität (`process_priority`) aller laufenden Prozesse berechnet. Die Anfrage nutzt aus, dass alle Prozesse eines Systems zum gleichen Zeitpunkt gemessen werden

und damit die Aggregate über alle zu einem Zeitpunkt aktiven Prozesse berechnet werden.

```
SELECT   system_key,
         sum(process_cpu),
         sum(process_memory),
         avg(process_priority)
FROM     ProcessSensor
GROUP BY system_key;
```

Anfrage 3.4: Aggregierte Informationen zu Systemen berechnen

Mit Hilfe von Anfrage 3.5 lassen sich alle Prozesse (`process_key`) in der überwachten Cloud filtern, die in den letzten 20 Sekunden durchschnittlich mehr als 50 % der CPU ihrer Maschine in Anspruch genommen haben.

```
SELECT   process_key, avg(process_cpu)
FROM     ProcessSensor WINDOW(RANGE 20 SECONDS)
GROUP BY process_key
HAVING   avg(process_cpu) > 50;
```

Anfrage 3.5: Prozesse mit hohem CPU-Verbrauch ermitteln

Aktionen

Mögliche Aktionen auf der Systemebene sind neben den Benachrichtigungen das (Neu-)Starten und Stoppen von Prozessen sowie das Ändern von einzelnen Prioritäten. Anfrage 3.4 ist ein guter Ausgangspunkt, um Einfluss auf die Lastverteilung in der überwachten Cloud zu nehmen. Aufbauend auf der Kenntnis des Gesamtverbrauchs der einzelnen Systeme, können diese so umverteilt werden, dass die Belastung jeder Maschine gleich ist. Dadurch lassen sich unter anderem Überlastungen einzelner Maschinen vermeiden, was ein wichtiger Punkt im Hinblick auf die Einhaltung von SLAs ist. Des Weiteren kann erkannt werden, ob zu viele Maschinen in der Cloud in Betrieb sind. Wenn die aktuelle Arbeitslast auf weniger Maschinen aufgeteilt werden kann, dann können durch die Abschaltung der überflüssigen Maschinen Kosten eingespart werden. Dies kann sowohl für den Betreiber (physische Maschinen) als auch für die Nutzer (virtuelle Maschinen) einer Cloud gelten. Die in Anfrage 3.5 ermittelten Prozesse können für weitere Untersuchungen einer entsprechenden Komponente oder einem Administrator gemeldet werden. Dort kann dann entschieden werden, ob der hohe Verbrauch normal oder abnormal ist und ob ein Eingreifen notwendig ist.

3.3.5.3 Anwendungsebene

Auf der Anwendungsebene werden einzelne laufende Anwendungen direkt und virtualisierte Anwendungen zusätzlich über ihre Laufzeitumgebungen überwacht und gesteuert. Sowohl für Anwendung als auch für Laufzeitumgebungen müssen individuelle Sensor-Typen bereitgestellt werden, die relevante Ausgaben der überwachten Objekte als Datenströme zur Verfügung stellen. Zur Steuerung muss auf öffentlich zugängliche Methoden zurückgegriffen werden. In Abschnitt 2.1.2.3 wurde erwähnt, dass der Einsatz von JVMs ein populärer Ansatz zur Anwendungsvirtualisierung ist und im Cloud Computing verwendet wird. Aus diesem Grund wurden in *CEP4Cloud* Sensor-Typen zur Überwachung von JVMs implementiert. Dadurch lässt sich jede in einer JVM laufende Anwendung indirekt überwachen. Eine JVM bietet eine breite Palette an Kennzahlen, von denen die wichtigsten als Datenströme in *CEP4Cloud* durch entsprechende Sensor-Typen verfügbar sind. Die notwendigen Schnittstellen und Methoden zum Auslesen der Werte werden von einer JVM selbst angeboten, sodass die Implementierung der Sensor-Typen wenig Aufwand erfordert. Dazu werden mit JMX (Java Management eXtensions) [JSRb] über entsprechende Methoden, deren Schnittstellen für einzelne Komponenten einer JVM fest definiert sind, die jeweils aktuellen Werte einer Metrik abgefragt. In den ersten Versionen von JMX war es nur möglich, die Daten innerhalb einer JVM abzufragen. Mittlerweile ist es aber auch möglich, JVMs aus anderen JVMs heraus zu überwachen. Dadurch können die Sensor-Typen vollständig in Java entwickelt werden. Darüber hinaus bietet eine JVM – ebenfalls mit Hilfe von JMX – öffentlich zugängliche Methoden an, um ihre Komponenten zu steuern. Beispielsweise kann auf diese Weise die Speicherbereinigung (engl. *Garbage Collection*, GC) von außen angestoßen werden.

Metriken
Für einen effizienten Transport der gemessenen Werte zu den in *CEP4Cloud* verfügbaren Metriken einer JVM, die in Tabelle 3.4 aufgelistet sind, kommen sowohl die Bündelung von Ereignissen als auch die Bündelung von Datenströmen zum Einsatz. In dem Bündel `JVMSensor` werden die Werte zu allen einer JVM direkt betreffenden Metriken parallel gemessen und als eine große Nachricht versendet. Innerhalb von *CEP4Cloud* wird dieses Bündel in die thematischen Datenströme `JVMClassesSensor` und `JVMMemorySensor` aufgeteilt. Weil zu einer einzelnen JVM mehrere GCs und mehrere Threads gehören können, werden diese jeweils in einem Durchgang gemeinsam ausgelesen und als Menge von Ereignissen in einer Nachricht gebündelt. Nach dem Transport der Nachricht über die Datenströme `GCSensor` bzw. `ThreadSensor` kann die Menge wieder in einzelne Ereignisse (pro Thread und pro GC jeweils ein Ereignis) zerlegt und in die Datenströme `JVMGCSensor` bzw. `JVMThreadSensor` eingefügt werden.

Bündel	Datenstrom	Metrik[3]
JVMSensor	JVMClassesSensor	Anzahl aktuell geladener Klassen
		Anzahl insgesamt geladener Klassen
		Anzahl insgesamt entladener Klassen
	JVMMemorySensor	Initiale Größe des Heap-Speichers
		Maximale Größe des Heap-Speichers
		Aktuell belegte Größe des Heap-Speichers
		Initiale Größe des Nicht-Heap-Speichers
		Maximale Größe des Nicht-Heap-Speichers
		Aktuell belegte Größe des Nicht-Heap-Speichers
GCSensor {	JVMGCSensor	Anzahl ausgeführter Speicherbereinigungen
		Gesamtdauer aller Speicherbereinigungen
ThreadSensor {	JVMThreadSensor	Name und ID
		Gegenwärtiger Zustand
		CPU-Verbrauch in %
		Insgesamt im Zustand „Wartend“ verbrachte Zeit
		Anzahl Transitionen in Zustand „Warten“
		Insgesamt im Zustand „Blockiert“ verbrachte Zeit
		Anzahl Transitionen in Zustand „Blockiert“

Tabelle 3.4: Metriken der Anwendungsebene von CEP4Cloud

Speichermanagement der JVM

Bevor passende Anfragen und Aktionen zu den verfügbaren Metriken vorgestellt werden, muss ein etwas tieferer Einblick in das Speichermanagement einer JVM [Sunb; Sun06] erfolgen. Wie jede andere Maschine auch, verfügt eine JVM über einen fest begrenzten Arbeitsspeicher, dem sogenannten Heap-Speicher. In diesem Speicher werden nicht mehr benötigte Objekte automatisch von der Speicherbereinigung entfernt. Die Größe des Heap-Speichers wird beim Start einer JVM entweder über einen Parameter vorgegeben oder automatisch berechnet und kann sich zur Laufzeit nicht mehr ändern. Nach dem Start einer JVM steht nicht zwangsläufig der gesamte reservierte Heap-Speicher zur Verfügung, sondern nur ein Teil davon. Die Größe dieses initialen Heap-Speichers kann ebenfalls über einen Parameter vorgegeben oder automatisch von der JVM gesetzt werden. Neben den Parametern für die maximale und die initiale Größe des Heap-Speichers gibt es noch zwei weitere wichtige Parameter, mit denen jeweils der Faktor angegeben werden kann, um den der verfügbare Heap-Speicher bei Bedarf vergrößert bzw. verkleinert wird. Aus Sicht des Speichermanagements einer JVM ist der verfügbare Heap-Speicher logisch in zwei Teile aufgeteilt. Der erste Teil mit der Bezeichnung „Young Generation“ ist für die Allokation von Speicher für neue Objekte gedacht. Wenn ein Objekt in diesem Teil eine gewisse Anzahl an

[3] Alle Speichergrößen werden in Megabyte und alle Zeitwerte in Millisekunden angegeben.

Speicherbereinigungen ohne entfernt zu werden überstanden hat, dann wird es in den zweiten Teil, der sogenannten „Tenured Generation“, verschoben. Ein als „Permanent Generation“ bezeichneter zusätzlicher Speicher ist für Daten vorgesehen, die die JVM benötigt und auf denen keine dynamische Speicherbereinigung notwendig ist. In der Permanent Generation werden zum Beispiel Klassen geladen und entladen. Der eigentliche Heap-Speicher besteht nur aus der Young bzw. Tenured Generation. Die Permanent Generation bekommt einen extra Speicherbereich zugewiesen, der als Nicht-Heap-Speicher bezeichnet wird. Hinter der Idee der Gruppierung von Objekten im Heap-Speicher in jung (Young Generation) und alt (Tenured Generation) steht die Erwartung, dass in der Gruppe der jungen Objekte viel häufiger eine Bereinigung, bei der immer ein großer Anteil entfernt wird, vorgenommen werden muss als in der Gruppe der alten, in der ein nur kleiner Teil bei einer Speicherbereinigung entfernt wird. Dies führt dazu, dass zur Bereinigung der Young Generation ein anderer Algorithmus optimal ist, als ein optimaler Algorithmus zum Bereinigen der Tenured Generation. Daher gibt es, wie bereits im Datenmodell aus Abbildung 3.3 dargestellt, innerhalb einer JVM mehrere verschiedene Speicherbereinigungen. Die dynamischen Speicherbereinigungen der JVM sind als letztes Mittel anzusehen, um eine teure Reorganisation des Heap-Speichers zu vermeiden. Aus diesem Grund werden sie auch nur dann automatisch ausgeführt, wenn mindestens ein Teil des Heap-Speichers keinen Platz für ein neues Objekt mehr zur Verfügung hat und die Alternative eine globale Reorganisation durch Vergrößerung der Teile wäre.

Anfragen

Um den Aufwand aller ausgeführten Speicherbereinigungen zu messen, wird in Anfrage 3.6 die durchschnittliche Ausführungsdauer jeder vorhandenen GC berechnet. Dazu muss lediglich die Gesamtdauer (`gc_collection_time`) durch die Gesamtanzahl aller ausgeführten Speicherbereinigungen (`gc_total_collections`) dividiert werden. Anhand dieser Informationen kann die Konfiguration der einzelnen GCs über diverse Parameter optimiert werden (siehe dazu [Sunb; Sun06]).

```
SELECT gc_key, gc_collection_time / gc_total_collections
FROM   JVMGCSensor;
```

Anfrage 3.6: Durchschnittliche Dauer der Bereinigungen berechnen

In Anfrage 3.7 wird für jede JVM jeweils die Veränderung des Verbrauchs an Heap-Speicher berechnet. Aus dem aktuellen und dem in den letzten 100 ms durchschnittlichen Verbrauch an Heap-Speicher wird die Differenz der beiden Verbräuche in Prozent ermittelt. Mit dieser Information lässt sich feststellen, mit welcher Geschwindigkeit der Verbrauch gestiegen oder gefallen ist.

```
SELECT j.jvm_key, (used_heap_memory - avg_heap_memory)
                                / avg_heap_memory * 100
FROM   JVMMemorySensor j,
       (SELECT jvm_key,
               avg(used_heap_memory) AS avg_heap_memory
        FROM JVMMemorySensor WINDOW(RANGE 100 MILLISECONDS)
        GROUP BY jvm_key) past
WHERE  j.jvm_key = past.jvm_key;
```

Anfrage 3.7: Geschwindigkeit der Verbräuche an Heap-Speicher berechnen

Zu jeder JVM wird in der Unteranfrage `past` der durchschnittliche Verbrauch an Heap-Speicher (`used_heap_memory`) in den letzten 100 Millisekunden berechnet. Über die Bildung des kartesischen Produkts zwischen Unteranfrage und `JVMMemorySensor` sowie der anschließenden Selektion werden die Datensätze, die zu der gleichen JVM gehören, miteinander verbunden. Zum Abschluss kann mit der verallgemeinerten Projektion die prozentuale Differenz berechnet werden.

Aktionen

In [Sunb; Sun06] werden als Nebenbedingungen der dynamischen Speicherbereinigung ein sparsamer Umgang mit der CPU, hohe Reaktionszeiten der laufenden Anwendung und ein niedriger Speicherverbrauch genannt. Die Prioritäten, mit denen eine JVM mit Standardeinstellungen diese drei Nebenbedingungen zu erfüllen versucht, entsprechen der genannten Reihenfolge. Im Folgenden soll dem niedrigen Speicherverbrauch mehr Beachtung geschenkt werden. Neben einem als verständlich geltenden sparsamen Umgang mit Ressourcen sind im Fall einer JVM zwei Dinge besonders hervorzuheben. Zum einen läuft sie als Prozess unter einem Betriebsystem. Wenn die JVM mehr Speicher verbraucht, dann verbraucht damit auch der entsprechende Prozess mehr Speicher. Mit steigendem Verbrauch steigt die Wahrscheinlichkeit, dass das Betriebsystem einen Teil des genutzten Speichers von dem schnellen Hauptspeicher der physischen Maschine in den deutlich langsameren virtuellen Speicher auf einem externen Speichermedium auslagert. Eine weitere Folge von hohem Speicherverbrauch ist, dass die einzelnen Teile des Heap-Speichers vergrößert werden, um einer wachsenden Anzahl an Speicherbereinigungen entgegenzuwirken. In beiden Fällen ist zu erwarten, dass die Anwendung in der JVM langsamer ausgeführt wird. Im ersten Fall muss auf Speicheranforderungen aus dem virtuellen Speicher gewartet werden und im zweiten Fall dauert eine Speicherbereinigung länger, da die zu bereinigenden Teile vergrößert wurden. In [Suna] stellt Sun zusätzlich einen allgemein positiven Zusammenhang zwischen dem Verbrauch an Heap-Speicher und der Geschwindigkeit der ausgeführten Anwendung her. Aus diesen Gründen sind viele freie und kommerzielle Werkzeuge sowie Beratungsleistungen diverser Unternehmen zum Reduzieren des Speicherverbrauchs eigener Anwendungen für die

