Universität Trier

Diplomarbeit

# Kooperative verteilte Fehleranalyse auf der Basis von Crowdsourcing

*vorgelegt von*

David Christ
MatNr.: 889921
chri4702@uni-trier.de

*vorgelegt zum*

29. Mai 2012

*vorgelegt am*

Lehrstuhl für
Systemsoftware und
verteilte Systeme
der Abteilung Informatik
der Universität Trier

Prof. Dr. Peter Sturm

Ich möchte an dieser Stelle einigen Personen danken.

Professor Sturm für die Idee der Arbeit und die gute Betreuung.

Guido und Peter für Tipps und Denkanstöße.

Pol, Tim und Tina für das Korrekturlesen der Arbeit.

Laura und Martin,
die mich während des letzten halben Jahres bei Verstand gehalten haben

Dem Fachschaftsrat Informatik und der *Doctor Who*-Selbsthilfegruppe Trier,
ohne die mein Studium nicht das gleiche gewesen wäre.

# Abriss

Ziel der vorliegenden Arbeit ist die Implementierung eines Programms zum entfernten Sammeln von Fehlerberichten möglichst vieler unterschiedlicher Computer über das Internet, das verteilte Speichern der gesammelten Informationen auf den Computern der Teilnehmer und die offene und kooperative Auswertung dieser Fehlerberichte.

Die Hoffnung dabei ist eine breite Akzeptanz der Hersteller, die Gleichstellung unterschiedlichster Entwicklerfirmen und die Teilnahmemöglichkeit für firmenexterne Personen. Es sollen Synergieeffekte ermöglicht werden die entstehen können, wenn bezahlte Experten gleichermaßen wie Hobbyisten gemeinsam an einer Aufgabe arbeiten.

Die zentrale Forschungsfrage ist, welche Voraussetzungen erfüllt werden müssen, um die gezeichnete Idee zu realisieren und welche Probleme dabei auftreten und zu bewältigen sind. Es ist zu klären, was die nötigen Elemente eines solchen Systems wären und wie die Anforderungen an jedes dieser Elemente sind. Zur Orientierung werden im Einsatz befindliche Systeme sowie existierende Forschungsergebnisse betrachtet, wo sinnvoll werden vorhandene Technologien wiederverwendet.

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Motivation

Der in Abbildung 1.1 gezeigte, weitreichend bekannte Dialog repräsentiert den aktuellen Stand der Technik in der entfernten Analyse von Programmfehlern und ist das Ergebnis einer langen Evolution.

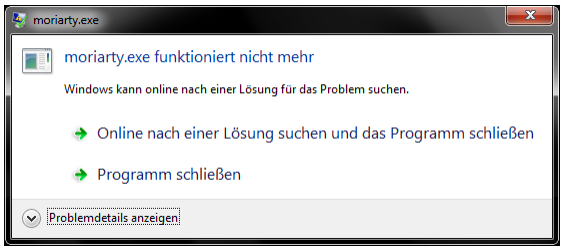


Abbildung 1.1: Programmfehler

Programmfehler gibt es spätestens seit der Entwicklung der ersten Programme. Auf den Computern der Entwickler hat die Suche nach diesen bei einfachen Ausgaben auf der Kommandozeile angefangen und über Debugger ihren Weg hin zu semi-automatisierten Unittests genommen.
Die Entwicklung des Internets ermöglichte, Debugging auf eine neue Ebene zu führen: Fehler, die auf dem Computer eines normalen Endanwenders automatisiert erkennbar sind, lösen das Erstellen eines Fehlerberichtes aus, der über das Internet zum Hersteller gesendet wird. Dieser verfügt über Wege, die Menge eingehender Berichte automatisiert vorzusortieren, damit sie von Menschen analysiert werden kann. Die Berichte geben außerdem Auskunft über die Häufigkeit bestimmter Fehler und decken auch selten auftretende Fehler auf.

Das Konzept funktioniert. Warum also überhaupt eine erneute Beschäftigung mit der Materie? Fehleranalyse ist seit jeher ein zentrales Problem in der Softwareentwicklung. Warum also nicht mit dem grundlegenden Konzept, dem aktuellen Stand der Technik und einer neu hinzukommenden Idee bei Null anfangen? Trotz des ausgefeilten Mechanismus sind einige Fragen noch immer nicht befriedigend gelöst oder werden in Zukunft neue Probleme aufwerfen.
Die Menge an Fehlerberichten wächst durch die steigende Anzahl an Computern weiter. Dies fordert bei der momentanen Client-Server-Architektur Verfahren für die Organisation mehrerer Server und die Verteilung der Last auf selbige.
Außerdem wirft das Aufkommen immer größerer und komplexerer Software neue Probleme auf. Durch die Kombination unterschiedlicher Software in Form von Bibliotheken, Frameworks und Softwarekomponenten werden Zusammenhänge komplexer und

das Kaskadieren eines Fehlers macht diesen für Menschen nicht selten schwerer erkennbar.
Weiter ist die verbreitete Methode die Sammlung der Fehlerberichte durch die Hersteller der Betriebssysteme. In den meisten Fällen sind diese Berichte nur unter Auflagen oder gar nicht von Externen einsehbar – Vergleich [Winqual] und [CrashReporter], genauer erläutert im Grundlagen-Kapitel.
Zuletzt verdeutlicht das Aufkommen unterschiedlicher Plattformen der vergangenen Jahre einen Mangel an der aktuellen Methodik: Während noch vor einigen Jahren in den meisten Fällen „Betriebssystem“ mit Windows gleichzusetzen war, steigt heute der Anteil von Mac OS X und auch Linux. Auch haben sich die Systeme auf Embeded-Geräten von Firmwares hin zu vollständigen Betriebssystemen entwickelt. Dieser Umstand hat oft zu einem eigenen Mechanismus zum Sammeln von Fehlerberichten speziell für die jeweilige Plattform geführt, sogar zu unterschiedlichen Systemen für verschiedene Linux-Distributionen und Desktopumgebungen. Eine große Anzahl an Bibliotheken und Programmen wird jedoch für eine Vielzahl an Betriebssystemen angeboten. Einige Hersteller solcher Programme haben daher ihrerseits einen eigenen Mechanismus entwickelt, der einerseits plattformübergreifend ist und andererseits die Auflagen der Hersteller der Betriebssysteme umgeht – so zum Beispiel der Browser Firefox aus dem Hause Mozilla mit [Breakpad]. Die Ressourcen für die Entwicklung eines solchen Mechanismus sowie der nötigen Infrastruktur können jedoch vor allem von kleinen Firmen nicht immer aufgebracht werden.

Im Rahmen dieser Arbeit wird das bekannte Konzept des entfernten Sammelns von Fehlerberichten um zwei Elemente erweitert. Zum Ersten um das Auslagern der zu bewältigenden Aufgaben auf die breite Masse. So ist neben dem Sammeln von Fehlerberichten auch das zur Verfügung stellen des Speicherplatzes und die Analyse der Fehlerberichte eine Aufgabe, die von jedem angenommen werden kann. Zum Zweiten um Plattformunabhängigkeit, so dass Softwareentwickler ein einheitliches, übergreifendes System zu deren Verfügung haben.
Die Hoffnung dabei ist eine breite Akzeptanz der Hersteller, die Gleichstellung von Entwicklerfirmen unterschiedlicher Größe und die Teilnahmemöglichkeit für firmenexterne Personen. Es sollen Synergieeffekte ermöglicht werden die entstehen können, wenn bezahlte Experten gleichermaßen wie Hobbyisten gemeinsam an einer Aufgabe arbeiten.

Nicht erst seit dem Bau der Pyramiden ist klar, dass große Aufgaben in Gemeinschaftsarbeit besser bewältigt werden können, als alleine. Forschungswettbewerbe, aber auch der Publikumsjoker einer bekannten Quizshow sind Anzeichen dafür, dass die Weisheit der Masse real ist und nicht selten sogar bessere Ergebnisse produziert, als das Anrufen eines einzelnen Experten. Laut [Surowiecki, 2004] Seite 4 lag die Erfolgsquote der in der Quizshow angerufenen Experten bei durchschnittlich 65%, die des Publikums bei 91%.
Mit dem Internet wurde eine Plattform geschaffen, welche die Möglichkeiten dieses Konzepts *Crowdsourcing* auf ein neues Niveau hebt. [Wikipedia] etwa nutzt das Wissen der Masse, um eine Online-Enzyklopädie zu erstellen. Das Projekt [OpenStreetMap]

nutzt die Arbeitskraft der Bevölkerung und sammelt Geodaten zur Erstellung von Kartenmaterial. Die Teilnehmer tragen deren Ergebnisse auf für diese Zwecke geschaffenen Plattformen zusammen. Dabei kann jeder die Arbeiten der anderen einsehen, sich selbst einen Arbeitsbereich auswählen und an Diskussionen teilnehmen.

Das Einbringen der ersten beiden Elemente in das Modell der entfernten Fehleranalyse führt zur Notwendigkeit eines dritten Elements, dem Datenschutz. Fehlerberichte enthalten unter Umständen sensible Daten. Stürzt ein Programm etwa ab und wird von diesem Programm der Inhalt des Arbeitsspeichers aufgezeichnet, so kann dieser laut [Broadwell et al., 2003] Seite 273 Kreditkartennummern, Passwörter oder (bei Browsern) Cookies enthalten. Informationen über eine Netzwerkkarte können deren IP- oder MAC-Adresse enthalten, Informationen über eine gekaufte Software Name und Anschrift des Benutzers. Alle diese Daten fließen potentiell in Fehlerberichte ein. Trotz der öffentlichen Einsehbarkeit dieser sollen die sensiblen Daten geschützt und die Privatsphäre der Benutzer gewahrt bleiben.

## 1.2 Forschungsfrage

Ziel der vorliegenden Arbeit ist die Implementierung eines Programms zum entfernten Sammeln von Fehlerberichten über das Internet, das verteilte Speichern der gesammelten Informationen auf den Computern der Teilnehmer und die offene und kooperative Auswertung dieser Fehlerberichte. Die beteiligten Geräte der Endanwender sollen dabei nicht nur eine möglichst große Anzahl, sondern auch möglichst unterschiedliche Fehlerberichte sowohl verschiedener Betriebssysteme (Windows, Linux, Mac OS X, ...) als auch unterschiedlicher Geräteklassen (Dektop, Laptop, Tablet, Smartphone, ...) zusammentragen. Die Speicherung der zusammengetragenen Berichte soll ebenfalls von Endanwendern übernommen werden, wofür Konzepte verteilter Datenbanken betrachtet werden. Die Möglichkeit zur Analyse der Fehlerberichte soll im letzten Schritt nicht nur, wie sonst üblich, nur dem Betriebssystem- oder Produkthersteller gegeben sein, sondern nach dem Konzept des Crowdsourcing auch der Allgemeinheit. Im Zuge der breiten Zugänglichkeit dieser Daten sollen auch Fragen des Schutzes persönlicher Daten diskutiert werden.

Die zentrale Forschungsfrage, welche sich aus der Motivation ergibt, beschäftigt sich damit, welche Voraussetzungen erfüllt werden müssen, um die gezeichnete Idee zu realisieren und welche Probleme dabei auftreten und zu bewältigen sind. Es ist zu klären was die nötigen Elemente eines solchen Systems wären und wie die Anforderungen an jedes dieser Elemente sind. Zur Orientierung werden im Einsatz befindliche Systeme aus den erwähnten Teilbereichen sowie existierende Forschungsergebnisse betrachtet, wo sinnvoll sollen vorhandene Technologien wiederverwendet werden.

Die breite Verwendung von Programmen zum entfernten Sammeln von Fehlerberichten in Betriebssystemen wie Microsofts Windows, Apples Mac OS X und Canonicals Ubuntu Linux soll für diese Arbeit als empirischer Nachweis der Nützlichkeit der gesammelten

Fehlerberichte sowie deren Repräsentativität für die Masse der Benutzer ausreichen. Ein formaler Nachweis dieser Repräsentativität fehlt an dieser Stelle und ist für die Zwecke dieser Arbeit nicht notwendig.

## 1.3 Nutzungsszenario

Auf jedem internetfähigen Gerät weltweit läuft eine Software, die nach dem bekannten Konzept Softwarefehler erkennt, einen Bericht anfertigt und einschickt. Statt an einen zentralen Server jedoch werden die Berichte an ein verteiltes Netzwerk von Datenspeichern übergeben, an dem wiederum jedes Gerät weltweit teilnimmt, welches über hinreichend viele freie Ressourcen verfügt. Aus diesem verteilten Datenspeicher bekommen Teilnehmer, reichend von Firmenmitarbeitern bis hin zu versierten Laien, die Berichte angezeigt, für die sie zuständig sind oder für die sie Interesse angemeldet haben. Sie können mit anderen Teilnehmern über diese Berichte Diskussionen führen und Lösungsvorschläge einreichen.

Tritt auf einem Gerät ein Fehler auf zu dem Lösungsvorschläge existieren, bekommt der Endanwender diese präsentiert. Probiert er einen von diesen aus, kann er für ihn abstimmen. Hat ein Lösungsvorschlag hinreichend viele Stimmen gesammelt, so wird er automatisch als Lösung angenommen und fortan weiteren Endanwendern als einziger präsentiert – die Masse hat eine Entscheidung getroffen.

Firmen können auf bestimmte Fehlerursachen „Kopfgelder“ aussetzen, die sie firmenintern nicht lösen können oder wollen. Auch Endanwender können Belohnungen für die Lösung von Fehlerursachen aussprechen, an deren Behebung sie interessiert sind. Der Teilnehmer, der die über das oben beschriebene Verfahren gewählte Lösung eingereicht hat, bekommt die versprochene Gesamtsumme.

Das System führt Statistiken darüber, welcher Teilnehmer zu welchen Problemen Lösungen gefunden hat. So kann ein Teilnehmer vom System als Experte für eine bestimmte Gruppe von Fehlerberichten gekennzeichnet werden. Über diesen Bewertungsmechanismus können Firmen beispielsweise auf Teilnehmer aufmerksam werden, die häufig Lösungen zu Fehlern in ihren Produkten finden, und diesen ein Jobangebot unterbreiten.

Nebenbei erstellt das System Statistiken über die aktuelle Häufigkeit bestimmter Fehler und ist so auch ein Indikator für die Menge an betroffenen Nutzern. Als Seiteneffekt ließen sich außerdem repräsentative Statistiken über Nutzung und Verteilung von Betriebssystemen und Programmen erstellen.

## 1.4 Aufbau der Arbeit

Nach der oben gegebenen Beschreibung der Idee dieser Arbeit werden im folgenden Grundlagen-Kapitel Programme vorgestellt, die Teile der für diese Arbeit benötigten

Funktionen erbringen. Im darauffolgenden Kapitel werden die so gewonnenen Einblicke zu einem konkreten Entwurf zusammengetragen und ergänzt. In der Implementierung wird ein Teilaspekt des Entwurfes programmatisch umgesetzt um die Machbarkeit zu belegen und im anschließenden Kapitel die Basis für eine Leistungsbewertung zu schaffen. Der letzte Teil dieser Arbeit verdeutlicht noch einmal die erlangten Erkenntnisse und konkretisiert Ansatzpunkte für weiterführende Arbeiten.

# 2 Grundlagen

## 2.1 Terminologie

Die Bezeichnung für die Computer der Teilnehmer des im Rahmen der Arbeit erstellten Netzwerks orientiert sich an der Terminologie des Grid-Computing. Die Grid-Architektur bezeichnet einen Zusammenschluss lose gekoppelter, potentiell heterogener Computer. Für die Bezeichnung der teilnehmenden Computer wird das Wort Knoten (engl.: Nodes) verwendet, das physisch verbindende Netzwerk wird das Internet sein. Auch wird die auf die Infrastruktur des physischen Netzwerkes aufgesetzte logische Struktur als Netzwerk bezeichnet. Ein Grid-Netzwerk benötigt idealerweise keine zentrale Instanz (beispielsweise zur Koordination), im Gegensatz etwa zur Client-Server-Architektur, sondern besteht aus gleichberechtigten Knoten. Im nächsten Schritt bedeutet dies, dass einzelne Knoten ausfallen können, ohne dass das Netzwerk nicht mehr funktioniert, es gibt keinen SPOF (engl.: single point of failure).

Unerwartetes oder fehlerhaftes Verhalten eines Programmes wird wie üblich einfach als Fehler (engl.: Error) bezeichnet, die außerplanmäßige Beendigung eines Programmes zusätzlich auch als Absturz (engl.: Crash). Dem gegenüber steht die Ursache für den Fehler oder Absturz, bezeichnet als Fehlerursache (engl.: Bug oder Root-Cause). Die Suche nach der Fehlerursache ist die Fehleranalyse (engl.: Debugging). Deren Ergebnis kann, zumindest temporär, ein Weg zur Fehlerumgehung (engl.: Workaround) sein, oder eine Fehlerkorrektur (engl.: Patch oder Update).

Die zur Fehleranalyse benötigten Daten sind unter anderem das Arbeitsspeicherabbild des Prozesses (engl.: Memory Dump, kurz Memdump oder nur Dump), die Prozessorregister, die Liste der geladenen Bibliotheken oder die Meta-Informationen wie Name und Version des fehlerhaften Programms. In der Fachliteratur, zum Beispiel in [Kinshumann et al., 2011] oder [Pacheco, 2011], wird häufig der Begriff des *postmortem debugging* verwendet – debugging nach dem Absturz (Tod) des Prozesses. Dieser ist jedoch für die Zwecke dieser Arbeit zu restriktiv, da ein erkennbarer Fehler mehr als nur ein Programmabsturz sein kann. Die (unmittelbar) nach dem Auftreten eines Fehlers sammelbaren Informationen werden daher in dieser Arbeit als *post error debug informations* bezeichnet. Dieser Begriff hat seinen Ursprung wahrscheinlich in [Wiki WER], wo seine Bedeutung jedoch auf Speicherabbilder beschränkt wird, denn die entsprechende Textpassage ist im exakten Wortlauf auf vielen Internetseiten wiederzufinden. Trotz seines Ursprungs aus einer wenig fachlichen Quelle, eignet der Begriff sich wegen seiner Allgemeingültigkeit für alle Fehler in dieser Arbeit besser als *postmortem*

*debugging*. *Post error debug informations* (kurz PEDIs) werden benutzt, um die Fehlerursache zu einem späteren Zeitpunkt zu analysieren. Sie werden in Form sogenannter (Fehler-)Berichte zusammengefasst. Diese Berichte können in großem Umfang von vielen betroffenen Computern gesammelt werden, um eine breite Basis zu analysierender Daten zur Verfügung zu stellen. Im Bezug auf Abstürze wird in der Fachliteratur unter anderem der Begriff *remote crash reporting* (dt.: entfernte Absturz-Berichterstattung) benutzt (vgl. [Broadwell et al., 2003]).

Vom Programm protokollierte Laufzeitinformationen werden als Mitschnitt (engl.: Log) bezeichnet und können ebenfalls zur Fehleranalyse genutzt werden.

Diese Arbeit beschäftigt sich mit dem Thema Crowdsourcing. Bei Crowdsourcing handelt es sich um ein Kofferwort aus dem englischen Wort für Menschenmasse (engl.: crowd) und dem ökonomischen Begriff des Auslagerns (engl.: outsourcing). Crowdsourcing bezeichnet daher naheliegenderweise das Auslagern von Aufgaben auf eine große Menge an Menschen. Neben dem einfachen Verteilen von Arbeitslast steht der Begriff auch für das Nutzen der Intelligenz der Masse und für die Tatsache, dass die Aggregierung des Wissens vieler unter Umständen korrekter ist, als das Wissen eines einzelnen Experten – Vergleich [Surowiecki, 2004].

In dieser Arbeit wird das Themenbereich Datenschutz im Bezug auf informationelle Selbstbestimmung, Anonymität und Schutz vor Datenmissbrauch betrachtet.

Plattform, oder die Plattform auf der eine Software läuft, kann im Sinne dieser Arbeit dass verwendete Betriebssystem bezeichnen, aber auch eine Laufzeitumgebung für Bytecode-Sprachen (C#, Java) oder ein Interpreter für Skriptsprachen (PHP, Python), in welchen Programme (näherungsweise) unabhängig vom darunterliegenden Betriebssystem ausgeführt werden können. Plattformunabhängigkeit hingegen bezieht sich in dieser Arbeit nur auf die Lauffähigkeit unter verschiedenen Betriebssystemen.

Des Weiteren zieht diese Arbeit Vergleiche zu sozialen Netzwerken und entlehnt aus deren Terminologie zum Beispiel die Begriffe Profil und Blog im Bezug auf die Teilnehmer sowie den Zusammenschluss von Teilnehmern zu Gruppen.

Die Software, die auf den Knoten im zu erstellenden Netzwerk laufen wird, teilt sich in drei Kategorien:

- Software, die Fehlerberichte sammelt und einschickt – und gegebenenfalls später Lösungsvorschläge präsentiert bekommt und bewertet
- Software, die die gesammelten Fehlerberichte entgegennimmt und speichert
- Software, mit der eine Person diese Fehlerberichte abrufen kann, vor allem um sie zu analysieren

Die Programme, welche diese Funktionen implementieren, werden entsprechend als Sammel-Software, Speicher-Software und Analyse-Software bezeichnet, die Computer im

Grid, auf welchen die Software läuft, als Sammel-Knoten, Speicher-Knoten und Analyse-Knoten. Ein Knoten kann dabei auch mehrere der drei Funktionen übernehmen, wobei sie in dieser Arbeit (wo nicht anders angegeben) disjunkt betrachtet werden.

Für diese Arbeit wird der Begriff Komponente für die einzelnen Teile eines Computer-Systems verwendet, gleich ob Hard- oder Software. Da jeder Teil eines Systems für die Fehlerursache von Bedeutung sein kann, ist eine weitere Unterteilung in dieser Arbeit, wo nicht anders angegeben, nicht von Bedeutung.

### 2.1.1 Relationale und NoSQL-Datenbanken

Für den Aspekt der Speicherung von Daten, insbesondere der Fehlerberichte, beschäftigt sich diese Arbeit mit dem Thema Datenbanken. Eine detaillierte Erläuterung der Materie übersteigt den Rahmen, jedoch soll an dieser Stelle kurz ein Überblick über relationale sowie NoSQL-Datenbanken gegeben werden. Zunächst seien jedoch das ACID-Paradigma und das CAP-Theorem erwähnt.

[Kemper und Eickler, 2006] beschreibt auf Seite 273 die folgenden vier Eigenschaften für Transaktionen auf einer Datenbank als das ACID-Paradigma.

**Atomicity** „[Es werden entweder] alle Änderungen der Transaktion in der Datenbasis festgeschrieben oder gar keine."

**Consistency** „Eine Transaktion hinterlässt nach Beendigung einen konsistenten Datenbasiszustand. Andernfalls wird sie komplett (siehe Atomarität) zurückgesetzt."

**Isolation** „Jede Transaktion muss – logisch gesehen – so ausgeführt werden, als wäre sie die einzige Transaktion, die während ihrer gesamten Ausführungszeit auf dem Datenbanksystem aktiv ist."

**Durability** „Die Wirkung einer erfolgreich abgeschlossenen Transaktion bleibt dauerhaft in der Datenbank erhalten."

Das CAP-Theorem besagt, dass es für ein verteiltes System nicht möglich ist, alle drei der folgenden Eigenschaften gleichzeitig zu erfüllen (nach [Gilbert und Lynch, 2002] Seite 52f):

**Consistency** Trotz der Verteiltheit des Systems verhält es sich, als würden alle Interaktionen seriell von einer einzelnen Instanz verarbeitet.

**Availability** Jede Anfrage an einen nicht ausgefallenen Knoten gibt stets eine Antwort zurück.

**Partition-Tolerance** Anfragen an das verteilte System werden auch bei Verlust von Nachrichten oder Ausfall von Knoten noch immer korrekt Beantwortet.

Das klassische Datenbankkonzept ist das der relationalen Datenbanken. Transaktionen auf diesen erfüllen in der Regel die Eigenschaften des ACID-Paradigmas.
Relationale Datenbanksysteme sind traditionell nicht verteilt. Sie müssen somit, im Bezug auf das CAP-Theorem, keine Partitionstoleranz (P) garantieren und erfüllen automatisch Konsistenz (C), nicht zu verwechseln mit der Konsistenz-Eigenschaft des ACID-Paradigmas. Des Weiteren bieten sie in der Regel auch Verfügbarkeit (A).
Relationale Datenbanken speichern Datensätze üblicherweise in nach festgelegten Schemata strukturierten Tabellen, beispielsweise:

$$Studenten : \{[\underline{MatrNr}, Name, Semester]\}$$

Zeilen (Einträge) in Tabellen können über eindeutige Schlüssel verfügen, die aus einer oder mehreren festgelegten Spalten der jeweiligen Tabelle bestehen – in den Beispielen je der unterstrichene Text, oben etwa die Matrikelnummern der Studenten (MatrNr). Über diese Schlüssel ist es auch möglich verschiedene Tabellen einer Datenbank in Relation zueinander setzen – daher der Name *relationale* Datenbank. Über einen sogenannten JOIN kann man so zum Beispiel mit einer Tabelle

$$Vorlesungen : \{[\underline{VorlNr}, Titel, SWS, gelesenVon]\}$$

eine Beziehung

$$hören : \{[\underline{MatrNr, VorlNr}]\}$$

erstellen. Die aufgeführten Beispiele sind [Kemper und Eickler, 2006] Seiten 69 bis 82 entnommen.
Mit einer relationalen Datenbank kann man meist mittels sogenannter SQL-Befehle (engl.: Structured Query Language) interagieren.

Mit der Verbreitung des Cloud-Computing wurde es nach [Pokorny, 2011] Seite 278 aufgrund der Nichtverteiltheit klassischer Datenbanksysteme nötig, in immer potentere und somit teurere Datenbankserver zu investieren – so genanntes *vertikales Skalieren*. Ein verteiltes System hingegen würde es ermöglichen, die aufkommende Last auf mehrere günstige Rechner zu verteilen – so genanntes *horizontales Skalieren*. Mit dieser Idee entstanden neue Datenbanksysteme, die heute unter dem Begriff NoSQL zusammengefasst werden.

Der Begriff NoSQL steht nach [Pokorny, 2011] Seite 280 für „not only SQL“ und umfasst eine lose Klasse verschiedener Datenbankkonzepte, die vom relationalen abweichende Ansätze verfolgen. Sie erfüllen in der Regel die Eigenschaften Verfügbarkeit und Partitiontoleranz (AP), folglich jedoch nicht Konsistenz (C). Statt der Konsistenz-Eigenschaft des CAP-Theorems realisieren NoSQL-Datenbanken laut [Pokorny, 2011] Seite 282 *eventual consistency* (zu dt.: letztendliche Konsistenz). Wortgemäß wird eine Änderung, die an einem einzelnen Knoten der verteilten Datenbank vorgenommen wird, zwar zu allen anderen Knoten repliziert werden, aber erst nach einer gegebenen Zeit. Dies bedeutet insbesondere, dass bei zwei gleichen Abfragen an die verteilte Datenbank, aber an verschiedene Instanzen dieser, eventuell unterschiedliche Werte – ein älterer und

ein neuerer, jedoch nie ein niemals geschriebener – zurückgegeben werden können, ohne das sich der Datenbestand zwischen den Anfragen ändert.
Die Idee bricht in direkter Folge gezwungenermaßen mit dem ACID-Paradigma – einer Grundfeste relationaler Datenbanken. Das Konzept dieser Datenbanken ist für viele heutige Anwendungen, vor allem im Web 2.0, jedoch vollkommen ausreichend – im Vergleich zu Anwendungen in Banken und Börsen. Dem Schlagwort ACID wird bei NoSQL-Datebanken BASE gegenüber gesetzt: Basically Available, Soft state, Eventually consistent ([Pokorny, 2011] Seite 279).
Durch ihr grundlegend neues Design war die Möglichkeit, an manchen Punkten auch die Notwendigkeit, gegeben, mit NoSQL-Datenbanken vom relationalen Datenmodell abzuweichen. Sie sind etwa häufig weniger streng schematisiert. So könnten Datensätze für Studenten, im Vergleich zum obigen Beispiel, innerhalb der gleichen Struktur einmal ein Tupel (MatrNr, Name, Semester) sein, ein anderes Mal (MatrNr, Name, Gesamtsemester, Fach, Fachsemester) oder auch (Name, Fach, Immatrikulationsdatum). Im Gegenzug unterstützen NoSQL-Datenbanken meist keine JOINs.
Während relationale Datenbanken ihre Stärken bei Systemen mit kleinen oder seltenen großen Transaktionen sowie häufigen Lesezugriffen haben, sind NoSQL-Datenbanken demgegenüber für viele Lese- und Schreibanfragen konzipiert.
Im Rahmen ihrer Auslegung für verteilte Systeme besitzen die meisten NoSQL-Datenbanken Replikationsmechanismen, die einen Datensatz auf mehrere Teilnehmer replizieren, um Ausfallsicherheit zu bieten. Viele verfügen außerdem über Verteilungsmechanismen, die die Menge der zu speichernden Datensätze möglichst gleichmäßig auf die Teilnehmer verteilt. Jede Instanz im Netzwerk muss dann nur

$$\frac{AnzahlDerDatensätze * ReplikationsFaktor}{AnzahlDerKnoten}$$

Datensätze speichern.

Für diese Arbeit relevant sind die in [Pokorny, 2011] Seite 280 aufgeführten häufigsten zwei Vertreter der NoSQL-Familie:

**Dokumentenorientierte Datenbanken** Speichern semistrukturierte Dokumente, ähnlich wie JSON oder XML

**Key-Value-Stores** Speichern lediglich Schlüssel-Wert-Paare, vergleichbar mit Hashtabellen

## 2.2 Stand der Technik

Zur Realisierung des Konzeptes werden nun existierende Methoden und Programme betrachtet, die Teilen der Ziele dieser Arbeit entsprechen. Sie sind erste Anhaltspunkte für den Entwurf.

