Seminarausarbeitung

Kjell Otto

Scaling and Concurrent Data Processing

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

1 Einführung

Die Menge an Computern und ihrer Sensorik nimmt kontinuierlich zu. Nicht nur in Firmennetzwerken ist die Anzahl von CPUs und Computern gestiegen, sondern auch in privaten Wohnräumen kommt immer mehr Rechenleistung zum Einsatz. Die Leistungsfähigkeit wird nur in Ausnahmen voll ausgeschöpft und bietet damit Spielraum für Mechanismen, die solche ungenutzten Ressourcen instrumentalisieren. Auch die Menge an Daten, die zur Auswertung oder Verarbeitung heute zur Verfügung steht, ist gestiegen. Die Skalierbarkeit der Infrastruktur ist in vielen Firmen ein Aspekt, der erst in den letzten Jahren wichtiger wird und in älteren Systemen meist nicht beachtet wurde.

Der Living Place Hamburg ist ein Forschungsprojekt der Hochschule für Angewandte Wissenschaften Hamburg im Kontext zukünftigen Lebens. Das Labor ist eine voll funktionsfähige Privatwohnung mit angeschlossenen Kontroll- und Büroräumen auf einer Grundfläche von 130 m^2, vgl. Abb. 1.1.

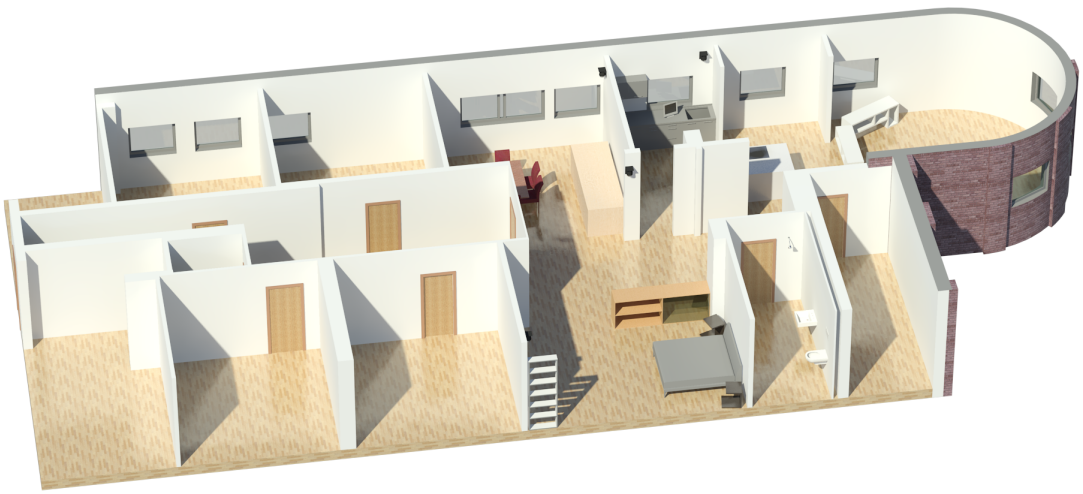

Abb. 1.1: 3D Rendering des Living Place Hamburg

In diesem Forschungsrahmen werden aktuell Masterprojekte realisiert, die in unterschiedlichen Bereichen der Informatik praxisnahe Anwendungen entwickeln. Die meisten dieser Projekte werden untereinander Informationen austauschen und weiterverarbeiten. Die Kommunikationsinfrastruktur wurde in [13] und [14] entwickelt und stellt die grundlegenden Funktionen für die Interaktion zwischen Anwendungen und die Persistierung der Nachrichten. Durch die Persistierung jeder Nachricht wird die Menge der archivierten Daten stetig steigen und mit der Anzahl an Masterprojekten zukünftig noch zunehmen. Da zunächst

jedes Projekt einen eigenen Computer zur Realisierung gestellt bekommt, steigt mit den Projekten auch die Anzahl der Computer im Labor. Die meisten Anwendungen benötigen jedoch nicht alle Ressourcen die auf ihrer Plattform zur Verfügung stehen und somit können viele ungenutzte Ressourcen entstehen. Auch die Skalierbarkeit der Anwendung steht bei den meisten Projekten nicht im Vordergrund und kann daher als Möglichkeit für Untersuchungen verschiedenster Aspekte in diesem Gebiet genutzt werden.

In dieser Ausarbeitung wird keine Masterarbeit vorgestellt, sondern viel mehr ein Rahmen abgesteckt, in dem sich die Masterarbeit ansiedeln wird. Die Lösungen, die heute zur Skalierung großer Dienste dienen und bei der Verarbeitung großer Datenmengen eine entscheidende Rolle spielen, werden den Rahmen einer Masterarbeit bilden. Zum aktuellen Zeitpunkt steht das konkrete Thema nicht fest.