JVM verfügbar. Stellvertretend sollen an dieser Stelle zwei Arbeiten von IBM erwähnt werden. Zum einen wurde eine Strategie entwickelt, bei der die Speicherbereinigung von außen angestoßen wird, sobald der Verbrauch einen einstellbaren Schwellwert überschreitet [IBM11]. Damit wird verhindert, dass die Bereiche des Heap-Speichers vergrößert werden, weil dies nur bei automatischen Speicherbereinigungen möglich ist. Ein weiterer Vorteil ist, dass der Zeitpunkt der Bereinigung so gewählt werden kann, dass keine kritische Phase der laufenden Anwendung davon beeinträchtigt wird (natürlich muss dazu die exakte Semantik der Anwendung bekannt sein). Zum anderen gibt es mit dem freien *Memory Analyzer* [MAT] ein sehr mächtiges Werkzeug, mit dem Schnappschüsse des Heap-Speichers analysiert werden können, um problematische Stellen im Programmcode zu identifizieren. Ein Beispiel für Speicher-ineffizienten Programmcode ist die Java-Schleife aus Programm-Fragment 3.8.

```
for(int i = 0; i < loops; i++) {
     // Deklarieren von Hilfsvariablen
     KlasseA hilfsVariable1 = new KlasseA();
     KlasseB hilfsVariable2 = new KlasseB();
           ⋮                ⋮
     // Anfang des Schleifen-Codes
           ⋮                ⋮
}
```

Programm-Fragment 3.8: Speicher-ineffiziente Schleife in Java

In der dargestellten Schleife werden in jeder einzelnen Iteration die Hilfsvariablen neu deklariert, womit der im Schleifenkörper allozierte Speicherplatz linear mit der Anzahl der Iterationen wächst. Deutlich effizienter wäre eine einmalige Deklaration aller Hilfsvariablen vor dem Eintritt in die Schleife und die Wiederverwendung innerhalb der Schleife durch Überschreiben. Allerdings gibt es viele Situationen, in denen der Quelltext nicht vorliegt und damit der ineffiziente Programmcode zwangsweise ausgeführt werden muss. Zum Beispiel hat der Betreiber einer Laufzeitplattform für Java-Anwendungen in einer Cloud keinen Einfluss auf die Anwendungen, die seine Kunden bei ihm ausführen. Um den Einfluss von ineffizienten Programmcode dennoch in Grenzen zu halten, soll die Idee der erzwungen Speicherbereinigung zu bestimmten Zeitpunkten aufgegriffen werden. Die Grundlage dazu bildet die Anfrage 3.7, die in der Lage ist, Speicher-ineffiziente Schleifen zu erkennen. Wann immer diese Anfrage ein Anwachsen des Verbrauchs um mehr als 10 % einer JVM meldet, wird dort durch eine entsprechende Aktion im Action Framework eine Speicherbereinigung von außen erzwungen. Um ein permanentes Ausführen dieser Aktion zu verhindern, wird die Aktion für eine konkrete JVM nach einer erzwungen Speicherbereinigung für die nächsten 200 ms blockiert. Zur Evaluierung dieser Stra-

tegie wurde sie in einem Experiment, bei dem eine Anwendung in einer JVM eine Speicher-ineffiziente Schleife ausführt, der Standardkonfiguration gegenübergestellt. Zum Einsatz kam dabei die JVM von Apple[4] mit 16 MB Speicherplatz für die Young Generation. Die Ergebnisse aus diesem Experiment sind in Abbildung 3.4 dargestellt.

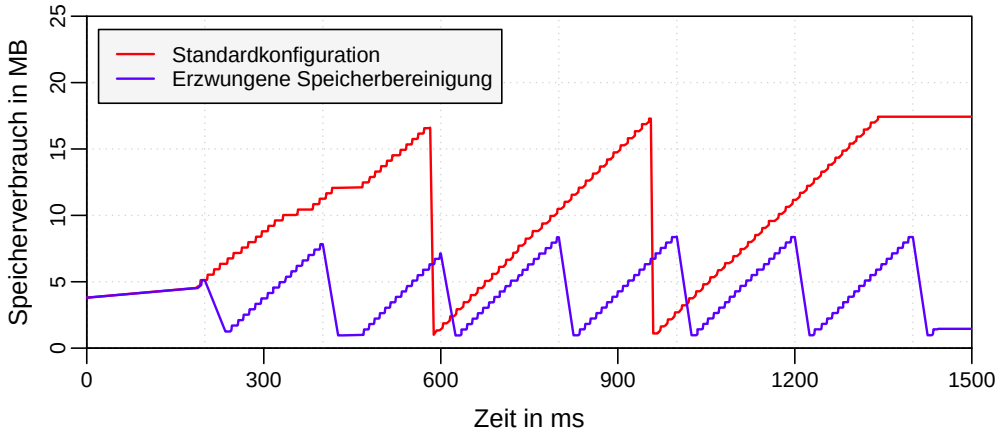

Abbildung 3.4: Optimierung des Java-Heaps

Es fällt sofort auf, dass eine automatische Speicherbereinigung bei voller Young Generation mit durchschnittlich 6 ms deutlich schneller ausgeführt wird, als eine erzwungene Speicherbereinigung mit durchschnittlich 26 ms. Der Grund dafür ist, dass bei einer automatischen Speicherbereinigung nur der Teil des Heap-Speichers bearbeitet wird, der voll ist (in diesem Fall die Young Generation) [Ort01]. Eine erzwungene Speicherbereinigung hat hingegen immer zur Folge, dass der komplette verfügbare Heap-Speicher gesäubert wird. Das Ziel, den Verbrauch möglichst gering zu halten, wurde mit durchschnittlich 4,4 MB (erzwungene Speicherbereinigung) gegenüber 9,6 MB (Standardkonfiguration) erreicht. Während die vollständige Abarbeitung der Schleife in der Standardkonfiguration nach 1.341 ms erreicht ist, dauert es mit 1.425 ms bei den erzwungenen Speicherbereinigungen 84 ms länger. Wird nur die reine Ausführungszeit für die Schleife betrachtet, dann stehen 1.329 ms in der Standardkonfiguration 1.243 ms bei den erzwungenen Speicherbereinigungen gegenüber. Das entspricht einer Steigerung der Geschwindigkeit um 7 %, wodurch nachgewiesen ist, dass ein kleinerer Speicherverbrauch schnellere Laufzeiten zur Folge hat. Das wichtigste Ergebnis des Experiments macht sich nach Abarbeitung der Schleife bemerkbar. Während in der Standardkonfiguration die JVM mit einer vollständig verunreinigten Young Generation weiterläuft, befindet sich der Heap-Speicher bei den erzwungenen Speicherbereinigungen in einem nahezu optimalen Zustand. Die sich daraus ergebenden Vorteile wurden bereits diskutiert. Aus dem Experiment lassen sich für jede mög-

[4] in der Version 1.6.0_26-b03-383

liche Situation konkrete Anweisungen zur Optimierung ableiten. Die erste Situation ist, dass der Quelltext der auszuführenden Anwendung vorliegt. In diesem Fall können ineffiziente Stellen im Programmcode durch eine Überwachung der Ausführung identifiziert und im Quelltext korrigiert werden. Anschließend kann der Speicherbedarf der korrigierten Anwendung, ebenfalls durch Überwachen der Ausführung, ermittelt werden. Ist der Speicherbedarf bekannt, dann kann bei allen zukünftigen Ausführungen die Größe des Heap-Speichers auf den Bedarf der Anwendung abgestimmt werden. Die schnelle automatische Speicherbereinigung wird dann für einen kleinen Speicherverbrauch zu geringen Kosten sorgen. Der zweite mögliche Fall ergibt sich immer dann, wenn zwar die Anwendung bekannt ist, aber der Quelltext nicht vorliegt. In diesem Fall müssen Speicher-ineffiziente Phasen während der Ausführung hingenommen werden. Allerdings kann auch hier durch eine Überwachung der Ausführung der eigentliche Bedarf ermittelt und bei zukünftigen Ausführungen der Heap-Speicher entsprechend begrenzt werden. Für die ersten beiden Situationen, in denen eine bekannte Anwendung mehrfach ausgeführt wird, sind die offiziellen Empfehlungen von Sun, nach denen die Größe des Heap-Speichers angepasst werden soll und auf erzwungene Speicherbereinigungen komplett zu verzichten ist, optimal geeignet. Es gibt aber noch eine dritte mögliche Situation, in der Anwendungen unbekannt sind oder nur selten bis einmalig ausgeführt werden. Dieser Fall trifft zum Beispiel auf den Betreiber einer Laufzeitplattform für Bytecode in einer Cloud zu, in der alle Nutzer ihre Anwendungen selbstständig ausführen. Für den Betreiber ist es unmöglich, die Größe der Heap-Speicher der JVMs auf den Bedarf von einzelnen und unbekannten Anwendungen abzustimmen. Den angebotenen JVMs müssen im Gegenteil sogar deutlich größere Heap-Speicher zur Verfügung gestellt werden, denn sollte der verfügbare Heap-Speicher nicht ausreichen, dann führt die erste fehlgeschlagene Allokation von Speicher zu einem Absturz von Anwendung und JVM. Ein großer Heap-Speicher bedeutet allerdings, dass aufgrund der sehr wenigen automatischen Speicherbereinigungen, die bei den wenigen Ausführungen allerdings die laufende Anwendung sehr lange blockieren werden, sich die negativen Auswirkungen von Speicher-ineffizienten Programmen optimal entfalten können und den Heap-Speicher anwachsen lassen. Die Folgen sind unter anderem dauerhaft langsamere Laufzeiten der Anwendung. Für die Betreiber von Laufzeitplattformen ist ein gelegentliches Anstoßen der Speicherbereinigung von außen eine sehr gute Strategie, um unbekannte Anwendungen effizient ausführen zu können.

3.3.6 Erweiterte Analysen

Bisher bezogen sich alle Anfragen auf immer nur eine einzelne Überwachungsebene. Die besondere Stärke von CEP kommt dann zum Ausdruck, wenn Metriken von verschiedenen Ebenen gemeinsam betrachtet oder Muster gesucht werden.

3.3.6.1 Korrelationen

Die isolierte Betrachtung der einzelnen Ebenen reicht für eine optimale und ganzheitliche Überwachung nicht aus, da innerhalb einer Cloud Abhängigkeiten und Beziehungen zwischen verschiedenen Ebenen bestehen. Um Zusammenhänge (insbesondere Verknüpfungen zwischen Problemen und ihren Ursachen) erkennen zu können, müssen Messwerte von unterschiedlichen Ebenen miteinander korreliert werden. Damit sinnvolle Korrelationen möglich werden, muss das Design der Datenströme auf einem Datenmodell (wie dem aus Abbildung 3.3) basieren, in dem alle vorhandenen Abhängigkeiten berücksichtigt sind. Bei Anfrage 3.9 handelt es sich um eine Korrelation zwischen den Hauptspeichern auf der Infrastrukturebene und den Prozessen auf der Systemebene.

```
SELECT process_key, process_memory
FROM   ProcessSensor p,
       (SELECT   ip_address, max(process_memory) AS maximum
        FROM     ProcessSensor WINDOW(RANGE 2 SECONDS)
        GROUP BY ip_address) sub,
       MemorySensor WINDOW(RANGE 2 SECONDS) m
WHERE  p.ip_address = sub.ip_address
  AND  p.ip_address = m.ip_address
  AND  process_memory > 0.8 * maximum
  AND  mem_free / mem_total < 0.2;
```

Anfrage 3.9: Prozesse mit kritischem Speicherverbrauch ermitteln

Die Anfrage kann wie folgt interpretiert werden. In einem ersten Schritt werden auf der Infrastrukturebene diejenigen Maschinen gefiltert, bei denen nur noch weniger als 20 % des totalen Hauptspeichers frei sind (`mem_free / mem_total < 0.2`). Für jede dieser Maschinen werden dann in einem zweiten Schritt auf der Systemebene pro Maschine alle Prozesse gesucht, die auf ihr ausgeführt werden (`p.ip_address = m.ip_address`) und die zusätzlich einen hohen Verbrauch an Hauptspeicher haben (`process_memory > 0.8 * maximum`). Dazu wird in der Unteranfrage `sub` pro Maschine der größte Speicherverbrauch eines einzelnen laufenden Prozesses in der Variable `maximum` gespeichert. Zum Abschluss werden alle Prozesse, deren Speicherverbrauch mindestens 80 % von dem zu dieser Maschine gehörenden Wert (`p.ip_address = sub.ip_address`) in `maximum` beträgt, an die Ausgabe weitergeleitet. Mit anderen Worten meldet die Anfrage automatisch die Prozesse (und zwar unabhängig von ihrem System) mit den höchsten Speicherverbräuchen auf einer Maschine, sobald dort der noch freie Hauptspeicher knapp wird.

3.3.6.2 Mustererkennung

Neben der Korrelation ist die Mustererkennung ein weiteres mächtiges Verfahren von CEP, um komplexe Ereignisse finden zu können. Muster können nicht nur auf den rohen Sensorströmen erkannt werden, sondern auch auf von dem CEP-System abgeleiteten Datenströmen, wodurch noch mächtigere Analysen möglich werden. Mit Hilfe von Anfrage 3.10 können alle Threads erkannt werden, bei denen sich über eine Zeitspanne hinweg (fünf Messungen in Folge) der Zustand nicht geändert hat.

```
SELECT   threadKey, threadState
FROM     JVMThreadSensor
MATCHING( PARTITION BY thread_key
          MEASURES     threadKey String, threadState String
          PATTERN      'ab{4}'
          WITHIN       1 MINUTE
          DEFINE       a DO  threadKey    = thread_key,
                             threadState  = thread_state
                       b AS  thread_key   = threadKey
                         AND thread_state = threadState );
```

Anfrage 3.10: Threads ohne Transitionen erkennen

In *PIPES* kann in der **FROM**-Klausel ein beliebiger Eingabe- oder abgeleiteter Datenstrom angegeben werden, auf dem die Mustererkennung, die durch das Schlüsselwort **MATCHING** eingeleitet wird, ausgeführt werden soll. Weil in dem Sensorstrom `JVMThreadSensor` die Messdaten aller Threads vorkommen, wird durch die **PARTITION BY**-Klausel für jeden individuellen Thread ein separater Teilstrom erzeugt und die Mustererkennung auf jeden dieser Teilströme angewandt. Da mehrere aufeinander folgende Datenstromelemente auf Gleichheit überprüft werden sollen, werden Variablen zum Zwischenspeichern von Werten für nachfolgende Datenstromelemente benötigt. Daher werden für den Schlüssel und den Zustand eines Threads in der **MEASURES**-Klausel jeweils eine Variable für die Aufnahme einer Zeichenkette deklariert. In der **PATTERN**-Klausel wird das zu erkennende Muster als regulärer Ausdruck angegeben. Gesucht wird in der Anfrage das Muster `ab{4}`, bei dem auf ein `a` viermal ein `b` folgen muss. Die **WITHIN**-Klausel gibt an, wie groß der Zeitraum ist, in dem nach dem Muster gesucht werden soll. Zum Schluss wird über **DEFINE** festgelegt, dass das `b` aus dem Muster mit dem `a` identisch sein soll. Dazu werden die angelegten Hilfsvariablen benötigt, in denen der Schlüssel und der Zustand des Datenstromelements `a` abgelegt werden. Danach wird für die nachfolgenden Datenstromelemente definiert, dass sie genau dann dem `b` im festgelegten Muster entsprechen, wenn ihre Zustände und Schlüssel mit den von `a` zwischengespeicherten Werten identisch sind.