### 2.2.1 Methoden zur Fehleranalyse

Während der Entwicklung und auf Seiten des Entwicklers ist eine der am weitest verbreiteten Methoden zur Erkennung von Fehlern das umgangssprachlich so genannte *printf*-Debugging – benannt nach dem entsprechenden C-Befehl. Dies bezeichnet das Ausgeben von Informationen, wie etwa dem Programmstatus oder dem Inhalt von Variablen, auf der Konsole. Außerdem werden dem Entwickler während der Entwicklungsphase einer Software programmiersprachenspezifische Werkzeuge für die Analyse des Programmablaufs zur Laufzeit an die Hand gelegt – beispielsweise [GDB], der GNU Debugger für eine Reihe an Programmiersprachen und Betriebssystemen.
Beide Methoden erfordern jedoch Modifikationen am Programm, die in den meisten Fällen nicht für einen Produktiveinsatz geeignet sind. Ebenso ist die Masse an Testsystemen stark beschränkt. Dafür entfällt jedoch die aufwändige Erhebung von Daten entfernter Computer.

Beim Testen im großem Rahmen können hingegen auch selten auftretende Fehler besser erkannt und Korrelationen – beispielsweise zu bestimmten Systemkomponenten – besser aufgedeckt werden. Im Gegenzug erfordert die Aggregierung der Daten einen aufwändigeren Mechanismus. Das einfache Sammeln von vom Benutzer verfassten Fehlerberichten in Prosa führt beispielsweise zu Teils sehr ungenauen, aber vor allem schlecht automatisch verarbeitbaren Daten.

Ein Problem beim Testen ohne Debugger oder Debug-Ausgaben stellt die Aufgabe dar, ein Fehlverhalten eines Programms überhaupt automatisch zu erkennen. Semantisch falsche, jedoch syntaktisch korrekte Ausgaben sind beispielsweise vom Benutzer erkennbar, von Programmen jedoch nur schwer – etwa die Ausgabe „$1 + 1 = 3$“.
Der Rückgabewert eines Programms nach dessen Beendigung kann hingegen vom aufrufenden Programm ausgewertet werden, wenn dieses von der Semantik des Rückgabewerts weiß. Wiederum andere Symptome von Fehlerursachen, wie die Nicht-Erreichbarkeit eines Prozesses oder dessen außerplanmäßige Beendigung (Absturz) sind recht einfach von der ausführenden Plattform erkennbar.

Bei nicht automatisiert erkennbaren Fehlern sind verständlicherweise Nutzerberichte sowie Programm-Logs meist die Möglichkeit der Wahl. Logs können dabei nur Einträge enthalten, die das Programm zur Laufzeit einfügen kann. Abstürze betreffend sind Logs daher vor allem dort aufschlussreich, wo kurz vor der fehlerhaften Stelle im Programm vom Entwickler eine Log-Ausgabe implementiert wurde, wo der Entwickler also eventuell sogar schon eine potentielle Fehlerursache vermutet hat. Speicherabbilder hingegen enthalten Informationen vom konkreten Zeitpunkt des Absturzes. Während Log-Dateien jedoch leicht und jederzeit zu erhalten sind, können Speicherabbilder nur von der ausführenden Plattform erstellt werden.

Gleich ob Nutzerbericht, Log oder PEDI, müssen die Daten einen Weg zu einem Analysten finden. Neben dem naiven Ansatz direkten Kontakts via Telefon, E-Mail oder

ähnlichem besteht die Möglichkeit des automatischen Erstellens und Versendens von Berichten. Erfahrungsgemäß verhält sich dabei die Menge der beim Entwickler eingehenden Berichte umgekehrt proportional zu dem vom Endanwender zu leistenden Arbeitsaufwand.

Der Weg einer Fehlerbehebung zurück zum Benutzer kann wiederum sowohl manuell als auch automatisiert ablaufen. Beispielsweise kann eine Antwort auf eine E-Mail gegeben werden. Ein verändertes Programm kann manuell vom Benutzer besorgt oder vom zuständigen Administrator verteilt werden. Bei vielen Programmen und Plattformen existieren separate programminterne Updateroutinen – etwa bei Firefox oder Windows. Manche Plattformen können auch allgemein das gesamte System auf den aktuellen Stand bringen – zum Beispiel mit [APT], dem Paketverwaltungssystem unter Debian und dessen Derivaten.

### 2.2.2 Programme zur Fehleranalyse

Nun werden einige in der Praxis verwendete Programme betrachtet, welche die Prozesse der Fehlererkennung, Datenerhebung und -aggregation sowie der Informationsauswertung unterstützen und teilweise automatisieren. Betrachtet werden auch deren Eigenschaften im Bezug auf diese Arbeit.

#### syslog

Logging-Mechanismen stehen heutzutage auf allen gängigen Betriebssystemen zur Verfügung und ermöglichen es einem Programm, zur Laufzeit Informationen in ein zentral von der Plattform verwaltetes digitales Logbuch zu schreiben. Klassischerweise bleiben diese Mitschnitte lediglich lokal auf den Systemen der Endbenutzer. Mit einigen Systemen – wie beispielsweise mit dem unter unixoiden Systemen verbreiteten Syslog-Protokoll ([RFC 5424]) und einer dazugehörigen Implementierung, wie etwa [Syslog-NG] – ist es ebenfalls möglich, Logbuch-Einträge über ein Netzwerk an andere Rechner zu senden. So werden beispielsweise in Firmen Mitschnitte über das interne Netzwerk auf einem zentralen Server gespeichert. Der Mechanismus bietet also eine Übertragungsmöglichkeit, die Sammlung und Auswertung wird jedoch von einer zentralen Instanz übernommen, beispielsweise dem Systemadministrator.

#### Bugzilla

Von der Möglichkeit abgesehen, Dringlichkeitsstufen oder Zugehörigkeiten festzulegen, handelt es sich bei Ticketsystemen und Bugtrackern in der Regel um einen Weg, Prosa-Berichte zu sammeln. Projekte wie [Bugzilla] sind meist für eine konkrete Anwendung zu konfigurieren und dann auf diese beschränkt. Sie werden üblicherweise von eher versierten Anwendern zum Einsenden von Fehlerberichten benutzt, wodurch sie nicht die

mit dieser Arbeit angestrebte Breite an Fehlerbericht-Sammlern bietet. Andererseits ist oft freier und uneingeschränkter Einblick in die eingereichten Tickets beziehungsweise Beiträge möglich, was ein gesetztes Ziel dieser Arbeit ist. Die Identität des Berichtenden kann oft über einen Benutzernamen „verschleiert“ werden. Der Schutz von persönlichen Daten obliegt hier dem Einzelnen.

### Nagios

Monitoring-Systeme, wie etwa [Nagios], haben ihren Fokus auf der Überwachung der Erreichbarkeit von Servern und Serverdiensten. Es ist mit Nagios dann etwa möglich, bei Nicht-Erreichbarkeit einer Internetseite deren Administrator via SMS zu benachrichtigen. Der reine Ansatz dieser Systeme ist zwar interessant, jedoch haben diese Projekte im Allgemeinen keine Auslegung für Endanwender-Computer. Auch werden eher boolsche erreichbar-/nicht-erreichbar-Informationen gesammelt und keine Logs oder PEDIs. Aufgrund der Auslegung dieser Systeme für deren Administratoren gibt es logischerweise auch keinen Grund, Fehlerberichte (mit der Masse) zu teilen.

### Windows Error Reporting

Unter Microsofts Betriebssystemen gibt es seit Windows XP mit dem *Windows Error Reporting* (WER) ein System, um automatisiert Fehlerberichte einer großen Menge an Benutzern über das Internet zu sammeln. Es ist eines der ältesten und umfangreichsten *remote crash reporting*-Systeme. Hinter dem in der Einleitung gezeigten Dialog aus Abbildung 1.1 verbirgt sich eine Windows-Komponente, die beim Absturz einer Software automatisch einen maschinell vorverarbeitbaren Fehlerbericht und je nach Konfiguration ein Speicherabbild erstellt. Der Benutzer muss anschließend nur noch gefragt werden, ob er mit der Übermittlung dieser Informationen einverstanden ist. Das System selbst erkennt dabei Abstürze nativer oder .NET-Anwendungen, kann jedoch auch von Programmen aufgerufen werden, wenn diese selbst einen internen Fehler erkennen und diesen berichten wollen.
Die durch das WER gesammelten Fehlerberichte werden von Microsoft in einer nicht frei zugänglichen Datenbank gesammelt. Zugriff auf diese Datenbank erhalten neben Microsoft nur Winqual-zertifizierte Firmen. Diese Zertifizierung benötigt man im Übrigen auch, um einer Komponente den Titel „Certified for Windows“ geben zu dürfen oder bei Treibern den „Logo-Test“ zu bestehen. Unter [Winqual] wird ein digitales Zertifikat von VeriSign als Bedingung für eine Winqual-Mitgliedschaft gefordert. Dieses schlägt (zum Zeitpunkt des Schreibens dieser Arbeit) mit 499 $/Jahr zu Buche. Ist eine Firma nun in diesem Programm angemeldet, darf sie auf die Fehlerberichte der sie betreffenden Software zugreifen. Es ist der Umgang Microsofts mit der Tatsache, dass Fehlerberichte sensible Daten enthalten können. Nur Vertrauenswürdige sollen Zugriff haben und zwar nur auf die Daten, die sie betreffen. Details zum Umgang mit der Privatsphäre der Nutzer im Bezug auf die eingereichten Fehlerberichte findet man auch auf [WER Privacy].

Hier heißt es unter anderem „Reports might [...] contain personal information [...] is not used to identify you [...]“.
Da nur die Hersteller Einsicht in die Fehlerberichte bekommen, kann hier nicht von *Crowdsourcing* gesprochen werden. Es existiert auch keine auf diese Arbeit übertragbare Lösung zu den Fragen der Datensicherheit. Möchte eine Firma keine Winqual-Zertifizierung erwerben oder möchte man als Privatperson teilnehmen, bleibt man bei dieser Methode außen vor. Außerdem ist WER sehr stark mit Windows verzahnt und somit stark plattformgebunden.

### Apport

Bei Apport handelt es sich um einen Mechanismus für die Linux-Distribution Ubuntu, der dem Konzept von WER ähnelt: Eine im Betriebssystem verankerte Komponente erkennt Abstürze und sendet sie nach Zustimmung des Nutzers an eine zentrale Datenbank. Im Unterschied zum WER ist die Komponente jedoch nur in den Entwicklerversionen der Distribution aktiviert. Auf diese Weise übertragen die Ubuntu-Entwickler die Verantwortung für den potentiell sensiblen Inhalt der gesendeten Daten auf das Technikverständnis der Benutzer. Auf der Seite [Apport] heißt es: „We can reasonably expect developers and technically savvy users, who run the development release, to be aware of this [...]“. Auf Seiten der Entwickler wird, wie bei WER, auf die Vertrauenswürdigkeit der die Berichte lesenden Personen gesetzt. Ein Ubuntu-internes Team überprüft die Berichte laut [CrashReporting] vor deren Veröffentlichung auf sensible Daten: „Those bug reports are now inspected by a trusted Ubuntu developer before marking it public.“
Die Auswertung der Fehlerberichte erfolgt im öffentlichen Ubuntu-Bugtracker. Auf diese Weise werden zwar keine Fehlerberichte zur Laufzeit der Distribution gesammelt, jedoch können zur Entwicklungsphase bereits viele Fehler behoben werden.
Das Analysieren der Berichte ist im Vergleich zum WER deutlich offener gestaltet, wenngleich die Probleme des Datenschutzes auch hier umschifft werden. Mit seiner Bindung an Ubuntu ist es außerdem wenig plattformunabhängig. Aufgrund des Open-Source-Charakters des Projektes ist der Mechanismus aber deutlich einfacher einzusehen. [Pitt, 2007] beschreibt beispielsweise den Aufbau des Dateiformats für Fehlerberichte. Dieser ist allgemein für alle Distributionen anwendbar und verständlicherweise besser an Linux angepasst als das Pendant von Microsoft.

### CrashReporter

In einer Apple Technote auf [CrashReporter] heißt es zu dem zu WER und Apport analogen Projekt von Apple: „[...] currently no way for third party developers to access [...]“. Auch über den internen Aufbau der Software oder die serverseitige Infrastruktur konnten keine Informationen gefunden werden.

#### Breakpad

Bei Breakpad handelt es sich um eine Softwarekomponente, die als Open-Source-Projekt von Mozilla und Google entwickelt wird. Breakpad muss in ein Programm eingebunden und von diesem gesteuert werden. Es kann Berichte auf Basis von Logdateien erstellen, aber auch auf die Bordmittel des Betriebssystems zurückgreifen (unterstützt werden Windows, Linux und Mac OS X) und somit auch im Falle eines Programmabsturzes Dumps zur Verfügung stellen. Die Fehlerberichte werden an die Serversoftware Socorro gesendet. Verwendung findet das System laut [Breakpad] unter anderem in Firefox ab Version 3.
Es ist in der hier vorgestellten Liste die erste plattformübergreifende Lösung und wird daher erwähnt. Da die Komponente jedoch in jede einzelne Software integriert werden muss, ist sie für diese Arbeit ungeeignet.

#### Mobile Endgeräte

WER ist auch in Windows Mobile ab Version 5.0 respektive Windows Phone 7 integriert. Für Android existiert mit [ACRA] eine Bibliothek mit der Fehlerberichte von einem Programm dessen Entwickler zugänglich gemacht werden können. Ähnlich wie Breakpad muss es in das jeweilige Programm integriert werden. Neben dem Ablegen der Berichte in GoogleDocs oder dem Google-Developer-Account stehen auch selbstdefinierte Aktionen zur Verfügung, wie etwa das Versenden per Mail oder der Upload auf einen Server.
Unter iOS werden Berichte auf dem Gerät selbst gespeichert (vgl. [iOS]). Durch eine Synchronisation mit iTunes landen diese gegebenenfalls auf dem Computer des Besitzers – sind also nur nützlich, wenn dieser auch der Entwickler ist. Ein Verteilungsmechanismus existiert nicht.

### 2.2.3 Datenbanksysteme

Eines der gesetzten Ziele dieser Arbeit ist, die entstehende Menge an Fehlerberichten auf die teilnehmenden Computer zu verteilen. Es kommen zwei mögliche Ansätze in Frage:

- Datenbanken ohne inhärente Verteilungs- und Replikationsmechanismen, oder
- Datenbanken mit inhärenten Verteilungs- und Replikationsmechanismen.

Eine Verwendung von ersteren würde die Implementierung eines Mechanismus zur Verteilung und zur Redundanz der gespeicherten Daten im Rahmen dieser Arbeit erfordern.

Produkte aus dem Segment der relationalen Datenbanken gehören im Allgemeinen der ersten Kategorie an. Wie im vorherigen Abschnitt festgestellt, sind sie nicht auf Anwendung in verteilten Netzwerken ausgelegt. Als prominente Vertreter wären an dieser Stelle [MySQL] und [PostgreSQL] zu nennen.

Zwei der populärsten Vertreter der NoSQL-Datenbanken sind die dokumentorientierten [CouchDB] und [MongoDB]. Sie replizieren ihre Daten nur im Ganzen auf alle Knoten, teilen die Datenmenge jedoch nicht auf. Das Konzept gleicht Sicherungkopien, es kann aber auch die Lese-Last aufgeteilt werden. Beide würden einen separaten Mechanismus zur Aufteilung der aufkommenden Datenmenge unter den Teilnehmern benötigen.
[Project Voldemort] ist eine NoSQL-Datenbank aus dem Bereich der Key-Value-Stores. Sie verfügt über Replikations- und Verteilungsmechanismen, bietet eine Grid-Infrastruktur ohne single point of failure und kann dynamisch erweitert oder verkleinert werden. Project Voldemort folgt erwartungsgemäß dem Modell der eventual consistency und erfüllt A- und P-Eigenschaften. Als Java-Anwendung ist das Programm praktisch plattformunabhängig und von diversen Programmiersprachen über eine API ansprechbar.
Als letzter Kandidat sei an dieser Stelle das Apache-Projekt [Cassandra] vorgestellt. Im Bezug auf die verwendete Datenstrukturierung bewegt sich die Datenbank zwischen Dokumentorientierung und Key-Value-Store. Es werden zwar Schlüssel-Wert-Paare gespeichert, diese lassen sich jedoch in eine vier- oder fünfdimensionale Struktur einordnen und sind vergleichbar mit dimensionsbeschränkten dokumentenorientierten Datenmodellen. Bezeichnet wird das Modell von den Herstellern als *Structured-Key-Value-Store*.
Cassandra verfügt ebenfalls über die Möglichkeit, zur Laufzeit Knoten hinzuzufügen oder zu entfernen, liefert eine Infrastruktur ohne SPOF und skaliert linear. Ebenso verfügt sie über inhärente Verteilungs- und Replikationsmechanismen, ist in Java programmiert und hat Bibliotheken zum Ansprechen der API für diverse Programmiersprachen. Mit CQL verfügt Cassandra außerdem über eine an SQL angelehnte Abfragesprache, welche aber beispielsweise über keinen JOIN-Operator verfügt. Der Algorithmus zur Kommunikation der einzelnen Instanzen im Grid basiert auf dem im folgenden Abschnitt vorgestellten Gossip-Protokoll.

### 2.2.4 Netzwerktopologien

Im klassischen Client-Server-Modell trägt ein zentraler Server die komplette Last der zu erledigenden Aufgaben im Netzwerk. Einerseits skaliert diese Architektur nicht gut horizontal, andererseits ist das Modell nicht ausfallsicher. Es existieren viele Wege, diese Mängel zu lindern. Beispielsweise schafft man sowohl Redundanzen als auch Lastverteilung durch die Bildung von Clustern.

Seit geraumer Zeit existieren Peer-to-Peer-Modelle (P2P) mit einem grundlegend neuen Design. In einem solchen Netzwerk kommunizieren mehr oder weniger gleichberechtigte Instanzen untereinander und jede erledigt einen Teil der Aufgaben. Der Ausfall einzelner Instanzen hat somit nicht den Ausfall der gesamten Infrastruktur zur Folge und nur ein Teil der Aufgaben kann nicht mehr erfüllt werden.
Das Modell wird ebenfalls genutzt, um Server-artige Dienste zur Verfügung zu stellen. Clients können sich dabei zu einer beliebigen Instanz einer Menge angebotener Computer verbinden. Bei steigender Last können weitere Instanzen zum P2P-Netzwerk hinzugefügt

werden. Auf diese Weise können Dienste von einem zentralen Anbieter zur Verfügung gestellt werden, ohne Einbußen bei der horizontalen Skalierbarkeit tragen zu müssen.

Nach [Allavena et al., 2005] Seite 293 ist ein oft übersehenes Problem bei P2P-Netzwerken, dass auch heartbeat-Signale (regelmäßige „Ich bin noch hier“-Mitteilungen an andere Teilnehmer oder „Bist du noch hier“-Abfragen von anderen Teilnemern) ein Problem bei der Skalierbarkeit sind. Die stete Kommunikation aller Knoten miteinander erzeugt ein nicht zu vernachlässigendes Datenaufkommen im Netzwerk.
Das in [Allavena et al., 2005] als funktionierend nachgewiesene „Gossip Based Membership Protocol“ – in dieser Arbeit kurz als Gossip bezeichnet – will diesen Umstand umgehen. Es verbinden sich jeweils nur kleine Untermengen von Knoten aus dem Grid. Diese

- tauschen Informationen über die Erreichbarkeit anderer Knoten aus und
- bringen sich untereinander auf den neusten Stand.

Die Erreichbarkeitsinformationen, die ein Knoten dabei angibt, ist die Menge der Knoten, die dieser beim letzten Gossip-Treffen erreicht hat. Aus der Gesamtmenge der von allen Gossip-Partnern als erreichbar angegebenen Teilnehmer wählt ein Knoten später zufällig die Menge seiner nächsten Verbindungspartner. Bei jeder Verbindung werden außerdem alle aktualisierten Inforationen des verteilten Netzwerks ausgetauscht – im Falle von Datenbanken etwa die Menge veränderter Datensätze.
Mittels dieses probabilistischen Ansatzes breiten sich neue Informationen viral im Netzwerk aus – vergleichbar mit dem Tratsch (engl. ugs.: Gossip) unter Menschen. Außerdem wird die Kommunikationslast im Netzwerk deutlich reduziert, ohne die Verbreitung neuer Informationen zu behindern – es müssen sich nicht alle Menschen auf einmal treffen, nur wiederholt kleine Gruppen.

Als prominenten Vertreter benutzt die im vorigen Abschnitt vorgestellte Cassandra-Datenbank das Gossip-Protokoll. So reicht es einer Instanz der Datenbank bereits aus, initial nur die Adresse einer einzelnen weiteren Instanz für ein erstes Gossip-Treffen zu kennen.

### 2.2.5 Ausprägungen von Crowdsourcing

Crowdsourcing kann vielerlei Ausprägungen haben. Nach dem Beispiel in der Einleitung von [Surowiecki, 2004] auf Seiten XI bis XIII etwa ließ man Menschen bei einem Wettbewerb auf einer Messe im Jahre 1906 das Gewicht eines Ochsen schätzen. Hierzu gaben knapp 800 Teilnehmer Schätzungen ab, die auf Loskarten notiert waren. Das eigentliche Ziel des Wettbewerbs war ein Preisgeld für die Person, die dem Gewicht am nächsten kam. Vom Ergebnis des Wettbewerbs abgesehen verfehlte der Durchschnitt der abgegebenen Schätzungen das Gewicht des Ochsen nur um ein britisches Pfund.

Heutzutage existiert mit dem Internet ein Medium, das wie geschaffen für die Aufgabe des Crowdsourcing ist, da es mit vergleichsweise geringem Aufwand eine riesige Anzahl an Menschen erreicht.
Ein Beispiel für Crowdsourcing ist in einem CAPTCHA-Dienst zu finden. CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) ist ein Verfahren, welches von vielen Webseiten eingesetzt wird um sicherzustellen, dass ein Mensch mit der Seite interagiert und kein Computerprogramm. Anwendung findet dies unter Anderem bei der Erstellung von Accounts bei E-Mail-Anbietern oder in Foren. Zu diesem Zwecke wird eine Aufgabe gestellt, die für einen Computer idealerweise unlösbar ist. Klassischerweise handelt es sich dabei um einen verzerrten Text, der nicht von einer Texterkennungssoftware erkannt werden können sollte.

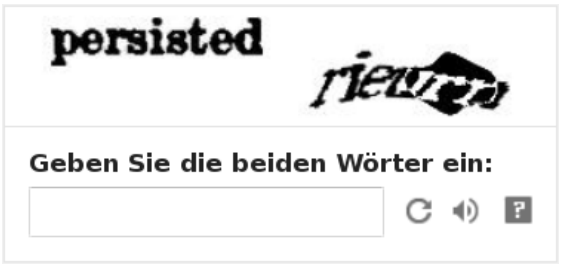


Abbildung 2.1: reCAPTCHA Beispiel

Google digitalisiert in seinem Projekt „Google Books“ in großem Umfang gedruckte Medien – 15 Millionen laut dem Blog-Eintrag eines Google-Mitarbeiters auf [Google Books] zwischen 2004 und Oktober 2010. Dabei passiert es immer wieder, dass Textstellen nicht von der eingesetzten Texterkennungssoftware erkannt werden können. Hier kommt der Dienst reCAPTCHA ins Spiel, der mittlerweile ebenfalls zu Google gehört. Dem Nutzer bietet er augenscheinlich lediglich die Funktionalität eines normalen CAPTCHAs. CAPTCHAs von reCAPTCHA zeigen statt computergenerierten verzerrten Texten jedoch nicht erkannte digitalisierte Textstellen an. Ein Beispiel ist auf Abbildung 2.1 zu sehen. Laut [reCAPTCHA] wird ein dem System bereits bekanntes Wort gezeigt, welches die ursprüngliche Funktion des CAPTCHAs erfüllt, und ein neues. Wird das bekannte Wort korrekt eingegeben ist das CAPTCHA gelöst und das System geht davon aus, dass die Gegenseite ein Mensch ist. Nun wird die Eingabe für das neue Wort in einer Kandidatenliste aufgenommen. Mittels hinreichend vieler Angaben für das neue Wort kann dann ein Kandidat nach Mehrheit aus der Liste ausgewählt werden. Dieses kann nun als ein bekanntes Wort verwendet werden.
Auf diese Weise wird mit dem CAPTCHA-Verfahren nach dem Konzept des Crowdsourcing eine große Masse von Menschen zur Texterkennung genutzt. Obendrein erfüllt das System die eigentliche Aufgabe eines CAPTCHAs sehr gut, da eine Maschine bereits an der Erkennung gescheitert ist.

# 3 Entwurf

## 3.1 Drei Grundsäulen

Der folgende Abschnitt stellt die drei Grundsäulen dieser Arbeit vor. Crowdsourcing als zentrales Werkzeug der Fehleranalyse, dessen Potential es zu nutzen gilt. Plattformunabhängigkeit, die eine breite Basis sowohl an sammelnden, speichernden als auch analysierenden Teilnehmern ermöglichen soll. Und Datenschutz als zu erfüllendes Entwurfsziel, welches mit dem Konzept des Crowdsourcing schlecht vereinbar scheint.

### 3.1.1 Crowdsourcing

Mit dem Internet existiert die ideale Basis für effizientes Crowdsourcing. Ein formeller Nachweis für diese intuitive Annahme ist kaum im Rahmen dieser Arbeit durchführbar. Für einen empirischen Nachweis wäre ein sehr weitreichend angelegter Test von Nöten. Dennoch sind die Annahmen fundiert durch den Erfolg von Projekten wie [Wikipedia] – mit im Mai 2012 allein über 1,3 Millionen deutschen Artikeln – oder reCAPTCHA – mit laut [Google Books] 15 Millionen digitalisierten Büchern in sechs Jahren.

Warum oder wie Crowdsourcing funktioniert, ist Teilgebiet der Sozialwissenschaften und eine ausführliche Einführung in die Thematik übersteigt den Rahmen dieser Arbeit. Jedoch sollen in diesem Abschnitt die von [Surowiecki, 2004] auf den Seiten XVII bis XVIII hervorgehobenen drei Probleme und deren Bezug auf diese Arbeit betrachtet werden:

**„Cognition Problems“** Eine Antwort auf oder Erkenntnisse über eine Frage finden.

**„Coordination Problems“** Die Selbstorganisation einer Gruppe.

**„Cooperation Problems“** Eine Menge an Individuen zur Zusammenarbeit bewegen.

#### Kognitionsprobleme

[Surowiecki, 2004] schreibt, dass die kognitive Vielfalt der Individuen innerhalb der Gruppe deutlich wichtiger für das Finden einer Antwort oder das Erlangen einer Erkenntnis ist, als deren Intelligenz allein. Auf Seite 31 schreibt er, dass einer Gruppe von Leuten verschiedenen Grades von Wissen und Einsicht bei wichtigen Entscheidungen

eher zu trauen sei, als ein oder zwei Personen, egal wie schlau diese seien. Dies bedeute aber nicht, dass die Intelligenz der einzelnen Personen innerhalb der Gruppe vollkommen irrelevant sei.
Eine Forderung an die datensammelnden Teilnehmer stellt das Kognitionsproblem zunächst nicht. Die Teilnehmer würden bei einer hinreichend breiten Nutzerbasis eine repräsentative Menge Fehlerberichte liefern, leisten im Modell aber bisher keine kognitive Arbeit. Die Anforderungen an die Daten speichernden Teilnehmer sind ebenfalls vorwiegend quantitativer Natur. Relevanz für diese Arbeit hat das Kognitionsproblem in erster Linie auf die Zusammensetzung der die Berichte analysierenden Gruppe von Teilnehmern. In zweiter Linie ist die Zusammensetzung jener Gruppe betroffen, welche die von den Analysten geäußerten Lösungsvorschläge anwendet und bewertet – eine Untergruppe der datensammelnden Teilnehmer. Für die beiden betroffenen Gruppen wird die Diversität der Internetnutzer als Grundlage für die geforderte kognitive Vielfalt genutzt. Diese Diversität folgt aus der Repräsentativität, die im Abschnitt zur Forschungsfrage definiert wurde.
Die Fachkompetenz der Analysen ist dabei allein durch die Hürde des Lesens der Fehlerberichte gegeben – ein Laie wird etwa kaum wissen, was mit einem Speicherabbild zu tun ist. Die Kompetenz bei der Bewertungen eines Lösungsvorschlags stützt sich auf die Einfachheit der Aufgabe, festzustellen ob ein gegebener Lösungsvorschlag das Problem beseitigt.