Zusammenfassung

Aktuell verfügbare Produkte zum Überwachen von IKT beschränken sich auf das Sammeln, Darstellen und Speichern einer breiten Palette von Messdaten. Kein Produkt erlaubt effektive Analysen der Daten oder benutzerdefinierte Manipulationen an überwachten Objekten. Aufgrund der Größe, Komplexität und Dynamik von Clouds ergeben sich ergänzend zu den funktionalen Anforderungen aus Kapitel 1 aufgrund von großen Mengen an Daten und Überwachungsregeln weitere Herausforderungen, die ein Cloud Monitoring bewältigen muss. CEP-Systeme sind ideale Kandidaten für die technologische Basis solcher Werkzeuge. Die allgemeine Architektur von CEP-Anwendungen kann für Cloud Computing angepasst und erweitert werden. Um die für eine ganzheitliche Überwachung notwendigen Korrelationen zwischen verschiedenen Metriken und Ebenen zu ermöglich, wird ein umfangreiches Datenmodell benötigt, das die in einer Cloud vorhandenen Beziehungen und Abhängigkeiten auf das Design der Datenströme abbildet. Damit die für Cloud Computing wertvollen Netzwerkkapazitäten von einem Cloud Monitoring nicht zu sehr belastet werden, kann der Transport der Messdaten durch Reduzierung der Anzahl und Größe von Nachrichten stark optimiert werden, ohne dass dabei ein Informationsverlust in Kauf genommen werden muss. Zur Ausführung einer großen Masse an parallel ablaufenden Anfragen eignet sich der Einsatz einer deklarativen Anfragesprache mit einer kontinuierlichen Optimierung der gesamten Anfragenmenge. Neben einer hohen Effizienz von deklarativen Sprachen hat das Kapitel anhand vieler Beispiele zusätzlich gezeigt, dass auch komplizierte Analysen komfortabel und in wenigen Zeilen durch sie ausdrückbar sind.

4 Erweiterungen

In diesem Kapitel wird anhand einer typischen Problemstellung im Cloud Computing, die in Abschnitt 4.1 vorgestellt wird, demonstriert, dass sich mit Hilfe einer Überwachung auf der Basis der vorgestellten Architektur mit Datenströmen und Complex Event Processing einfach und effizient nützliches Wissen generieren sowie effektiv darauf reagieren lässt. Die zur Problemlösung notwendigen Anfragen und Algorithmen werden in Abschnitt 4.2 am Beispiel von *CEP4Cloud* systematisch entwickelt und in Abschnitt 4.3 evaluiert. Am Ende des Kapitels werden in Abschnitt 4.4 mögliche Optimierungen besprochen.

4.1 Problemstellung

In Rahmen der Definition von Cloud Computing in Abschnitt 2.1.1.3 wurde das Zusammenfassen einzelner Ressourcen zu größeren Pools als eine charakteristische Eigenschaft von Cloud Computing vorgestellt. Innerhalb dieser Ressourcenpools müssen die ständig eintreffenden Nachfragen so verteilt werden, dass zum einen jede Nachfrage wie versprochen bedient werden kann (keine Einbußen in der Dienstqualität durch Überlastung einzelner Ressourcen) und dass zum anderen die vorhandenen Ressourcen möglichst optimal ausgenutzt werden, um durch eine hohe Auslastung die Dienste mit minimalen Kosten anbieten zu können. Aufgrund der hohen Dynamik in einer Cloud muss kontinuierlich entschieden werden, wie eine Menge von Verbrauchern auf eine meistens deutlich kleinere Menge von Ressourcen (z. B. viele virtuelle auf wenige physische Maschinen) abzubilden ist und wie bei Problemen (z. B. plötzliches und unerwartetes Ungleichgewicht oder Ausfall einzelner Ressourcen) durch eine dynamische Neuverteilung zu reagieren ist, anstatt wie derzeit in der Praxis üblich nur zu Beginn eines neuen Dienstes einen passenden Platz für ihn zu suchen. Bei allen Situationen dieser Art handelt es sich um die eindimensionale Variante der „Bin-Packing“-Aufgabe, die als NP-vollständiges Problem bekannt ist [Kar72]. Erschwerend zu der NP-Vollständigkeit kommt hinzu, dass sich die Datengrundlage permanent verändert und dadurch das Problem ständig neu zu lösen ist. Eine brauchbare Lösung der Aufgabe muss deshalb sehr schnell (nahe Echtzeit) vorliegen. Im nächsten Abschnitt soll mit den Mitteln von *CEP4Cloud* eine allgemeine Lösung für solche Probleme am Beispiel der folgenden Problemstellung erarbeitet werden.

Für dieses Problem wurden aus dem Datenmodell von *CEP4Cloud* (siehe Abbildung 3.3) die grobgranularste Entität als Ressource und die feingranularste Entität als Verbraucher ausgewählt, weil bei dieser Kombination die Anzahl der Verbraucher sehr viel höher als die der Ressourcen ist. Im Blickpunkt der weiteren Betrachtungen steht somit eine Anwendung, die alle auszuführenden Aufgaben als JVM-Threads innerhalb einer Menge von Maschinen aufteilt. Die Anwendung arbeitet als „Black Box“ und vollkommen willkürlich, wodurch Threads auf jeder der zur Verfügung stehenden Maschinen plötzlich entstehen oder terminieren und Threads während ihrer Lebenszeit für unterschiedliche Belastungen sorgen. Im nächsten Abschnitt muss zunächst durch eine vorangehende Untersuchung mehr Wissen über die Anwendung erstellt werden. Aufbauend auf diesem Wissen kann anschließend durch kontinuierliches Verschieben der Threads ihrer ineffizienten Verteilung entgegengewirkt werden.

4.2 Dynamische und proaktive Lastbalancierung

Für einen schnellen und ersten Überblick ist es in vielen Fällen hilfreich, zu untersuchende Daten in geeigneter Art und Weise visuell darzustellen (in *CEP4Cloud* können in der Visualisierungskomponente (Abschnitt 3.3.4) Datenströme direkt grafisch als Web-Dokument abgerufen werden). In dem vorliegenden Fall von CPU-konsumierenden Threads ist ein Liniendiagramm, das den jeweiligen Verbrauch über einen Zeitraum wiedergibt, eine passende Form der Darstellung. Dabei fällt auf, dass es grob zwei Typen von Threads gibt. Die Threads des einen Typs erzeugen eine annähernd konstante CPU-Last während die Threads des anderen Typs in ihrem Verbrauch an Rechenzeit schwanken und dadurch ein Muster erzeugen. In Abbildung 4.1 sind jeweils die Messdaten eines konstanten und eines variablen Threads dargestellt.

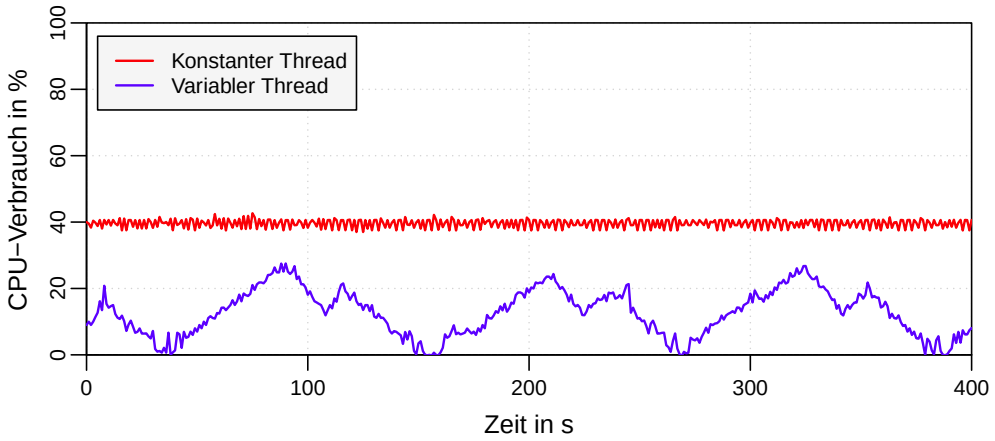

Abbildung 4.1: Unterschiedliche Typen von Threads

Nach noch genauerer Inspektion der variablen Threads kann festgestellt werden, dass das entsprechende Muster pro Thread periodisch wiederholt wird und dass bereits gesehene Muster immer wiederkehren (beides sind Annahmen, die für den weiteren Verlauf von der Anwendung erfüllt werden). Das Auftreten von kennzeichnenden Mustern ist typisch für eine Cloud. Zum Beispiel werden Nutzer identische Dienste wiederholt für die gleichen Aufgaben einsetzen und dadurch ein Nutzerprofil erzeugen. Darüber hinaus ist eine Cloud häufig saisonalen Schwankungen (Tageszeit, Werktag/Wochenende, Feiertage, etc.) in der Anzahl der Nachfragen unterworfen. Es ist daher vorteilhaft, solche unbekannten Muster systematisch zu suchen und zu merken, denn wann immer aktuell gemessene Daten dem Anfang eines bereits bekannten Musters entsprechen (und bekannt ist, dass mit hoher Wahrscheinlichkeit Muster wiederholt auftreten können), dann kann das Wissen über die Muster zum Treffen von Vorhersagen verwendet werden. Das zukünftige Verhalten einer Cloud wird dadurch nicht nur berechenbar, sondern auch planbar.

4.2.1 Vorbereitungen

Am Beispiel der variablen Threads sollen kurz die Grundlagen von Mustermanagement skizziert werden. Bevor Muster erkannt werden können, muss als erstes eine repräsentative und persistente Datenbasis bestehend aus bekannten Mustern aufgebaut werden. Dazu müssen Perioden zuverlässig erkannt werden können, denn sobald die Periode eines Datenstroms mit wiederkehrendem Muster vorliegt, können die Daten einer einzelnen Periode als Referenz dauerhaft abgespeichert werden. Ein sehr effizientes und speicherschonendes Verfahren zum Erkennen von Perioden in Datenströmen stammt aus dem Algorithmus *STAGGER* [EAE06]. Abbildung 4.2 gibt die Arbeitsweise der Periodendetektion von *STAGGER* auf einem idealen variablen Thread wieder.

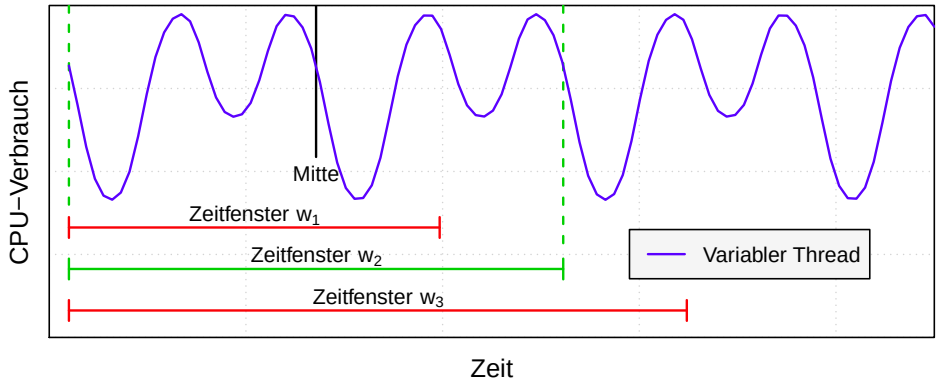

Abbildung 4.2: Detektion von Perioden

Auf die strömenden Sensordaten zu einem variablen Thread werden mehrere Zeitfenster (w_1, w_2 und w_3 in Abbildung 4.2) mit wachsenden Fenstergrößen $\mathcal{W}_1 < \mathcal{W}_2 < \mathcal{W}_3$ gelegt. Aufgrund der Annahme, dass ein variabler Thread aus einem festen Muster besteht und dieses sich periodisch wiederholt, müssen sich die Fenster nicht bewegen, sondern können an dem ersten Messwert fixiert werden. Dadurch ist es nicht mehr notwendig mit mehreren Zeitfenstern zu starten. Stattdessen kann mit einem einzigen Zeitfenster angefangen werden, das so lange vergrößert wird, bis die Periode erkannt wurde. Sobald in einem Zeitfenster alle benötigten Daten nach $\mathcal{W}_i$ Zeiteinheiten verfügbar sind, werden innerhalb des entsprechenden Zeitfensters die Daten genau in der zeitlichen Mitte $\mathcal{W}_i/2$ geteilt und die beiden resultierenden Hälften auf Ähnlichkeit untersucht (siehe dazu Seite 83). Wenn die zwei Ausschnitte aus einem Zeitfenster w_i als ähnlich angesehen werden, dann wurde eine Periode mit der Frequenz $\mathcal{W}_i/2$ erfolgreich gefunden und die Daten aus einer der beiden Hälften können als Referenzmuster für diesen variablen Thread in einer Datenbank dauerhaft abgelegt werden. In Abbildung 4.2 trifft dies auf das Zeitfenster w_2 zu. Die Zeitfenster w_1 und w_3 hingegen können die vorhandene Periode nicht erkennen, weil w_1 zu klein und w_3 zu groß ist. Nachdem eine Periode erfolgreich erkannt und das Muster gemerkt wurde, kann die Periodendetektion für diesen variablen Thread beendet werden. Ansonsten muss das Verfahren mit einem etwas größeren Zeitfenster fortgesetzt werden. Die Implementierung einer Periodendetektion ist in *CEP4Cloud* einfach, denn es werden nur Zeitfenster benötigt. Diese sind bereits in *PIPES* vorhanden und können so modifiziert werden, dass sie dynamisch mit einem Datenstrom wachsen. Auf der durch das Zeitfenster erzeugten Datenmenge kann die Untersuchung auf Ähnlichkeit direkt ausgeführt werden. In Situationen, in denen tatsächlich viele verschieden große Zeitfenster parallel benötigt werden, ist die folgende Modifikation der Zeitfenster in *PIPES* hilfreich. Um beim Erkennen von Perioden keinen Speicher unnötig zu verschwenden, sollten Daten lediglich einmal zwischengespeichert und von den Zeitfenstern gemeinsam verwendet werden. Daher genügt es, nur die Daten von dem größten der Zeitfenster zwischenspeichern zu lassen. Alle kleineren Zeitfenster benötigen eine Teilmenge der Daten des größten Zeitfensters und können sich diese aus seinem Zwischenspeicher besorgen. Da die Menge der möglichen Muster von variablen Threads endlich ist, sind in der Lernphase ab einem bestimmten Zeitpunkt alle Muster in der Datenbank abgelegt. In einigen Anwendungen kommt es vor, dass sich periodische Muster mit der Zeit verändern. In diesen Fällen wird das Verfahren nicht beendet, nachdem eine Periode erkannt wurde, sondern sehr viele Zeitfenster gleiten mit dem entsprechenden Datenstrom weiter, um neue Perioden erkennen zu können. Der dauerhafte Verbleib der Zeitfenster ist aufgrund der gemeinsamen Nutzung einer Datenbasis nicht kritisch. Darüber hinaus ist es durch die Speicherteilung möglich, sehr viele und sich nur geringfügig in ihrer Größe unterscheidende Fenster anzulegen, die auch notwendig sind, um Perioden zuverlässig finden zu können.