Nach der in [Surowiecki, 2004] diskutierten Theorie des Crowdsourcings ist die zusammengefasste Aussage einer Gruppe im Allgemeinen korrekt – wie etwa bei dem im Kapitel Grundlagen erwähnten Beispiel zum Gewicht eines Ochsen. Einige Individuen hingegen liegen mit ihren einzelnen Aussagen aber mitunter sehr falsch. Auf Seite 43 wird hierzu beschrieben, wie falsches Wissen sukzessive durch die Welt kaskadieren kann: Starrt eine Gruppe Menschen mitten in der Innenstadt gebannt in den Himmel ist die Wahrscheinlichkeit groß, dass vorbeigehende Passanten ihren Blick ebenfalls gen Himmel richten. Je größer die Gruppe, desto mehr Passanten lassen sich beeinflussen (empirisch nachgewiesen laut Surowiecki im Jahre 1968 von S. Milgram, L. Bickman und L. Berkowitz). Der soziologische Begriff des „sozialen Beweises“ – ein einzelner Mensch vertraut in die Weisheit mehrerer anderer Menschen. Diese Grundfeste menschlichen Instinktes ist nicht grundsätzlich schlecht, zeigt aber das Risiko bei der Nutzung des Crowdsourcings. Und häufig genug, so zeigt die Erfahrung, ist das Verhältnis der Qualität einer Aussage eher umgekehrt proportional zur Lautstärke ihres Sprechers.
Eine einfache Lösung gibt [Surowiecki, 2004] auf Seite 64: Die Antworten werden simultan gegeben. Im Normalfall sind einige Menschen zögerlicher und können daher durch vorhergehende Antworten beeinflusst werden. Die Idee hinter dem Konzept der Gleichzeitigkeit ist also, dass die Teilnehmer ihre Antworten unabhängig voneinander geben. Außerdem ist das Verfahren schneller, denn das Abwarten einzelner Teilnehmer wird unterbunden.
Im Bezug auf diese Arbeit hat dieses Problem jedoch wenig Relevanz. Eine Aggregierung der Lösungsvorschläge, die zu einem gemeinsamen Konsens führen soll, findet nicht statt. Überdies können die Lösungsvorschläge von Analysten ohnehin als unabhängig voneinan-

der angesehen werden. Es gibt für Analysten weder einen Grund, einen funktionierenden Vorschlag erneut zu äußern, noch einen Grund, sich an einem nicht funktionierenden Lösungsvorschlag zu orientieren. Der existierende Vorschlag eines Analysten kann einem Anderen höchstens einen Tipp geben, wo dieser seine Suche beginnt, beziehungsweise wo nicht. Des Weiteren gibt es für einen Computernutzer keinen Grund, für einen nicht hilfreichen Vorschlag zu stimmen, egal wie viele andere für ihn gestimmt haben.

### Koordinierungsprobleme

Die Zielgruppe dieser Arbeit – Teilnehmer aus aller Welt, die über das Internet verbunden sind – liefert einen hohen Grad an Dezentralisierung. Nach [Surowiecki, 2004] Seite 71 garantiert dieser Umstand zwar unabhängige und spezialisierte Beiträge, gibt aber keine Garantie, dass eine wertvolle Information aus einem Teil des Systems den Weg in die anderen Teile findet – sprich eine gefundene Fehlerursache ihren Weg in einen Patch oder ein Update findet. Schlimmstenfalls würde Arbeit mehrfach getan oder ungenutzt bleiben, was gerade durch diese Arbeit vermieden werden soll. Auf Seite 74f stellt Surowiecki fest, dass alle erbrachte Schwarmintelligenz ungenutzt bleibt, solange es kein Mittel gibt, die gewonnenen Erkenntnisse zu sammeln. Er stellt daraufhin die Frage, ob eine bündelnde Instanz dem Konzept der Dezentralisierung zuwider stehe – ob bündeln also mit zentralisieren gleichzusetzen sei. Er verneint diese Frage und weist auf die Notwenigkeit eines Mechanismus hin, mit welchem eine kollektiv gefundene Lösung umgesetzt werden kann. Als Referenz verweist er auf Linux. Programmierer aus aller Welt tragen ihre Verbesserungen zu diesem Projekt bei. Die beigetragenen Verbesserungen werden ihrerseits dann gesammelt und diskutiert. Die schlussendliche Ausführung der Veränderung obliegt jedoch seit jeher Linus Torvalds selbst.

Dies zeichnet einen sehr genauen Rahmen von einer der Aufgaben dieser Arbeit: Es soll allen Teilnehmern ein Instrument an die Hand gegeben werden, mit dem Daten gesammelt, ausgewertet und die Ergebnisse der Auswertung diskutiert werden können, mit dem es den Teilnehmern möglich ist, sich zu koordinieren. Die Ergebnisse dieser Diskussionen umzusetzen obliegt jedoch den Herstellern der jeweiligen Software. Dies kommt darüber hinaus der Tatsache entgegen, dass auch nicht-freie Software durch diese Arbeit abgedeckt werden soll. Da der Quellcode nicht einsehbar ist, kann nur der Hersteller ihn ändern.
Dies hindert die Teilnehmer natürlich nicht, im Rahmen ihrer Lösungsvorschläge selbst Workarounds zur Verfügung zu stellen, aber an der vom Hersteller herausgegeben Software ändert sich dabei nichts.

### Kooperationsprobleme

Kooperationsproblemen wohnt laut [Surowiecki, 2004] Seite XVIII die Aufgabe inne, selbstsüchtige, misstrauische Menschen zur Zusammenarbeit zu motivieren, die auf den

ersten Blick keinen Grund dazu haben. Bei globalen Problemen wie etwa Umweltverschmutzung oder der von ihm auf Seite 85 beschriebenen Bewegung in einer Großstadt ist diese Motivation leicht erkennbar. Im Gegensatz zu den vorhergehenden zwei Problemen, deren (theoretische) Lösungen vergleichsweise simpel und allgemeingültig sind, sind diese Probleme laut Surowiecki mannigfaltig und komplex und deren Lösungen fallspezialisierter Natur. Auf Seite 92f beschreibt er lange erprobte Lösungen für einige dieser Probleme, beispielsweise durch Gesetzesvorgaben oder gesellschaftliche Konventionen und Normen die Menschen dazu bewegen, zu kooperien. Er nennt etwa das Anstehen in einer Schlange, das Rechtsfahrgebot oder eine zufällige kurze Pause, wenn zwei Personen gleichzeitig einen Satz beginnen.

Es ergibt sich die Frage: Mit welcher Motivation sollten Menschen im Rahmen dieser Arbeit kooperieren? Während etwa bei Problemen der Umweltverschmutzung eine Kooperation lebensnotwendig ist, ist die Motivation der Masse für die Teilnahme an dieser Arbeit noch undeutlich – ungeachtet der Tatsache, dass auch der Umweltschutz unübersehbar mit Kooperationsproblemen zu kämpfen hat. Zur Klärung werden die potentiellen Nutzer in drei Gruppen aufgeteilt:

- Firmen
- Endanwender mit altruistischen Zügen
- Endanwender ohne altruistische Züge

Firmen dürfte es am wenigstens an einer Motivation mangeln, am vorgestellten Netzwerk teilzunehmen. Eine Firma möchte (idealerweise) ihre eigenen Produkte verbessern und hat in Annahme ohnehin Arbeitskräfte, welche sich dieser Aufgabe annehmen. Diese müssten lediglich von ihren bisherigen Methoden der Datenerhebung und -aggregierung auf das hier entworfene System umstellen. Dieser Umstieg wird motiviert, wenn das neue System Vorteile birgt. Die möglichen Vorteile wiederum sind im Abschnitt zur Motivation in der Einleitung dargelegt. Für die Speicherung von Daten wäre eine Firma in der Lage ihre Server und die Arbeitsplatzrechner ihrer Mitarbeiter zur Verfügung zu stellen. Auf den Rechnern ihrer Mitarbeiter könnte des Weiteren die Sammel-Software installiert sein.
Die Geschichten von Projekten wie Linux, Wikipedia oder OpenStreetMap zeigen, dass eine nicht zu vernachlässigende Menge an Menschen existiert, die aus eigenem Antrieb einem System wie dem gezeichneten ihre Ressourcen je nach Möglichkeit und Wissen freiwillig zur Verfügung stellt. Die Annahme, dass Freiwillige einen Beitrag auch jenseits des alleinigen Sammelns von Fehlerberichten leisten, ist daher nicht unbegründet.
Die für das Sammeln von Fehlerberichten wichtigste, aber wohl auch am schwersten zu motivierende Gruppe sind „normale“ Endanwender. Der von klassischen Systemen wie WER oder Breakpad beschrittene Weg ist, die Hürde für das Senden eines Fehlerberichtes so gering wie möglich zu halten. Der Bericht wird automatisch erstellt und der Anwender nur noch nach der Sendeerlaubnis gefragt. Alleine die Existenz der Projekte

zeigt, dass eine hinreichend große Menge an Berichten eintrifft, um die Umsetzung solcher Systeme zu rechtfertigen. Trotz allem werden die Mitglieder dieser Gruppe kaum ihren Festplattenspeicher zur Verfügung stellen, noch werden die technisch versierten unter ihnen Fehlerberichte auswerten. Ein denkbares Modell wäre eine Form der Vergütung für eine bestimmte Zahl eingesendeter Berichte, einer bestimmten Menge zur Verfügung gestellten Speichers oder für eingebrachte Lösungen ähnlich der im Nutzungsszenario vorgestellten „Kopfgelder“.

### 3.1.2 Plattformunabhängigkeit

Die grundlegende Idee, das im Rahmen der Arbeit erstellte System unter allen Betriebssystemen anbieten zu können, erscheint die am wenigsten problematische. Lediglich einiges an Fleißarbeit müsste geleistet werden, um das Konzept der Arbeit auf unterschiedliche Plattformen zu portieren. Zu beachten ist lediglich, dass keine Fähigkeiten gefordert werden dürfen, die nicht von allen Plattformen gewährleistet werden können.

Da die Systeme zum Erkennen von Abstürzen und zum Sammeln von Programminformationen sehr plattformnah sind, ist es nicht möglich, eine allgemeingültige Sammel-Software in einer betriebssystemunabhängigen Sprache zu schreiben, welche mit diesen interagiert. Wenn die im Rahmen der Arbeit erstellte Sammel-Software jedoch hinreichend modular und zumindest in weiten Teilen mit plattformunabhängigen Technologien entwickelt ist, müssen nur sehr kleine Teile angepasst werden, um sie auf einer anderen Plattform lauffähig zu machen.
Trotz allem muss die Sammel-Software als plattformspezifisch angesehen werden. Eine Grundvoraussetzung um Interoperabilität zwischen den Knoten zu sichern, ist daher die Verwendung von einheitlichen (und idealerweise offenen) Protokollen und Formaten.

Ein tatsächliches Problem stellen jedoch dynamische Umgebungen dar, also Laufzeitumgebungen für Bytecodes oder Interpreter für Skripte. Nach [Pacheco, 2011] Seite 3 sind Systeme zum sammeln von PEDIs gut entwickelt und weit verbreitet, wenn es sich um Betriebssysteme und deren native Ausführungsumgebungen handelt. Er begründet dies damit, dass es sich bei diesen Anwendungen um die Kern-Infrastruktur handelt, deren Ausfall kostenintensiv ist und damit früh erforscht wurde. Außerdem existiert diese Art der Programme schon deutlich länger.
Erfährt ein Programm in einer dynamischen Umgebung einen Fehler, bekommt das zugrundeliegende Betriebssystem davon in der Regel nichts mit. Tritt etwa in einem Java-Programm eine Exception auf, so fängt die JVM diese ab und beendet sich infolgedessen sauber. Python und Ruby im Vergleich besitzen die Möglichkeit, das laufende Programm zu pausieren und eine interaktive Interpreter-Konsole zu öffnen (vergleich ebenda Seite 9), was weder im Produktiveinsatz praktikabel, noch im Rahmen der in dieser Arbeit betrachteten entfernten Sammlung von Fehlerberichten nützlich ist.
[Pacheco, 2011] führt auf Seite 8 außerdem an, dass selbst, wenn das Betriebssystem einen Fehler erkennen könnte und ein Speicherabbild erstellen würde, das resultierende Abbild nicht zwingend von Nutzen wäre: Threadstacks, lokale Variablen und Objekte

innerhalb einer JVM sind nicht (zwangsweise) Threads, Variablen und allokierter roher Speicher, welche von der JVM gegenüber dem Betriebssystem genutzt werden. Die JVM erweitert oder ersetzt Strukturen mit Java-Abstraktionen. Analoges gilt für andere dynamische Umgebungen. Für diese müssen daher Dumps auf deren Ebene erstellt werden. Wie in Betriebssystemen müssen also auch in Interpretern und Laufzeitumgebungen Automatismen zur Erkennung von Fehlern und zur Erstellung von Berichten eingebaut werden. Als Positivbeispiel sei an dieser Stelle die von Pacheco erwähnte Java HotSpot VM genannt, die in der Java-Distribution von Oracle verwendet wird und Java-Dumps erstellen kann. Außerdem die Common Language Runtime (CLR), welche die Laufzeitumgebung für .NET-Programme unter Windows ist und sich in das Windows Error Reporting integriert.

Da die Modifikation von dynamischen Umgebungen hier nicht in Frage kommt, fällt die Betrachtung entsprechender Programme vorerst aus dem Rahmen dieser Arbeit. Außerdem macht der unterschiedliche Aufbau der einzelnen Umgebungen eine allgemeine Lösung des Problems gleichermaßen unmöglich, wie dies für native Programme unterschiedlicher Betriebssysteme selbst der Fall ist. Das Problem ist jedoch bekannt und wird durch die steigende Zahl an Programmen für diese Umgebungen an Aktualität gewinnen. Ist eine entsprechende Lösung für ein einzelnes Produkt aber erst einmal gefunden, ließe es sich bei entsprechender Einhaltung der Protokolle und Formate leicht in die Ergebnisse dieser Arbeit integrieren.

### 3.1.3 Datenschutz

Bereits auf den ersten Blick steht das Konzept dieser Arbeit – Fehlerberichte, welche sensible Daten enthalten können, zur freien Einsicht bereitzustellen – im Widerspruch zur informationellen Selbstbestimmung und zum Schutz privater Informationen.
Auch bei Systemen wie WER, Apport oder dem CrashReporter gelangen potentiell sensible Daten in fremde Hände. Doch zum Einen sollten die Mitarbeiter der Hersteller einer Verschwiegenheitsverpflichtung unterliegen, zum Anderen verlassen diese Berichte nicht (vgl. [CrashReporter]) oder nur unter sehr restriktiven Bedingungen (vgl. [Winqual]) das Haus des Herstellers.
Die Frage nach dem Datenschutz bei der Weitergabe von Fehlerinformationen ist bekannt, die breite Zugänglichkeit der Informationen setzt die Frage im Hinblick auf diese Arbeit jedoch in eine zentrale Position. Das Problem hierbei ist, dass es für einen Computer nicht möglich ist, sensible Daten selbst zu erkennen und zu filtern.

[Broadwell et al., 2003] beschreibt unter dem Titel „Scrash“ zu dieser Problematik eine Methode, mittels derer ein automatisches Filtern von Memory Dumps in zwei Schritten ermöglicht wird (Seite 276ff):

1. Modifizieren des Programms mittels neu eingeführter Befehle (wie etwa smalloc = secure malloc) derart, dass sensible Daten in einem gekennzeichneten Bereich

des Arbeitsspeichers abgelegt werden, statt mit nicht-sensiblen Daten gemischt im normalen Stack oder Heap.

2. Automatisiertes Herausfiltern der gekennzeichneten Bereiche aus dem Dump vor dessen Versenden.

Das System bietet trivialerweise die gewünschte Sicherheit. Den Programmierern obläge die Aufgabe einzuschätzen, welche Eingaben sensibel sein könnten und somit gekennzeichnet werden müssten. Des Weiteren müssten alle existierenden Compiler derart erweitert werden, dass sie die neuen Befehle verstehen. Als Alternative zur Modifikation der Compiler zeigt [Broadwell et al., 2003] auf Seite 275 einen Präprozessor für C-Programme, welcher in 1.200 Zeilen OCaml realisiert ist. Dieser übersetzt erweiterten C-Quellcode, welcher die neu eingeführten Befehle enthält, in normalen C-Quellcode, welcher mit üblichen Compilern übersetzt werden kann.
Für die Zwecke dieser Arbeit ist dies aber verständlicherweise ein nicht zu realisierender Aufwand. Das zu entwickelnde System sollte ohne weitreichende Eingriffe in existierende Plattformen eingefügt werden können. Außerdem müsste man Steve Ballmer, Linus Torvalds und Tim Cook an einen runden Tisch bringen um die in dieser Arbeit angestrebte Plattformunabhängigkeit der von Broadwell gezeichneten Methode zu erreichen.

Eine automatische, dafür aber eher heuristische Lösung ist der so genannte „Minidump“, der beispielsweise im Windows Error Reporting verwendet wird. Der normale Minidump enthält dabei (in der Variante von Microsoft) nur den Stacktrace. Er kann aber so konfiguriert werden, dass er weitere definierte Adressbereiche, wie etwa das Code-Segment, enthält. Eine vollständige Liste aller hinzufügbaren Segmente ist unter [MiniDump] zu finden.
Bereits der Stacktrace liefert oft schon die nötigen Hinweise zum Debuggen. Verständlicherweise liefert diese Lösung aber keine absolute Sicherheit, denn mit Hinzukommen weiterer Segmente zum Minidump steigt neben der Nützlichkeit der Informationen die Wahrscheinlichkeit, sensible Daten aufzunehmen. Nach [Kinshumann et al., 2011] Seite 112 wird, wenn über das WER ein Dump vom Nutzer angefordert wird, zuerst ein Minidump übertragen – nur der Stacktrace, ohne weitere Segmente. Erst wenn dieser nicht ausreicht, sammelt das System angepasste oder volle Dumps. Solche Dumps wiederum sammeln nur Endanwendersysteme, die explizit darauf konfiguriert wurden.

Selbst unter Annahme eines perfekten Schutzes sensibler Daten in Fehlerberichten muss dem Benutzer stets die Möglichkeit gegeben werden darüber zu entscheiden, welche seiner Daten gesendet werden. Um informationelle Selbstbestimmung zu gewährleisten, haben die Hersteller daher nicht grundlos den Weg gewählt, vor jedem Senden von Daten nach dem Einverständnis des Anwenders zu fragen.
Für die Interaktion mit dem Benutzer sollen im Rahmen dieser Arbeit folgende Regeln gelten:

1. In den Standardeinstellungen wird nur ein Minimum an Daten gesendet.

2. In den Standardeinstellungen wird der Benutzer vor jedem Senden von Informationen um Erlaubnis gefragt.
3. Es kann gewählt werden, welche Daten über das Minimum hinaus gesendet werden dürfen und welche nicht. Dabei ist der Benutzer zu warnen, dass diese Daten sensible Informationen enthalten können.
4. Beim Senden von Daten über das Minimum hinaus muss separat um Zustimmung gefragt werden.
5. Es soll einstellbar sein, dass bestimmte oder alle Informationen automatisch ohne Nachfrage gesendet werden dürfen.
6. Das Senden jedweder Informationen muss komplett deaktivierbar sein.

Bestimmte sensible Daten wie etwa IP- oder MAC-Adressen von Netzwerkkarten, Seriennummern von Software, et cetera sind leicht erkennbar und sollten nie in einen Bericht eingeschlossen werden. So kann die Anonymität eines Teilnehmers gewährleistet werden.

## 3.2 Architektur

Nun soll unter Berücksichtigung der drei Grundsäulen ein Netzwerk entworfen werden, mit dem Fehlerberichte gesammelt und analysiert werden können. Wie bereits erläutert, wird die physische Verbindung auf Basis des Internets realisiert und der logische Aufbau im Form eines Grids modelliert. Die Kommunikationswege und -muster sollen den Prinzipien sozialer Netzwerke folgen. Die Teilnehmer sind dabei potentiell alle mit dem Internet verbundenen Computer weltweit.

Für jeden am Netzwerk teilnehmenden Sammel-Knoten existiert ein mit dem Profil aus sozialen Netzwerken vergleichbarer Eintrag im System. Dieser enthält Informationen über die Konfiguration des entsprechenden Knotens.
Idealerweise gibt es eine 1:1-Abbildung von Fehlerursachen zu Gruppen. Die Mitglieder einer Gruppe sind die Sammel-Knoten, auf denen ein entsprechender Fehler aufgetreten ist. Tritt ein Fehler auf einem Computer auf, zu dem es noch keine Gruppe gibt, wird diese automatisch erstellt. Existiert eine Gruppe zum aufgetretenen Fehler, tritt der Computer dieser automatisch bei.
Teilnehmer können in einer Gruppe im Folgenden Informationen „posten“ – Speicherabbilder statt Partybilder.
Analyse-Knoten verfügen über eine Software, mit der man Gruppen und Profile durchstöbern kann. Über eine Suchmaske ist es möglich, gezielt nach Gruppen oder Rechnern mit bestimmten Eigenschaften zu suchen. Da von den Analysten technische Versiertheit angenommen werden kann, erlaubt die Suchmaske auch komplexe Anfragen – etwa nach Fehlern im Zusammenhang mit einer bestimmten Grafikkarte beim Darstellen von Videos mit bestimmter Kodierung.
In den Gruppen können Analysten Nachrichten und Lösungsvorschläge veröffentlichen.

So erhalten Gruppen auch die Funktion einer Diskussionsplattform, für die an der Lösung des Problems beteiligten Analysten.
Gespeichert werden die Daten in dieser Arbeit nicht, wie bei sozialen Netzwerken üblich, auf einem zentralen Server, beziehungsweise Servercluster, sondern auf Speicher-Knoten. Dem verteilten Prinzip entsprechend können sich Sammel- wie Analyse-Knoten zu beliebigen unterschiedlichen Speicher-Knoten verbinden und greifen trotzdem auf den gleichen Datenbestand zu – abgesehen von den Defiziten durch *eventual consistency*.

Eine naive Abschätzung der zu tragenden Last eines einzelnen Speicher-Knotens ist dabei die Summe der Last auf den Servern eines *remote crash reporting*-Systems (wie etwa WER) und der eines sozialen Netzwerkes geteilt durch die Menge an teilnehmenden Speicher-Knoten. Je mehr Teilnehmer es gibt, desto geringer ist die Last für den Einzelnen. Je geringer die Last für den Einzelnen ist, desto höher ist die Wahrscheinlichkeit, dass ein potentieller Teilnehmer seine Ressourcen zur Verfügung stellt.

## 3.3 Sammel-Knoten

Das Konzept des remote crash reporting von WER und Apport soll als Vorbild für die Sammel-Knoten dienen. Der Mechanismus, lokal Fehler zu erkennen, einen Bericht zu erstellen, sowie den Bericht an eine nächste Instanz weiterzureichen, trifft genau eines der Ziele dieser Arbeit, Fehlerberichte in großem Ausmaß zu sammeln.
Diese Systeme sammeln jedoch ausschließlich Daten im direkten Zusammenhang mit dem Fehler, während beispielsweise in Bugtrackern auch oft Log-Dateien angefordert werden. Die Integration eines Betriebssystem-inhärenten Logging-Mechanismus könnte die verfügbaren Informationen daher sinnvoll erweitern.

### 3.3.1 Anforderungen an die Software

Der auf den Sammel-Knoten laufende Mechanismus unterliegt einer Reihe an Anforderungen, die eine Erweiterung des entworfenen Netzwerks erfordern.
Eine Zusammenfassung über die Erfahrungen von Microsoft mit dem Windows Error Reporting findet sich in [Kinshumann et al., 2011]. Den Seiten 112f sind folgende „fünf Strategien“ entnommen:

1. „Automatic bucketing“
2. „Progressive data collection“
3. „Minimizing human interaction“
4. „Preserving user privacy“
5. „Providing solutions to users“

Ebenfalls in Betracht gezogen werden die fünf „Auflagen“ für ein entsprechendes System von [Pacheco, 2011] Seite 3:

1. „Application software must not require modifications that cannot be used in production [...]“
2. „The facility must be always on“
3. „The facility must be fully automatic“
4. „[...] the facility must provide enough information to be useful for nontrivial problems“
5. „The dump must be transferable to other systems for analysis“

Die in Strategie 1 erwähnte Einordnung in „Eimer“ (engl.: bucketing) bezeichnet die automatisierte Zusammensortierung von Fehlerberichten. So sollen idealerweise in einem Eimer nur Fehlerberichte landen, welche die gleiche Ursache haben – und dies verständlicherweise ohne dass diese Ursache bereits bekannt ist. Im Text von Kinshumann wird darauf hingewiesen, dass ein solcher Algorithmus natürlich nicht existiert, Microsoft dafür aber Heuristiken entwickelt hat. Auf der Seite der Endbenutzer-Computer kann beim Auftreten eines Fehlers mittels einer solchen Heuristik deterministisch die eindeutige Eimer-ID des zuständigen Eimers aus dem dort verfügbaren Bericht berechnet werden. Weiter hat Microsoft laut [Kinshumann et al., 2011] Seite 113 über 500 Heuristiken entwickelt, um eine weitere serverseitige Sortierung vorzunehmen. Diese Heuristiken sind leider, bis auf sehr wenige, nicht öffentlich zugänglich – die clientseitige Heuristik zur Erstellung der Eimer-ID wird im Abschnitt zum Speicher-Knoten dargelegt.
In dieser Arbeit sind die Gruppen das Pendant zu Eimern im WER und auch Gruppen bekommen eine eindeutige Gruppen-ID. Es muss aber möglich sein, vorgenommene Zuordnungen zu verändern. Wird bei mehreren Gruppen angenommen, dass deren Fehler die gleiche Ursache besitzen, soll es möglich sein, diese zusammenzuschließen. Analog sollte aus Untermengen von Berichten einer Gruppe eine neue Gruppe erzeugt werden können, wenn mehrere Fehlerursachen vermutet werden. Da sich diese Annahmen später als fehlerhaft herausstellen können, dürfen die Originalgruppen dabei nicht verloren gehen. Statt dessen werden in einer Gruppe Verknüpfungen zu Unter- oder Übergruppen angezeigt. Neue Berichte in den Urgruppen werden automatisch in ihren Übergruppen angezeigt. Analog müssen auch Diskussionsbeiträge aus Untergruppen zu deren Urgruppen durchgereicht werden.
Mit Rückverweis auf das Koordinationsproblem aus der Grundsäule Crowdsourcing sei hier vermerkt, dass dieser Mechanismus zusätzlich für Parallelität und Unabhängigkeit sorgt und vor allem einen langwierige Diskussion über das Zusammenlegen oder Spalten einer Gruppe umgeht.

Strategie 2 („Progressive data collection“) fordert, dass Daten sukzessive die bereits existierenden ergänzen können müssen. Das bedeutet, dass zunächst nur das erstmalige Auftreten einer potentiellen Fehlerursache zusammen mit einem minimalen Bericht auf den WER-Servern gespeichert wird – ein Eimer wird erstellt. Im weiteren Verlauf wird

bei jeder Meldung seitens eines Clients nur die Eimer-ID an die WER-Server übertragen, so dass dort lediglich der entsprechende Zähler erhöht wird. Für einen Eimer kann dann im System vermerkt werden, dass mehr Informationen gefordert werden. Sind für einen Eimer Daten gefordert, so werden diese von einem Computer erbeten, wenn bei diesem der Fehler auftritt. Der Bestand der verfügbaren Daten wächst *progressiv*.
Eine einzelne Ursache bewirkt mitunter sehr viele Auftreten eines Fehlers, zum Beheben sind aber im Vergleich nur sehr wenige weiterführende Informationen nötig. So wird schnell klar, dass ein Großteil der übertragenen Informationen lediglich Eimer-IDs sind – die Teilnehmer versenden nicht jedes Mal die maximal mögliche Menge an Informationen aus einem Fehlerbericht. Die in WER eingesetzte Methode verringert so den Netzwerkverkehr auf ein Minimum ohne die Funktionalität einzuschränken.
Das bisherige Konzept wird nach dieser Vorlage derart erweitert, dass ein Computer beim Auftreten eines Fehlers zunächst nur der Gruppe beitritt, welche die entsprechende Fehlerursache repräsentiert. Befasst sich ein Analyst mit dem Fehler, kann dieser in der Gruppe die Bitte nach mehr Daten äußern. Erst dann werden die Mitglieder aufgefordert, weitere Daten zu „posten“. Statt einer Prosa-Nachricht in der Gruppe, die wohl kaum ein Endbenutzer lesen würde, bekommt eine Gruppe lediglich eine Markierung. Die Sammel-Software ist wiederum in der Lage eine entsprechende Markierung zu erkennen und, je nach Konfiguration der Software und Zustimmung des Nutzers, die entsprechenden Informationen zu senden.

Strategie 3, die Minimalisierung der nötigen Benutzerinteraktion impliziert Auflage 3, nach der das System vollautomatisch sein muss. Je höher die Hürden für einen Endanwender sind, einen Beitrag zum System zu leisten, desto unwahrscheinlicher ist seine Teilnahme. Mit dem Vorbild der automatischen Erstellung von Berichten, bei denen der Benutzer lediglich um die Sendeerlaubnis gefragt wird, ist der vorgestellte Ansatz jedoch innerhalb der Anforderungen von Kinshumann und Pacheco.

Um gemäß Strategie 4 die Privatsphäre der Benutzer zu gewährleisten, sollten Analysten so wenige Informationen wie möglich anfordern – etwa erst Minidumps und nur wenn unbedingt nötig angepasste oder volle Dumps. Bei der Implementierung der Sammel-Software ist außerdem zu beachten, die erwähnten erkennbaren sensiblen Daten aus den Berichten zu entfernen – etwa IPs, MACs oder Seriennummern.
Das im Abschnitt zur Grundsäule Datenschutz erwähnte Minimum an zu übertragenden Informationen kann nach den bisherigen Betrachtungen auf die Informationen beschränkt werden, die zum Erstellen einer Gruppe notwendig sind. Existiert eine Gruppe zu einem aufgetretenen Fehler noch nicht, so kann sie vom Betroffenen aus diesen Informationen erstellt werden. Existiert sie bereits, so werden diese Informationen benötigt, um auf dem Sammel-Knoten die Gruppen-ID zu bestimmen und alleine ein Beitritt zu dieser Gruppe offenbart das Vorhandensein der entsprechenden Informationen auf dem beigetretenen Computer.

Zur Erfüllung von Strategie 5 soll es möglich sein, die Lösung für einen gefundenen Fehler an die Betroffenen zu propagieren. Dazu könnten Sammel-Knoten die sie betreffenden Gruppen in regelmäßigen Abständen überwachen und den Nutzer bei einer verfügbaren

Lösung benachrichtigen. Man könnte das System auch um Privat- und Gruppennachrichten erweitern. Bei der Lösung eines Problems könnte das System dann automatisch eine Nachricht an alle Mitglieder der Gruppe senden.