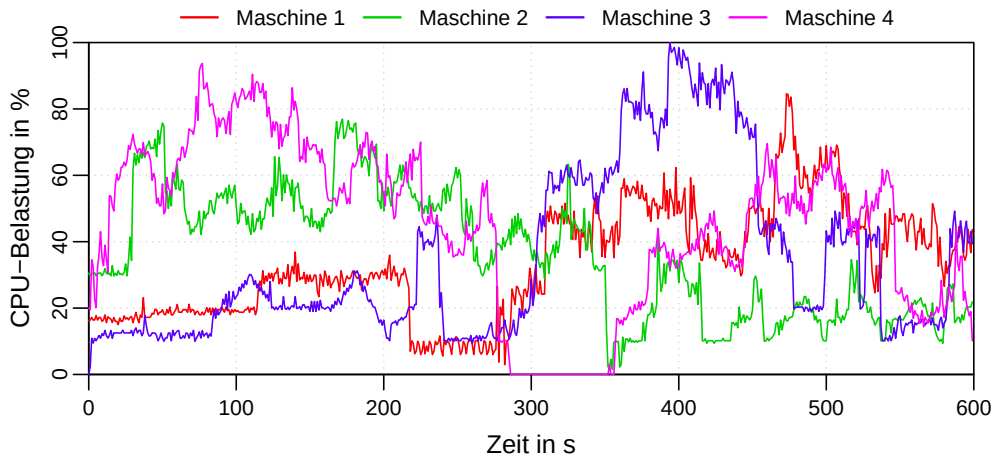

Abbildung 4.3: Lastenverteilung ohne Balancierung

Nach dem Herausarbeiten des Vorhandenseins von zwei Typen von Threads und dem Ablegen aller Muster der variablen Threads in der Datenbank von *CEP4Cloud*, kann eine Gegenmaßnahme für die ungleiche Verteilung der Last, von der ein repräsentativer Ausschnitt über einen Zeitraum von 10 Minuten auf vier Maschinen in Abbildung 4.3 dargestellt ist, erarbeitet und im Action Framework implementiert werden. In dem gezeigten Ausschnitt sind an den Flanken gut die Zeitpunkte erkennbar, an denen auf einer Maschine ein Thread gestartet bzw. beendet wurde. Zwischen zwei Flanken ist die CPU-Belastung entweder konstant (z. B. Maschinen 1 und 3 am Anfang), periodisch variierend (z. B. Maschinen 2 und 4 am Anfang) oder chaotisch. Letzteres lässt sich dadurch erklären, dass mehrere variable Threads gleichzeitig aktiv sind und sich überlagern. Die durchschnittliche CPU-Belastung beträgt 37,3 % (Maschine 1: 33,5 %, Maschine 2: 36,2 %, Maschine 3: 34,8 % und Maschine 4: 44,6 %) und könnte problemlos von der Hälfte der Maschinen erledigt werden.[1] Maschine 4 wird sogar (vor und nach dem Zeitpunkt 300) für einen Zeitraum von über einer Minute überhaupt nicht verwendet. Durch eine kontinuierliche Überwachung dieser Metriken können einfach überflüssige Maschinen erkannt und abgeschaltet werden, um unnötige Kosten zu vermeiden. In den meisten Fällen gravierendere Auswirkungen hat allerdings das umgekehrte Extrem, bei dem eine Maschine überlastet wird (Maschine 3 um den Zeitpunkt 400 herum). Durch Verzögerungen der Prozesse und Threads auf dieser Maschine kommt es zu einer schlechteren Dienstqualität, die im vorliegenden Fall hätte vermieden werden können, da auf allen anderen Maschinen genügend freie Rechenkapazitäten zur Verfügung standen. Aufgrund der schweren Folgen von Überlastungen hat ihre Vermeidung durch eine gleichmäßige Belastung höchste Priorität.

[1] Die verwendeten Maschinen sind identisch und dadurch erzeugt ein Thread auf jeder der Maschinen die gleiche prozentuale Last. In nicht homogenen Umgebungen muss eine Übersetzungstabelle zwischen je zwei Maschinen erstellt und verwendet werden.

Zum Abschluss der Vorbereitungen müssen die Messwerte des Sensor-Typs, der für die CPU auf der Infrastrukturebene verantwortlich ist, aufbereitet werden. In Abbildung 4.3 sind bei den Messwerten ganzer Maschinen deutlich Oszillationen, die zu teilweise starken Differenzen zwischen zwei aufeinander folgenden Werten führen, zu beobachten. Die Gründe dafür sind neben unvermeidbaren Messfehlern weitere nicht zu beeinflussende Faktoren (z. B. der Scheduler des Betriebsystems). Deswegen werden die Werte für die aktuelle CPU-Belastung einer Maschine nicht über den direkt von den Sensoren erzeugten Datenstrom `CPUSensor`, sondern über den von Anfrage 4.1 davon abgeleiteten Datenstrom `CPUBelastung`, abgefragt. Diese Anfrage glättet die Werte, indem sie durch ein partitionierendes Zählfenster (siehe Seite 32) für jede Maschine (`ip_address`) einen eigenen Teilstrom erzeugt, daraus die letzten fünf Messwerte (`cpu_total`) nimmt und den Durchschnitt von ihnen als aktuelle CPU-Belastung für diese Maschine ausgibt.

```
SELECT   ip_address, avg(cpu_total) AS cpu
FROM     CPUSensor WINDOW(PARTITION BY ip_address
                                  ROWS 5)
GROUP BY ip_address;
```

Anfrage 4.1: Datenstrom CPUBelastung ableiten

4.2.2 Zusammenspiel der Komponenten

Mit Hilfe der besprochenen Vorbereitungen und aller in *CEP4Cloud* vorhandenen Komponenten kann eine geeignete Lastbalancierung, die durch gleichmäßiges Verteilen der Last die Leistungsfähigkeit und Verfügbarkeit der Dienste sicherstellt, entwickelt und implementiert werden. Weil ein möglichst gutes und schnelles Verfahren, das sich kontinuierlich anwenden lässt, anstatt eines perfekten benötigt wird, besteht die grundlegende Idee darin, durch sukzessives Verschieben von einzelnen Threads zwischen zwei Maschinen die Last gleichmäßig unter ihnen aufzuteilen. Da Threads in ihrem Verbrauch schwanken und jederzeit Threads neu entstehen und terminieren können, gibt es keine allgemein optimale Lösung, sondern nur optimale Lösungen zu jedem festen Zeitpunkt, gegen die das Verfahren konvergiert. Trotz der Dynamik in der Menge der aktiven Threads, werden die meisten zu einem bestimmten Zeitpunkt aktiven Threads auch noch zum nächsten Zeitpunkt aktiv sein und ihr CPU-Verbrauch zum nächsten Zeitpunkt wird von ihrem aktuellen nicht sehr stark abweichen. Dadurch hat die Konvergenz des Verfahrens gegen die optimale Lösung eines festen Zeitpunkts auch positive Auswirkungen auf die nachfolgenden Zeitpunkte, da deren optimale Lösungen untereinander und zu der des festen Zeitpunkts mit hoher Wahrscheinlichkeit ähnlich sein werden.

4.2.2.1 Auswahl von Maschinen

Der inkrementelle Verfahrensschritt ist nur auf zwei Maschinen anwendbar. Deshalb müssen in allen Fällen, in denen mehr Maschinen zur Verfügung stehen, zwei geeignete ausgewählt werden. Es ist leicht ersichtlich, dass die jeweils am stärksten bzw. am schwächsten belasteten Maschinen eine gute Wahl sind, um durch Verschieben eines einzelnen Threads zwischen ihnen einem Gesamtausgleich aller Maschinen näher zu kommen. Weil es sich bei der Bestimmung dieser Maschinen um eine Filterung handelt, kann die Auswahl effizient und einfach von einer Anfrage im CEP-System durchgeführt werden.

```
SELECT CPUBelastung.*
FROM   CPUBelastung,
       (SELECT max(cpu) AS maxCPU, min(cpu) AS minCPU
        FROM   CPUBelastung WINDOW(RANGE 1 SECOND)) s
WHERE  cpu = maxCPU OR cpu = minCPU;
```

Anfrage 4.2: Extrem belastete Maschinen bestimmen

Anfrage 4.2 bestimmt die jeweils extrem belasteten Maschinen, indem nur diejenigen Datensätze aus dem Datenstrom `CPUBelastung` an die Ausgabe weitergeleitet werden, deren Wert in `cpu` dem maximalen oder minimalen aller aktuellen Werte von `cpu` in `CPUBelastung` entspricht. Damit die Selektion ausgeführt werden kann, werden zuvor in der Unteranfrage `s` das Maximum (**`max`**`(cpu)`) und Minimum (**`min`**`(cpu)`) in der letzten Sekunde berechnet und als zusätzliche Attribute durch das kartesische Produkt allen Datensätzen aus `CPUBelastung` hinzugefügt. Da es mehrere Maschinen geben kann, die maximal bzw. minimal belastet sind, müssen eventuell aus diesen Teilmengen zufällig je eine Maschine als Eingabe für die Lastbalancierung gezogen werden.

4.2.2.2 Klassifizieren von Threads

Nachdem die beiden Zielmaschinen bekannt sind, muss in einer zweiten Auswahl derjenige Thread von der höher belasteten Maschine bestimmt werden, der durch Verschieben den besten Ausgleich zur Folge hat. Aufgrund der vorhandenen variablen Threads führt ein Ausgleich zum Ausführungszeitpunkt der Lastbalancierung nicht zwangsläufig zu einem positiven Ergebnis, weil durch Veränderungen in dem CPU-Verbrauch der variablen Threads sich die Lastenverteilung schnell verändern kann. Aus diesem Grund ist ein Ausgleich zwischen zwei Maschinen zu einem späteren Zeitpunkt anzustreben, wozu der zukünftige CPU-Verbrauch von variablen Threads der höher belasteten Maschine bekannt sein muss.

Zur Vorhersage des CPU-Verbrauchs von variablen Threads müssen diese erst noch von den konstanten Threads, deren CPU-Verbrauch sich nicht ändern wird, unterschieden werden. Ein gutes Maß für die Unterscheidung von konstanten und variablen Threads ist die Varianz. Auf der Grundlagen der Varianz in einer Stichprobe (die Größe der Stichprobe wird über *stichprobenumfang* eingestellt) ist Anfrage 4.3 in der Lage, zuverlässig konstante und variable Threads voneinander zu trennen.

```
SELECT   thread_key, variance(cpu), avg(cpu)
FROM     JVMThreadSensor WINDOW(PARTITION BY thread_key
                                ROWS stichprobenumfang)
GROUP BY thread_key
HAVING   variance(cpu) > 1.1 * schwellwert
OR       variance(cpu) < 0.9 * schwellwert;
```

Anfrage 4.3: Thread-Typen bestimmen

Für jeden aktiven Thread wird mit Hilfe eines partitionierenden Zählfensters ein eigener Teilstrom erzeugt, der jeweils die in *stichprobenumfang* festgelegte Anzahl der jüngsten Messwerte beinhaltet. Anschließend werden für jeden Teilstrom die Varianz (**variance**(cpu)) und der Durchschnitt (**avg**(cpu)) des CPU-Verbrauchs berechnet und an die Ausgabe weitergeleitet. Eine Aktion im Action Framework nimmt das erste produzierte Ergebnis zu jedem Thread entgegen und ordnet anhand eines vorgegebenen Grenzwerts (*schwellwert*) dem zugehörigen Thread bei Unterschreiten dieses Grenzwerts den Typ „konstant“ und bei Überschreiten den Typ „variabel“ zu. Weil es bei Varianzen, die nur weniger als 10 % von dem festgelegten Grenzwert abweichen, zu Fehlern bei der Klassifizierung kommen kann, werden in der Anfrage nur die Ergebnisse ausgegeben, bei denen eine fehlerfreie Klassifizierung möglich ist. Die Anfrage verzögert daher bei einer schlechten Stichprobe die Ausgabe so lange, bis ein verwertbares Ergebnis vorliegt. Der durchschnittliche CPU-Verbrauch (Glätten der oszillierenden Messwerte) wird bei dieser Gelegenheit mitberechnet, da er die von einem konstanten Thread zukünftig erzeugte Belastung ausdrückt. Für variable Threads hingegen hat dieser Wert keinerlei Bedeutung und kann ignoriert werden.

Um den für einen zukünftigen Zeitpunkt zu erwartenden CPU-Verbrauch eines variablen Threads berechnen zu können, muss ihm das richtige Muster aus der Datenbank zugeordnet werden. Dazu werden die ersten Messwerte jedes variablen Threads, die direkt in die Datenbank fließen können, als Stichprobe benötigt. Anhand der Stichprobe und den entsprechend großen Anfängen aller bekannten Muster aus der Datenbank kann jeweils die Ähnlichkeit zwischen ihnen bestimmt werden. Am Ende wird das Muster mit der größten Ähnlichkeit zu der Stichprobe eines variablen Threads ihm als Muster zugeordnet. In Abbildung 4.4 sind eine Stichprobe und zwei bekannte Muster als interpolierte Kurven dargestellt.

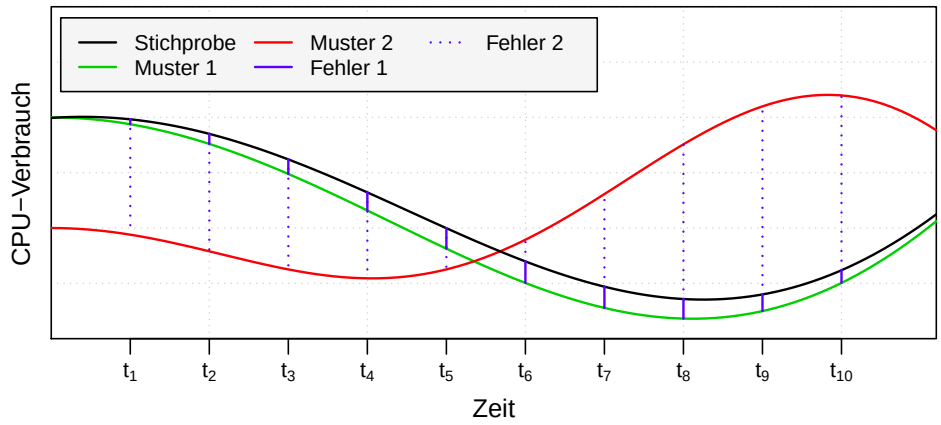

Abbildung 4.4: Musterklassifizierung

Zur Bestimmung der Ähnlichkeit zweier Kurven kann ein beliebiges Abstandsmaß verwendet und die Ähnlichkeit über den Abstand der Kurven definiert werden (je geringer der Abstand, desto größer die Ähnlichkeit). Die Grundlagen zur Berechnung des Abstands sind zu jedem Zeitpunkt jeweils die entsprechenden Messwerte bzw. die Differenzen zwischen ihnen (in Abbildung 4.4 sind die Differenzen als Fehler 1 bzw. Fehler 2 eingezeichnet). Gute Ergebnisse liefert der euklidische Abstand, der sich für eine Stichprobe der Größe n als zeitlich geordnete Liste $\mathcal{S}$ und einem Muster als zeitlich geordnete Liste $\mathcal{M}$ nach der Berechnungsvorschrift 4.1 ermitteln lässt. In dem in der Abbildung dargestellten Beispiel ist der euklidische Abstand der Stichprobe zu Muster 1 deutlich geringer als zu Muster 2, weshalb der Stichprobe Muster 1 zugeordnet werden würde.

$$abstand_{euklid}(\mathcal{S},\mathcal{M}) := \sqrt{\sum_{i\in\{1,\ldots,n\}} (\mathcal{S}[i]-\mathcal{M}[i])^2} \tag{4.1}$$

In manchen Anwendungen ist es nicht möglich, (periodische) Muster und Stichproben exakt aneinander auszurichten, um den Abstand berechnen zu können. Häufig fehlt der Anfangspunkt von der Stichprobe, von den Mustern oder von beiden. In diesen Fällen kann dennoch die Ähnlichkeit auf der Basis des Abstands bestimmt werden. Anstatt den Abstand auf der Grundlage der Differenzen zwischen Messwerten zu berechnen, wird in [TÖ09] vorgeschlagen, jeweils die Häufigkeitsverteilung der Werte von Stichprobe und Muster zu erstellen und den Abstand zwischen den Verteilungen zu berechnen. Im Allgemeinen wird allerdings davon auszugehen sein, dass für gleich gute Ergebnisse bei einem Vergleich der Häufigkeitsverteilungen im Gegensatz zu einem direkten Vergleich der Werte mehr Eingabedaten erforderlich sind (in Abhängigkeit von der Größe der Domäne, aus der die Werte stammen können).

4.2.2.3 Algorithmen im Action Framework

Nach der Besprechung der grundlegenden Schritte einer Musterklassifizierung können diese in der in Algorithmus 4.4 angegebenen Funktion umgesetzt werden, um über einen einfachen Funktionsaufruf zu einem beliebigen Thread als Eingabe einfach und schnell das zugehörige Muster ermitteln zu können.

Eingabe: Zu klassifizierender Thread t.
Daten: Alle Muster als zeitlich geordnete Listen in Menge $\mathcal{M}$.

```
1  besterAbstand ← null; muster ← null;
2  Zeitlich geordnete Liste stichprobe ← HOLESTICHPROBE(t);
3  für jedes m ↩ M tue
4      abstand ← 0;
5      für i ← 1 bis |stichprobe| tue
6          abstand ← abstand + (stichprobe[i] − m[i])²;
7      abstand ← √abstand;
8      wenn abstand < besterAbstand dann
9          muster ← m;
10         besterAbstand ← abstand;
```

Ausgabe: Das ähnlichste Muster *muster*.

Algorithmus 4.4: Muster von einem Thread bestimmen

Der Algorithmus benötigt alle bekannten Muster aus der Datenbank in Form von zeitlich geordneten Listen. Zuerst wird die zu dem übergebenen Thread passende Stichprobe, die durch Aufzeichnen der ersten gemessenen CPU-Verbräuche von ihm entstanden ist, aus der Datenbank geladen. Danach wird sukzessive der Abstand zwischen jedem Muster und der Stichprobe mit Hilfe des euklidischen Abstands (Berechnungsvorschrift 4.1) berechnet und mit dem Abstand des bis zu diesem Zeitpunkt ähnlichsten Musters verglichen. Sofern das aktuelle Muster einen kleineren Abstand hat, müssen das Muster sowie der berechnete Abstand zwischengespeichert werden. Nachdem alle Muster verarbeitet wurden, steht das ähnlichste Muster zu dem Thread in der Eingabe fest und kann als Ergebnis ausgegeben werden.

Mit Hilfe der Strategien und Verfahren zur Auswahl geeigneter Maschinen und Threads sowie einer zuverlässigen Musterklassifizierung können schließlich die für eine dynamische und proaktive Lastbalancierung notwendigen Algorithmen entworfen und als Aktionen im Action Framework implementiert werden.

Daten: Menge aller aktiven Threads $\mathcal{T}_{aktiv}$,
Menge aller klassifizierten Threads $\mathcal{T}_{klassifiziert}$.

```
1  T_klassifiziert ← T_klassifiziert ∩ T_aktiv;
2  Menge T_unklassifiziert ← T_aktiv \ T_klassifiziert;
3  für jedes t ↩ T_unklassifiziert tue
4      varianz ← HoleVarianz(t);
5      wenn varianz ≠ null dann
6          wenn varianz < schwellwert dann
7              durchschnitt ← HoleDurchschnitt(t);
8              KlassifiziereKonstant(t, durchschnitt);
9          sonst
10             muster ← HoleMuster(t);
11             KlassifiziereVariabel(t, muster);
12         t ↪ T_klassifiziert
```

Ergebnis: Aktualisierte Menge $\mathcal{T}_{klassifiziert}$.

Algorithmus 4.5: Threads klassifizieren

Als erstes müssen kontinuierlich neue Threads klassifiziert werden, um aus einer stets aktuellen Menge von klassifizierten Threads geeignete Kandidaten für eine Verschiebung auswählen zu können. Algorithmus 4.5 stellt den Ablauf eines Durchlaufes der Klassifizierung dar. Als Datenbasis benötigt der Algorithmus die Menge aller zurzeit aktiven Threads und alle von ihm bereits klassifizierten Threads. Zu Beginn wird durch Bildung des Schnitts zwischen den Mengen der aktiven und der klassifizierten Threads letztere um alle zwischenzeitlich abgeschlossenen Threads bereinigt. Danach kann versucht werden, jeden aktiven und unklassifizierten Thread zu klassifizieren. Die erste zu fällende Entscheidung ist, ob sich ein Thread konstant oder variabel verhält. Dazu wird die von Anfrage 4.3 berechnete Varianz abgefragt und mit dem festgelegten Schwellwert verglichen. Liegt für einen Thread die Varianz noch nicht vor, dann bricht die Klassifizierung für ihn ab. Wenn die Varianz geringer als der Schwellwert ist, dann handelt es sich um einen konstanten Thread und es kann der von Anfrage 4.3 ebenfalls berechnete Durchschnittswert diesem Thread als konstanter CPU-Verbrauch zugeordnet werden. In allen anderen Fällen liegt ein variabler Thread vor und es muss noch durch Algorithmus 4.4 das zu ihm gehörende Muster ermittelt werden, um ihn erfolgreich klassifizieren zu können. Für ein kontinuierliches Verhalten muss der Algorithmus mit Pausen zwischen je zwei Durchläufen permanent ausgeführt werden. Die Länge der Pausen richtet sich nach der Frequenz, mit der neue Threads entstehen. Wird der Algorithmus zu selten ausgeführt, dann stehen nur wenige bis keine Threads zum Verschieben bereit. Bei einer zu häufigen Ausführung werden aufgrund von Durchläufen, die nichts an dem Ergebnis verändern, Ressourcen verschwendet.

Auf der Basis der von Algorithmus 4.5 bereitgestellten Menge von aktiven und klassifizierten Threads sowie der von Anfrage 4.2 ermittelten Maschinen mit der größten bzw. kleinsten Belastung kann nun die eigentliche Lastbalancierung realisiert werden.

Daten: Menge aller klassifizierten Threads $\mathcal{T}_{klassifiziert}$,
Maschine $maschine_{max}$ mit maximaler Last cpu_{max},
Maschine $maschine_{min}$ mit minimaler Last cpu_{min}.

```
1  solange true tue
2      wenn maschine_max ≠ maschine_min dann
3          optimum ← (cpu_max − cpu_min)/2; auswahl ← null;
4          besterAbstand ← null; veränderung ← 0;
5          für alle t ↩ T_klassifiziert tue
6              maschine ← HoleMaschineZuThread(t);
7              wenn maschine = maschine_max dann
8                  cpu ← null;
9                  wenn IstKonstant(t) dann
10                     cpu ← HoleKonstantenVerbrauch(t);
11                 sonst
12                     cpu ← HoleCpuZukunft(t, wartezeit/2);
13                 wenn |cpu − optimum| < besterAbstand dann
14                     besterAbstand ← |cpu − optimum|;
15                     veränderung ← cpu; auswahl ← t;
16         wenn |max − min − 2·veränderung| < 0.9 · (max − min) dann
17             VerschiebeThread(t, maschine_max, maschine_min);
18     Warte(wartezeit)
```

Ergebnis: Lastenausgleich durch Verschieben eines Threads.

Algorithmus 4.6: Dynamische und proaktive Balancierung der Last

Die in Algorithmus 4.6 dargestellte Lastbalancierung prüft zuerst, ob die gemeldeten Maschinen verschieden sind, weil sich sonst ein vollständiger Durchlauf nicht lohnt. Danach wird der Mittelwert der beiden CPU-Lasten als optimaler Wert für einen zu verschiebenden Thread zwischengespeichert, da ein Thread mit genau diesem Wert als CPU-Verbrauch zu einem perfekten Ausgleich der Lasten führen würde. Für jeden klassifizierten Thread, der zu der maximal belasteten Maschine gehört, wird anschließend sein zukünftiger CPU-Verbrauch abgefragt. Bei einem konstanten Thread ist der in Anfrage 4.3 berechnete Mittelwert der Verbrauch für alle zukünftigen Zeitpunkte. Im Fall eines variablen Threads hingegen muss ein exakter zukünftiger Zeitpunkt angegeben werden, für den sein CPU-Verbrauch vorhergesagt werden soll. In Experimenten hat sich herausgestellt, dass die zeitliche Mitte zwischen zwei

Durchläufen der Lastbalancierung zu den besten Resultaten führt. Daher wird die Wartezeit bis zum nächsten Durchlauf (*wartezeit*) halbiert und als zukünftiger Zeitpunkt gewählt. Abschließend kann für den erwarteten CPU-Verbrauch des aktuell betrachteten Threads festgestellt werden, ob er näher an dem optimalen Wert liegt als der bisher beste gefundene CPU-Verbrauch (bei heterogenen Maschinen muss über eine Übersetzungstabelle der Wert unter Umständen an die abweichende Leistungsfähigkeit der potentiellen Ziel-Maschine angepasst werden). In allen positiven Fällen kann der für bisher am geeignetsten gehaltene Thread durch den aktuell betrachteten ersetzt werden. Nachdem alle klassifizierten Threads auf der maximal belasteten Maschine verarbeitet wurden, steht derjenige Thread fest, der bei einem Verschieben für den besten Ausgleich sorgt. Dennoch muss überprüft werden, ob ein Verschieben dieses Threads tatsächlich zu einer Verbesserung führt, denn falls auf der maximal belasteten Maschine zum Beispiel nur ein Thread läuft und auf der minimal belasteten beliebig viele, dann würde durch ein Verschieben das Ungleichgewicht noch vergrößert werden. Dieser Fall wird in Zeile 16 ausgeschlossen. Zusätzlich wird gefordert, dass das bestehende Ungleichgewicht um mindestens 10 % reduziert wird, damit es sich um eine echte Verbesserung handelt. Ein Grund für das Verlangen einer bestimmten Mindestverbesserung kann sein, dass das Verschieben von Objekten immer mit Kosten verbunden ist, die mit eingeplant werden müssen. Die Migration einer virtuellen Maschine ist beispielsweise mit dem Transfer ihres Haupt- und Externspeichers verbunden, wodurch zum Teil erhebliche Datenmengen über das Netzwerk bewegt werden müssen. Hat der ausgewählte Thread auch die letzte Überprüfung passiert, dann kann er verschoben werden. Danach wird bis zum nächsten Durchlauf der Lastbalancierung eine Pause eingelegt. Die Dauer der Pause wird nach jedem Durchlauf in Abhängigkeit von bestimmten und sich möglicherweise verändernden Parametern neu berechnet . Dadurch wird die gesamte Lastbalancierung dynamisch skalierbar. Die in den Experimenten verwendete Berechnungsvorschrift ist in der Formel 4.2 dargestellt.

$$wartezeit := \max\left(\frac{dauerFürZweiMaschinen}{anzahlAktiverMaschinen - 1}, 1\right) \tag{4.2}$$

In allen Experimenten wurde angenommen, dass sich die Anzahl der Verbraucher in etwa proportional zu der Anzahl der Ressourcen verhält. Daher ist die Anzahl der aktiven Maschinen der wichtigste zu berücksichtigende Parameter. Aufgrund dieser Proportionalitätsannahme ist es ausreichend, eine Dauer (*dauerFürZweiMaschinen*) für die Pausen zu ermitteln, bei der die Lastbalancierung bei nur zwei vorhandenen Maschinen optimal (das heißt weder zu selten noch zu häufig) ausgeführt wird. Diese Zeitdauer kann mit der Anzahl der aktiven Maschinen (*anzahlAktiverMaschinen*) linear zur Laufzeit skaliert werden. Für alle Fälle, in denen weitere Gesetzmäßigkeiten auf

der Seite der Verbraucher oder Ressourcen ausfindig gemacht werden können, müssen diese durch entsprechende Parameter mit berücksichtigt werden. Im Allgemeinen sollte eine dynamische Skalierung durch Anpassen der Pausen jedoch in den meisten Situationen gut funktionieren.

Zur Bestimmung des zukünftigen CPU-Verbrauchs eines variablen Threads wurde die Funktion HOLECPUZUKUNFT aufgerufen, die detaillierter beschrieben werden muss. Da der hinter dieser Funktion stehende Algorithmus jedoch leicht aus Algorithmus 4.4 abgeleitet werden kann, wird nur das prinzipielle Vorgehen, das in Abbildung 4.5 dargestellt ist, skizziert.

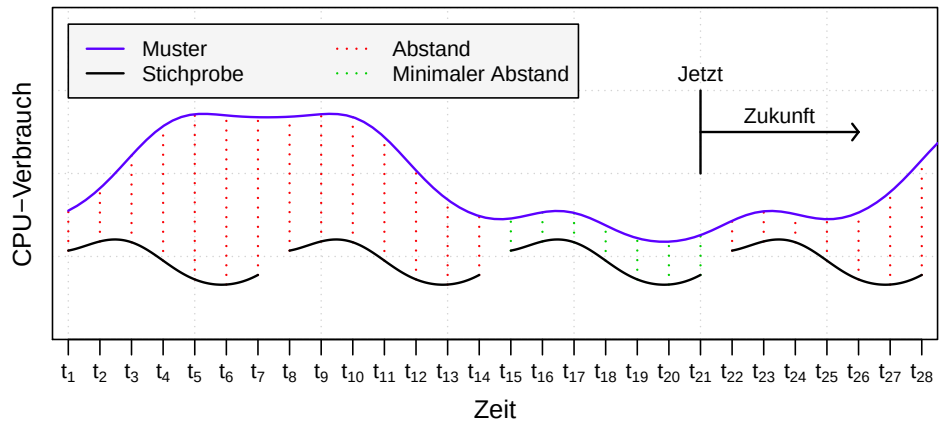

Abbildung 4.5: Zukünftigen CPU-Verbrauch von variablen Threads bestimmen

Zu jedem variablen und klassifizierten Thread ist das zugehörige Muster bekannt. Zusätzlich zu seinem Muster werden von einem übergebenen variablen Thread noch die jüngsten Messwerte seines CPU-Verbrauchs als Stichprobe benötigt. Die Stichprobe wird dann an den Anfang seines Musters (t_1) ausgerichtet und es wird der Abstand berechnet. Danach wird der Anfang der Stichprobe an den nächsten Zeitpunkt des Musters (t_2) ausgerichtet und der Abstand wird erneut berechnet. Die Stichprobe wird so lange an dem Muster entlang bewegt, bis das Ende erreicht ist (aus Gründen der Übersichtlichkeit sind nur die Zwischenschritte für die Zeitpunkte t_1, t_8, t_{15} und t_{22} eingezeichnet). Die Position, an der der Abstand am geringsten war (in der Abbildung t_{15}), zeigt auf den Beginn der zuletzt gesehene Sequenz des Musters in dem aktuellen Sensorstrom. Das Ende der Stichprobe (t_{21}) zeigt folglich den aktuellen Zeitpunkt an. Von dieser Position aus kann das Muster soweit weiterverfolgt werden, bis der gesuchte zukünftige Zeitpunkt erreicht ist (bei Erreichen des Endes kann aufgrund der Periode das Muster wieder von vorne abgelaufen werden). Der Wert des Musters an dieser Position gibt den zu erwartenden CPU-Verbrauch an.