Zu Auflage 1 erwähnt Pacheco unter anderem, dass weder unoptimierter Code, noch Debug-Daten oder sonstige Modifikationen von Nöten sein dürfen, um PEDIs zu sammeln. Diese Modifikationen wären für einen Einsatz auf Produktivsystemen nicht tragbar. Die im Rahmen der Arbeit zu erstellende Software darf dementsprechend keine weiteren Ansprüche an die Plattform oder die ausführbaren Dateien äußern. Lediglich die verfügbaren Bordmittel (Logger, Fehlererkennungssoftware, et cetera) werden als Voraussetzungen angenommen um auf praktisch allen Computern mit dieser Plattform lauffähig zu sein.

Aus Auflage 2, das System müsse stets aktiv sein, ist zu folgern, dass der Ressourcenverbrauch von Sammel- und Speicher-Software möglichst gering zu halten ist. Eine im Hintergrund laufende Anwendung, wie es das Überwachen des Systemablaufs und das crowdsourcingbasierte Speichern von Daten auf Endanwender-Rechnern sind, darf den Nutzer nicht einschränkten. Da ein Analyst aktiv am System teilnimmt, ist es für die Analyse-Software nicht nötig, transparent im Hintergrund zu laufen, ohne dass der Nutzer eine Einschränkung seiner im Vordergrund benutzten Anwendung(en) erfährt.
Die reine Machbarkeit eines Sammel-Knotens ist bereits durch die Existenz von WER gegeben. Eine verteilte Datenbank hingegen, die keine merkliche Systemlast (außer dem verbrauchten Festplattenspeicher) verursacht, ist im angestrebten Ausmaß mit keinem existierenden Projekt zu vergleichen und bedarf genauerer Betrachtung im Rahmen dieser Arbeit.

Die Transferierbarkeit der PEDIs auf andere Systeme, Auflage 5, ergibt sich durch die Orientierung dieser Arbeit an existierenden *remote crash reporting*-Systemen, welche diese Auflage per Definition erfüllen.

Der folgende Unterabschnitt beschäftigt sich mit Auflage 4: Welche Informationen sind zur Behebung einer Fehlerursache nötig und daher auf den Rechnern der Teilnehmer zu sammeln?

### 3.3.2 Anforderungen an die gesammelten Daten

Unter Windows Error Reporting nach [Kinshumann et al., 2011] und Apport nach [Pitt, 2007] werden folgende wichtige Informationen gesammelt, die das Programm selbst betreffen:

- Die Metainformationen des fehlerbehafteten Programms (Name, Version, ...)
- Status der Prozessor-Register, vor allem des Program-Counters
- Programmend- oder Absturzinformationen (Rückgabecode, Typ der Exception, ...)
- Speicherabbild

- Geladene Bibliotheken (bei WER) beziehungsweise Paketabhängigkeiten (bei Apport) und deren Metainformationen
- Zeitpunkt des Auftretens des Fehlers

Da auf dem Analyse-System die entsprechenden Dateien, wie Binärdateien oder Debug-Symbole, nicht vorhanden sein könnten, wäre eine Möglichkeit sinnvoll, diese hinzuzufügen. Hierzu muss jedoch zunächst die rechtliche Lage für das Verteilen derselben betrachtet werden. Eine entsprechende Funktionalität wird daher an dieser Stelle außer Acht gelassen.
Weitere relevante Informationen betreffen die Systemsoftware:

- Log-Nachrichten (vor allem die des fehlerhaften Programms)
- Die Liste der installierten Komponenten des Systems
- Die Liste der Treiber (auch der inaktiven, vgl. [Kinshumann et al., 2011] Seite 111) unter unixoiden Betriebssystemen entsprechend die Kernel-Module

Von Nutzen könnten außerdem sein: Konfigurationsdateien, der Prozessbaum, Last-Informationen (CPU, RAM, HDD, ...), Sensordaten (Temperatur, Stoß, ...) und da bereits ein speziell geformtes Netzwerkpaket ein Betriebssystem zum Abstürzen bringen kann, müsste auch der komplette Netzwerkverkehr mitgeschnitten werden, ...
Prinzipiell sind sämtliche sammelbaren Daten potentiell nützlich. Einige von ihnen sind nachvollziehbarerweise aber nicht mit vertretbarer Belastung des Systems zu erhalten. An dieser Stelle sei darauf hingewiesen, dass ein Kompromiss zwischen Auflage 2 (always on), Strategie 4 (privacy) und Auflage 4 (enough information) gefunden werden muss.

### 3.3.3 On-demand-Loglevel

Einerseits gilt: Je mehr Daten gesammelt werden sollen, desto höher die Systemlast. Andererseits gilt aber auch: Je komplexer ein Fehler ist, desto mehr Daten können notwendig sein um seine Ursache zu erkennen. Aus diesem Grunde ist es sinnvoll, den Umfang der protokollierten Daten wenn nötig dynamisch erhöhen zu können – im Englisch das Loglevel *on demand* zu verändern.
Läuft ein System in normalen Parametern, so werden keinerlei Informationen zur Laufzeit mitgeschnitten. Beim Auftreten eines Fehlers werden lediglich die PEDIs nach Ermessen des Anwenders zusammengetragen und versendet.
Wie beschrieben kann ein Analyst in einer Gruppe die Bitte nach mehr Informationen äußern. Diese Informationen können nicht immer durch PEDIs abgedeckt werden. Tritt nun auf einem Computer in der Gruppe der entsprechende Fehler häufig auf, wird die Menge der gesammelten Daten auf diesem um die geforderten Informationen erweitert. So kann der Computer die erbetenen Daten beim nächsten Auftreten des Fehlers senden. Wird etwa der Absturz einer Sensor-Software bei einer bestimmten Systemtemperatur vermutet, so werden Messdaten der Temperatursensoren in den Fehlerbericht aufgenommen. Wird eine Fehlerursache behoben oder wurden genug Daten gesammelt, werden die

Teilnehmer der jeweiligen Gruppe automatisch wieder auf das geringste Loglevel heruntergestuft – vorausgesetzt natürlich, dies kollidiert nicht mit den Forderungen anderer Gruppen.

Im Unterabschnitt zum Datenschutz wurde eine Einstellungsmöglichkeit für jene Daten gefordert, die über ein Minimum hinausgehend aufgezeichnet werden dürfen. Diese Regel kann nun leicht in ein Modell übertragen werden, im welchem das Loglevel nur bis auf ein einstellbares Maximum erhöht werden darf. Bei einer Veränderung des Loglevels sollte der Benutzer darüber natürlich informiert werden.

## 3.4 Speicher-Knoten

Unter den vorgestellten Datenbanken erfüllt Apache Cassandra einige wichtige geforderte Funktionen für Speicher-Knoten:

- Es existieren inhärente Verteilungs- und Replikationsmechanismen.
- Die Verwendung von Gossip lässt die Datenbank linear skalieren und liefert eine Grid-Struktur ohne single point of failure.
- Eine Verbindung zur Datenbank kann zu jedem beliebigen Knoten geschehen. Dadurch kann die Last durch Lese- und Schreibzugriffe aufgeteilt werden.

Eine Verwendung von Cassandra würde, sofern praktikabel, eine eigenständige Implementierung unnötig machen.

### 3.4.1 Verfügbarkeit der Speicher-Knoten

Bei einem Auslagern des Speicherplatzes auf die Masse ist die Verfügbarkeit der Teilnehmer ein ernstes Problem. Während bisherige Datenbankserver, -cluster und -grids für Verfügbarkeiten über 90% je Rechner ausgelegt sind, ist im vorgestellten Fall eher von 9% Verfügbarkeit je Speicher-Knoten die Rede. Daher ist die Wahl der Anzahl an Replikaten pro Datensatz in der Datenbank für diese Arbeit wichtig. Existieren zu wenige Replikate, fehlen Daten, wenn die zuständigen Computer offline sind. Wird die Zahl zu hoch gewählt, steigt die Netzwerkauslastung beim Verteilen der Replikate und die Festplattennutzung. Die zuerst zu klärende Frage ist also: Was sind die Zielrechner und wie lange sind diese online?

Laut einer Pressemitteilung der BITKOM zu deren Studie, „surfen [die Deutschen] im Schnitt 135 Minuten pro Tag“, wobei nur gut 70% der Befragten überhaupt gelegentlich online sind. Die Onlinezeiten verteilen sich laut [BITKOM, 2010] wie in Tabelle 3.1 dargestellt.

| Anteil der Befragten (in %) | Onlinezeit (in Stunden pro Tag) |
|---|---|
| 30% | $<$ 1 h |
| 35% | 1 - 2 h |
| 26% | 2 - 5 h |
| 8% | 5 - 10 h |
| 1% | $>$ 10 h |

Tabelle 3.1: Onlinezeiten der Deutschen

Es ist zu beachten, dass in der Studie nur Informationen über die Zeit erhoben wurden, welche die Nutzer tatsächlich im Web aktiv sind. Ist der gewählte Tarif im Haushalt eine Flatrate, ist die Wahrscheinlichkeit hoch, dass der Computer während seiner kompletten Laufzeit mit dem Internet verbunden ist, auch wenn der Anwender gerade nicht explizit ein Programm nutzt, das auf das Internet zugreift. Die Zeit, die der betrachtete Computer aus Sicht des Datenbank-Grids verfügbar ist, ist daher unter Umständen höher, als die in Tabelle 3.1 angegebenen Zeiten. Dies gibt jedoch die Sicherheit, dass es sich bei diesen Zahlen näherungsweise um untere Schranken handelt.

Es seien für diese Arbeit zwei grobe Nutzergruppen definiert:

1. „Normal“-Anwender, die nur kurz Nachrichten oder E-Mails lesen – Onlinezeit bis fünf Stunden pro Tag, 91% der Befragten
2. Dauernutzer, welche beispielsweise viel chatten oder in sozialen Netzwerken aktiv sind – Onlinezeit ab fünf Stunden pro Tag, 9% der Befragten

Wie bereits im Unterabschnitt Crowdsourcing erwähnt, könnten Firmen durchaus ihre Ressourcen zur Verfügung stellen, da die Ziele dieser Arbeit auch in ihren Interessen liegen. Von Arbeitsplatz-Computern kann man annehmen, dass sie sieben bis acht Stunden täglich fünf Tage die Woche laufen (also durchschnittlich fünf bis sechs Stunden pro Tag). Auch beispielsweise Poolrechner in Universitäten können ähnliche Verfügbarkeiten aufweisen. Beide wären also in etwa Kategorie 2 zuzuordnen.

Computer von Nutzern der Kategorie 1 eignen sich in Annahme nicht für die Verwendung als Speicher-Knoten. Bei ihrer geringen Online-Zeit stünde alleine der Verwaltungsaufwand, sie regelmäßig auf den aktuellen Datenbestand zu bringen, in keiner Relation zu der Zeit, die sie effektiv nutzbar sind. Daher soll in dieser Arbeit untersucht werden, ob alleine die weniger als 10% ausmachenden Computer der Nutzer aus Kategorie 2 ausreichen, um das Speicher-Grid aufzubauen. Aus den Zahlen sei daher als Grundlage für diese Arbeit eine Onlinezeit der Speicher-Knoten von sechs Stunden pro Tag angenommen. 6 $h/d = 25\%$ Verfügbarkeit der Rechner.
Damit ergibt sich Tabelle 3.2 für die Verfügbarkeit eines Datensatzes in Relation zur gewählten Anzahl an Replikaten je Datensatz. Ein Datensatz in der Datenbank entspricht dabei zunächst einer Gruppe und deren Inhalten. Da die entstehende Datenmen-

| Replikationsfaktor | Verfügbarkeit | Ausfallzeit |
|---|---|---|
| 1 | 25% | 18 Stunden pro Tag |
| 2 | 44% | 13 Stunden 26 Minuten pro Tag |
| 4 | 68% | 7 Stunden 40 Minuten pro Tag |
| 8 | 90% | 2 Stunden 24 Minuten pro Tag |
| 16 | 99% | 14 Minuten am Tag |
| 25 | 99,9% | 44 Minuten pro Monat |
| 31 | 99,99% | 4 Minuten pro Monat |
| 39 | 99,999% | 5 Minuten pro Jahr |

Tabelle 3.2: Rechner-Verfügbarkeits-Relation

ge mindestens linear mit dem Replikationsfaktor steigt, ist dieser als „teuer“ anzusehen – zumindest aus der Sicht eines Endanwenders.

Zu klären bleibt, welche Forderungen diese Arbeit überhaupt an die Verfügbarkeit stellt. Ist eine Gruppe nicht erreichbar, kann der betroffene Computer ihr logischerweise nicht beitreten. Die Wahrscheinlichkeit ist jedoch hoch, dass der Fehler ein zweites und drittes Mal auftreten wird. Ein Beitritt zur entsprechenden Gruppe sowie gegebenenfalls das Senden erbetener Informationen kann dann erneut versucht werden. Dies führt zwar zum Verlust der Repräsentativität der absoluten Zahlen für die Häufigkeit eines Fehlers, da das Problem aber für alle Gruppen gilt, bleibt die Aussage über die relative Häufigkeit statistisch gesehen korrekt. Für Sammel-Knoten ist die Verfügbarkeit also als vergleichsweise unkritisch anzusehen.
Auf der anderen Seite kann ein für eine Firma tätiger Analyst kaum ein bis zwei Stunden Pause pro Tag vor seinem Vorgesetzten rechtfertigen. Für die Software der Analyse-Knoten wäre daher gegebenenfalls ein Caching-Mechanismus denkbar, der in regelmäßigen Abständen aktuelle Daten zu einer geforderten Menge von Gruppen lokal zwischenspeichert und Schreibvorgänge verzögern kann. So können Ausfallzeiten von der Analyse-Software überbrückt werden.

Ein Replikationsfaktor von acht scheint ein guter Kompromiss zwischen Erreichbarkeit und aufkommendem Datenvolumen für Speicher-Knoten. Offen bleibt bis zur Leistungsbewertung jedoch die Frage, ob die weniger als 10% der Netzteilnehmer, die laut der BITKOM-Pressemitteilung die nötige Onlinezeit gewährleisten, für die auftretende Datenmenge ausreichen.

### 3.4.2 Strukturierung der Daten

Im Windows Error Reporting werden laut [WER Klassifizierung] folgende Informationen eines Fehlers zur Erstellung einer eindeutigen Eimer-ID benutzt:

- „Application name – for example, winword.exe“
- „Application version – for example, 10.0.2627.0“
- „Module name – for example, mso.dll“
- „Module version – for example, 10.0.2613.1“
- „Offset into module – for example, 00003cbb“

Für die Zwecke dieser Arbeit werden diese Informationen noch um die Plattform ergänzt – beispielsweise „Win32NT 6.1.7601.65536 x64“, alias Microsoft Windows 7 64-bit Service Pack 1, oder „java 1.7.0_03-icedtea x86_64“, alias OpenJDK 7 für 64-bit Linux. Folglich ist ein Fehler als festes Tupel zu behandeln, welches für die (vermutete) Fehlerursache eindeutig ist und die Gruppen-ID ergibt. Eine Gruppen-ID, und somit die minimale Menge der von einem Sammel-Knoten benötigten Informationen, wäre also beispielsweise:

$$winword.exe, 10.0.2627.0, mso.dll, 10.0.2613.1, 00003cbb, Win32NT\ 6.1.7601.65536\ x64$$

Komponenten in einem Computersystem sind mitunter sehr unterschiedlich – eine Grafikkarte hat beispielsweise einen Anschlusstyp, ein Browser einen Hersteller. Sie müssen daher sehr flexibel modelliert werden. Des Weiteren müssen diese Informationen auch auf unterschiedlichen Plattformen in gleicher Art und Weise erfasst werden können, da manche Software und insbesondere Hardware unabhängig vom verwendeten Betriebssystem eingesetzt werden kann.
Wir definieren für diese Arbeit eine Komponente als eine Menge von Attribut-Wert-Paaren, beispielsweise

$$\{Hersteller : Mozilla\ Foundation\}, \{Name : Firefox\}, \{Version : 11.0.1\}, ...$$

oder

$$\{Hersteller : NVidia\}, \{Modell : GeForce\ 9600M\ GT\}, \{Anschlusstyp : PCIe\}, ...$$

Ein Computer ist seinerseits wiederum eine Menge von Komponenten.
Man kann nun leicht nachvollziehen, dass auch die Plattform selbst eine Komponente ist – wie im Beispiel oben zusammengesetzt aus Name, Version und Architektur.

Eine weitere Anforderung an das System ist, dass Änderungen über die Zeit modellierbar sein müssen. Ein Computer kann durch Installation, Deinstallation, Update oder Umbau eine Veränderung seiner Komponenten erfahren. Dabei kann eine Komponente einen Computer beliebig oft verlassen und wieder betreten, wie etwa bei USB-Geräten üblich. Korreliert ein Fehler aber mit einer bestimmten Komponente, die aktuell im betroffenen System nicht vorhanden ist, muss die Systemkonfiguration zum Zeitpunkt des Fehlers rekonstruierbar sein. Die Mitgliedschaft von Gruppen bezieht sich also nicht nur auf den Computer, sondern auch auf die Zeitpunkte, zu denen der Fehler aufgetreten ist.

Um dies zu gewährleisten wird zu jeder Komponente in einem Computer ihr Aktivierungszeitpunkt gespeichert sowie beim Entfernen der Deaktivierungszeitpunkt. Eine Komponente selbst darf nie aus der Datenbank entfernt werden.

## 3.5 Analyse-Knoten

Zum Abrufen der verfügbaren Informationen von Speicher-Knoten liefern uns Projekte wie [Bugzilla] einen ersten Anhaltspunkt. Ein Ticket in einem Bugtracker enthält die verfügbaren fixen Informationen – für diese Arbeit etwa jene Informationen, welche für die Gruppen-ID zuständig sind –, ein Feld für Anhänge – hier unter anderem Log-Dateien und Memory Dumps – und einen Bereich für Diskussionen – hier insbesondere für die Analysten. Für die Zwecke dieser Arbeit müssen zusätzlich die betroffenen Computer dargestellt werden. Neben der reinen Auflistung dieser wäre es hilfreich, direkt eine Menge von Komponenten aufzuzeigen, die allen betroffenen Computern gemeinsam ist, um Korrelationen leicht erkennbar zu machen.

Neben der Darstellung der Daten ist es die zentrale Aufgabe der Analyse-Software, einem Analysten das Durchsuchen des Datenbestandes zu ermöglichen. Bugzilla lässt hier in der Standardausführung neben der einfachen Schlagwort-Suche nur nach Produkt, Komponente, Berichts-Status (NEW, ASSIGNED, VERIFIED, CLOSED, ...) und aktueller Lösungs-Kategorie (FIXED, INVALID, WONTFIX, DUPLICATE, ...) suchen.
Bei der großen Anzahl und breiten Fächerung der mit den Methoden dieser Arbeit gesammelten Fehlerberichte müssen Suchanfragen in der Analyse-Software jedoch präziser formuliert werden können. Eine Suchanfrage sollte beispielsweise folgende Kriterien modellieren können:

- Alle betroffenen Computer haben eine Entität mit dem Paar {Hersteller: NVidia}
- Wildcard-Anfragen wie {Programmversion: 1.4.*}
- Reguläre Ausdrücke wie {Plattformname: (Free|Open|Net)BSD}
- Anfragen mit oder ohne Berücksichtigung der Groß- und Kleinschreibung
- Suche nach Fehlern eines bestimmten Zeitraums
- Auflistung aller Fehler eines konkreten Computers

Die Kombination mehrerer Kriterien mit UND, ODER und NICHT sollte ebenfalls realisierbar sein.
Es könnte auch möglich sein, direkte Abfragen an die Speicher-Knoten zu formulieren – vergleichbar mit SQL-Anfragen an relationale Datenbanken – oder Teile der Datenbank zum herunterladen zur Verfügung zu stellen – als Weiterführung der Idee einer Cache-Funktion für die Analyse-Software. Vorsicht ist allerdings geboten, da hier gegebenenfalls eine hohe Last für die Speicher-Knoten entstehen kann, wenn viele Teilnehmer die Datenbank regelmäßig und umfangreich auslesen.

Soziale Netzwerke warten mit Methoden auf, einer großen Menge an Teilnehmern eine Plattform zur Koordination (im Sinne des Koordinationsproblems) zur Verfügung zu stellen. Der Austausch von persönlichen Nachrichten ist in sozialen Netzwerken eine Grundfunktion, welche auch in die Funktionalität der Analyse-Software übernommen werden sollte, um die direkte Kommunikation zwischen Analysten zu ermöglichen. Wie etwa bei [Facebook] könnte für einen Analysten auf einer Art Startseite eine Übersicht über die ihn betreffenden Neuigkeiten existieren. Er würde so beispielsweise auf neue Beiträge in Gruppen hingewiesen, in denen er tätig ist. Die aus Facebook bekannte Chronik (englische Bezeichnung: Timeline) für Benutzerprofile kann als Vorlage dienen, um die Veränderung der Komponenten eines Computers sowie das Auftreten von Fehlern auf diesem über die Zeit darzustellen. So ist es beispielsweise leicht erkennbar, wenn ein konkreter Fehler erst nach dem Verändern einer bestimmten Komponente auftritt.

Die Analyse-Software muss so modelliert sein, dass sie die aufgeführten Ideen implementiert, auf Endanwender-Computern läuft und sich – wie die Sammel-Knoten – zu einem beliebigen Speicher-Knoten verbindet um auf den Datenbestand zuzugreifen. Da im Gegensatz zu Sammel-Knoten hier keinerlei Interaktion mit der Plattform von Nöten ist, kann sie ausnahmslos mit betriebssystemunabhängigen Technologien realisiert werden.

# 4 Implementierung

Das folgende Kapitel beschäftigt sich mit der programmatischen Umsetzung des ausgearbeiteten Entwurfs.

## 4.1 Designentscheidungen

### 4.1.1 Programmauswahl

Als Plattform für die im Rahmen dieser Arbeit erstellte Software wurde nach Vorgabe Microsoft Windows 7 verwendet, hier in der Version Professional x64 mit Service Pack 1. Geschrieben ist die Software in C# 4.0 für .NET 4.0, in der Hoffnung Teile dieser Arbeit können mit Mono auf andere Plattformen übertragen werden. [Mono] ist eine quelloffene Implementierung des .NET-Frameworks, welche auch unter Linux und Mac OS X lauffähig ist. Die verwendete IDE war Microsoft Visual Studio 2010 Service Pack 1. Die abgebildeten UML-Diagramme wurden mit Altova UModell 2011 erzeugt.

Zur Erkennung von Softwarefehlern und zum Sammeln der zugehörigen Daten wird der auf Endanwender-Rechnern vorhandene Teil des Windows Error Reporting verwendet.

Die eingesetzte Datenbank ist Apache Cassandra, da sie wie im letzten Kapitel beschrieben viele für diese Arbeit geforderte Eigenschaften von Speicher-Knoten bietet.

### 4.1.2 Einschränkungen

Bisher wurde ein sehr umfangreicher und ausführlicher Entwurf gezeichnet. Dessen Umsetzung würde natürlich den Rahmen dieser Arbeit um ein Vielfaches übersteigen. Die folgenden Einschränkungen haben es ermöglicht, die in diesem Kapitel vorgestellten, umgesetzten Teile tiefgreifender und sauberer auszuarbeiten. Die Idee dieser Arbeit wird so auf eine solide Basis gestellt, um die Grundlage für weiterführenden Arbeiten zu legen.

- Von der Erstellung einer Analyse-Software wurde abgesehen. Die Anbindung freier Software für Foren oder soziale Netzwerke liefert zwar eine gute Grundidee, doch alleine beispielsweise die Methoden über das Suchen, Finden und Darstellen von Gruppen (= Fehlern) könnte im Umfang bereits eine eigene Arbeit ausfüllen. Die Schnittstellen, um eine solche Software in aufbauenden Arbeiten zu implementieren, werden jedoch weiter unten in diesem Kapitel beschrieben.

- Die Bedienung der Software beschränkt sich auf ein Konsolen-Interface
- Eine neue Cassandra-Instanz benötigt zum Starten, wie im Kapitel Grundlagen beschrieben, die Adresse mindestens einer anderen bereits laufenden Instanz für ein erstes Gossip-Treffen. Bei der angestrebten Netzwerkstruktur kann jedoch nicht einfach eine feste Adresse gewählt werden – diese müsste ein Server sein, was der Idee widerspricht. Ein entsprechender Mechanismus zur Bekanntmachung der Adressen von Cassandra-Instanzen wird außen vor gelassen.
- Datenschutz wurde und wird nur im Bezug auf informationelle Selbstbestimmung und Schutz sensibler Daten betrachtet, nicht im Bezug auf Angriffs- und Manipulationsszenarien. Auch gibt es keinen Authentifizierungsmechanismus zwischen den Knoten.
- Es werden nur die durch Windows-Bordmittel (WER) erhaltbaren PEDIs verwendet – ein kurzer textueller Bericht sowie ein optionaler Dump.
- Die Erkennung von Systemkomponenten beschränkt sich auf über die Wege des Betriebssystems installierte Software – wie sie beispielsweise in der Systemsteuerung einsehbar ist. Anknüpfungspunkte für das Einbinden weiterer Erkennungsmechanismen sind jedoch in der Software vorhanden.
- Das Loglevel ist statisch und verändert sich nicht on-demand, lediglich Dumps werden nur auf Anfrage gesendet.
- Speicherabbilder von Programmen werden in der erstellten Software nur von User-Mode-Programmen erstellt. Dies ergab sich im Verlaufe der Arbeit und ist weiter unten erläutert.
- Die Software wurde nur unter Windows 7 in der beschriebenen Version getestet. Die Minimalanforderung ist in Annahme Windows XP. Dies ist die erste Windows-Version mit WER und sollte somit von den implementierten Routinen zur Interaktion mit der Plattform angesprochen werden können.

Ideen für die Realisierung der fehlenden Elemente sind im Ausblick am Ende der Arbeit angerissen.

## 4.2 Modellierung

### 4.2.1 Kommunikation

Wie bereits im Entwurf erörtert, teilt sich die Arbeit in drei Knotentypen (respektive Programme), je zum sammeln, speichern und analysieren von Fehlerberichten. Alle gemeinsam bauen auf eine vierte Komponente, die Netzwerkinfrastruktur.

Die Kommunikationslogik soll durch Gossip realisiert werden. Dieses Protokoll findet sich bereits in Cassandra implementiert, der Software für Speicher-Knoten. Da eine Verbindung zu einem Cassandra-Grid über eine Verbindung zu jedem beliebigen seiner Knoten geschehen kann, braucht es keine weitere Logik zur Realisierung einer Kommunikation zwischen Sammel-Knoten und/oder Analyse-Knoten untereinander.
Die im Rahmen dieser Arbeit erstellte Sammel-Software integriert sich in die Plattform des Sammel-Knotens, nimmt dort Fehlerberichte entgegen und verarbeitet diese zu einem ebenfalls in diesem Kapitel vorgestellten Format. Anschließend gibt sie diese Daten zu einem Speicher-Knoten. Der Analyse-Knoten kann die Daten von einem Speicher-Knoten beziehen und entsprechend seine Daten auch wieder zu einem solchen zurückschreiben. Die resultierenden, in der Datenbank eingetragenen Lösungsvorschläge und Bitten um Daten kann die Sammel-Software wiederum aus den Einträgen auf den Speicher-Knoten herauslesen.

Die Cassandra-API basiert laut [Cassandra API] auf Thrift. Nach [Slee et al., 2007] ist Thrift ein sprachneutraler, in zahlreichen Programmiersprachen implementierter Softwarestack und eine zugehörige Quellcodeerzeugungsengine für jede dieser Sprachen. Diese Engine transformiert einen in einer Schnittstellen- und Datenbeschreibungsspache (engl.: interface and data definition language) geschriebenen Code in Client- und Server-RPC-Bibliotheken. Diese ermöglichen RPC-Kommunikation zwischen in verschiedenen Sprachen geschriebenen Programmen.
Durch die Verfügbarkeit von [Thrift] für eine breite Basis an Programmiersprachen und dessen quelloffener Lizensierung entspricht das Cassandra-Thrift-API der im Kapitel Entwurf geforderten Plattformunabhängigkeit und wird hier als Basis für die Kommunikation zwischen Sammel- und Speicher-Knoten respektive Analyse- und Speicher-Knoten definiert.

### 4.2.2 Sammel-Software

#### Gliederung

Die Sammel-Software ist in folgende vier Komponenten aufgegliedert:

- Controller
- CrowdConnector
- PlattformConnector
- UserConnector

Für die Kommunikation der einzelnen Komponenten der Sammel-Software untereinander wird ein eventbasiertes System gewählt.

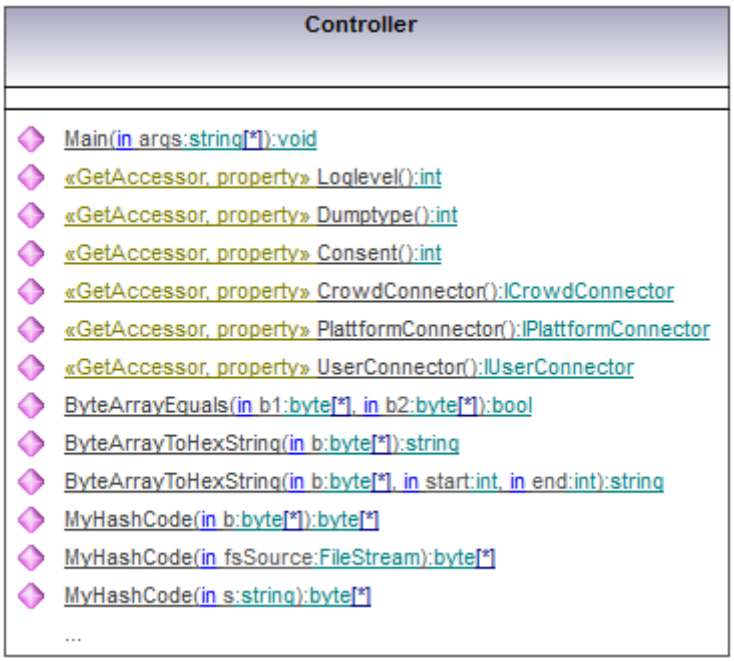


Abbildung 4.1: Controller

Abbildung 4.1 zeigt den statischen Controller. Er erstellt und konfiguriert die drei Connectoren. Anschließend übergibt er den Kontrollfluss an den UserConnector. Der Controller empfängt Events der Connectoren und leitet diese gegebenenfalls weiter. Neben Hilfsfunktionen für die Hashgenerierung bietet der Controller den Connectoren Lese-Zugriff für die aktuelle Konfiguration. Verweise auf alle Connectoren ermöglichen es einem einzelnen Connector, auf die öffentlichen Methoden der anderen Connectoren zuzugreifen.

Die in Abbildung 4.2 gezeigten Connectoren wurden durch Interfaces definiert. So kann ein einfaches modulares Austauschen dieser gewährleistet werden.
Der CrowdConnector stellt die Verbindung zu einem gegebenen Speicher-Knoten her und sendet die ihm übergebenen Fehlerberichte an diesen. Ein separater Thread kann den Speicher-Knoten periodisch auf Lösungsvorschläge zu Fehlern durchsuchen, die den Computer betreffen.
Der PlattformConnector konfiguriert die Plattform entsprechend den im Controller einsehbaren Einstellungen. Er registriert sich bei der Plattform als Empfänger bei Events für auftretende Fehler und Veränderungen der Komponenten am System. Der PlattformConnector bringt diese Meldungen in ein vereinheitlichtes Format und leitet sie an den Controller weiter.
Der UserConnector realisiert die Interaktionsschnittstelle mit dem Benutzer. Er gibt Möglichkeiten zur Konfiguration und leitet getätigte Einstellungswünsche an den Controller weiter. Auch zeigt er Benachrichtigungen an, die vom Controller oder einem

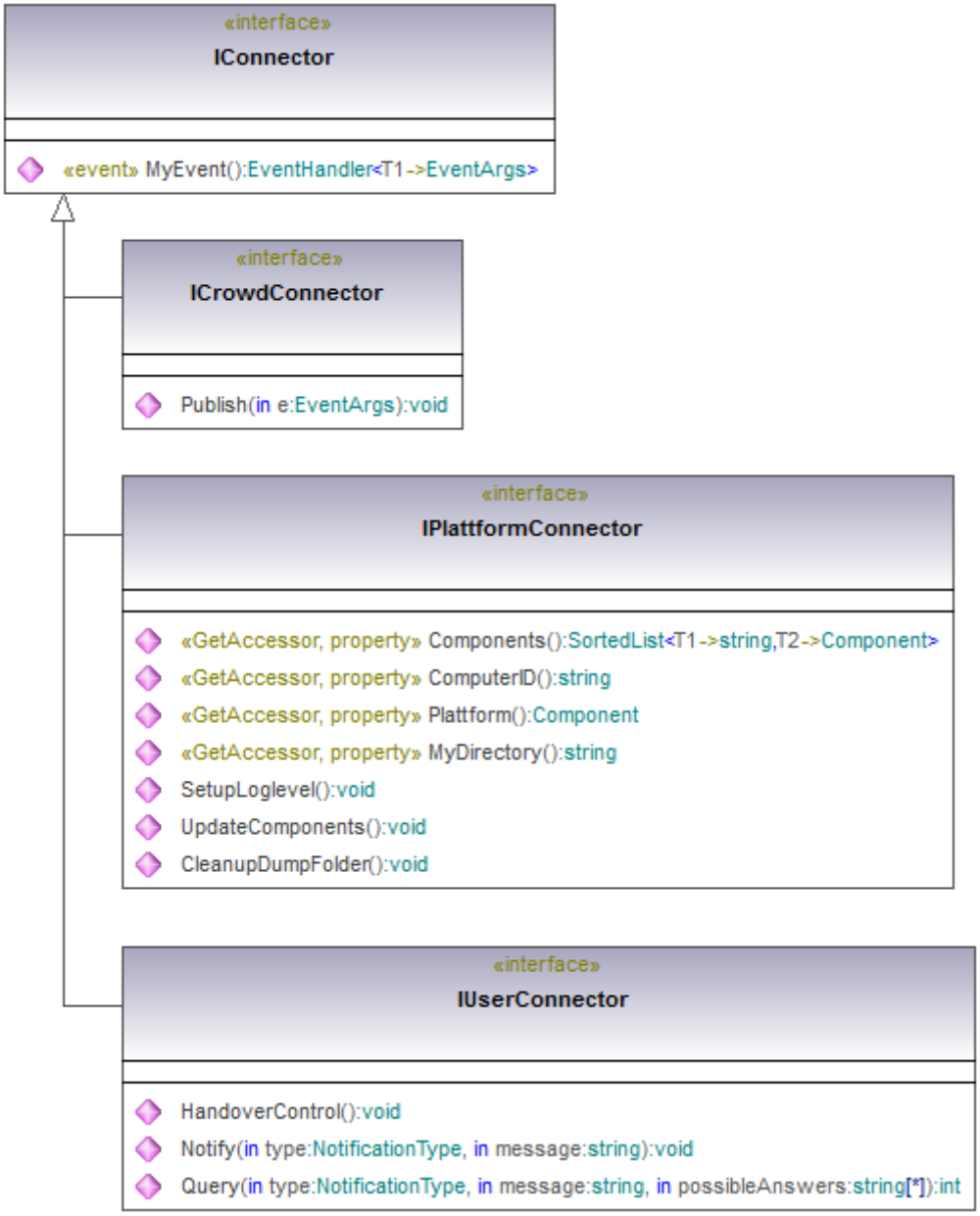


Abbildung 4.2: Connector

beliebigen Connector kommen können. Des Weiteren implementiert der UserConnector den „Möchten Sie den Fehlerbericht einschicken?“-Dialog.
Für diese Arbeit wird ein CrowdConnector erstellt, welcher über die Cassandra-Thrift-API eine Verbindung zu einer Cassandra-Instanz herstellt; ein PlattformConnector, welcher sich in Windows 7 Professional x64 SP1 integriert und ein UserConnector, welcher ein Kommandozeileninterface zur Verfügung stellt.

#### Loglevel

In der Implementierung stehen folgende Einstellungsmöglichkeiten zur Verfügung:

**Loglevel 0** Sendet keine Fehlerberichte

**Loglevel 1** Sendet den Satz minimal benötigter Informationen zum Erstellen einer Gruppe oder zum Beitritt zu einer solchen

**Loglevel 2** Erstellt ein Profil des Computers mit der auf diesem registrierten Software

Das Versenden eines Speicherabbildes, sofern erbeten, wird separat eingestellt.

**Dumptyp 0** Sende nie ein Speicherabbild, nur die im Loglevel angegebenen Daten

**Dumptyp 1** Sende einen Minidump (nur Stacktrace)

**Dumptyp 2** Sende einen vollen Dump

Schließlich ist noch konfigurierbar, ob und wie das Einverständnis des Nutzers eingeholt wird.

**Consent 0** Jedwedes Senden verweigern – ermöglicht es, trotzdem in einer Gruppe nach Lösungsvorschlägen bezüglich aufgetretener Fehler zu suchen

**Consent 1** Für jede Sende-Operation nachfragen – getrennt für Bericht und Dump

**Consent 2** Automatisches Senden aller Daten

### 4.2.3 Speicher-Software

#### Wahl der Schlüssel

Für die Erstellung von Einträgen in Cassandra muss jedem Eintrag ein Schlüssel (ID) zugeordnet werden – nicht nur Gruppen, sondern auch beispielsweise Komponenten und Computern. Da dieser Schlüssel logischerweise eindeutig sein muss, sollte er sich aus den jeweiligen eindeutigen Informationen für den entsprechenden Eintrag deterministisch ergeben. So kann er von jedem Teilnehmer berechnet werden, der über die jeweiligen Informationen verfügt – wenn etwa die Komponente Teil seines Systems ist. Anders

könnte man für einen konkreten Eintrag nicht nachvollziehen, ob dieser bereits im System eingetragen ist, ohne alle existierenden Einträge abzurufen.

Für Komponenten wird die nach den Attributen alphabetisch sortierte Konkatenation der Attribut-Wert-Paare als eindeutige Information gewählt – ohne Berücksichtigung der Groß- und Kleinschreibung bei der Sortierung.
Für Gruppen werden die bereits als Gruppen-ID erläuterten Werte für deren Schlüssel gewählt, in der im Entwurf verwendeten Reihenfolge. Informationen wie betroffene Computer (sprich Gruppenmitglieder), Dumps und Lösungsvorschläge werden in einer Gruppe eingetragen, ohne Teil der Gruppen-ID zu sein.
Für die Zwecke dieser Arbeit ist es wichtig, dass Computer (gleich ob Sammel-, Speicher- oder Analyse-Knoten) eindeutige Elemente im System sind. Wie bereits im Unterabschnitt Datenschutz der drei Grundsäulen erwähnt, soll das System aber trotzdem Anonymität gewährleisten. Von Seiten der anderen Teilnehmer des Systems ist es jedoch vollkommen ausreichend, wenn ein Computer von den anderen unterscheidbar ist – beispielsweise ist für einen Analysten nur die Information wichtig, wie viele verschiedene Computer von der Fehlerursache betroffen sind. Für den jeweiligen Computer selbst hingegen ist es wichtig zu erkennen, welches Element im System für ihn selbst steht – beispielsweise zur Verifikation einer Gruppenzugehörigkeit. Benötigt wird also eine Einweg-Funktion, deren Ergebnis aus konstanten Informationen des Computers auf diesem deterministisch errechenbar ist – kurz, ein Hash. In dieser Arbeit wird der Hash aus der Konkatenation von Hostname, Plattformname, Plattformarchitektur und MAC-Adresse der ersten Netzwerkkarte gewählt – Informationen, die vergleichsweise selten wechseln. Durch die Auslegung dieser Arbeit für die Kooperation über das Internet kann das Vorhandensein einer Netzwerkkarte angenommen werden. Der Vorteil dieser Methode ist, dass ein Computer für jedes installierte Betriebssystem als separater Teilnehmer auftritt, aber ein Update des Betriebssystems keine Veränderung der Computer-ID hervorruft.

Um Schlüssel uniformer und kurzer Länge zu erhalten, ergeben sich auch die Gruppen- und Komponenten-Schlüssel aus dem Hash der jeweiligen Informationen. Zudem werden ab diesem Punkt die auch in der Implementierung benutzten englischen Bezeichnungen verwendet. Für die vorliegende Arbeit wurde MD5 verwendet, welcher 128 Bit lange Hashwerte erzeugt.
Daraus ergibt sich (mit „+“ als String-Konkatenation) beispielsweise:

**ComponentID** MD5(Hersteller + Mozilla + Name + Firefox + Version + 11.0.1)

**ComputerID** MD5(MyComputer + Win32NT + x64 + 00:11:22:33:44:55)

**GroupID** MD5(firefox.exe + 11.0.1 + somelib.dll + 4.2 + deadbeef + ComponentID der Plattform)

Die $2^{128}$ verschiedenen Werte eines MD5-Hashes sollten für die möglichen IDs ausreichen und es sollte zu keinen unbeabsichtigten Kollisionen kommen. Für diese Arbeit

relevant ist jedoch die Möglichkeit, das Urbild des MD5-Hashes einer ComputerID zu bestimmen, was den Hostnamen und die MAC-Adresse des Computers offenbart und somit die Anonymität des Teilnehmers gefährdet. Für Hash-Algorithmen wie MD5 oder SHA-1 existieren Regenbogentabellen, um das Zurückrechnen des Urbildes eines Hashes zu vereinfachen.
Die Wahl für MD5 in dieser Arbeit wurde aufgrund seiner Verbreitung getroffen. Das modulare Design der erstellten Software erlaubt aber ein einfaches Austauschen. Die finale Auswahl eines Hash-Algorithmus' sollte auf die im Rahmen dieser Arbeit ausgelassene Sicherheitsanalyse verschoben werden, in welcher auch Angriffsszenarien betrachtet werden müssen.
Schlüssel in Cassandra erlauben höchstens eine Länge von 64 kB.

### Datenstruktur

Nach dem bereits in den Grundlagen erläuterten CAP-Theorem kann ein verteiltes System, welches verfügbar und partitionstolerant ist, Konsistenz unter den Netzwerkteilnehmern nicht mit vernachlässigbarer Latenz garantieren. Im konkreten Fall bedeutet dies, dass ein Datensatz, der auf eine Cassandra-Instanz geschrieben wird, erst mit Verzögerung im restlichen System verfügbar ist. Existieren zwei Datensätze gleichen Attributs, aber unterschiedlicher Werte, so wählt Cassandra den neueren Wert.
Am Beispiel dieser Arbeit sei in Annahme für die Gruppen nur ein einziger Zähler implementiert, der die Anzahl der Auftreten eines Fehlers zählt. Aktuell sei der Zähler 5. Sind vier Computer mit einer konkreten Datenbank-Instanz verbunden während ein Fehler auf ihnen auftritt, wird deren Zähler sukzessive auf 9 erhöht. Verbindet sich zu einer anderen Instanz wenig später, noch vor einem Gossip-Treffen der betreffenden Instanzen, ein anderer Computer, erhöht dieser den dortigen Zähler auf 6. Da bei einer folgenden Synchronisation der Wert 6 zeitlich später geschrieben wurde, wird dieser im System propagiert.
Aufgrund dieses Umstandes wäre es für das gezeichnete Modell ideal, wenn alle Daten nur ein einziges Mal geschrieben und ab diesem Zeitpunkt nur noch gelesen würden. Auf diese Weise bekäme man schlimmstenfalls einen veralteten Datensatz, jedoch nie einen inkonsistenten oder falschen. Aus dieser Überlegung ergibt sich näherungsweise der Entwurf von Tabelle 4.1 – Texte in spitzen Klammern werden durch ihre jeweiligen Werte ersetzt und kommen mehrfach vor. Für diese Datenstruktur gilt, dass auf alle Daten nur von einem einzigen Teilnehmer lesend und schreibend zugegriffen wird, von anderen Teilnehmern ausschließlich lesend.

In Annahme verbindet sich ein Knoten zu dessen Laufzeit nur zu einer einzigen Cassandra-Instanz. Da sich *eventual consistency* lediglich auf die Konsistenz des Datenbestandes über die verschiedenen Knoten hinweg bezieht, sind für einen Teilnehmer die von ihm geschriebenen Einträge verzögerungsfrei konsistent zugänglich. Weiterhin wird angenommen, dass zwischen zwei Phasen, in denen ein Sammel- oder Analyse-Knoten mit dem Netzwerk verbunden ist, ein hinreichend großes Zeitfenster existiert, um zwischen

<table>
<tr><th colspan="5">Entwurf einer hierarchischen Datenstruktur</th></tr>
<tr><td rowspan="12"><GroupID></td><td rowspan="6">Detail</td><td colspan="3">{AppName: Programmname}</td></tr>
<tr><td colspan="3">{AppVersion: Programmversion}</td></tr>
<tr><td colspan="3">{ModuleName: Modulname}</td></tr>
<tr><td colspan="3">{ModuleVersion: Modulversion}</td></tr>
<tr><td colspan="3">{Offset: Offset im Modul}</td></tr>
<tr><td colspan="3">{Plattform: ComponentID der Plattform}</td></tr>
<tr><td>DumpRequest</td><td colspan="3">{<ComputerID>: Anzahl der vom Computer ComputerID erbetenen Dumps}</td></tr>
<tr><td>Component</td><td><ComponentID></td><td colspan="2">{<ComputerID>: Wie oft die Komponente ComponentID auf dem Computer ComputerID am Fehler GroupID beteiligt war.}</td></tr>
<tr><td>Computer</td><td colspan="3">{<ComputerID>: Anzahl Auftreten auf dem Computer ComputerID}</td></tr>
<tr><td>Dump</td><td colspan="3">{<DumpID>: Memory Dump (Binärdaten)}</td></tr>
<tr><td rowspan="2">Solution</td><td rowspan="2"><SolutionID></td><td>Description</td><td>{Description: Beschreibungstext für den Lösungsvorschlag}</td></tr>
<tr><td>Voting</td><td>{<ComputerID>: Bewertung des Lösungsvorschlags SolutionID vom Computer ComputerID}</td></tr>
<tr><td><ComputerID></td><td><ComponentID></td><td colspan="3">{<Index>: Datum der Änderung mit der Nummer Index an der Komponente ComponentID}</td></tr>
<tr><td><ComponentID></td><td colspan="4">{<Attribut>: Wert}</td></tr>
</table>

Tabelle 4.1: Entwurf Datenstruktur

den Speicher-Knoten eine Synchronisation durchzuführen. Aus diesem Grunde kann gefolgert werden, dass für jeden einzelnen Teilnehmer am Netzwerk Konsistenz über seine eigenen Einträge angenommen werden kann.
Folglich befindet sich in Tabelle 4.1 bei Einträgen, die von mehreren Teilnehmern geschrieben werden könnten, eine Unterstruktur nach deren ComputerID. Äußert ein Entwickler beispielsweise die Bitte nach einem einzelnen Speicherabbild, so wird unterhalb des Feldes DumpRequest ein Eintrag mit dessen ComputerID als Attribut und der Zahl 1 als Wert angelegt – statt etwa nur einen einzelnen Zähler für DumpRequest zu inkrementieren. Ein anderer Analyst kann mit einer anderen Datenbank-Instanz verbunden sein und zeitlich parallel seinen eigenen Eintrag erstellen, ohne dass bei einer Synchronisation der Instanzen ein Konflikt auftritt oder Informationen verloren gehen.
Dies gilt für die Einträge unter:

- <GroupID> / DumpRequest / <ComputerID>
- <GroupID> / Component / <ComponentID> / <ComputerID>
- <GroupID> / Computer / <ComputerID>
- <GroupID> / Solution / <SolutionID> / Voting / <ComputerID>
- <ComputerID>

Da Komponenten per Definition eindeutig sind, können unter <ComponentID> keine Konflikte auftreten. Selbst wenn eine Komponente zur gleichen Zeit an unterschiedlichen Konten eingefügt wird, sind die erstellten Einträge identisch.
Das System hat ferner den Vorteil, dass man entweder die Summe aller Zahlen oder die Summe aller disjunkten Einträge betrachten kann – beispielsweise die Summe aller aufgetretenen Fehler oder die Zahl aller betroffenen Computer.

Einträge für Komponenten werden in Tabelle 4.1 durch deren oben erläuterte ID identifiziert. Jede Komponente besteht aus einer von Menge Attribut-Wert-Paaren.
Ein Computer besteht aus einer Menge an Komponenten. Unter jedem Eintrag mit einer ComputerID werden dessen Komponenten aufgelistet, repräsentiert durch ComponentIDs, welche sich auf ihre entsprechenden Einträge beziehen. Die Indices bei den Komponenten eines Computers enthalten das Datum des Hinzufügens beziehungsweise des Entfernens der Komponente. Per Definition muss eine Komponente einem Computer zuerst hinzugefügt werden – nötigenfalls beim ersten Anmelden des Computers im System. Damit stehen ungerade Indices für Hinzufügen, gerade für Entfernen der Komponente. So kann eine Komponente mehrmals ihren Status im System ändern.
Die Details einer Gruppe stehen für die Informationen, wie sie im Abschnitt zum Sammel-Knoten des Kapitels Entwurf erläutert wurden. Unter DumpRequest erstellen Analyse-Knoten Einträge mit ihrer jeweiligen ComputerID, sowie der Anzahl der von ihnen erbetenen Memory Dumps. Die Einträge unter Component enthalten die ComponentIDs der von der fehlerhaften Software geladenen Module. Jeder Computer erstellt unterhalb der jeweiligen ComponentID einen eigenen Eintrag mit einem privaten Zähler für

jedes Mal, dass das entsprechende Modul bei einem Fehler auf ihm beteiligt war. Die Einträge unter Computer hingegen sind Zähler für die reine Anzahl des Auftretens der Fehler auf dem jeweiligen Computer. Unterhalb von Dump können Computer Memory Dumps einreichen. DumpID ist dabei eine Konkatenation aus der ComputerID des Quellcomputers mit dem Erstellungsdatum des Dumps. So kann ein Computer mehrere Dumps zur Verfügung stellen. Unter Solution können Analysten Einträge für Lösungsvorschläge erstellen. Der Schlüssel SolutionID ist ein Hash über den unter Description eingetragenen Beschreibungstext für den Lösungsvorschlag. Würde ein einfacher Zähler verwendet, könnte es wieder zu den erwähnten Kollisionen kommen, wenn parallel an zwei verschiedenen Instanzen ein Lösungsvorschlag mit dem gleichen Index erstellt wird. Unter Voting kann jeder Computer einen Eintrag mit seiner ID und einer Pro-Stimme für den Lösungsvorschlag anlegen.

Für die Anwendung auf Cassandra muss eine weitere Einschränkung betrachtet werden: Cassandra unterstützt ausschließlich vier- oder fünfdimensionale Hierarchien, wobei die ersten beiden organisatorische Gruppierungen sind:

- Cluster / Keyspace / ColumnFamily / Column
- Cluster / Keyspace / SuperColumnFamily / SuperColumn / Column

Jede Zeile (engl.: Column) besteht aus einem Tripel der Form {Attribut: Zeitstempel: Wert}. Jede zusammengehörige Menge von Columns in einer ColumnFamily beziehungsweise jede zusammengehörige Menge SuperColumns in einer SuperColumnFamily verfügt über einen Spaltenschlüssel (engl.: RowKey), der sie identifiziert – im obigen Beispiel die IDs. SuperColumns sind Columns, deren Elemente normale Columns sind. (Super)-ColumnFamilies ihrerseits sind wiederum Mengen von (Super)Columns. Die Mischung von Columns und SuperColumns in einer Family ist nachvollziehbarerweise nicht erlaubt.
Keyspaces entsprechen in etwa den Datenbanken aus dem relationalen Datenbankmodell – dem logischen Zusammenschluss von Tabellen, respektive (Super)ColumnFamilies. Da auf einem einzelnen Computer mehrere logisch getrennte Datenbestände vorliegen können, die jeweils aus mehreren Keyspaces bestehen können, gehören Keyspaces wiederum zu einem Cluster. Verbinden sich zwei laufende Cassandra-Instanzen, die nicht zum gleichen Cluster gehören, so kommt kein Datenaustausch zustande.
Da der Zeitstempel in dieser Arbeit von Cassandra automatisch verwaltet wird (um Konflikte bei der Synchronisation aufzulösen) wird er im weiteren Verlauf dieser Arbeit nicht weiter aufgeführt. Der Vollständigkeit halber sei aber erwähnt, dass auch eine manuelle Manipulation des Zeitstempels möglich ist. Des Weiteren befinden sich alle (Super)ColumnFamilies dieser Arbeit im selben Keyspace im selben Cluster. Daher werden diese Ebenfalls nicht explizit erwähnt.
Aufgrund der Restriktionen bezüglich der Verschachtelungstiefe von Cassandra wird der Ansatz aus Tabelle 4.1 zu jenem in Tabelle 4.2 verändert.

| (Super)CF | RowKey | (Super)Column | Column |
|---|---|---|---|
| Group | <GroupID> | Detail | {AppName: Programmname} |
| | | | {AppVersion: Programmversion} |
| | | | {ModuleName: Modulname} |
| | | | {ModuleVersion: Modulversion} |
| | | | {Offset: Offset im Modul} |
| | | | {Plattform: ComponentID der Plattform} |
| | | Computer | {<ComputerID>: Anzahl Auftreten des Fehlers auf dem Computer *ComputerID*} |
| | | Dump | {<DumpID2>: Memory Dump (Binärdaten)} |
| GroupComponent | <GroupID> | {<ComputerID>: Wie oft die Komponente *ComponentID* auf dem Computer *ComputerID* am Fehler *GroupID* beteiligt war.} | |
| GroupSolutionVote | <GroupID> | <SolutionID> | {ComputerID: Bewertung des Lösungsvorschlags *SolutionID* vom Computer *ComputerID*} |
| Computer | <ComputerID> | <ComponentID> | {<Index>: Datum der Änderung mit der Nummer *Index* an der Komponente *ComponentID*} |
| GroupSolution | <SolutionID> | {<ComputerID>: Beschreibungstext für den Lösungsvorschlag} | |
| Component | <ComponentID> | {<Attribut>: Wert} | |

Tabelle 4.2: Datenstruktur für Cassandra

DumpID2 ist hier die im ersten Ansatz verwendete DumpID, plus einem Zähler. Nach [Cassandra Limits] muss ein einzelner Datensatz in den Arbeitsspeicher passen. Daher wird empfohlen, große Daten zu teilen. Als Beispiel werden auf der gleichen Seite 64 MB angeführt. Ist daher ein Dump im hier verwendeten System größer als 64 MB, so wird er in 64 MB-Blöcke geteilt, die ab 0 durchnummeriert werden – nicht zu teilende Dumps erhalten dabei trotzdem den Zähler 0 zu ihrem Index.
Wie man sieht, wurden diverse Einträge unter Group / <GroupID> auf die oberste Ebene vorgezogen, um tiefere Hierarchien aufzulösen. Dabei ist in diesen (Super)ColumnFamilies der RowKey der gleiche, wie jener der entsprechenden Gruppe. Aufgrund der Tatsache, dass bei Cassandra die Verteilung von Datensätzen auf die verschiedenen Speicher-Knoten auf Basis des RowKeys geschieht, bleiben daher zusammengehörige Datensätze dem gleichen Speicher-Knoten zugeordnet und werden nicht physisch fragmentiert.
Cassandra nimmt die Aufteilung in getrennte Dateien auf der Festplatte anhand der (Super)ColumnFamilies vor – eine Datei je Family.

Die in Tabelle 4.2 erstellte Strukturierung der Daten definiert ein plattformübergreifendes Format für Fehlerberichte, welches den im Kapitel Entwurf geäußerten Ansprüchen genügt.

### Verteilung der Daten

Ein Cassandra-Grid besteht aus vielen Knoten. Für jeden Datensatz ist eindeutig bestimmt, welcher dieser Knoten für ihn zuständig ist. Ein einstellbarer Verteilungsmechanismus determiniert außerdem die Knoten, auf denen die Replikate gespeichert werden. Alle zusammen werden im Folgenden als Replika-Knoten für den entsprechenden Datensatz bezeichnet.

Da sich ein Client zu jeder beliebigen Instanz des Grids verbinden kann, ist nicht gegeben, dass diese für einen eingehenden Datensatz zuständig ist. Die entsprechende Instanz reicht den Datensatz in diesem Fall nur weiter. Sollten eine, mehrere oder (ab Cassandra 1.0) gar alle Replika-Knoten nicht erreichbar sein, so kann der Mechanismus des HintedHandoff verwendet werden. Die Nachricht wird dabei für eine bestimmte Zeit im System vorgehalten, ohne auf den oder die Replika-Knoten geschrieben zu werden, bis sich der zuständige Knoten wieder verbindet.
Lese-Anfragen werden jedoch ausschließlich an einen der zuständigen Replika-Knoten gerichtet – alle im System existierenden Knoten zu fragen wäre verständlicherweise unpraktikabel. Dadurch ist der betreffende Datensatz in diesem Zustand nicht lesbar.

### Konsistenzanforderungen für Zugriffe

Für Lese- und Schreibzugriffe gibt es die ConsistencyLevel ONE, QUORUM (zu dt.: beschlussfähige Mehrheit) und ALL, für Schreibvorgänge zusätzlich ANY. QUORUM

ist hierbei $N/2 + 1$, wobei N die Gesamtzahl der Replika-Knoten ist. Dies bedeutet laut [Cassandra API], dass der Zugriff für *eine*, *mehr als die Hälfte* oder *alle* Replika-Knoten erfolgreich abgeschlossen werden muss, um insgesamt als erfolgreich angesehen zu werden. Bei einem ANY-Schreibvorgang gilt das Schreiben des Datensatzes auch als erfolgreich abgeschlossen, wenn ein beliebiger Knoten ein HintedHandoff entgegengenommen hat. Nachvollziehbarerweise ist ein vergleichbarer Mechanismus beim Lesen nicht möglich.
Es ergibt sich somit selbst in einem *eventual consistency*-Modell Konsistenz (bezüglich des CAP-Theorems), wenn dass Verhältnis Lese-ConsistencyLevel : Schreib-ConsistencyLevel

1. ONE : ALL,
2. QUORUM : QUORUM oder
3. ALL : ONE

ist. Je höher das ConsistencyLevel, desto höher ist verständlicherweise die Latenz eines Zugriffs.

Ungeachtet des verwendeten Lese-ConsitencyLevels wird durch einen Lesezugriff stets ein Gossip-Treffen der für den Datensatz zuständigen Replika-Knoten ausgelöst. Dieser Vorgang wird als ReadRepair bezeichnet. Hierdurch entspricht das zweite Ergebnis, bei einem zweimaligen Lesen mit ConsistencyLevel ONE, bei hinreichend großem Zeitabstand zwischen den Zugriffen, einem einmaligen ALL-Lesen – sofern alle Replika-Knoten verfügbar sind.