4.3 Evaluation

Um die vorgestellten Anfragen und Algorithmen auf ihre Effektivität zu überprüfen, wurde die Thread-erzeugende Anwendung erneut auf vier Maschinen ausgeführt und dieses mal die Lastbalancierung eingeschaltet. Die dabei verwendeten Parameter sind in Tabelle 4.1 aufgelistet.

Parameter	Wert
stichprobenumfang	30
schwellwert	10
klassifizierung	kontinuierlich
dauerFürZweiMaschinen	5 s

Tabelle 4.1: Parameter der Evaluation

Bei einem Schwellwert von exakt 10 und einem Stichprobenumfang von 30 Messwerten werden weniger als 1 % aller Threads falsch klassifiziert. Die Klassifizierung (Algorithmus 4.5) wird kontinuierlich ausgeführt. Das bedeutet, dass nach den ersten 30 erforderlichen Datensätzen in dem Sensorstrom eines neuen Threads die Klassifizierung direkt angestoßen wird, denn es hat sich gezeigt, dass in nur sehr wenigen Fällen die ersten 30 Werte kein verwertbares Ergebnis liefern. Die Lastbalancierung arbeitet auf zwei Maschinen mit einer Pause von fünf Sekunden noch zuverlässig. Für vier Maschinen verkürzt sich die Wartezeit entsprechend der Berechnungsvorschrift 4.2 auf 1,67 Sekunden. In Abbildung 4.6 ist die Lastenverteilung der Anwendung erneut über einen Zeitraum von 10 Minuten dargestellt.

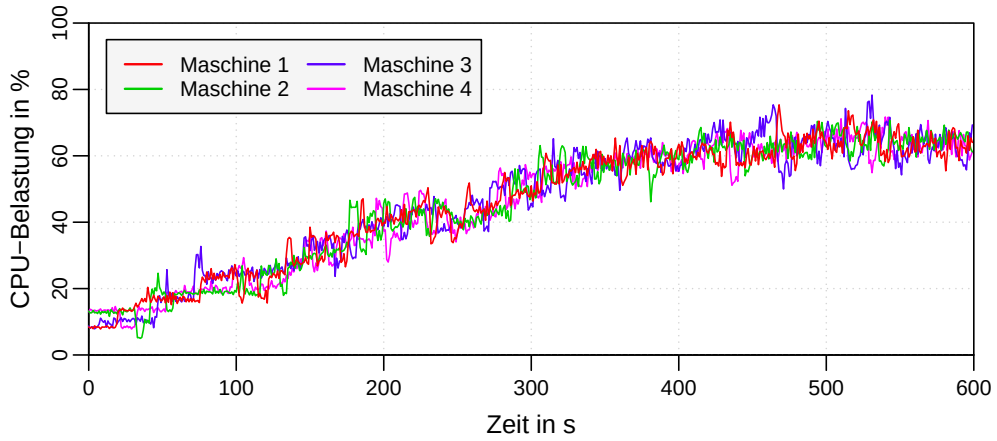

Abbildung 4.6: Lastenverteilung mit Balancierung

Es wurde ein Ausschnitt aus dem Experiment gewählt, in dem mehr neue Threads entstehen als vorhandene terminieren. Dadurch steigt der Gesamtbedarf allmählich. Die Lastbalancierung sorgt zu jedem Zeitpunkt für eine gleichmäßige Auslastung der Maschinen und Leistungsspitzen auf einer der Maschinen werden vollständig vermieden. Es sind deutlich Paarungen von Flanken zu erkennen, bei denen eine Flanke kurze Zeit später von einer entsprechend entgegengerichteten Flanke gefolgt wird. An diesen Stellen wird das Eingreifen der Lastbalancierung sichtbar.

Eine abschließende Feststellung ist, dass sich die Architektur auf der Basis von Datenströmen und CEP sowie der Einsatz von *CEP4Cloud* als Grundlage für dringend benötigte Anwendungen, unter anderem zu den wichtigen Themen Leistungsfähigkeit, Sicherheit und Verfügbarkeit, als nützlich erwiesen haben. Alle Komponenten kamen zum Einsatz und die kontinuierlich zu erledigenden Aufgaben konnten auf das CEP-System, die Datenbank und das Action Framework verteilt werden. Durch die Auslagerung einiger Berechnungen in das CEP-System als kontinuierliche Anfragen wurde mühsam zu schreibender und ineffizienter Programmcode vermieden.

4.4 Optimierungen

In diesem Abschnitt sollen zwei mögliche Optimierung der vorgestellten Lastbalancierung diskutiert werden. Die erste Optimierung betrifft einen effizienteren Umgang mit den zur Verfügung stehenden Ressourcen während die zweite Optimierung zu genaueren Ergebnissen führt.

Die Anfrage 4.3 zur Berechnung des Durchschnitts und der Varianz von Threads als Entscheidungsgrundlage zur Klassifizierung hat einen großen Nachteil. Für jeden Thread werden diese Werte nur einmal zur Klassifizierung benötigt. Dennoch berechnet die Anfrage die Aggregate bei jedem Eintreffen eines neuen Datensatzes zu einem Thread neu und verschwendet dadurch unnötigerweise Rechenkapazitäten in dem CEP-System. Viel gravierender jedoch ist das zusätzliche Problem, dass nicht benötigte Ergebnisse produziert werden und das Netzwerk belasten. Die Probleme werden durch eine Modifikation, die in Anfrage 4.7 dargestellt ist, vermieden.

```
SELECT   thread_key, variance(cpu), avg(cpu)
FROM     (SELECT *
          FROM   JVMThreadSensor
          EXCEPT
          SELECT s.*
          FROM   JVMThreadSensor s, ClassifiedTable t
          WHERE  s.thread_key = t.thread_key)
          WINDOW(PARTITION BY thread_key
                          ROWS stichprobenumfang) u
GROUP BY thread_key
HAVING   variance(cpu) > 1.1 * schwellwert
OR       variance(cpu) < 0.9 * schwellwert;
```

Anfrage 4.7: Thread-Typen effizienter bestimmen

Aufgrund der Schnappschuss-Reduzierbarkeit aller Operatoren von *PIPES* haben diese bei Ausblendung der Zeitdimension die identische Semantik wie die für statische Tabellen bestimmte erweiterte relationale Algebra. Aus diesem Grund können in dem SQL-Dialekt von *PIPES* Datenströme und Datenbanktabellen zusammen angefragt werden. Die einzelnen Zeilen einer Tabelle können direkt als Payloads verwendet werden. Da auf diese Weise allerdings die zeitliche Komponente in den Tupeln, die von den Datenstromoperatoren verarbeitet werden, unbestimmt bleibt, ist ein Verbund mit einem Datenstrom zwingend erforderlich, um die fehlenden Zeitstempel bei der Bildung des Kartesischen Produkts aus dem Datenstrom zu übernehmen.[2] Damit die Anfrage korrekt ausgeführt werden kann, muss die Tabelle, in der zu jedem Thread sein durchschnittlicher CPU-Verbrauch sowie die zugehörige Varianz abgelegt werden, näher definiert werden. Für alle nachfolgenden Überlegungen sei angenommen, dass diese Tabelle unter dem Namen `ClassifiedTable` angelegt wurde und die Attribute des Schemas mit `thread_key`, `durchschnitt` und `varianz` benannt sind. Damit wird der Verbund zwischen der Tabelle und dem Datenstrom `JVMThreadSensor` über das gemeinsame Attribut `thread_key` gebildet. Jedes Tupel in dem Datenstrom zu einem Thread, der bereits einen Eintrag in der Tabelle hat, erzeugt ein positives Ergebnis nach der Bildung des Verbunds. Genau diese Tupel werden mit Hilfe der Differenz aus dem Datenstrom entfernt, bevor die Berechnungen von Durchschnitt und Varianz ausgeführt werden. Auf den ersten Blick mag die Einsparung an Berechnungen durch Zugriffe auf die Datenbank teuer erkauft zu sein. Es sei daher darauf hingewiesen, dass diese Datenbankzugriffe auch bei der Anfrage 4.3 anfallen, weil für jedes produzierte Ergebnis ein Datenbankzugriff

[2] Aufgrund der fehlenden temporalen Information von Datenbanktabellen ist die Semantik nicht mehr eindeutig definiert. Es hängt von den Zeitpunkten der Aktualisierung einer in dem CEP-System zwischengespeicherten Tabelle ab, ob tatsächlich nach dem ersten Ergebnis keine weiteren mehr folgen. Weil es in der Optimierung allerdings nur um eine Reduzierung der Ergebnismenge geht, ist die unklare Semantik kein Problem.

notwendig ist. Die Datenbankzugriffe in Anfrage 4.7 hingegen lassen sich deutlich reduzieren. Das CEP-System hält eine vollständige Kopie der Tabelle vor, die über einen einstellbaren Wert regelmäßig aktualisiert wird. Weil die Tabelle als Filter dient, muss nur höchstens mit der Frequenz aktualisiert werden, mit der neue Threads entstehen. Zusätzlich zu weniger Datenbankzugriffen kommen mit weniger Berechnungen im CEP-System und weniger Datenverkehr im Netzwerk noch die eigentlich beabsichtigten Verbesserungen hinzu.

Ein anderes Problem, das sich direkt auf die Qualität der Lastbalancierung auswirkt, befindet sich in Algorithmus 4.6. Für ein proaktives Verhalten werden zwar die zukünftigen CPU-Verbräuche der variablen Threads berücksichtigt, aber die von Anfrage 4.2 gelieferten CPU-Lasten der beiden extremen Maschinen werden als konstant angenommen. Diese Annahme ist falsch, da sich auch die CPU-Last einer Maschine durch variable Threads zukünftig verändern kann. Unter Beachtung von zukünftigen Werten kann es sogar passieren, dass zwei völlig andere Maschinen extrem belastet sein werden. Entsprechend müssten die CPU-Lasten der Maschinen durch Algorithmus 4.8, der als Ersatz für Anfrage 4.2 gedacht ist, angepasst werden.

Daten: Menge aller aktiven Maschinen $\mathcal{M}$,
Menge aller als variabel klassifizierten Threads $\mathcal{T}$.

```
1  maschine_max ← null; cpu_max ← null;
2  maschine_min ← null; cpu_min ← null;
3  für alle m ↩ M tue
4      cpu ← HoleCpu(m);
5      Menge T_m ← HoleThreads(m);
6      für alle t ↩ (T ∩ T_m) tue
7          cpuAktuell ← HoleCpu(t);
8          cpuZukunft ← HoleCpuZukunft(t, wartezeit/2);
9          cpu ← cpu + (cpuZukunft − cpuAktuell);
10     wenn cpu > cpu_max dann
11         maschine_max ← m; cpu_max ← cpu;
12     wenn cpu < cpu_min dann
13         maschine_min ← m; cpu_min ← cpu;
```

Ausgabe: Maschine $maschine_{max}$ mit maximaler Last cpu_{max},
Maschine $maschine_{min}$ mit minimaler Last cpu_{min}.

Algorithmus 4.8: Extreme Maschinen prediktiv bestimmen

Der dargestellte Algorithmus iteriert über alle aktiven Maschinen. Für jede Maschine werden die auf ihr laufenden variablen Threads bestimmt und für jeden dieser

Threads wird die Veränderung seines CPU-Verbrauchs zu dem Zeitpunkt, für den optimiert werden soll, berechnet. Die CPU-Last der Maschine wird durch sukzessives Addieren der Differenzen angepasst. Aus den neuen CPU-Lasten können anschließend die zukünftig extremen Maschinen mit ihren vorhergesagten Lasten ausgewählt werden. Im Gegensatz zu der früheren Anfrage im CEP-System ist der Algorithmus mit hohen Kosten verbunden, weil für jeden Durchlauf der Lastbalancierung zuvor über alle Maschinen und Threads verschachtelt iteriert werden muss. Von daher sollte die Anfrage nur durch den Algorithmus ersetzt werden, wenn die von ihr produzierten Ergebnisse nicht zufriedenstellend sind.

Zusammenfassung

In diesem Kapitel wurden die Architektur aus Abschnitt 3.2 und das in Abschnitt 3.3 implementierte Überwachungssystem im praktischen Einsatz auf ihre Tauglichkeit für Cloud Computing überprüft. Durch die Modellierung von Abhängigkeiten und Beziehungen war es möglich, Probleme in einer Schicht der Cloud (Infrastrukturebene) auf Ursachen in einer anderen Schicht (Anwendungsebene) zurückzuführen und dort zu lösen. Aufgrund der Architektur konnten einzelne Verfahrensschritte auf verteilte Komponenten, die jeweils individuelle Stärken besitzen, übertragen werden. Zusätzlich zu den vorhandenen Komponenten wurde durch die zur Verfügung gestellten Datenströme und das CEP-System die Anwendung von effizienten und effektiven Algorithmen, die verstecktes Wissen aus Datenströmen extrahieren können, ermöglicht. Das Überwachungssystem erwies sich insgesamt als eine solide Grundlage, um Lösungen zur Beseitigung von Risiken in der Leistungsfähigkeit, Sicherheit und Verfügbarkeit einer Cloud und ihrer Dienste zu entwickeln und zu implementieren.

5 Diskussion und Ausblick

Zum Abschluss der Arbeit werden in diesem Kapitel die wichtigsten Ergebnisse in Abschnitt 5.1 zusammengefasst und interessante Fortsetzungen sowie offen gebliebene Fragen, die jeweils noch ausführlich zu erforschen sind, in Abschnitt 5.2 vorgestellt.

5.1 Zusammenfassung

In Cloud Computing steckt ein enormes Potential, das die Aufmerksamkeit von Industrie und Wissenschaft intensiv auf sich gelenkt hat. Teile von IKT und Gesellschaft wurden bereits dadurch verändert und die Bedeutung von Cloud Computing wird schon heute mit den Einführungen des PCs und des Internets gleichgesetzt [Bun10]. Auf der anderen Seite ist die neue Technologie noch mit vielen und hohen Risiken verbunden, die für die Nutzer eine ständige Gefahr darstellen. Zusätzlich können wegen der Risiken nur unkritische Anwendungen und Daten in eine Cloud gebracht werden, wodurch ein großer Nutzerkreis von Cloud Computing ausgeschlossen wird. Damit die positive Entwicklung auch zukünftig fortgesetzt werden kann, müssen die Risiken beseitigt werden. Die dazu benötigten Konzepte, Methoden und Technologien können allerdings erst dann entstehen, wenn Clouds zuverlässig überwacht werden können. Aufgrund der zentralen Rolle von Cloud Monitoring und der Untauglichkeit für Cloud Computing von existierenden Monitoring-Produkten, wurden in dieser Arbeit zuerst aus den Eigenschaften und Risiken von Cloud Computing die Anforderungen an ein Cloud Monitoring herausgearbeitet und anschließend darauf aufbauend eine Lösung entwickelt und implementiert. Bei der Auswahl der zu integrierenden Technologien und in anschließenden Experimenten haben sich Datenstrommanagementsysteme und CEP als ideale Kandidaten erwiesen. Die in dem Prototyp *CEP4Cloud* gemündete Implementierung wurde zur Evaluation als Basis für die Entwicklung von Lösungen einiger typischer Probleme und Risiken eingesetzt und stellte sich dabei als sehr geeignet heraus. Von *CEP4Cloud* bis zu einem produktiven System sind noch eine Reihe von Fragen und Problemen (siehe dazu den nächsten Abschnitt) zu klären. Darüber hinaus muss die prototypische Implementierung stark ausgebaut werden (z. B. deutlich mehr Metriken, Anfragen und Aktionen, vollständiges Datenmodell, Benutzerverwaltung, Mustermanagement, etc.).