Mit Rückblick auf die Ausführungen zur Verfügbarkeit der Speicher-Knoten im Kapitel Entwurf wird für die Sammel-Software ein Schreib-ConsistencyLevel ANY empfohlen. Selbst wenn alle Replika-Knoten offline sind, existiert so trotzdem noch eine reale Wahrscheinlichkeit, dass der Fehlerbericht später in die Datenbank eingetragen wird. Das Verfallen eines HintedHandoffs und somit des eingereichten Fehlerberichts kann wie erwähnt toleriert werden.
Für die Analyse-Software wird die Kombination eines Lese-ConsistencyLevels ONE und dem vorgeschlagenen Caching-Mechanismus empfohlen. Wird in hinreichend großem Zeitabstand mit dieser Methode ein zweites Mal gelesen, so liefert die Datenbank den abgefragten Datensatz nach Korrektur durch das ReadRepair zurück – also nach der Synchronisation aller momentan verfügbaren Replika-Knoten.
Es wird schnell klar, dass in mit diesen Empfehlungen keine Konsistenz gewährleistet werden kann, als Zugeständnis an die geringe Verfügbarkeit der einzelnen Speicher-Knoten.

## 4.3 Umsetzung

### 4.3.1 Windows Error Reporting

Für die Erkennung von Softwarefehlern und die Sammlung der benötigten PEDIs wird in dieser Arbeit die systeminhärente Infrastruktur von Windows genutzt: Das Windows Error Reporting. Die Fehlerverarbeitung verläuft in etwa folgendermaßen – gemäß den Standardeinstellungen von Windows 7:

1. In einem laufenden Prozess tritt ein Fehler auf.
2. a) Das Betriebssystem erkennt den Fehler – etwa bei Abstürzen.

   b) Das Programm ruft WER auf – wenn der Fehler intern erkannt wird.
3. Das Betriebssystem pausiert den Prozess.
4. WER sammelt die Details – Programm-Metainformationen, geladene Module et cetera, sowie einen Minidump.
5. Aus den Details wird wird ein Fehlerbericht in Form einer Report.wer-Datei erstellt und mit dem Dump in die so genannte ReportQueue eingereiht.
6. Der Benutzer wird nach Einverständnis für das Versenden des Berichts gefragt.
7. Bei Zustimmung kontaktiert der Computer den WER-Server und überträgt den Bericht.
8. Gegebenenfalls bittet der WER-Server den Computer um den Dump, in einem solchen Fall wird der Benutzer mit einer weiteren Nachfrage konfrontiert.
9. Nach Beenden der Verbindung wird die Datei Report.wer aus der ReportQueue in das so genannte ReportArchive verschoben, der Dump wird direkt gelöscht.
10. Wurde der Fehler vom Betriebssystem erkannt und war der Fehler kritisch (Absturz), wird der Prozess beendet und gegebenenfalls neu gestartet.

ReportQueue und ReportArchive sind dabei Ordner unterhalb von

- C:\Users\<Benutzername>\AppData\Local\Microsoft\Windows\WER\
  für Berichte von Programmen, welche mit Nutzerrechten ausgeführt werden, oder
- C:\ProgramData\Microsoft\Windows\WER\
  für Berichte von Programmen, welche mit Administratorrechen ausgeführt werden.

Sie enthalten je aufgetretenem Fehler einen Unterordner, welcher die jeweiligen Dateien enthält.

Diverse Schritte dieses Ablaufs lassen sich dabei in der Registry auf globaler oder auf Nutzerebene verändern. Die Beschreibung der meisten Einstellungsmöglichkeiten findet sich unter [WER Einstellungen].
Zu finden sind die Registry-Einträge für das Windows Error Reporting unter:

- „HKEY_CURRENT_USER\
  Software\Microsoft\Windows\Windows Error Reporting“
  Für den aktuellen Benutzer
- „HKEY_LOCAL_MACHINE\
  Software\Microsoft\Windows\Windows Error Reporting“
  Für systemweite Einstellungen

Eine Stolperfalle ist hier, dass unter den 64-Bit-Varianten von Windows für manche Belange „zwei verschiedene“ Registries existieren. Unter den Software-Schlüsseln gibt es dort jeweils einen Unterschlüssel Wow6432Node. Unterhalb dieser Unterschlüssel existiert je eine zweite Software-Hierarchie exklusiv für 32-Bit-Anwendungen. Registryzugriffe von 32-Bit-Anwendungen werden von Windows für manche Schlüssel transparent auf die jeweiligen Wow6432Node-Unterschlüssel umgeleitet, andere nicht, wiederum andere Schlüssel sind Verknüpfungen zu ihrem 64-Bit-Pendant. Der Mechanismus kann unter [Wow6432Node] nachgelesen werden.
Für diese Arbeit ist lediglich relevant, dass beim Absturz eines 32-Bit-Programmes die WER-Einstellungen aus folgenden Schlüsseln gelesen werden:

- „HKEY_CURRENT_USER\
  Software\Wow6432Node\Microsoft\Windows\Windows Error Reporting“
  Für den aktuellen Benutzer
- „HKEY_LOCAL_MACHINE\
  Software\Wow6432Node\Microsoft\Windows\Windows Error Reporting“
  Für systemweite Einstellungen

Es ist mit den folgenden zwei Registry-Einträgen möglich, den Abarbeitungsvorgang beim Auftreten eines Fehlers noch vor der Nachfrage beim Benutzer abzubrechen und alle Daten in der ReportQueue zu belassen.

**DontShowUI = 1** Unterbindet den „Fehlerbericht einschicken?“-Dialog. Keine Nachfrage, also auch kein Senden.

**ForceQueue = 1** Forciert das Abspeichern der Berichte und des Dumps in der ReportQueue.

Wichtig ist dies vor allem, um das Löschen des Dumps zu verhindern, der nicht ins ReportArchive übertragen wird.
Es gibt jedoch keine dokumentierte Möglichkeit, den Vorgang später fortzuführen und die Berichte in der ReportQueue abzuarbeiten. Da diese Methode also die eigentliche Funktionalität des Windows Error Reporting gegenüber Microsoft aushebelt, ist von dieser abzusehen.

Als zweite Möglichkeit findet man unter [WER Einstellungen] die Option, nicht nur den Bericht, sondern auch den Dump im ReportArchive zu speichern. So könnte WER seine Arbeit normal ausführen und alle benötigten Daten währen im Nachhinein leicht abrufbar. Erreicht wird dies laut der genannten Quelle über die Option „ConfigureArchive = 2“. Diese hatte in der Praxis jedoch wider Erwarten keinen Effekt.

Als dritte Lösung bietet WER die Möglichkeit, die Dumps aus den Berichten separat zu speichern. Art des Dumps und Speicherort sind dabei sogar frei konfigurierbar. Auf diese Weise kann man den Bericht aus dem ReportArchive verwenden und den Dump von dem separaten Speicherort beziehen. Leider funktioniert dieses Verfahren nur für Programme, die mit Benutzerrechten ausgeführt werden, während WER selbst bekanntermaßen auch Fehler von Systemprogrammen und dem Betriebssystem selbst behandeln kann. Da die oben angerissenen Lösungsvorschläge jedoch nicht praktikabel sind, beschränkt sich die Sammlung von Speicherabbildern hier auf User-Mode-Programme. Die Berichte selbst werden ohne diese Beschränkung erstellt.
Zum Sammeln von User-Mode-Dumps muss unter den erwähnten Registry-Schlüsseln für Windows Error Reporting ein Unterschlüssel LocalDumps angelegt werden. Sobald dieser existiert, werden Dumps gemäß den Standardeinstellungen gespeichert. Hier können folgende Einstellungen vorgenommen werden:

**DumpType** 1 (Minidump, Standard), 2 (voller Dump) oder 0 (angepasster Dump)

**CustomDumpFlags** Bei „DumpType = 0“ können Parameter für Custom Dumps angelegt werden, die wählbare feste Speicherbereiche zusätzlich zum Stacktrace beinhalten – Vergleich Unterabschnitt Datenschutz im Kapitel Entwurf respektive [MiniDump].

**DumpCount** Maximale Anzahl vorgehaltener Dumps – Standardeinstellung 10. Für diese Arbeit nicht relevant, da die Löschung von Dumps von der erstellten Software übernommen wird.

**DumpFolder** Speicherort – Standardeinstellung C:\Users\<Benutzername>\AppData\Local\CrashDumps

Zu beachten sind die auf dem jeweiligen Computer von den Standardeinträgen abweichenden Einstellungen, die absichtlich verändert worden sein können. Einen abweichenden Speicherort kann die erstellte Software aus der Registry auslesen und verwenden. DumpType wird entsprechend den Einstellungen in der erstellten Software überschrieben.

### 4.3.2 Events in .NET

Der folgende Unterabschnitt soll beispielhaft verdeutlichen, wie die Komponenten der im Rahmen dieser Arbeit erstellten Software ereignisbasiert kommunizieren.

Sender können bei Events Argumente übergeben. Diese Argumente sind in .NET durch die Klasse EventArgs realisiert. Nur Objekte diesen Typs oder dessen Kindklassen können bei Ereignissen gesendet werden. Dabei enthalten Objekte der EventArgs-Klasse selbst keine Daten. Dies ist sinnvoll, wenn die Tatsache, dass das Event ausgelöst wurde, die einzige Information ist, die kommuniziert werden muss. Für die Zwecke dieser Arbeit erweitern mehrere Klassen EventArgs, um Daten über die Events kommunizieren zu können.
Bei Änderung der Konfiguration über den UserConnector löst dieser beispielsweise ein Event aus, dessen Argument ein Objekt der Klasse ConfigChangedEventArgs aus Abbildung 4.3 ist.

```
1  class ConfigChangedEventArgs : EventArgs
2  {
3    public readonly int Loglevel, DumpType, Consent;
4
5    public ConfigChangedEventArgs(int loglevel, int dumpType, int consent)
6    {
7      this.DumpType = dumpType;
8      this.Loglevel = loglevel;
9      this.Consent = consent;
10   }
11 }
```

Abbildung 4.3: Argumente von Events

Ein Event in .NET hat den Typ eines Delegats. Die Signatur des Delegats wiederum enthält üblicherweise das Senderobjekt und ein Ereignisargument als Parameter, der Rückgabetyp ist void.
Es existieren zwei vordefinierte Delegate, die man anstelle selbst definierter Delegate verwenden kann:

- `public delegate void EventHandler(Object sender, EventArgs e)`
- `public delegate void EventHandler<T>(Object sender, T e)`
  `where T : EventArgs`

Ersteres erlaubt nur die Verwendung eines EventArgs-Objekts als Argument. Letzteres erlaubt mittels Generics die Verwendung eigener, von EventArgs abstammender Argumente.
Das in Abbildung 4.4 gezeigte Beispiel soll die Definition und das Senden von Events mit vordefinierten Delegaten verdeutlichen. Der UserConnector löst hier ein Event aus, wenn

der Benutzer die Konfiguration der Software verändern möchte. Das Event ist public, damit sich externe Empfänger registrieren können.

```
1  class MyUserConnector : IUserConnector
2  {
3    // Definiert das Event. Nutzt dabei das Delegat
4    // public delegate void EventHandler<T>(Object sender, T e)
5    public event EventHandler<ConfigChangedEventArgs> ConfigChangedEvent;
6
7    private void processUserInput(Object input)
8    {
9      ...
10     // Erstellen des Ereignisarguments.
11     // loglevel, dumptype und consent wurden aus der Benutzereingabe gelesen.
12     ConfigChangedEventArgs ccea = new ConfigChangedEventArgs(loglevel,
           dumptype, consent)
13     // Auslösen des Events.
14     ConfigChangedEvent(this, ccea);
15   }
16 }
```

Abbildung 4.4: Senden von Events

Potentielle Empfänger eines Ereignisses müssen nun zwei Dinge tun. Erstens eine Methode mit einer dem Delegat genügenden Signatur zur Verfügung stellen, welche das Ereignis verarbeitet. Zweitens müssen sie sich beim Sender als interessierte Empfänger registrieren. Die sei verdeutlicht durch das Codebeispiel aus Abbildung 4.5.

```
1  class Controller
2  {
3    // Methode zur Verarbeitung des Events.
4    private void configChangedEventHandler(object sender,
         ConfigChangedEventArgs ccea)
5    {
6      ...
7    }
8
9    public static void Main(string[] args)
10   {
11     // Hinzufügen der Methode zu den interessierten Empfängern.
12     UserConnector.ConfigChangedEvent += new
13       EventHandler<ConfigChangedEventArgs>(configChangedEventHandler);
14   }
15 }
```

Abbildung 4.5: Empfangen von Events

Ein Event erzeugt bei jedem Empfänger einen separaten Kontrollfluss (engl.: Thread).

### 4.3.3 Windows EventLog

Das EventLog unter Microsoft Windows ist ein zentraler Dienst, unter welchem Programme einen Mitschnitt anlegen können – vergleichbar mit syslog unter unixoiden Systemen. Laut [EventLog] verfügt Windows über mehrere Protokolle, unter anderem Application (für normale Anwendungen), Security (u.A. für Login-Versuche) und System (für Systemkomponenten). Weitere existieren, sind jedoch unter den verschiedenen Windowsversionen (ab XP) uneinheitlich. Einträge unter diesen Protokollen haben die folgenden für diese Arbeit relevanten Felder: Beschreibung (freier Text), Datum, Uhrzeit, Quelle (Programm, welches das Ereignis eingetragen hat) und Event-ID. Die Event-ID eines Ereignisses ist eine frei wählbare Zahl. Sie wird meist für eine Kategorisierung des Eintrags genutzt.

Anhand des PlattformConnectors veranschaulicht der Code in Abbildung 4.6, wie man jedwede im Application-Protokoll eintreffenden Einträge empfängt.

```
1  using System.Diagnostics;
2
3  class MyPlattformConnector : IPlattformConnector
4  {
5    public PlattformConnector()
6    {
7      EventLog el = new EventLog("Application");
8      el.EntryWritten += new EntryWrittenEventHandler(eventLogHandler);
9      el.EnableRaisingEvents = true;
10   }
11
12   private void eventLogHandler(object sender, EntryWrittenEventArgs e)
13   {
14     ...
15   }
16 }
```

Abbildung 4.6: Abonnieren des EventLogs

Der Prozess funktioniert analog dem im vorangehenden Unterabschnitt vorgestellten Mechanismus. Das EventLog sendet jedoch nur dann Ereignisse an den Empfänger, wenn dieser zusätzlich EnableRaisingEvents auf true gesetzt hat (vgl. Zeile 9).
Für diese Arbeit sind Einträge nach zwei Mustern relevant. Die für das Auftreten eines zu berichtenden Fehlers, sowie die Veränderung an der installierten, registrierten Software – zum Beispiel durch Ausführen einer setup.exe oder uninstall.exe.

Beim Eintreten eines Softwarefehlers schreiben zwei Programme dieses Ereignis in das Application-Protokoll: Application Error und Windows Error Reporting. Das Beschreibungsfeld von letzterem enthält den Pfad zu dem Ordner, in welchem der Fehlerbericht gespeichert wurde, der von der Sammel-Software verarbeitet werden muss.
Entry.ReplacementStrings ist in den übergebenen EntryWrittenEventArgs ein Array von

Strings, von denen jedes Element einer Zeile des Beschreibungsfelds entspricht. Dieses wird durch den in Abbildung 4.7 aufgeführten Code nach dem entsprechenden Pfad durchsucht.

```
1  class MyPlattformConnector : IPlattformConnector
2  {
3    private void eventLogHandler(object sender, EntryWrittenEventArgs e)
4    {
5      // Filtern von Protokoll-Einträgen nach der Quelle.
6      if (e.Entry.Source.Equals("Windows Error Reporting"))
7      {
8        string reportPath = null;
9        // Durchsuchen des Beschreibungstextes nach dem Pfad des Berichts.
10       foreach (string s in e.Entry.ReplacementStrings)
11       {
12         if (s.Contains("\\Microsoft\\Windows\\WER\\"))
13         {
14           reportPath = s + "\\Report.wer";
15           break;
16         }
17       }
18       if (reportPath != null)
19       {
20         CrashOccuredEventArgs coea = parseWerReport(reportPath);
21         // Erzeugen des Events, welches der Controller empfängt.
22         CrashOccuredEvent(this, coea);
23       }
24     }
25   }
26 }
```

Abbildung 4.7: EventLog-Einträge zum Windows Error Reporting behandeln

Das Programm MsiInstaller schreibt jedwede Veränderung an der installierten Software, auch die Intention einer solchen, ins Application-Protokoll. Den vorgemerkten Beginn einer Veränderung, deren Beginn, das Vormerken des Beendigens, das Beendigen mit oder ohne Erfolg, und vieles mehr. Hierbei reicht es also nicht mehr aus, den bloßen Namen der Anwendung aus dem Logeintrag zu lesen. Statt jedoch den Beschreibungstext zu analysieren, welcher nach Systemsprache lokalisiert eingetragen wird und daher schwer allgemeingültig zu verarbeiten ist, kann man einfach auf die Event-IDs zurückgreifen, die das Ereignis charakterisieren. Die Schnittmenge der Event-IDs, welche laut [MSI IDs] relevant sind, und empirischen Beobachtungen ist:

**1033** Das Produkt wurde durch Windows Installer installiert

**1034** Das Produkt wurde durch Windows Installer entfernt

**1035** Das Produkt wurde durch Windows Installer rekonfiguriert

Daraus ergibt sich die Implementierung in Abbildung 4.8, die Event-ID wird dabei in .NET als InstanceId bezeichnet.

```
1  class MyPlattformConnector : IPlattformConnector
2  {
3    private void eventLogHandler(object sender, EntryWrittenEventArgs e)
4    {
5      if ((e.Entry.InstanceId == 1033 || e.Entry.InstanceId == 1034 ||
6           e.Entry.InstanceId == 1035) && e.Entry.Source.Equals("MsiInstaller"))
7      {
8        // UpdateCompnents() löst wenn nötig ein ComponentsChangedEvent aus.
9        UpdateComponents();
10     }
11   }
12 }
```

Abbildung 4.8: EventLog-Einträge zum MSI Installer behandeln

### 4.3.4 Apache Cassandra

Neben der Möglichkeit der direkten Verwendung der Cassandra-Thrift-Bibliotheken existiert eine Vielzahl programmiersprachenspezifischer Wrapper um diese. Die Wrapper bieten eine einfachere Handhabung und meist eine bessere Integration in die jeweilige Programmiersprache.
In dieser Arbeit wurde [FluentCassandra] für die Implementierung des CrowdConnectors verwendet. Diese Bibliothek bot unter den für C# verfügbaren Wrappern bei einem groben Vergleich die intuitivste Bedienung, einen schnellen Einstieg und gute Integration in C#, etwa durch Verwendung des dynamic-Schlüsselworts und LINQ. Zudem ist das Projekt OpenSource und verspricht Mono-Kompatibilität.
Einen ersten Fallstrick findet man in den unterschiedlichen Versionen von FluentCassandra. Die zum Zeitpunkt des Erstellens der Arbeit aktuelle Revision des Git-Repositories ließ sich nicht kompilieren, daher wurde die zum Download bereitstehende Version 0.7.0.3 verwendet. FluentCassandra 0.7.* ist jedoch nur mit den Cassandra-Versionen 0.7.* kompatibel, verwendet wurde Cassandra 0.8.10. Dokumentiert ist dieser Fakt nicht, einzig ein Kommentar unter einem nicht mehr auffindbaren Blog-Eintrag verweist auf diese Tatsache. Weiß man von diesem Umstand nicht, sind die bei komplexeren Datenbankzugriffen aufgetretenen Exceptions kaum erklärlich.
Der nächste Mangel ist die fehlende Dokumentation der Bibliothek. Es existiert keine Online-Dokumentation und der Quellcode ist gänzlich unkommentiert (wodurch auch die IntelliSense-Funktion von VisualStudio ungenutzt bleibt). Da man mit den Beispielen auf dem Blog des Entwicklers und im Git-Repository jedoch ein sehr breites Spektrum an Anwendungen abgedeckt hat, fiel dieser Umstand zunächst nicht auf. Auch hier treten die Mängel erst bei der Erstellung komplexerer Methoden zu Tage.
Zuletzt lässt sich mit der verwendeten Version von FluentCassandra das Consistency-

Level für Lese- und Schreibzugriffe nicht verändern. Praxistests haben ergeben, dass für beide Zugriffsarten ALL verwendet wird.
Zusammenfassend lässt sich sagen, dass FluentCassandra in Aussicht stellt, eine umfassende und einfach zu handhabende Bibliothek für die Verwendung von Apache Cassandra in .NET zu werden. Zum Zeitpunkt des Schreibens dieser Arbeit ist ihr Status jedoch als frühe Betaphase anzusehen.
Leider war die Integration der Bibliothek schon so weit fortgeschritten, dass ein Umschwenken auf eine anderen Bibliothek im zeitlichen Rahmen der Arbeit nicht mehr möglich war.

Für den Fortgang dieser Arbeit soll an dieser Stelle eine Cassandra-Konfiguration „intelligent geraten“ werden. Die resultierende Konfiguration wird im Abschnitt zur Leistungsbewertung für den Test verwendet. Es existieren mehrere Konfigurationsdateien.

Die Datei cassandra-env.sh enthält als für diese Arbeit wichtige Option die Einstellmöglichkeit für die maximale Größe des Heaps im Arbeitsspeicher. Die Standardeinstellung für die Heapgröße ist $RAM/2$. Für Endbenutzersysteme würde dies jedoch zu untragbaren Einschränkungen führen. Ein empirisch getestetes Minimum liegt hier bei 500 MB.

Die Datei cassandra-topology.properties ist zu leeren. Sie enthält eine Beschreibung der Netzwerktopologie. Da die intendierte Netzwerktopologie jedoch dynamisch ist, können keine Aussagen getroffen werden.

cassandra.yaml ist die zentrale Konfigurationsdatei. Im Folgenden sind die Einträge aufgeführt, die für diese Arbeit von den Standardeinstellungen abweichend gewählt wurden:

**cluster_name** Bezeichner für die erste Hierarchieebene der Datenstruktur. Frei wählbar, muss aber wie beschrieben eindeutig sein, damit die Instanzen miteinander kommunizieren.

**initial_token: „“** Der für einen Datensatz zuständige Knoten wird durch das so genannte Token bestimmt – einer Art ID für den Knoten. Wird kein Token für einen Knoten festgelegt, wählt dieser beim ersten Beitritt ins System dieses so, dass er die Hälfte des Datenbestandes des am meisten belasteten Knotens übernimmt.

**auto_bootstrap: true** Ein neuer Speicher-Knoten tritt automatisch dem System bei und übernimmt Datensätze. Zu beachten ist, dass auto_bootsrap bei der allerersten Instanz false sein muss, da sie logischerweise keinen anderen Instanzen beitreten kann.

**max_hint_window_in_ms: 172800000** (= 48 Stunden, Standardeinstellung 30 Minuten) Zeit, die ein HintedHandoff vorgehalten wird. Wird binnen dieser Zeit kein zuständiger Replika-Knoten aktiv, wird der Datensatz verworfen.

**partitioner: ByteOrderedPartitioner** Der ByteOrderedPartitioner ordnet die Datensätze dem Knoten zu, dessen Token dem RowKey am nächsten ist. Der RandomPartitioner erzeugt Hashwerte der RowKeys und ordnet nach der Nähe dieser zu

den Tokens. Der erste Partitioner unterstützt Bereichsabfragen, da die Einträge sortiert abgelegt sind. Letzterer verspricht eine gleichmäßige Verteilung, gegeben durch die näherungsweise zufällige und gleichverteilte Streuung der Hashwerte. Schlüssel sind oft an ihr Datum gebunden – beispielsweise bei der Sortierung von Personen nach dem ersten Buchstaben des Nachnamens. Eine näherungsweise Gleichverteilung der Schlüssel auf dessen Wertebereicht ist daher nicht immer gegeben. Da in dieser Arbeit die RowKeys (respektive die IDs) jedoch bereits Hashwerte sind, kann dieser Zwischenschitt ausgelassen werden, ohne die gleichmäßige Verteilung der Datensätze auf die Knoten zu beeinflussen.

**seeds: <Liste>** Eine Liste von IP-Adressen oder Hostnamen anderer Datenbankinstanzen, zu denen sich diese Instanz initial verbinden kann.

**concurrent_reads: 16** Zahl paralleler Lesevorgänge, Empfehlung laut Kommentar in der Konfigurationsdatei ist das 16-fache der Anzahl an Festplatten im System. In Annahme haben Endbenutzerrechner nur eine Festplatte. Ein Reduzieren dieser Zahl verringert die Festplattenlast, erhöht jedoch die Latenz.

**concurrent_writes: 16** Zahl der parallelen Schreibvorgänge, Empfehlung laut Kommentar in der Konfigurationsdatei ist die achtfache Anzahl an Prozessorkernen. Per Annahme haben Endbenutzerrechner einen Dualcore Prozessor. Da Schreibaufträge eher CPU- und RAM- als IO-intensiv sind, richtet sich dieser Wert nicht nach der Anzahl der Festplatten.

**rpc_server_type: async** Erstellt nur einen Thread, welcher RPC-Verbindungen verarbeitet, um Ressourcen zu sparen.

**rpc_timeout_in_ms: 30000** (= 30 Sekunden, Standardeinstellung 10 Millisekunden) Erhöhung des Limits für die Zeitüberschreitung bei RPC-Verbindungen als Tribut an die hohe Latenz der Verbindung über das Internet.

**dynamic_snitch_update_interval_in_ms: 10000** (= 10 Sekunden, Standard 100 Millisekunden) Ein *Snitch* in Cassandra ist für das Routing von Anfragen innerhalb des Netzwerkes zuständig. Der Knoten, an den eine Anfrage gerichtet wurde, muss bekanntermaßen nicht jener sein, der für den angeforderten Datensatz zuständig ist. Beim standardmäßig aktivierten dynamischen SimpleSnitch protokolliert Cassnadra die Lese-Latenzen von Knoten um Lese-Anfragen auf schnelle Replika-Knoten umzuleiten. Zur Entlastung der Verbindung wird dieses Prüf-Intervall gesenkt. Andere Snitches können die Netzwerk-Struktur für dieses Routing berücksichtigen, wozu eine entsprechende cassandra-topology.properties-Datei benötigt wird.

Der für die Daten dieser Arbeit erstellte Keyspace erhält folgende Einstellungen:

**ReplicaPlacementStrategy: SimpleStrategy** Nach [DataStax] entscheidet diese Strategie, welche Knoten die Replikate eines Knotens erhalten. Die verwendete Strategie wählt den Teilnehmer mit dem nächsthöheren Token.

Komplexere Strategien, wie etwa die NetworkTopologyStrategy, erlauben es, Replikate in anderen Rechenzentren zu platzieren, um den Ausfall eines einzelnen Rechenzentrums zu kompensieren. Für solche Strategien müssten wiederum die Netzwerkinfrastrukturen in der Datei cassandra-topology.properties definiert werden.

**ReplicationFactor: 8** Wie bereits im Kapitel Entwurf erläutert, wird ein Replikationsfaktor von acht gewählt, welcher einer Verfügbarkeit der Datensätze von 90% entspricht.

# 5 Leistungsbewertung

Ein realitätsnahes Testen der vorgestellten Software im Rahmen dieser Arbeit war nicht realisierbar, weil die Masse an Testrechnern, für welche diese Arbeit ausgelegt ist, nicht verfügbar war. Abgezielt wird auf Millionen Heimrechner, verbunden über haushaltsübliche Breitbandanschlüsse, die unter Windows 7 laufen. Zur Verfügung stand jedoch nur ein lokales Netzwerk mit 57 Linux-Rechnern. Zur Rechtfertigung des auf dieser Basis ausgeführten Tests muss vor allem die Annahme getroffen werden, dass Cassandra unter dem eingesetzten OpenJDK 6 für Linux ein ähnliches Lastverhalten aufweist, wie unter dem verbreiteten Oracle Java SE 6 für Windows.

## 5.1 Bewertungskriterien

Bei der Betrachtung der als Speicher-Knoten in Frage kommenden Teilnehmer wurde im Kapitel Entwurf schnell klar, dass ein Verhältnis von Sammel- zu Speicher-Knoten nicht im Verhältnis 1:1 zu erwarten ist: Weniger als 10% aller in Frage kommenden Rechner bieten alleine die gewünschte Online-Zeit. Dies lässt die Frage nach der Bereitschaft der Teilnehmer, freiwillig ihre Rechner zu Verfügung zu stellen, aber noch außen vor. Für die noch verbleibenden potentiellen Teilnehmer wird sich dann außerdem die Frage stellen, wie stark ihr System durch das Speichern, Senden und Empfangen von Daten ausgelastet wird. Benutzer lastintensiver Anwendungen werden beispielsweise kaum ihre Rechnerleistung zur Verfügung stellen wollen, für andere sollte die Verwendung der Speicher-Software idealerweise keine merklichen Einschränkungen zur Folge haben.

Damit einher geht die Frage, wie viele Datensätze pro Zeiteinheit erzeugt werden, die von den Speicher-Knoten entgegengenommen werden müssen. Oder anders: Wie oft tritt auf dem durchschnittlichen Computer pro Tag ein Fehler auf? Jeder Computer für sich erfährt dabei im Normalfall kaum ständig Fehler, eine fehlerhafte Software auf einem Computer wird aber potentiell sehr häufig gestartet.
Laut [Kinshumann et al., 2011] Seite 114 verzeichnen die 60 im Einsatz befindlichen WER-Server im Durchschnitt 100 Millionen Fehlerberichte pro Tag. Rund 2,2 Milliarden Menschen nutzen laut [Internetnutzer] weltweit das Internet. Bei darunter schätzungsweise mindestens 1,5 Millarden Windows-Rechnern ergäbe dies 0,0667 Berichte pro Tag und Rechner beziehungsweise einen Bericht je 15 Rechner pro Tag.

Als nächstes müssen die Cassandra-Seeds betrachtet werden, also jene Instanzen der verteilten Datenbank, zu denen sich ein Speicher-Knoten initial verbindet, wenn er online

geht. Im gleichen Zug müssen auch jene Instanzen betrachtet werden, zu denen sich die Sammel- und Analyse-Knoten verbinden.
Im schlimmsten Fall gibt es eine öffentliche Liste sehr weniger, bekannter Speicher-Knoten, welche die Verbindungen aller anderen Knoten entgegennehmen. Die erwartete Last auf diesen im Folgenden so genannten Seed-Knoten ist ungleich höher.
Da Cassandra-Instanzen die Seed-Knoten nur zu einer initialen Verbindung benötigen, ihre nächsten Verbindungspartner gemäß des Gossip-Protokolls aber zufällig neu wählen, handelt es sich hierbei um jeweils einmalige Belastungen. Im Gegensatz dazu kommunizieren die Sammel- und Analyse-Knoten für die Dauer ihrer Verbindung sämtliche Daten über Seed-Knoten mit dem Daten-Grid.

Überhaupt besteht Klärungsbedarf über die Verwendung von Cassandra in dem gezeichneten Netzwerk. Die erprobten Nutzungsszenarien weichen deutlich vom intendierten Nutzungsszenario ab. Im Gegensatz zu üblicherweise grob 90% Verfügbarkeit je Knoten werden hier 25% angenommen. Statt über ein lokales Netzwerk hoher Bandbreite und geringer Latenzen sind die Knoten in diesem Netzwerk über asymmetrische haushaltsübliche Breitbandanschlüssen verbunden. Des Weiteren ist das Zurverfügungstellen der Datenbank in der Regel eine der Hauptaufgaben der teilnehmenden Rechner, während die Datenbank auf Speicher-Knoten eine Hintergrundanwendung sein soll.

Aus diesen Aussagen ergeben sich mehrere Kriterien für die Leistungsbewertung.

### 5.1.1 Kriterium 1: Ist das Betreiben der Sammel-Software tragbar?

Die Last, welche durch das alleinige Sammeln von Fehlerberichten entsteht, ist vernachlässigbar. Am Beispiel des Windows Error Reporting, welches sogar auf Smartphones seit Windows Mobile 5.0 zum Einsatz kommt, sei dies verdeutlicht. Begründet liegt dies in der Tatsache, dass das Betriebssystem ohne merklichen Mehraufwand einige Fehlerarten (vor allem Abstürze) leicht erkennen kann und die entsprechenden Komponenten zur Fehlerverarbeitung erst bei Bedarf aktiv werden.
Ein breit angelegter Test kann an dieser Stelle daher entfallen. Es würde lediglich auf eine schlechte Implementierung hinweisen, wenn eine für den Endbenutzer nicht tragbare Systemlast gemessen würde. Zudem würde ein entsprechender Test keine interessanten Ergebnisse und neuen Erkenntnisse liefern.

Durch Beobachtungen mittels des Taskmanagers während der Entwicklungsphase der im Rahmen dieser Arbeit erstellten Sammel-Software wurde keine messbare CPU-Last und ein Arbeitsspeicherverbrauch von weniger als 15 MB während der passiven Phasen festgestellt. Während des Sendens eines Fehlerberichtes betrug die CPU-Last auf einem Core2 Duo 2,88 GHz für weniger als eine Sekunde 15%, der Arbeitsspeicherverbrauch stieg für mehrere Sekunden um 2 MB zuzüglich gegebenenfalls der Größe des verarbeiteten Arbeitsspeicherabbilds.

### 5.1.2 Kriterium 2: Ist das Betreiben der Speicher-Software tragbar?

Um zu klären, wie hoch die Systembelastung auf einem Speicher-Knotens ist, muss zunächst Klarheit über das Anwendungsszenario geschaffen werden. Wichtig hierfür sind vor allem die Fragen, wie das Verhältnis zwischen Sammel- und Speicher-Knoten ist, wie viele der Speicher-Knoten Seed-Knoten sind und wie viele Fehlerberichte ein Sammel-Knoten pro Zeiteinheit einreicht.

Wie festgestellt wurde, ist das Verhältnis zwischen Speicher- und Sammel-Knoten bestenfalls 1:10, eher deutlich geringer. Hier müssen verschiedene Verhältnisse zwischen den Knoten betrachtet werden.
Die Anzahl erwarteter Fehlerberichte liegt bei etwa 0,0667 Berichten pro Computer und Tag. Schlimmstenfalls werden jedoch sehr viele Berichte von sehr wenigen Speicher-Knoten entgegengenommen und erst dann auf das Grid verteilt.

Der resultierende Versuchsaufbau wird im nächsten Abschnitt erläutert.

### 5.1.3 Kriterium 3: Eignet sich Cassandra für die gestellte Aufgabe?

Zur Klärung der Eignung müssten folgende drei Unterpunkte untersucht werden:

- Skaliert Cassandra auch auf 100.000 oder gar 1.000.000 Instanzen?
- Arbeitet das System bei asynchronen 1 bis 100 Mbit/s-Leitungen von Breitbandanschlüssen und vergleichsweise hohen Latenzen zuverlässig?
- Funktionieren die Verteilungs- und Replikationsmechanismen bei Verfügbarkeit der Knoten von 25%?

Wie bereits erwähnt, entsprach die zur Verfügung stehende Testumgebung nicht den Anforderungen, die ein entsprechender Test stellen würde. Aufgrund der Relevanz der Antworten auf diese Fragen wäre ein solcher jedoch der nächste logische Schritt beim weiteren Verfolgen des in dieser Arbeit gezeichneten Konzepts.

## 5.2 Versuchsaufbau

Für die Zwecke des Tests wird die im Rahmen dieser Arbeit erstellte Sammel-Software derart erweitert, dass sie Fehlerberichte zufällig würfelt. Da hierbei kein real existierendes Programm gestartet wird, entfallen die dadurch sonst entstehenden Latenzen und die Software ist in der Lage, eine deutlich höhere Menge an Daten pro Zeiteinheit zu produzieren. Da ausschließlich Linux-Maschinen zur Auswahl standen, während die Software selbst für Windows geschrieben ist, kamen virtuelle Maschinen zum Einsatz – im Folgenden Writer-Knoten genannt.

Zur Simulation von Analyse-Knoten wurde ein Bash-Skript geschrieben, welches den kompletten Datenbestand ausliest – im Folgenden als Reader-Knoten bezeichnet.

Die Testgeräte waren die 57 Computer in den zwei Rechnerpools der Abteilung Informatik an der Universität Trier, im Folgenden bezeichnet mit poolpc10 bis poolpc66, und ein weiterer Computer, der für den Fernzugriff eingerichtet war, im Folgenden bezeichnet mit poolpc-remote. Auf den Computern lief Debian Testing (Wheezy) x86_64 unter Kernel 3.0. Apache Cassandra 0.8 lief unter dem dort Installierten OpenJDK 6b24. Die Sammel-Software lief über ein virtualisiertes Windows mit VirtualBox 4.1. Zum Sammeln der Systemstatistiken wurde dstat 0.7.2 verwendet.
Die in dieser Arbeit aufgeführten Statistiken stammen von Computern mit Intel Core2 Duo 3 GHz Prozessor und 4 GB Arbeitsspeicher. poolpc41-50, deren Statistiken sich auf dem beigelegten Datenträger befinden, verfügten über einen Intel Core2 Quad 2,50 GHz und 8 GB Arbeitsspeicher.
Die virtuellen Maschinen verfügten über einen CPU-Kern, 1 GB Arbeitsspeicher und eine 20 GB große virtuelle Festplatte in Form eines Festplattenimages. Das Festplattenimage befand sich auf einer 19 GB großen lokalen Partition. Der tatsächlich verwendete Speicherplatz betrug knapp 10 GB.
Das verwendete Windows in den virtuellen Maschinen entsprach dem für die Entwicklung verwendeten Windows 7 Professional x64 SP1. Unter diesem wurde ausschließlich die .NET-Laufzeitumgebung, sowie die im Rahmen dieser Arbeit erstelle Software installiert.
Die Konfiguration der Software war Loglevel 2, Dumptyp 0 und Consent 1. Da für die Behebung einer Fehlerursache nur vergleichsweise wenige Speicherabbilder benötigt werden, wurden diese im Test außer Acht gelassen. Durch die Einrichtung im Autostart und die Verwendung von Consent 1 war kein manueller Eingriff in die virtuellen Maschinen zur Laufzeit notwendig. Vom Befehl, die Maschine zu starten, bis zum Senden der ersten Daten verstrich eine Zeit von zirka sieben Minuten.
Die Konfiguration der Speicher-Knoten entsprach der im Kapitel Implementierung ausgeführten. Beim Testen des Versuchsaufbaus ergab sich das Problem, dass zwei Instanzen das gleiche Token zugewiesen bekamen, wenn sie sich zum gleichen Zeitpunkt mit dem Seed-Knoten verbanden. Cassandra war zwar in der Lage, diesen Missstand zu erkennen, aber alle Knoten mit dem gleichen Token wurden dann, bis auf einen, nicht verwendet. Aus diesem Grunde wurden die einzelnen Instanzen in Sieben-Minuten-Abständen gestartet.

Die Aufgaben der Rechner verteilten sich wie folgt:

**poolpc-remote** Fernsteuerung der anderen Rechner sowie Reader-Knoten

**poolpc10** Seed-Knoten

**poolpc11-25** Dedizierte Speicher-Knoten

**poolpc26-50** Speicher- und Writer-Knoten

**poolpc51-66** Dedizierte Writer-Knoten

Im Voraus wurde die Datenbank auf poolpc10 mit einem Datenbestand von über einer Millionen Einträgen initialisiert, damit bereits in der Anfangsphase des Experiments beim Beitritt einer Cassandra-Instanz eine nicht vernachlässigbare Menge an Daten ausgetauscht werden konnte.
Die folgenden Aktionen wurden von poolpc-remote via SSH ausgeführt:

- Zunächst wurde auf poolpc10-66 dstat gestartet.
- Im nächsten Schritt wurden die Cassandra-Datenbanken auf poolpc10-20 gestartet – in Sieben-Minuten-Abständen. Dann wurde für 20 Minuten die durchschnittliche Last auf den Speicher-Knoten gemessen, während keine Daten eingespeist wurden.
- Im Anschluss wurden die virtuellen Maschinen der Writer-Knoten auf poolpc26-66 gestartet. Nach dem Start von je 10 Maschinen wurde je 20 Minuten Zeit gelassen, die Last beim resultierenden Verhältnis zwischen Speicher- und Writer-Knoten zu messen.
- Nach dem Start aller 41 Writer wurden sukzessive die Cassandra-Instanzen von poolpc21-50 gestartet, wieder in 10er-Schritten mit anschließenden 20 Minuten Messphase, um herauszufinden, ob durch das Hinzukommen weiterer Knoten die vorhanden entlastet werden.
- Anschließend wurden alle Writer gestoppt.
- Im letzten Schritt wurde mehrmals der Reader gestartet. Zwischen dessen einzelnen Läufen wurden je 10 Datenbankinstanzen gestoppt um die Last verschiedener Verhältnisse zwischen Reader und Speicher-Knoten zu ermitteln.

Testbeginn war um 16:50 Uhr, der Test wurde abgeschlossen um 6:40 Uhr.

## 5.3 Testergebnisse

Bei der Deutung der Testergebnisse sind poolpc19 und poolpc20 nicht zu berücksichtigen, da auf diesen die Cassandra-Instanzen aufgrund eines Fehlers nicht korrekt starteten. Auf poolpc38 und 39 sowie auf poolpc41 und 42 führte ein anderer Student zum Messzeitpunkt lastintensive Aufgaben durch, daher werden die Statistiken dieser Rechner ebenfalls nicht betrachtet. Zuletzt ist anzumerken, dass zwischen 5:30 Uhr und 5:45 Uhr auf allen Systemen lastintensive Aufgaben erledigt wurden, wahrscheinlich durch einen Cron-Job. Die Messdaten zur vierten und fünften Laufphase des Readers bleiben somit außen vor.
Auf poolpc26 bis 66 waren zwischen 18:30 Uhr und 0:00 Uhr die virtuellen Maschinen in Betrieb.

### 5.3.1 Übersicht

Die CPU-Auslastung lag den Großteil der Zeit bei den Speicher-Knoten durchschnittlich unter 5%. Lastspitzen von 10%, in selteneren Fällen auch bis 20%, traten beim Lesen vieler Datensätze durch den Beitritt neuer Cassandra-Instanzen zum Grid und durch den Reader auf. Zeitlich nicht einem Ereignis zuzuordnende Lastspitzen sind auf Cassandra-interne Aufgaben zurückzuführen. Der Seed-Knoten hingegen weist eine deutlich höhere CPU-Auslastung auf und erfährt sehr häufige Lastspitzen mit 10% bis 20% Auslastung.

Die Menge des verwendeten Arbeitsspeichers steigt nach dem Start einer Cassandra-Instanz nahezu streng monoton auf 1,2 GB bis 1,3 GB an und verweilt dort konstant über den restlichen Verlauf. Für den Seed-Knoten gilt selbiges für 1,5 GB.
Cassandra verwendet den Arbeitsspeicher eines Computers als Cache für Festplattenzugriffe.

Bei den Festplattenzugriffen gibt es keine signifikanten Unterschiede zwischen dem Seed-Knoten und den restlichen Cassandra-Instanzen. Die Speicher-Knoten haben häufig Schreib-Peaks, selten Lese-Peaks und außerhalb der Peaks kaum lesende oder schreibende Zugriffe.
Die schreibenden Zugriffe sind unregelmäßig, keinem konkreten Ereignis im Test zuzuschreiben und nutzen die volle Bandbreite des Festplattenanschlusses. Wahrscheinlich handelt es sich um Flushes der Daten aus dem Arbeitsspeicher zur Festplatte. Dies erklärt, warum der Seed-Knoten nicht stärker belastet wird: Obwohl alle Daten über ihn empfangen und weitergeleitet werden, schreibt er nur jene Daten fest, für die er verantwortlich ist. Die Tatsache, dass die Auslastung des Arbeitsspeichers nicht sinkt, lässt sich damit erklären, dass Cassandra den Speicher alloziert lässt – für die nächsten eintreffenden Daten.
Beim Start einer Cassandra-Instanz ist auf genau einer einzelnen anderen Cassandra-Instanz ein Lese-Peak festzustellen. Dies ist darauf zurückzuführen, dass die hinzukommende Cassandra-Instanz die Hälfte der Datensätze der meistbelasteten Instanz übernimmt. Die neu startende Cassandra-Instanz hat einen entsprechenden Schreib-Peak.
Weitere Lese-Peaks stellen die Zugriffe des Readers dar.

Analog zur Belastung der Festplatten erzeugt der Beitritt einer neuen Cassandra-Instanz zum Grid hohe Auslastung der Download-Bandbreite des beitretenden Computers beziehungsweise Upload-Bandbreite des Knotens, der seine Daten an die neue Instanz abgibt – bis zu 80 Mbit/s bei einem 100 Mbit/s-Anschluss. Das Starten des Readers führt zu einer erkennbaren Upload-Auslastung aller Knoten.
Bei den Messwerten ist eine deutlich stärkere Belastung des Seed-Knotens auszumachen, der die Daten aller Writer-Knoten entgegennimmt und verteilt, respektive die Daten aller Speicher-Knoten an den Reader ausgibt. Im Schnitt hat der Seed-Knoten bei eintreffenden Fehlerberichten eine Auslastung von 20 Mbit/s up- und downstream. Während der aktiven Phase der Writer lässt sich bei den anderen dedizierten Speicher-Knoten im Schnitt nur eine Last von 2 Mbit/s up- und downstream feststellen. Es werden sowohl Daten entgegengenommen, als auch Replikas verteilt.

Ungeachtet der angegebenen Durchschnittswerte lasten die Rechner ihre Netzwerkanschlüsse nur punktuell auf die genannten 80 Mbit/s aus.

Der Load aller nicht-Seed-Rechner im 5-Minuten-Durchschnitt ist stets unter 0.05, im 1-Minuten-Durchschnitt existieren Peaks, selten mit einem Load größer als 0.2. Beim Seed-Knoten sind die Peaks deutlich häufiger, wodurch auch der 5-Minuten-Durchschnitt oft die 0.05 überschreitet.
Lesevorgänge erhöhen für ihre Dauer den Load aller Speicher-Knoten auf über 0.1, den des Seed-Knotens auf über 0.2 mit Peaks bis zu 0.5.

Auslagerungsspeicher wurde auf keinem Speicher-Knoten verwendet.

Insgesamt wurden außerhalb der Reader-Phasen auf den Rechnern keine erwähnenswerten Datenmengen von der Festplatte gelesen. Im Schnitt wurden knapp 2 GB geschrieben. Bei bis zu 1 GB tatsächlich vorhandenen Daten ist anzunehmen, dass hier veränderte Datensätze auf der Festplatte aktualisiert wurden – beispielsweise durch hinzugefügte Einträge unter Crash / <CrashID> / Computer. Der Seed-Knoten wich von diesen Daten erwartungsgemäß nicht ab.
Es existierten insgesamt 28 GB Daten auf allen 44 Speicher-Knoten. Beim verwendeten Replikationsfaktor von acht ergeben sich grob 3,5 GB Daten für die über 1.700.000 Datensätze und somit grob 2,1 kB je Datensatz (Durchschnitt für alle (Super)Column-Families).

Die empfangene Datenmenge via Netzwerk beläuft sich über den gesamten Zeitraum auf 1,5 bis 2 GB. Dies deckt sich mit der auf die Festspeicher geschriebenen Datenmenge, abzüglich jener Operationen, die noch vor dem Festschreiben der Datensätze im Arbeitsspeicher getätigt werden konnten. Die vom Seed-Knoten empfangene Datenmenge hingegen beläuft sich auf 33 GB.
Die von jedem Knoten gesendete Datenmenge umfasst durchschnittlich 1 GB, die des Seed-Knotens 40 GB.
Die Werte verdeutlichen, dass einem nicht-Seed-Knoten bei der aufkommenden Datenmenge über einen hausüblichen Breitbandanschluss ausreichend Bandbreite zur Verfügung gestellt werden könnte – die Latenzen durch den Anschluss und die zeitliche Ausdehnung des Sendens der Daten bei geringer Bandbreite außer Acht gelassen.

### 5.3.2 Konkrete Auszüge

Im Folgenden werden vier ausgewählte Graphen gezeigt, welche die Daten aus der Übersicht verdeutlichen. Auf dem beigelegten Speichermedium befinden sich Graphen zu allen Testcomputern.

Wie auf Abbildung 5.2 erkennbar, sendet der Seed-Knoten um etwa 17 Uhr die Hälfte seines Datenbestandes zu einem anderen Knoten. Abbildung 5.1 zeigt zur gleichen Zeit einen CPU-Peak.
Analog empfängt der nicht-Seed-Knoten bei dessen Start um etwa 17:50 Uhr seine Daten

von einem anderen Knoten, wie auf Abbildung 5.4 erkennbar, mit einem entsprechenden CPU-Peak, wie auf Abbildung 5.3 zu sehen.

Um etwa 19:45 Uhr übernimmt eine neu hinzukommende Instanz die Daten des nicht-Seed-Knotens, wodurch er zunächst eine große Menge Daten sendet, danach aber sichtlich weniger Daten empfängt und eine merklich geringere CPU-Auslastung hat, erkennbar auf den Abbildungen 5.3 und 5.4. Eine erneute Aufteilung seiner Last um circa 22 Uhr führt zu keiner weiteren merklichen Entlastung der CPU.

Auf den Abbildungen 5.1, 5.2 und 5.4 ist gut zu erkennen, wie die Belastung zwischen 18:30 Uhr und 19:30 Uhr in vier Stufen ansteigt. Jede Stufe entspricht dem Start von je zehn Writer-Knoten. Auf dem Seed-Knoten ist ein Wegfallen dieser Last um 0:00 Uhr erkennbar. Hier werden die Writer beendet.

Auf Abbildungen 5.1 und 5.2 ist ab 0:20 Uhr die Belastung durch den Reader für den Seed-Knoten erkennbar, Phase 1 endet etwa um 2:30 Uhr, Phase 2 um 3:50 Uhr und Phase 3 um 5 Uhr. Zu den aktiven Zeiten des Readers können auf dem nicht-Seed-Knoten Peaks erkannt werden, die nicht während der Pausen des Readers auftreten.

Da zwischen den einzelnen Läufen des Readers sukzessive Speicher-Knoten abgeschaltet werden, müssen die verbleibenden Replika-Knoten deren Aufgaben übernehmen. Auf dem nicht-Seed-Knoten ist entsprechend um 3:30 Uhr und um 4:30 Uhr ein Anstieg der auftretenden Systembelastung zu erkennen.

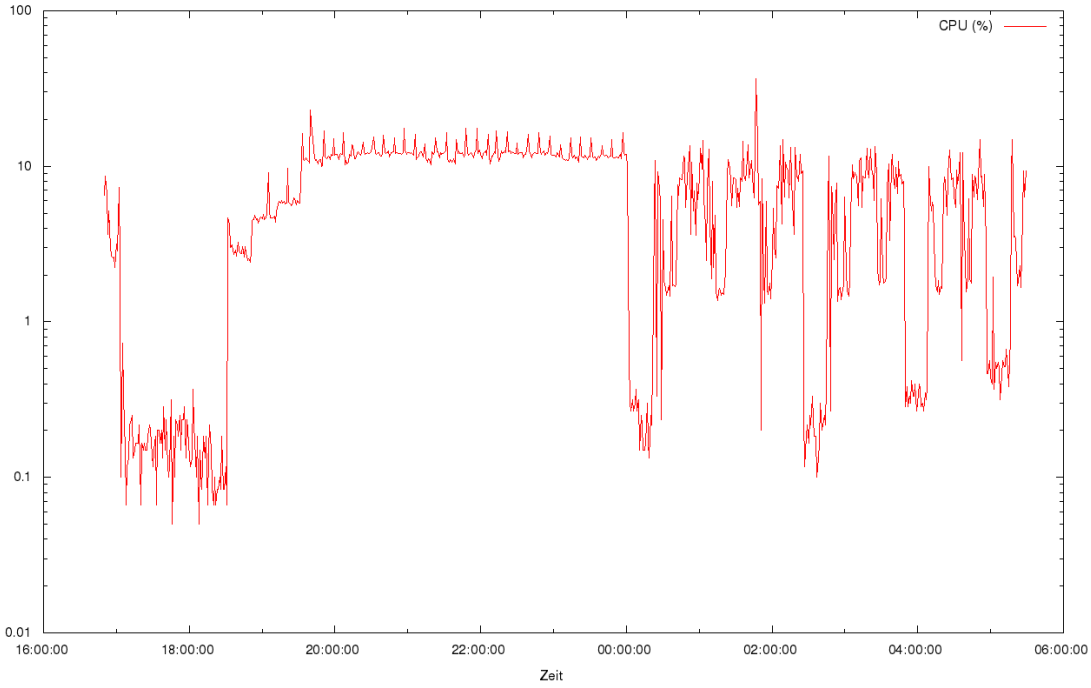


Abbildung 5.1: Seed-Knoten CPU-Belastung

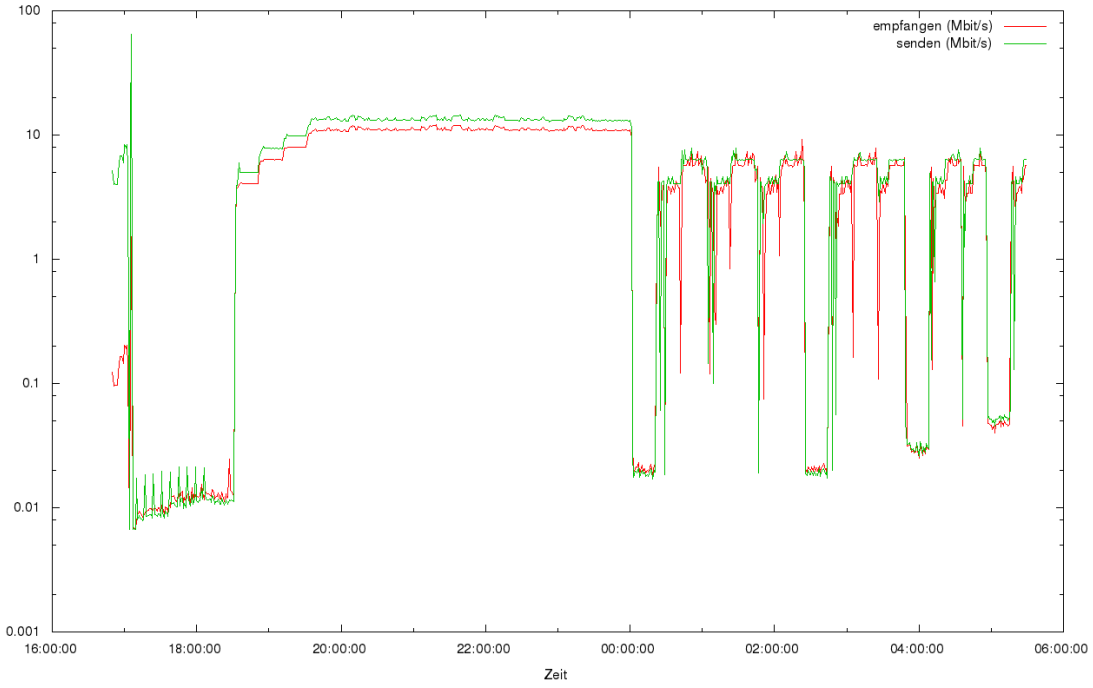


Abbildung 5.2: Seed-Knoten Netzwerk-Belastung

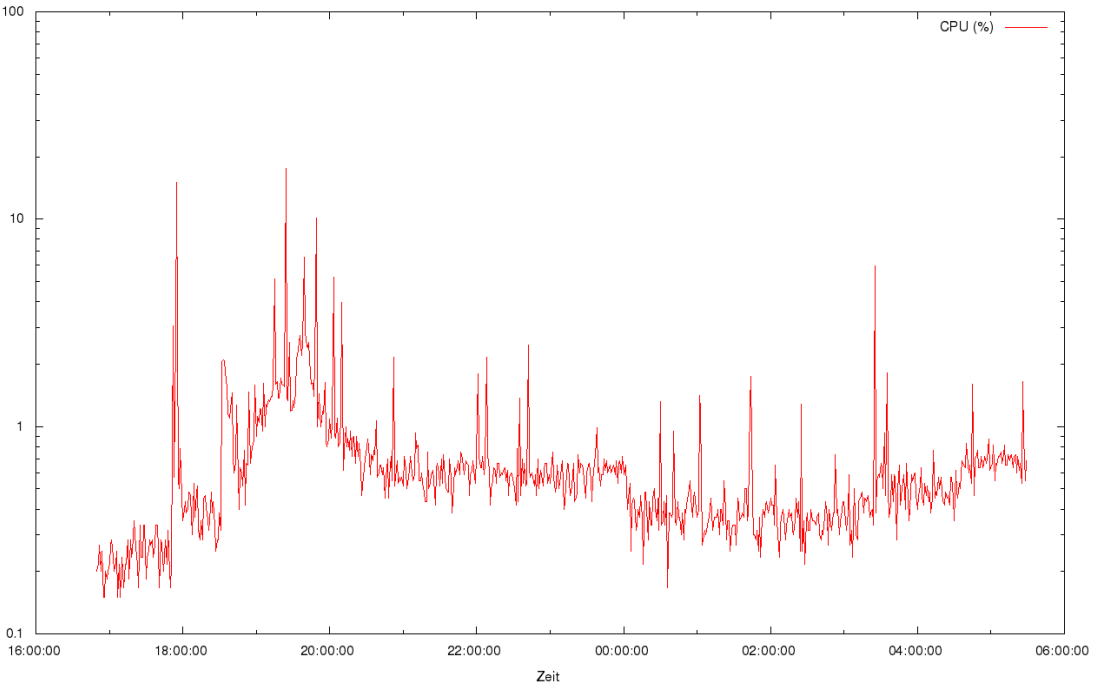


Abbildung 5.3: nicht-Seed-Knoten CPU-Belastung

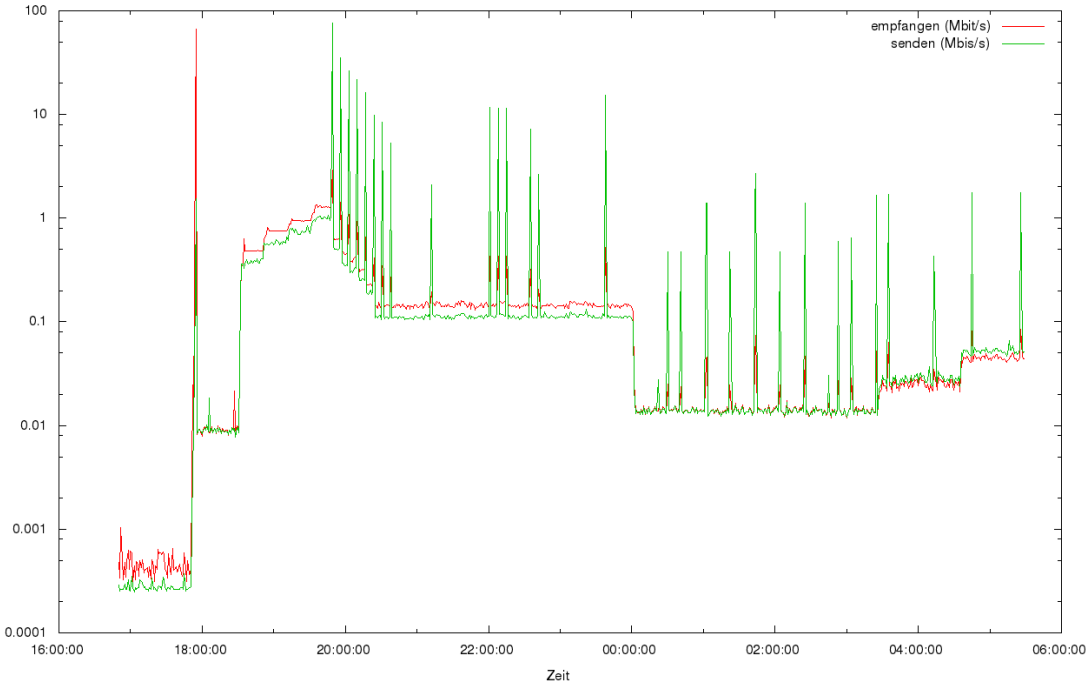


Abbildung 5.4: nicht-Seed-Knoten Netzwerk-Belastung

## 5.4 Auswertung

In der gesamten Messzeit wurden 698.429 Datensätze geschrieben. Die Laufzeit je Writer-Knoten betrug nach Tabelle 5.1 durchschnittlich 4,9 Stunden. Daraus ergibt sich eine Produktionsleistung je Knoten von 3.492 Datensätzen pro Stunde.
Da laut [Kinshumann et al., 2011] ein einzelner Computer durchschnittlich 0,0667 Berichte pro Tag produziert, simuliert ein Writer-Knoten dieses Experiments 1.257.120 Sammel-Knoten. Um 2,2 Milliarden Internetnutzern weltweit zu simulieren, wären demnach 1.750 Writer-Knoten notwendig – das 43-fache der Anzahl Writer-Knoten aus dem Versuchsaufbau.