5.2 Offene Forschungsfragen

Dieser Abschnitt stellt die interessantesten und noch unbeantworteten Fragen vor, die während des Einsatzes von *CEP4Cloud* entstanden sind.

5.2.1 Anfrageassistent

In Experimenten mit *CEP4Cloud* kam es bei jeder neuen Anfrage und Aktion immer wieder zu der Frage, wie Parameter (z. B die Größe von Fenstern in Anfragen) und Schwellwerte (ein Über-/Unterschreiten dieser Grenzen führt zur Ausführung von Aktionen) korrekt zu setzen sind. Hierbei wäre eine Unterstützung durch das jeweilige System wünschenswert. Beispielsweise könnte ein Assistent auf einer Stichprobe der Datenströme Anfragen und Aktionen mit unterschiedlichen Parameter-Werten wiederholt ausführen, um die zu vorgegebenen Qualitätskriterien (z. B. Anzahl der Anfrageergebnisse oder ausgeführte Aktionen pro Zeiteinheit) besten Werte zu bestimmen und dem Nutzer vorzuschlagen. In einer verbesserten Variante könnte ein Assistent auch vollkommen neue Anfragen und Aktionen vorschlagen sowie existierende Anfragen und Aktionen als unbrauchbar einstufen. Ein einfacher Anwendungsfall soll die grundlegende Idee kurz skizzieren. In einem Überwachungssystem sei eine Vielzahl von Datenströmen verfügbar, die jeweils Informationen zu dem CPU-Verbrauch von irgendwelchen Objekten beinhalten. Ein Nutzer habe eine Menge von Anfragen formuliert, die jeden dieser Datenströme bis auf einen verarbeiten. Weil der Nutzer offensichtlich an Informationen zu dem CPU-Verbrauch von Objekten interessiert ist, kann ein Assistent eine weitere Anfrage vorschlagen, die den (wahrscheinlich übersehenen) fehlenden Datenstrom anfragt und mit allen passenden Aktionen verknüpft.

Ein idealer Anfrageassistent könnte nicht nur Vorschläge machen, sondern automatisch Anfragen, Aktionen, Parameter und Schwellwerte zur Laufzeit verändern. Unter der Voraussetzung eines perfekt arbeitenden Assistenten würde jede von einem Nutzer vorgegebene Menge von Anfragen und Aktionen mit der Zeit immer gegen eine optimale Menge konvergieren. Die Ausarbeitung eines Assistenten wäre für die Datenstromverarbeitung insgesamt ein Gewinn und ist ein interessantes Gebiet, in dem weitere Forschungen notwendig sind. Ein erster Ansatz könnte darin bestehen, SLAs so zu formulieren, dass sie maschinell verarbeitet werden können. Ein Anfrageassistent kann dann als Eingabe einen SLA entgegennehmen und als Ausgabe alle Anfragen liefern, die zur Überwachung dieses SLAs notwendig sind.

5.2.2 Dynamische Skalierbarkeit

Alle Überwachungssysteme, die die in Abschnitt 3.2 vorgeschlagene Architektur umsetzen, sind beliebig skalierbar. Die Architektur besteht aus verteilten Komponenten, von denen jede auf einer eigener Maschine betrieben werden kann. Damit können bereits größere Clouds überwacht werden. Falls es dennoch zu Engpässen kommen sollte, dann ist jede einzelne Komponente in der Lage, ebenfalls verteilt auf einem Cluster von Maschinen zu arbeiten. Die Sensoren bzw. Agenten sind auf natürliche Weise innerhalb der Cloud verteilt, verteilte Datenbanksysteme sind schon sehr lange bekannt und verfügbar, viele für den Broker in Frage kommende Produkte, wie das in *CEP4Cloud* eingesetzte *HornetQ*, können auf einem beliebig großen Cluster verteilt werden und mehrere CEP-Systeme können hierarchisch angeordnet und jeweils auf einer eigenen Maschine betrieben werden. Unterschiedliche Visualisierungen können auf mehrere Web-Server verteilt werden und im Bedarfsfall können mehrere identische Web-Server eingesetzt werden, auf die die Nachfragen gleichmäßig aufgeteilt werden. Weil einzelne Aktionen im Action Framework vollkommen unabhängig voneinander sind, können Aktionen partitioniert und jede Gruppe auf einer eigenen Maschine abgelegt werden.

Während mit der beliebigen Skalierbarkeit eine wichtige Anforderung an ein Cloud-Monitoring erfüllt ist, gibt es noch einen entscheidenden Nachteil. Die Skalierung muss manuell vorgenommen werden und kann nicht automatisch sowie zur Laufzeit geschehen. Ein wesentliches Charakteristikum von Clouds und ihren Diensten ist aber, dass sie sehr schnell und dynamisch skalieren. Ein Überwachungssystem sollte bei Anwachsen der Anzahl von überwachten Objekten ebenfalls automatisch skalieren, um weiterhin zuverlässig arbeiten zu können. Auch bei einer Verkleinerung muss sich das Überwachungssystem anpassen, damit sich die Kosten für die Überwachung relativ zu der jeweils aktuellen Größe der Cloud verhalten. Weil das CEP-System diejenige Komponente ist, die mit Abstand am häufigsten in einem produktiven Umfeld skaliert werden muss, werden für CEP-Systeme entsprechende Automatismen dringend benötigt.

Die Skalierbarkeit von CEP-Systemen wird allerdings durch das Netzwerk deutlich eingeschränkt. Da zum Erkennen von gesuchten komplexen Ereignissen immer zuerst alle Kombinationen von einfachen Ereignissen erstellt werden müssen, müssen für viele CEP-Anfragen die Sensordaten zu jeder Maschine aus einem Verbund von CEP-Systemen gebracht werden. Um die Anzahl der benötigten CEP-Systeme in Grenzen zu halten, müssen auf einer einzelnen Maschine so viele Berechnungen wie möglich ausgeführt werden können. Deshalb ist zu erforschen, ob und wie die Algorithmen von CEP-Systemen auf unterschiedliche Arten von Co-Prozessoren portiert werden können (z. B. auf GPUs), um dort parallel zur CPU Berechnungen ausführen zu können.

5.2.3 Installation, Konfiguration und Aktualisierungen

In der prototypischen Implementierung *CEPCloud* waren die Sensor-Agenten unabhängige Anwendungen, die auf jeder Instanz in der Cloud manuell installiert und eingerichtet werden mussten. Im Gegensatz zu einem Prototypen müssen in einem produktiven Umfeld regelmäßig Änderungen und Aktualisierungen der Agenten vorgenommen werden, wie z. B. neue Sensor-Typen in die Agenten einfügen oder die Agenten an Aktualisierungen der Hypervisor, Betriebssysteme und überwachten Objekt-Typen anpassen. Bei großen Infrastrukturen mit mehreren tausend physischen Maschinen ist ein solches Vorgehen nicht mehr realisierbar und es wird ein automatisches und selbstständiges Verteilen und Aktualisieren der Agenten benötigt. Eine Möglichkeit wäre die Integration der Agenten in die eingesetzten Hypervisor und Betriebssysteme. Für Aktualisierungen und Änderungen zur Laufzeit könnten die Agenten mit der Fähigkeit ausgestattet werden, dynamisch beliebigen Programmcode nachzuladen. Hier ist noch genauer zu klären, ob und wie diese Ideen umgesetzt werden können. Insbesondere die Sicherheit der überwachten Cloud darf dabei nicht gefährdet werden.

Literaturverzeichnis

[Agr+08] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom und Neil Immerman: „Efficient Pattern Matching over Event Streams“. In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (2008), S. 147–160.

[Arm+09] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica und Matei Zaharia: „Above the Clouds: A Berkeley View of Cloud Computing“. Techn. Ber. UCB/EECS-2009-28. UC Berkeley Reliable Adaptive Distributed Systems Laboratory, Feb. 2009.

[Bai+06] Yijian Bai, Hetal Thakkar, Haixun Wang, Chang Luo und Carlo Zaniolo: „A Data Stream Language and System Designed for Power and Extensibility“. In: *Proceedings of the 15th ACM international conference on Information and knowledge management (CIKM)* (2006), S. 337–346.

[Bau+11a] Christian Baun, Marcel Kunze, Tobias Kurze und Viktor Mauch: „Private Cloud-Infrastrukturen und Cloud-Plattformen“. In: *Informatik Spektrum* 34.3 (Juni 2011), S. 242–254.

[Bau+11b] Christian Baun, Marcel Kunze, Jens Nimis und Stefan Tai: „Cloud Computing – Web-basierte dynamische IT-Services“. 2. Aufl. Informatik im Fokus. Springer-Verlag, Feb. 2011.

[Bay02] Thomas Bayer: „REST Web Services – Eine Einführung“. Übersichtsartikel. Orientation in Objects GmbH. Nov. 2002.

[BDS00] Jochen van den Bercken, Jens-Peter Dittrich und Bernhard Seeger: „javax.XXL: A prototype for a Library of Query processing Algorithms“. In: *SIGMOD Conference* (2000), S. 588.

[Ber+01] Jochen van den Bercken, Björn Blohsfeld, Jens-Peter Dittrich, Jürgen Krämer, Tobias Schäfer, Martin Schneider und Bernhard Seeger: „XXL - A Library Approach to Supporting Efficient Implementations of Advanced Database Queries“. In: *Proceedings of the International Conference on Very Large Databases (VLDB)* (2001), S. 39–48.

[BL03] Bärbel Burkhard und Guido Laures:
„SOA – Wertstiftendes Architektur-Paradigma“.
In: *Objektspektrum* 6 (Nov. 2003), S. 16–22.

[BM11] Christian Baun und Viktor Mauch:
„Cluster-, Grid- und Cloud-Computing“.
Vorlesungsfolien. Hochschule Mannheim. Juli 2011.

[Bot+10] Irina Botan, Roozbeh Derakhshan, Nihal Dindar, Laura Haas, Renée J. Miller und Nesime Tatbul: „SECRET: A Model for Analysis of the Execution Semantics of Stream Processing Systems“.
In: *Proceedings of the VLDB Endowment* 3.1 (Sep. 2010), S. 232–243.

[Bun10] Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. (BITKOM):
„Cloud Computing mit extrem starkem Wachstum“. Presseinformation. Okt. 2010.

[Cam+03] Michael Cammert, Christoph Heinz, Jürgen Krämer, Martin Schneider und Bernhard Seeger: „A Status Report on XXL – a Software Infrastructure for Efficient Query Processing“.
In: *IEEE Data Engineering Bulletin* 26.2 (2003), S. 12–18.

[CGM09] Badrish Chandramouli, Jonathan Goldstein und David Maier:
„On-the-fly Progress Detection in Iterative Stream Queries“.
In: *Proceedings of the International Conference on Very Large Databases (VLDB Endowment)* 2.1 (Aug. 2009), S. 241–252.

[CH09] Daniele Catteddu und Giles Hogben:
„Cloud Computing Risk Assessment“. Techn. Ber.
European Network und Information Security Agency, Nov. 2009.

[Cha04] David A Chappell: „Enterprise Service Bus“. 1. Aufl.
O'Reilly Media, Juni 2004.

[Che76] Peter Pin-Shan Chen:
„The entity-relationship model – toward a unified view of data“.
In: *ACM Transactions on Database Systems (TODS)* 1.1 (März 1976), S. 9–36.

[CM11] Gianpaolo Cugola und Alessandro Margara: „Processing Flows of Information: From Data Stream to Complex Event Processing“.
In: *To appear in ACM Computing Surveys* (2011).

[Cod70] Edgar F. Codd:
„A Relational Model of Data for Large Shared Data Banks“.
In: *Communications of the ACM* 13.6 (Juni 1970), S. 377–387.

[DGK82] Umeshwar Dayal, Nathan Goodman und Randy H. Katz: „An Extended Relational Algebra with Control over Duplicate Elimination“. In: *Proceedings of the ACM Symposium on Principles of Database Systems* (März 1982), S. 117–123.

[EAE06] Mohamed G. Elfeky, Walid G. Aref und Ahmed K. Elmagarmid: „STAGGER: Periodicity Mining of Data Streams using Expanding Sliding Windows“. In: *6th International Conference on Data Mining (ICDM06)* (2006), S. 188–199.

[EB09] Michael Eckert und Francois Bry: „Complex Event Processing (CEP)“. In: *Informatik Spektrum* 32.2 (Apr. 2009), S. 163–167.

[Eug+03] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui und Anne-Marie Kermarrec: „The Many Faces of Publish/Subscribe“. In: *ACM Computing Surveys (CSUR)* 35.2 (Juni 2003), S. 114–131.

[Fie00] Roy Thomas Fielding: „Architectural Styles and the Design of Network-based Software Architectures“. Dissertation. University of California, Irvine, 2000.

[Fos+08] Ian Foster, Yong Zhao, Ioan Raicu und Shiyong Lu: „Cloud Computing and Grid Computing 360-Degree Compared“. In: *Grid Computing Environments Workshop 2008* (Nov. 2008), S. 1–10.

[GÖ03] Lukasz Golab und M. Tamer Özsu: „Issues in Data Stream Management“. In: *ACM SIGMOD Record* 32.2 (Juni 2003), S. 5–14.

[IBM11] IBM Corporation: „CICS Transaction Server for z/OS – Performance Guide“. Handbuch. 2011.

[Int09] International Organization for Standardization: „Structured Query Language (SQL)“. ISO/IEC 9075:2008. Jan. 2009.

[Kar72] Richard M. Karp: „Reducibility Among Combinatorial Problems“. In: *Complexity of Computer Computations* (1972), S. 85–103.

[KL04] Donald Kossmann und Frank Leymann: „Web Services“. In: *Informatik Spektrum* 27.2 (Apr. 2004), S. 117–128.

[Kle05] Leonard Kleinrock: „A vision for the Internet“. In: *ST Journal of Research* 2.1 (Nov. 2005), S. 4–5.

[Krä07] Jürgen Krämer: „Continuous Queries over Data Streams – Semantics and Implementation“. Dissertation. Philipps-Universität Marburg, 2007.

[Kre08] Stefan Krecher: „Mit dem ESB zur SOA“.
Foliensatz. PDS Programm & Datenservice GmbH. Juni 2008.

[KS04] Jürgen Krämer und Bernhard Seeger:
„PIPES – A Public Infrastructure for Processing and Exploring Streams“.
In: *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Juni 2004), S. 925–926.

[KS05] Jürgen Krämer und Bernhard Seeger:
„A Temporal Foundation for Continuous Queries over Data Streams“.
In: *Proceedings of the International Conference on Management of Data* (2005), S. 70–82.

[Lea09a] Neal Leavitt: „Complex-Event Processing Poised for Growth“.
In: *IEEE Computer* 42.4 (Apr. 2009), S. 17–20.

[Lea09b] Neal Leavitt: „Is Cloud Computing Really Ready for Prime Time?“
In: *IEEE Computer* 42.1 (Jan. 2009), S. 15–20.

[Lin09] Linux Magazin: „Netzwerke mit Zenoss überwachen“. Sonderheft.
Apr. 2009.