Im schlechtesten Fall standen den 41 Writer-Knoten im Versuch elf Speicher-Knoten und ein Seed-Knoten gegenüber. Auf das 43-fache hochgerechnet, könnten demnach schätzungsweise 473 Speicher-Knoten und 43 Seed-Knoten die weltweit auftretende Last an Fehlerberichten verarbeiten und würden dabei in etwa die höchste gemessene Last tragen müssen. Die geringste Last je Rechner trat bei 41 Speicher-Knoten im Test auf, was 1.763 Speicher-Knoten weltweit entspräche.

Vergleicht man die aufgetretene Last mit den laut [Kinshumann et al., 2011] 60 WER-Servern, erlaubt dies die Vermutung, dass 60 hochpotente Speicher-Knoten ebenfalls die

| Rechner | Laufzeit |
|---|---|
| poolpc26-36 | 11 Knoten * 5,3667 Stunden |
| poolpc37-46 | 10 Knoten * 5,0333 Stunden |
| poolpc47-56 | 10 Knoten * 4,7 Stunden |
| poolpc57-66 | 10 Knoten * 4,3667 Stunden |
| **gesamt** | 200,0333 Stunden / 41 Knoten |

Tabelle 5.1: Laufzeiten

auftretende Last tragen könnten, wenn das System auf 1,5 Milliarden Teilnehmer (der überschlagenen Anzahl an Windows-Rechnern) ausgeweitet wird.

Obwohl kein konkreter funktionaler Zusammenhang zwischen der Systemlast und dem Verhältnis zwischen Daten- und Writer-Knoten bestimmbar ist, erkennt man deutlich, dass mit der steigenden Anzahl Writer-Knoten die Belastung der Speicher-Knoten steigt und mit Hinzufügen zusätzlicher Speicher-Knoten wieder sinkt. Je mehr Teilnehmer die Menge der Speicher-Knoten also hat, desto eher ist die Last für den Einzelnen tragbar – was den Erwartungen entspricht – vergleiche dazu Abbhildungen 5.3 und 5.4 20 Uhr. Die Last lässt sich jedoch nicht unter ein bestimmtes Minimum senken – vergleiche 22 Uhr der gleichen Abbildungen sowie den empirisch ermittelten, minimal benötigten Heap von 500 MB.

Viele Heimrechner sind heutzutage für die Zwecke ihrer Anwender „übermotorisiert“. Bei der gemessenen Last haben diese durchaus das Potential, ihre Leistung zur Verfügung zu stellen, ohne dass der Arbeitsfluss ihrer Besitzer nennenswert beeinträchtigt wird. Poweruser hingegen, welche die Leistung ihrer Computer selbst auslasten, dürften ihre Rechner kaum als Speicher-Knoten zur Verfügung stellen wollen. Unglücklicherweise genügen aber aller Erwartung nach eher letztere den geforderten Onlinezeiten.
Auch ist die auftretende Last nicht vereinbar mit dem heute weit verbreiteten und sich immer weiter verbreitenden Modell mobiler Geräte. Laptops, Tablets und Handys können bei der steten Belastung kaum Energiesparmodi einnehmen und fallen somit auch aus der Riege verwendbarer Computer.

Bei geschätzten 500 bis 2.000 nötigen Speicher-Knoten im Modell werden sich trotz allem sehr wahrscheinlich hinreichend viele Freiwillige finden. Zu beachten ist jedoch, dass Writer und Reader aufeinanderfolgend gemessen wurden, während sie in Realität simultan auftreten werden. Außerdem wurde hier ein einzelner Reader modelliert, welcher die komplette Datenbank ausliest, im Gegensatz zum Lastverhalten in der Realität, wo potentiell sehr viele Analysten einzelne Datensätze auslesen. Doch selbst wenn bei parallelem Zugriff von Sammel- und Analyse-Knoten die auftretende Last um den Faktor 100 steigt, ist die Hoffnung realistisch, eine ausreichend große Menge an Teilnehmern für das Netzwerk gewinnen zu können. Betrachtet man die Tatsache, dass vor allem Herstellerfirmen eine hohe Motivation hätten, zum System beizutragen, so sind vor allem deren Arbeitsplatz-Computer potentielle Kandidaten für Speicher-Knoten.

Es ist klar erkennbar, dass der Seed-Knoten eine deutlich höhere Belastung erfährt. Diese Last könnte auf die anderen Speicher-Knoten aufgeteilt werden, wenn Analyse-, Sammel und nicht-Seed-Knoten eine gute Möglichkeit hätten, deren Verbindungspunkte aus der Menge aller existierenden Speicher-Knoten zu wählen.

Zusammengefasst eignen sich nur vergleichsweise wenige Computer von Endanwendern weltweit für die Verwendung als Speicher-Knoten. Glücklicherweise hat der Test gezeigt, dass auch nur vergleichsweise wenige Computer von Nöten sind, um die auftretende Last zu tragen.

# 6 Schlussfolgerung

## 6.1 Ausblick

Während der Leistungsbewertung wurde das Problem aufgedeckt, dass bei zeitgleichem Beitritt mehrerer Knoten zum Cassandra-Grid gleiche Tokens vergeben werden können. Dies wurde in der Arbeit durch zeitlich versetzte Starts der Datenbanken umgangen. Für die Anwendung des Konzeptes in großem Rahmen müssen hier jedoch Abhilfen geschaffen werden.
Ein erster Ansatz wäre, ein Mutex für den Tokenvergabemechanismus zu verwenden. Dieser würde erst wieder freigegeben, wenn ein Token vergeben wurde. Da jeweils die Last eines einzelnen Knotens aufgeteilt wird, könnte der Mechanismus in Cassandra so implementiert sein, dass die Vergabe lediglich auf einem einzelnen Rechner stattfindet und somit kein verteilter wechselseitiger Ausschluss umgesetzt werden müsste.
Zur Realisierung der Idee wäre jedoch ein Eingreifen in den Quellcode von Cassandra von Nöten. Da es sich um ein Open Source-Projekt handelt, ist dies zwar theoretisch möglich, aber im Rahmen dieser Arbeit nicht umsetzbar.

Ein weiteres Problem in der Verwendung von Cassandra fiel im Anschluss an den Test beim erneuten Starten des Seed-Knotens auf, wie Abbildung 6.1 zeigt. Ein permanentes Entfernen von Knoten ist nur durch explizites Abmelden des Selbigen möglich. Offline-Zeiten von Knoten können mittels HintedHandoff selbst dann überbrückt werden, wenn alle Replika-Knoten ausfallen, doch bleiben diese noch immer für den entsprechenden Datensatz verantwortlich.
Dies ist durchaus ein Vorteil. Würden die Datensätze eines nicht erreichbaren Knotens einem anderen zugewiesen werden, so würden die Daten des entsprechenden Knotens ungültig. Selbst unter der Annahme, dass ein Knoten beim Herunterfahren des entsprechenden Computers zuverlässig abgemeldet werden könnte, bei einer erneuten Anmeldung im System bekäme er ein neues Token, somit einen neuen Zuständigkeitsbereich, müsste diese neuen Daten herunterladen und die alten verwerfen. Auf der anderen Seite

```
$ ./bin/nodetool -h poolpc10 ring
Address         DC           Rack        Status State   Load            Owns    Token
                                                                                 116573014182068669348474486738428845727
[redacted].10   datacenter1 rack1       Up     Normal  42,44 KB        50,00%  31502422451834053482630834880486792863
[redacted].192  datacenter1 rack1       Down   Normal  51,1 KB         25,00%  74037718316951361415552660809457819295
[redacted].11   datacenter1 rack1       Up     Normal  46,7 KB         25,00%  116573014182068669348474486738428845727
```

Abbildung 6.1: Nodetool

jedoch können sich Teilnehmer nach und nach entschließen, nicht mehr am System teilnehmen zu wollen oder ihr Token geht verloren. Es ist demnach nicht davon auszugehen, dass sich alle Knoten bei längerer oder dauerhafter Offline-Zeit sauber abmelden.
Daher müsste eine separate Software auf den Speicher-Knoten die Erreichbarkeit der anderen Speicher-Knoten überwachen und diese ab einer bestimmten Offline-Zeit dauerhaft aus dem System abmelden. Natürlich sollte diese Software ebenfalls über eine Grid-Struktur ohne single point of failure verfügen. Sie müsste dementsprechend in periodischen Abständen über einen Election-Algorithmus einen „Master-Knoten“ für die Aufgabe des Abmeldens wählen. Die Offline-Zeit, nach der ein Knoten dauerhaft aus dem System abgemeldet wird, sollte max_hint_window_in_ms entsprechen – der Zeit, die ein HintedHandoff vorgehalten wird, bevor er verworfen wird.

Angriffsszenarien wurden in dieser Arbeit außen vor gelassen und bedürfen einer separaten wie ausführlichen Betrachtung. Es sei jedoch erwähnt, dass Cassandra über einen inhärenten Authentifizierungsmechnismus verfügt, der eventuell für die Zwecke dieser Arbeit nutzbar wäre. Der „Benutzername“ für einen Computer könnte dessen ComputerID sein.

Vor der Erweiterung der entwickelten Software und der Fortführung des umgesetzten Konzeptes sollte die Tauglichkeit von Cassandra im angestrebten Rahmen getestet werden, wie im dritten Kriterium der Leistungsbewertung geschildert. Zu testen ist dabei eher die Funktionalität der Datenbank, als die reine Belastung der Computer.

Für die Anwendung im großem Rahmen ist ein Mechanismus für die Bekanntmachung der Adressen von Speicher-Knoten zu realisieren, zu denen sich andere Knoten verbinden können. Das erste zu bewältigende Problem dabei ist, dass keine HotSpots, wie der Seed-Knoten aus der Leistungsbewertung, entstehen sollen.
Eine naive Lösung wäre eine lokal gespeicherte Liste zuletzt gesehener Speicher-Knoten. In Cassandra könnte eine Funktionalität implementiert werden, welche in jeder Iteration die Teilnehmer des aktuellen Gossip-Treffens dieser Liste hinzufügt.
Eine alternative Möglichkeit ergibt sich bei einem hybriden Modell, in dem Firmen ihre Server zur Verfügung stellen. Stehen dedizierte Server als Speicher-Knoten zur Verfügung, kann eine Liste derer statisch erzeugt und verteilt werden. Diese werden dann gezielt HotSpots, beziehungsweise Seed-Knoten. Denkbar wäre auch, dass ein Server einen größeren Anteil der Datensätze übernimmt, als ein normaler Speicher-Knoten. Hierzu müsste der Tokenvergabemechanismus verändert werden.
Vergleichbar mit einem Namensdienst wäre es ebenso möglich, einen oder mehrere Server zur Verfügung zu stellen, welche Verbindungsanfragen zu Speicher-Knoten entgegennehmen und diese gleichverteilend an andere Speicher-Knoten weiterleiten. Ein solcher Server müsste Teil des Grids sein, um stets eine aktuelle Liste der Teilnehmer liefern zu können, wäre aber nicht so ausgelastet wie ein dedizierter Seed-Knoten. Ein solcher Mechanismus könnte wieder durch Firmen zur Verfügung gestellt werden.

Viele Arbeiten beschäftigen sich mit Methoden zum Sammeln, Vorverarbeiten und Auswerten von Fehlerberichten, mit [Kinshumann et al., 2011] und [Pacheco, 2011] wur-

den hier nur zwei betrachtet. Weitere Arbeiten werden weitere Aufschlüsse geben.
Das Format für Fehlerberichte – Details, betroffene Computer und beteiligte Komponenten – orientiert sich stark an denen der WER-Berichte. Andere Quellen, wie beispielsweise [Pitt, 2007] orientieren sich an Linux-Systemen. Um eine wirklich plattformübergreifende Lösung zur Verfügung stellen zu können, müssen weitere Formate hinzugezogen und eine Vereinigung dieser Formate erstellt werden.

Die Ergebnisse dieser Arbeit und die Ergebnisse aus den Forschungen über die aufgetretenen Probleme und vorgeschlagenen Verbesserungen müssen in eine formale Definition einfließen, an der sich alle Implementierungen orientieren.

Cassandra-Instanzen könnten sich ihres geografischen Standorts bewusst gemacht werden. Durch die Verwendung eines Partitioners, einer ReplikaPlacementStrategie und eines Snitches in Cassandra, welche den Standort der Instanzen auf der Welt berücksichtigen, könnten einige Probleme der hohen Verteiltheit des Systems gelindert werden. Statt der im Kapitel Implementierung beschriebenen Strategien könnten Knoten ihre Replikate so verteilen, dass diese keine großen geographischen Strecken zurücklegen, um die Latenzen zu minimieren. Ähnlich der erwähnten NetworkTopologyStrategy könnten dann einige wenige Replikate möglichst weit vom Original entfernt platziert werden um die Datensicherheit zu maximieren. Durch die lokale Präferenz mancher Software könnten auch Berichte in den Regionen gespeichert werden, in denen sie am häufigsten auftreten.

Bisher gibt ausschließlich ein Feld für Lösungsvorschläge Rückmeldung an den Benutzer. Es wäre ein System denkbar, mit dem direkt Workaround-Skripte oder Links zu Patches verteilt werden können. Der Benutzer müsste, wenn bei ihm ein bereits behobener Fehler auftritt, nur noch mit einem Klick bestätigen, dass die Veränderungen auf seinem Computer angewendet werden darf. Die Software könnte so ein zentrales Werkzeug für die Aktualisierung des Systems werden.

Der nächste Schritt ist die Erstellung einer Analyse-Software. Die Forschung über seine solche befasst sich mit den Funktionsmustern von Crowdsourcing sowie *human computer interaction* (Mensch-Maschine-Kommunikation).
Wie kommen die Fehlerberichte zu passenden Analysten? Sucht sich jeder Teilnehmer selbst Probleme, mit denen er sicht befasst? Werden neue Probleme nach einem bestimmten Muster einem Analysten empfohlen? Steigt ein Problem eine Hierarchie hinauf, vom Hobby-Entwickler mit steigendem Schwierigkeitsgrad zu kompetenteren Experten? Für Analysten könnten Erfolgs- oder Reputationsstatistiken aufgestellt werden.
Wie werden Fehlerberichte überhaupt organisiert? Wie muss eine gute Eingabemaske zum Suchen nach konkreten Fehlerursachen aussehen? Wie kann man bereits in einer Suchmaske Anfragen zum Suchen von Korrelationen zwischen Komponenten modellieren? Die Suchmaske darf dabei nicht zu unübersichtlich werden.
Bei der Erstellung einer Analyse-Software ist es leicht möglich, Techniken zu verwenden, die unabhängig vom verwendeten Betriebssystem sind.

Ähnlich der von Apport verwenden Methode, eingegangene Fehlerberichte von einem Kernentwicklerteam kontrollieren zu lassen, bevor diese der breiten Öffentlichkeit zugänglich gemacht werden, könnte auch diesem System eine Moderation hinzugefügt werden. So könnte die Vertraulichkeit sensibler Daten im System um einen weiteren Faktor erhöht werden. Die Rolle der Moderatoren könnte beispielsweise von der für die jeweilige Fehlerursache zuständigen Firma übernommen werden. Der Mechanismus könnte optional sein, so ist es einer Firma freigestellt, Moderatoren einzusetzen. Der Benutzer könnte des Weiteren gewarnt werden, wenn eine von ihm erbetene Information ohne vorherige Revision im System verfügbar gemacht wird.

Von Apache Solr, einer auf Lucene basierenden Softwarekomponente zur Volltextsuche, existiert mit [Solandra] eine Variante mit Cassandra als Datenbasis. Mit diesem System könnte über die Daten der Speicher-Knoten ein Suchindex aufgebaut werden. Es können Suchanfragen formuliert werden, die jenen genügen, die im Abschnitt über den Analyse-Knoten gefordert sind. So muss eine solche nicht separat implementiert werden.

Das MapReduce-Framework [Hadoop] ist ebenfalls aus dem Hause Apache und kann seine Daten ebenfalls aus einer Cassandra-Datenbank beziehen. In einem ersten Schritt könnten hiermit die in [Kinshumann et al., 2011] auf Seite 113 erwähnten Heuristiken implementiert werden, welche die Eimer, respektive Gruppen, besser sortieren als deren bloße GroupID. In einem nächsten Schritt könnten Algorithmen automatisch Korrelationen aufdecken. Zusammen mit dem Grid-Gedanken würde eine neue Kategorie von Rechen-Knoten entstehen – vergleichbar mit SETI@home.

## 6.2 Fazit

In der Einleitung wurde das Konzept einer kooperativen verteilten Fehleranalyse auf der Basis von Crowdsourcing gezeichnet und ein Nutzungsszenario vorgestellt. Im Kapitel Grundlagen wurden existierende Programme gezeigt, die einen Teil der benötigten Funktionen erbringen. Auf Basis derselben wurde im Entwurf das gezeichnete Konzept erweitert und konkretisiert. Das Kapitel Implementierung wählt sich daraus einen Teilaspekt, der den Entwurf verdeutlicht. Die anschließende Leistungsbewertung gibt einen ersten Anhaltspunkt für die Umsetzbarkeit der Idee dieser Arbeit. In diesem Kapitel wurden noch einmal die Fragen revidiert, die während des gesamten Prozesses aufgedeckt wurden, sowie erste Lösungsansätze vorgeschlagen.

Es wurde verdeutlicht, dass das Konzept des Crowdsourcing auf Basis des Internets als Kommunikationsmedium funktioniert und eine breite Masse potentieller Teilnehmer angesprochen wird. Die Probleme bezüglich Kognition, Koordination und Kooperation wurden im Bezug auf diese Arbeit behandelt und lassen eine funktionierende Praxisumsetzung plausibel erscheinen.

Es existiert kein zuverlässiger Weg, alle potentiell sensiblen Daten aus einem umfangreichen Fehlerbericht zu entfernen, zumindest nicht, ohne jedwede existierende Software

zu modifizieren. Zwei wichtige Absicherungen liegen jedoch im Rahmen des Möglichen: Zum Einen werden bevorzugt reduzierte Informationen gesendet. Zum Anderen kann der Benutzer frei bestimmen, wann und was gesendet wird.

Da die Sammel-Software sehr plattformnah sein muss, ist diese nicht allgemeingültig realisierbar. Die weiteren Komponenten können jedoch mit plattformunabhängigen Technologien umgesetzt werden.
Das Konzept dieser Arbeit ist allgemeingültig. Die Verwendung definierter, offener Protokolle und Formate garantiert die Interoperabilität zwischen potentiellen, verschiedenen Implementierungen.

Durch den modularisierten Aufbau der erstellten Sammel-Software ist es leicht möglich, Connectoren auszutauschen und an die jeweilige Plattform anzupassen. Mit Mono kann die Software so auch unter Linux und Mac OS X genutzt werden.
Cassandra hat sich unter den im Grundlagen-Kapitel vorgestellten Alternativen als jene Datenbank gezeigt, welche dem Konzept dieser Arbeit am ehesten entspricht. Durch die Umsetzung in Java ist sie zudem als nahezu plattformunabhängig anzusehen. Cassandras Mängel für den intendierten Anwendungszweck wurden deutlich gemacht und Ideen für Verbesserungen erläutert. Diese Modifikationen sind eventuell leichter umzusetzen, als die komplette Neuentwicklung einer Speicher-Software.
Erste Überschlagsrechnungen haben gezeigt, dass nur vergleichsweise wenige Speicher-Knoten von Nöten wären, um das weltweite Datenaufkommen von Sammel-Knoten zu tragen.

Die Relevanz einer Praxisumsetzung zeigt sich in den Defiziten heutiger Methoden. Kooperative verteilte Fehleranalyse auf der Basis von Crowdsourcing ermöglicht eine neue Herangehensweise an das Problem und schöpft aus den Möglichkeiten aktueller Kommunikationsformen sowie der Weisheit der Masse. Die Arbeit hat die Machbarkeit des Konzepts gezeigt, solide Grundlagen geschaffen und motiviert ein weiteres Vertiefen der Thematik.

# 7 Literaturverzeichnis


[Allavena et al., 2005] Allavena, André, A. Demers und J. E. Hopcroft (2005). *Correctness of a gossip based membership protocol.* In: *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, PODC '05, S. 292–301, New York, NY, USA. ACM.

[BITKOM, 2010] BITKOM (2010). *Presseinformation.* Umfrage im Auftrag des BITKOM.

[Broadwell et al., 2003] Broadwell, Pete, M. Harren und N. Sastry (2003). *Scrash: a system for generating secure crash information.* In: *Proceedings of the 12th conference on USENIX Security Symposium - Volume 12*, SSYM'03, S. 19–19, Berkeley, CA, USA. USENIX Association.

[Gilbert und Lynch, 2002] Gilbert, Seth und N. Lynch (2002). *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.* SIGACT News, 33(2):51–59.

[Kemper und Eickler, 2006] Kemper, Alfons und A. Eickler (2006). *Datenbanksysteme – Eine Einführung.* Oldenbourg Wissenschaftsverlag GmbH, 6 Aufl.

[Kinshumann et al., 2011] Kinshumann, Kinshuman, K. Glerum, S. Greenberg, G. Aul, V. Orgovan, G. Nichols, D. Grant, G. Loihle und G. Hunt (2011). *Debugging in the (very) large: ten years of implementation and experience.* Commun. ACM, 54(7):111–116.

[Pacheco, 2011] Pacheco, David (2011). *Postmortem Debugging in Dynamic Environments.* Queue, 9(10):12:10–12:21.

[Pitt, 2007] Pitt, Martin (2007). *The Apport crash report format.*

[Pokorny, 2011] Pokorny, Jaroslav (2011). *NoSQL databases: a step to database scalability in web environment.* In: *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*, iiWAS '11, S. 278–283, New York, NY, USA. ACM.

[Slee et al., 2007] Slee, Mark, A. Agarwal und M. Kwiatkowski (2007). *Thrift: Scalable Cross-Language Services Implementation.*

[Surowiecki, 2004] Surowiecki, James (2004). *The wisdom of crowds: Why the many are smarter than the few and how collective wisdom shapes business, economies, societies, and nations.* Doubleday, New York, NY, USA.

[ACRA] Google Inc. (Stand: April 2012). *acra - Application Crash Report for Android - Google Project Hosting.* https://code.google.com/p/acra/

[Apport] Ubuntu Wiki (Stand: April 2012). *Apport - Ubuntu Wiki.* https://wiki.ubuntu.com/Apport

[APT] Debian (Stand: April 2012). *Apt - Debian Wiki/.* http://wiki.debian.org/Apt

[Breakpad] Mozilla Foundation (Stand: April 2012). *Crash reporting - MDN.* https://developer.mozilla.org/en/Crash_reporting

[Bugzilla] Mozilla Foundation (Stand: April 2012). *Home :: Bugzilla :: bugzilla.org.* http://www.bugzilla.org/

[Cassandra] Apache Software Foundation (Stand: April 2012). *The Apache Cassandra Project.* https://cassandra.apache.org/

[Cassandra API] Apache Software Foundation (Stand: April 2012). *API - Cassandra Wiki.* https://wiki.apache.org/cassandra/API

[Cassandra Limits] Apache Software Foundation (Stand: April 2012). *CassandraLimitations - Cassandra Wiki.* https://wiki.apache.org/cassandra/CassandraLimitations

[CouchDB] Apache Software Foundation (Stand: April 2012). *Apache CouchDB.* https://couchdb.apache.org/

[CrashReporter] Apple Inc. (Stand: April 2012). *Technical Note TN2123: CrashReporter.* https://developer.apple.com/library/mac/#technotes/tn2004/tn2123.html

[CrashReporting] Ubuntu Wiki (Stand: April 2012). *CrashReporting - Ubuntu Wiki.* https://wiki.ubuntu.com/CrashReporting

[DataStax] DataStax Inc. (Stand: April 2012). *About Replication in Cassandra | DataStax Cassandra 0.8 Documentation.* http://www.datastax.com/docs/0.8/cluster_architecture/replication

[EventLog] Microsoft Corporation (Stand: April 2012). *Event Viewer [Vista].* http://technet.microsoft.com/en-us/library/cc766042.aspx

[Facebook] Facebook Inc. (Stand: April 2012). *Willkommen bei Facebook - anmelden, registrieren oder mehr erfahren.* https://www.facebook.com/

[FluentCassandra] Managed Fusion (Stand: April 2012). *Fluentcassandra by managedfusion.* http://fluentcassandra.com/

[GDB] Free Software Foundation Inc. (Stand: April 2012). *GDB: The GNU Project Debugger/*. https://www.gnu.org/software/gdb/

[Google Books] James Crawford (Stand: April 2012). *Inside Google Books: On the Future of Books*. http://booksearch.blogspot.de/2010/10/on-future-of-books.html

[Hadoop] Apache Software Foundation (Stand: April 2012). *Welcome to Apache™Hadoop™!/*. https://hadoop.apache.org/

[Internetnutzer] Miniwatts Marketing Group (Stand: April 2012). *World Internet Usage Statistics News and World Population Stats*. http://www.internetworldstats.com/stats.htm

[iOS] Apple Inc. (Stand: April 2012). *Technical Note TN2151: Technical Note TN2151*. https://developer.apple.com/library/ios/#technotes/tn2151/_index.html

[MiniDump] Microsoft Corporation (Stand: April 2012). *MINIDUMP_TYPE enumeration*. http://msdn.microsoft.com/en-us/library/windows/desktop/ms680519%28v=vs.85%29.aspx

[MongoDB] 10gen Inc. (Stand: April 2012). *MongoDB*. http://www.mongodb.org/

[Mono] Xamarin (Stand: April 2012). *Mono*. http://www.mono-project.com/

[MSI IDs] Microsoft Corporation (Stand: April 2012). *Windows Installer Application Installation*. http://technet.microsoft.com/en-us/library/cc735598%28v=ws.10%29.aspx

[MySQL] Oracle Corporation (Stand: April 2012). *MySQL :: The world's most popular open source database*. https://www.mysql.com/

[Nagios] Nagios Enterprises (Stand: April 2012). *Nagios - The Industry Standard in IT Infrastructure Monitoring*. http://www.nagios.org/

[OpenStreetMap] OpenStreetMap Foundation (Stand: April 2012). *OpenStreetMap*. http://www.openstreetmap.org/

[PostgreSQL] PostgreSQL Global Development Group (Stand: April 2012). *PostgreSQL: Welcome*. http://www.postgresql.org/

[Project Voldemort] LinkedIn (Stand: April 2012). *Project Voldemort*. http://project-voldemort.com/

[RFC 5424] IETF (Stand: April 2012). *RFC 5424 - The Syslog Protocol*. https://tools.ietf.org/html/rfc5424

[reCAPTCHA] Google Inc. (Stand: April 2012). *What is reCAPTCHA?*. https://www.google.com/recaptcha/learnmore

[Solandra] Jake Luciani (Stand: April 2012). *tjake/Solandra · GitHub*. https://github.com/tjake/Solandra/

[Syslog-NG] BalaBit IT Security Ltd. (Stand: April 2012). *syslog-ng - Multiplatform Syslog Server and Logging Daemon*. http://www.balabit.com/network-security/syslog-ng

[Thrift] Apache Software Foundation (Stand: April 2012). *Apache Thrift*. https://thrift.apache.org/

[WER Klassifizierung] Microsoft Corporation (Stand: April 2012). *How WER Collects and Classifies Error Reports*. http://msdn.microsoft.com/en-us/windows/hardware/gg487468

[WER Privacy] Microsoft Corporation (Stand: April 2012). *Privacy Statement for the Microsoft Error Reporting Service*. http://oca.microsoft.com/en/dcp20.asp

[WER Einstellungen] Microsoft Corporation (Stand: April 2012). *WER Settings*. http://msdn.microsoft.com/en-us/library/windows/desktop/bb513638%28v=vs.85%29.aspx

[Wikipedia] Wikimedia Foundation (Stand: April 2012). *Wikipedia*. https://www.wikipedia.org/

[Wiki WER] Wikimedia Foundation (Stand: April 2012). *Windows Error Reporting - Wikipedia, the free encyclopedia*. https://en.wikipedia.org/wiki/Windows_error_reporting

[Winqual] Microsoft Corporation (Stand: Januar 2012). *Winqual Help*. https://winqual.microsoft.com/help/default.htm#winqual_requirements.htm

[Wow6432Node] Microsoft Corporation (Stand: April 2012). *Registry Keys Affected by WOW64 (Windows)*. http://msdn.microsoft.com/en-gb/library/aa384253%28v=VS.85%29.aspx

# 8 Tabellenverzeichnis

# 9 Abbildungsverzeichnis