[Luc02] David C. Luckham: „The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems“. 1. Aufl.
Addison-Wesley Professional, Mai 2002.

[Luc07] David C. Luckham: „A Brief Overview of the Concepts of CEP“.
Übersichtsartikel. 2007.

[Luc08] David C. Luckham:
„A Short History of Complex Event Processing Part 1-3“.
Übersichtsartikel. 2008.

[LWZ04] Yan-Nei Law, Haixun Wang und Carlo Zaniolo: „Query Languages and Data Models for Database Sequences and Data Streams“.
In: *Proceedings of the International Conference on Very Large Databases (VLDB)* 30 (2004), S. 492–503.

[LY99] Tim Lindholm und Frank Yellin:
„Java(TM) Virtual Machine Specification“. 2. Aufl.
Prentice Hall, Apr. 1999.

[Mar06] Jean-Louis Marechaux: „Combining Service-Oriented Architecture and Event-Driven Architecture using an Enterprise Service Bus“.
Übersichtsartikel. IBM Corporation. März 2006.

[McI10] Roger McIlmoyle: „Cloud Computing and Storage Optimization“. Foliensatz. SunGard Availability Services LP. Mai 2010.

[McK09] McKinsey & Company: „Clearing the Air on Cloud Computing“. Diskussionspapier. Apr. 2009.

[MG09] Peter Mell und Tim Grance: „The NIST Definition of Cloud Computing“. In: *National Institute of Standards and Technology* 53.6 (2009), S. 50.

[Mic06] Brenda M. Michelson: „Event-Driven Architecture Overview“. Übersichtsartikel. Patricia Seybold Group. Feb. 2006.

[Nat03] Yefim V. Natis: „Service-Oriented Architecture Scenario“. Techn. Ber. AV-19-6751. Gartner Research, Apr. 2003.

[Off11] Cabinet Office: „ITIL Continual Service Improvement“. Bd. 5. The Stationery Office, Juli 2011.

[Ort01] Ed Ort: „A Test of Java Virtual Machine Performance“. Techn. Ber. Sun Microsystems, Feb. 2001.

[Pat10] Michael Paterson: „Evaluation of Nagios for Real-time Cloud Virtual Machine Monitoring“. Arbeitsbericht. University of Victoria. Jan. 2010.

[PZL08] Cesare Pautasso, Olaf Zimmermann und Frank Leymann: „RESTful Web Services vs. „Big“ Web Services: Making the Right Architectural Decision“. In: *Proceedings of the 17th International World Wide Web Conference Committee (IW3C2)* (Apr. 2008), S. 805–814.

[Rie08] Tobias Riemenschneider: „Optimierung kontinuierlicher Anfragen auf Basis statistischer Metadaten“. Dissertation. Philipps-Universität Marburg, 2008.

[Rob86] Lawrence G. Roberts: „The Arpanet and computer networks“. In: *Proceedings of the ACM Conference on The history of personal workstations* (Jan. 1986), S. 51–58.

[RR07] Leonard Richardson und Sam Ruby: „RESTful Web Services“. 1. Aufl. O'Reilly Media, Inc., Mai 2007.

[Sac09] Matthew Sacks: „Application Performance Monitoring with Hyperic HQ“. Linux Pro Magazine. Mai 2009.

[SJS01] Giedrius Slivinskas, Christian S. Jensen und Richard T. Snodgrass: „A Foundation for Conventional and Temporal Query Optimization Addressing Duplicates and Ordering“. In: *IEEE Transactions on Knowledge and Data Engineering* 13.1 (Feb. 2001), S. 21–49.

[Sli01] Giedrius Slivinskas:
„A Middleware Approach to Temporal Query Processing“.
Dissertation. Aalborg University, 2001.

[SN96] Roy Schulte und Yefim Natis: „Service Oriented Architectures, Part 1“.
Techn. Ber. SPA-00-7425. Gartner Research, Apr. 1996.

[Sno87] Richard T. Snodgrass: „The Temporal Query Language TQuel“.
In: *ACM Transactions on Database Systems* 12.2 (Juni 1987), S. 247–298.

[Str11] Stratecast Research Group:
„Overcoming Obstacles to Cloud Computing“. Marktforschungsbericht.
Feb. 2011.

[Sug+00] Toshio Suganuma, Takeshi Ogasawara, Mikio Takeuchi, Toshiaki Yasue,
Motohiro Kawahito, Kazuaki Ishizaki, Hideaki Komatsu und
Toshio Nakatani: „Overview of the IBM Java Just-in-Time Compiler“.
In: *IBM Systems Journal* 39.1 (Feb. 2000), S. 175–193.

[Suna] Sun Microsystems: „J2SE 5.0 Performance White Paper“.
Übersichtsartikel.

[Sunb] Sun Microsystems:
„Java SE 6 HotSpot Virtual Machine Garbage Collection Tuning“.
Übersichtsartikel.

[Sun06] Sun Microsystems:
„Memory Management in the Java HotSpot Virtual Machine“.
Übersichtsartikel. Apr. 2006.

[Sym11] Symantec Corporation:
„Virtualization and Evolution to the Cloud Survey“.
Marktforschungsbericht. Juni 2011.

[TÖ09] Yingying Tao und M. Tamer Özsu:
„Mining Data Streams with Periodically Changing Distributions“.
In: *Proceeding of the 18th ACM conference on Information and knowledge management (CIKM)* (2009), S. 887–896.

[Tuc+03] Peter A. Tucker, David Maier, Tim Sheard und Leonidas Fegaras:
„Exploiting Punctuation Semantics in Continuous Data Streams“.
In: *IEEE Transactions On Knowledge And Data Engineering* 15.3 (Mai 2003).

[Vaq+09] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres und Maik Lindner:
„A Break in the Clouds: Towards a Cloud Definition“. In: *ACM SIGCOMM Computer Communication Review* 39.1 (Jan. 2009), S. 50–55.

[WS04] Thomas Walter und Peter Soth:
„Service Orientierte Architekturen mit BEA WebLogic“.
Foliensatz. BEA Systems (Central/Eastern Europe). Juli 2004.

Onlinequellen

[ACW] Amazon CloudWatch: `http://aws.amazon.com/cloudwatch/`

[AEB] Amazon Elastic Beanstalk:
`http://aws.amazon.com/elasticbeanstalk/`

[AH] Apache Hadoop: `http://hadoop.apache.org/`

[AS] AppScale: `http://appscale.cs.ucsb.edu/`

[AWS] Amazon Web Services: `http://aws.amazon.com/`

[BC] Bungee Connect: `http://www.bungeelabs.com/`

[CK] Cloudkick: `https://www.cloudkick.com/`

[CS] CloudStack: `http://www.cloud.com/`

[CW] Computerwoche (27.04.2011) – Amazon konnte nicht alle Daten retten:
`http://www.computerwoche.de/management/cloud-computing/2484197` (besucht am 04. 07. 2011)

[DF] Django Framework: `https://www.djangoproject.com/`

[Droa] Dropbox: `http://www.dropbox.com/`

[Drob] Dropbox: Where are my files stored?:
`https://www.dropbox.com/help/7/` (besucht am 28. 08. 2011)

[Euc] Eucalyptus: `http://open.eucalyptus.com/`

[Fla] Adobe Flash: `http://www.adobe.com/products/flash.html`

[FTD] Financial Times Deutschland (27.04.2011) – Amazons Wolke versagt tagelang: `http://www.ftd.de/it-medien/medien-internet/instabile-cloud-server-amazons-wolke-versagt-tagelang/60044292.html` (besucht am 04. 07. 2011)

[Gan] Ganglia: `http://ganglia.sourceforge.net/`

[GG] GoGrid: `http://www.gogrid.com/`

[Gg1] Google Storage: `http://code.google.com/apis/storage/`

[Gg2] Google App Engine: `http://code.google.com/appengine/`

[Gg3] Google Apps: `http://www.google.com/apps/`

[Gg4] Google Maps: `http://code.google.com/apis/maps/`

[Gol] Golem (27.01.2010) – Cloudkick - anbieterunabhängige Verwaltung von Cloud-Servern: `http://www.golem.de/1001/72682.html` (besucht am 04. 09. 2011)

[HO] Heise Online (27.08.2011) – 200.000 Festplatten in einem 120-Petabyte-Cluster: `http://heise.de/-1332160` (besucht am 16. 09. 2011)

[HQ] JBoss HornetQ: `http://www.jboss.org/hornetq/`

[Hyp] Hyperic: `http://www.hyperic.com/`

[Ici] Icinga: `https://www.icinga.org/`

[IWa] InfoWorld (12.07.2011) – Startup launches new SaaS cloud monitoring solution, RevealCloud: `http://www.infoworld.com/d/virtualization/startup-copperegg-launches-new-saas-cloud-monitoring-solution-called-revealcloud-743` (besucht am 04. 09. 2011)

[IWb] InfoWorld (16.12.2005) – Microsoft, IBM, SAP discontinue UDDI registry effort: `http://www.infoworld.com/d/architecture/microsoft-ibm-sap-discontinue-uddi-registry-effort-777` (besucht am 29. 07. 2011)

[JA] Java Applets: `http://java.sun.com/applets/`

[JFC] JFreeChart: `http://www.jfree.org/jfreechart/`

[JND] Java Naming and Directory Interface (JNDI): `http://www.oracle.com/technetwork/java/index-jsp-137536.html`

[JSRa] JSR 221 (JDBC): `http://www.jcp.org/en/jsr/detail?id=221`

[JSRb] JSR 3 (JMX): `http://www.jcp.org/en/jsr/detail?id=3`

[JSRc] JSR 914 (JMS): `http://www.jcp.org/en/jsr/detail?id=914/`

[KVM] Kernel-based Virtual Machine: `http://www.linux-kvm.org/`

[MAT] Memory Analyzer: `http://www.eclipse.org/mat/`

[Mor] Timothy Prickett Morgan (19.07.2010) – NASA and Rackspace open source cloud fluffer: http://www.theregister.co.uk/2010/07/19/nasa_rackspace_openstack/ (besucht am 16. 09. 2011)

[MZ] ARIS MashZone: http://www.mashzone.com/en/mashzone

[Nag] Nagios: http://www.nagios.org/

[Nim] Nimbus: http://www.nimbusproject.org/

[Nir] Nirvanix: http://www.nirvanix.com/

[ON] OpenNebula: http://www.opennebula.org/

[OS] OpenStack: http://www.openstack.org/

[OV] Opsview: http://www.opsview.com/

[Rac] Rackspace: http://www.rackspace.com/

[Rai] Costin Raiu (16.05.2011) – Chromebook: A new class of risks: http://www.net-security.org/secworld.php?id=11021 (besucht am 16. 09. 2011)

[RC] RevealCloud: http://www.copperegg.com/product/revealcloud/

[RFCa] RFC 2616 (HTTP/1.1): http://tools.ietf.org/html/rfc2616/

[RFCb] RFC 5321 (SMTP): http://tools.ietf.org/html/rfc5321/

[RFCc] RFC 791 (IP): http://tools.ietf.org/html/rfc791/

[RFCd] RFC 793 (TCP): http://tools.ietf.org/html/rfc793/

[RFCe] RFC 959 (FTP): http://tools.ietf.org/html/rfc959/

[RSS] Spezifikation von RSS: http://www.rssboard.org/rss-specification/

[Sal] Salesforce: http://www.salesforce.com/

[Sig] Sigar: http://www.hyperic.com/products/sigar/

[SL] Microsoft Silverlight: http://www.silverlight.net/

[TAE] TyphoonAE: http://code.google.com/p/typhoonae/

[Ter] 3tera: http://www.3tera.com/

[UDD] UDDI Spezifikation: http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm

[VMW] VMware: http://www.vmware.com/

[W3Ca] W3C Definition von Web Services: http://www.w3.org/TR/wsa-reqs/

[W3Cb] W3C Entwurf von XMLHttpRequest: http://www.w3.org/TR/XMLHttpRequest/

[W3Cc] W3C Spezifikation von SOAP: http://www.w3.org/TR/soap/

[W3Cd] W3C Spezifikation von WSDL: http://www.w3.org/TR/wsdl20/

[WA] Windows Azure: http://www.microsoft.com/windowsazure/

[WL] Windows Live: http://explore.live.com/

[WM] WatchMouse: http://www.watchmouse.com/

[Xen] Xen: http://xen.org/

[XXL] eXtensible and fleXible Library: http://code.google.com/p/xxl/

[Zen] Zenoss: http://www.zenoss.com/

[Zoh] Zoho: http://www.zoho.com/

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltext- und Algorithmenverzeichnis

Abkürzungsverzeichnis

AJAX . . . . . Asynchronous JavaScript and XML
API . . . . . Application Programming Interface
BPR . . . . . Business Process Redesign
BITKOM . . . . . Bundesverband IKT und neue Medien e.V.
CEP . . . . . Complex Event Processing
CRM . . . . . Customer Relationship Management
CPU . . . . . Central Processing Unit
DBMS . . . . . Data Base Management System
DDoS . . . . . Distributed Denial-of-Service
DHCP . . . . . Dynamic Host Configuration Protocol
DSMS . . . . . Data Stream Management System
EDA . . . . . Event Driven Architecture
ENISA . . . . . European Network and Information Security Agency
ER . . . . . Entity-Relationship
ESB . . . . . Enterprise Service Bus
FTP . . . . . File Transfer Protocol
GC . . . . . Garbage Collection
GPU . . . . . Graphics Processing Unit
HaaS . . . . . Human as a Service
HTTP . . . . . Hypertext Transfer Protocol
IaaS . . . . . Infrastructure as a Service
IBM . . . . . International Business Machines
IKT . . . . . Informations- und Kommunikationstechnologie
IP . . . . . Internet Protocol
JDBC . . . . . Java Database Connectivity
JMS . . . . . Java Messaging Service
JMX . . . . . Java Management eXtensions
JNDI . . . . . Java Naming and Directory Interface
JNI . . . . . Java Native Interface
JVM . . . . . Java Virtual Machine
KVM . . . . . Kernel-based Virtual Machine
LAN . . . . . Local Area Network
LDAP . . . . . Lightweight Directory Access Protocol
NASA . . . . . National Aeronautics and Space Administration

NIST . . . . . National Institute for Standards and Technology
NP . . . . . Nichtdeterministisch Polynomielle Zeit
P2P . . . . . Peer-to-Peer
PaaS . . . . . Platform as a Service
PC . . . . . Personal Computer
REST . . . . . Representational State Transfer
RFID . . . . . Radio Frequency Identification
RSS . . . . . Really Simple Syndication
SaaS . . . . . Software as a Service
SLA . . . . . Service Level Agreement
SMS . . . . . Short Message Service
SMTP . . . . . Simple Mail Transfer Protocol
SOA . . . . . Service Oriented Architecture
SOAP . . . . . Simple Object Access Protocol
SQL . . . . . Structured Query Language
TCP . . . . . Transmission Control Protocol
UDDI . . . . . Universal Description, Discovery and Integration
UDA . . . . . User Defined Aggregate
URI . . . . . Uniform Resource Identifier
VM . . . . . Virtual Machine
VMM . . . . . Virtual Machine Monitor
W3C . . . . . World Wide Web Consortium
WSDL . . . . . Web Services Description Language
XaaS . . . . . Everything as a Service
XML . . . . . eXtensible Markup Language
XXL . . . . . eXtensible and fleXible Library