1.1 Motivation

Speicherplatz von mehreren Terrabyte ist heute nicht mehr außergewöhnlich für handelsübliche Computer im Verbrauchermarkt. Firmen, die versuchen Daten aus dem Internet zu evaluieren, sehen sich aber zunehmend mit Datenmengen in Petabytegröße konfrontiert und müssen verschiedenste Technologien einsetzen, um diese zu speichern und zu prozessieren, vgl. [4]. Durch das ständige Sammeln von Daten und deren Archivierung wird die Datenmenge zunehmend steigen. Selbst redundante Systeme haben oft ein Skalierungsproblem ab einer bestimmten Anzahl von Festplatten, daher werden schon heute beim Aggregieren großer Datenmengen verteilte Dateisysteme eingesetzt, vgl. [2], [16]. Aber nicht nur Speicherplatz erfordert neue Techniken um mit der Datenmenge zu skalieren. Auch die Taktrate von CPUs wird nach heutigem Stand der Technik nicht mehr steigen und entsprechend müssen neue Konzepte zum parallelen Verarbeiten von Daten entwickelt werden. Einige Ansätze zielen auf den Einsatz vieler Computer ab und andere versuchen die Ressourcen eines einzelnen Computers optimal zu nutzen, vgl. [12]. Andere Ansätze versuchen Grafikkarten in das Prozessieren großer Datenmengen mit einzubeziehen und verfolgen den Ansatz der Instrumentalisierung von GPGPUs, vgl. [17], [10]. Es gibt Ansätze, die den Einsatz von verteilten Systemen mit dem Einsatz von Grafikkarten kombinieren, vgl. [1].

Ausfallsicherheit ist ein weiterer Aspekt, der heute immer häufiger zum Tragen kommt. Systeme die eine Vielzahl von Computern benutzen um das Problem der Skalierung auf Speicher- oder Taktebene zu lösen, birgen durch die Anzahl der beteiligten Computer ein höheres Potential an Ausfällen. Der in diesem Kontext meist verwendete Ansatz ist dezentral und bietet eine Lösung für Ausfälle von Koordinateuren und Arbeitern, vgl. [4]. Wie man diesen Ausfällen mit Virtualisierung vorbeugen kann wurde bisher wenig untersucht und bietet Potential für weitere Untersuchungen.

1.2 Einsatzgebiete

Der Living Place Hamburg bietet ein Umfeld, in dem eine große Anzahl an Sensoren zum Einsatz kommen wird und entsprechend schnell Daten aggregiert werden können, vgl. [13]. Die Infrastruktur zur Verteilung und Persistierung von Nachrichten ist bereits vorhanden und soll in Zukunft mit weiteren Abstraktionsleveln erweitert werden. Momentan ist die Datenhaltung im Living Place ohne Konzepte zur Skalierung realisiert worden und bietet damit großes Potential zur Forschung in diesem Bereich. Auch die Entwickler von Anwendungen in diesem Kontext werden der Einfachheit halber zunächst jeder einen eigenen Computer zur Verfügung haben. An dieser Stelle lässt sich untersuchen wieviel Ressourcen eventuell im Gesamtsystem ungenutzt sind und verwendet werden können um Daten aus der Sensorhistorie weiterführend zu untersuchen.

Ein weiterer Aspekt, der eventuell eine optimale Auslastung der Ressourcen fördern kann, ist die Untersuchung von Virtualisierungsmechanismen im Living Place. Anwendungen, die einen geringen Ressourcenaufwand haben und zur Kommunikation mit ihrer Sensorik lediglich eine Netzwerkanbindung benötigen, eignen sich besonders. Zum einen hätte es den Vorteil der Platzersparnis und zum anderen würde es den Geräuschpegel senken, sodass sich eventuelle Bewohner nicht beeinträchtigt fühlen.

2 Masterarbeit

Im Folgenden werden Vorarbeiten vorgestellt, die zu dem Rahmen dieser Masterbeit geführt haben. Im Anschluss werden die Gebiete erläutert, in denen Untersuchungen stattfinden sollen, um ein konkretes Thema für eine Masterarbeit festzulegen.

2.1 Vorarbeiten

In [11] wurde sich ausführlich mit der Programmiersprache Clojure auseinander gesetzt. Dieser dynamische, funktionale und auf der Java Virtual Machine (JVM) laufende Lisp Dialekt bietet eine Vielzahl von Konzepten zum Umgang mit Problemen der Softwareentwicklung im Bezug auf Parallelität. Ein Grundkonzept dieser Sprache sind die unveränderlichen und persistenten Datenstrukturen. Unveränderlich im Bezug auf Zuweisung und persistent im Bezug auf die Historie einer Variablen. Erstellt man eine Variable wird dieser kein neuer Wert zugewiesen, sondern vom ursprünglichen Inhalt ein neuer Wert berechnet. So entstehen viele Versionen einer Variablen. Sobald ein Wert in der Historie einer Variable nicht mehr referenziert wird, kann dieser von der Garbage Collection (GC) der JVM freigegeben werden. Dennoch ist Clojure nicht pur funktional, sondern erlaubt durch die nahtlose Integration von Java auch eine Änderung dieser Datenstrukturen. Werden Datenstrukturen benötigt, die veränderlich aber dennoch geschützt sind, wie bei einer Kommunikation zwischen mehreren Threads über die Inhalte von Variablen, muss man in Clojure die spracheigenen Mechanismen nutzen. Ein Multi Version Concurrency Control (MVCC) System mit Snapshot Isolation kontrolliert die Zugriffe auf alle geteilten Ressourcen, vgl. [18].

Clojure bietet vier Hauptkonzepte in Bezug auf den Schutz von geteilten Ressourcen. In Clojure kann nur ändernd auf Variablen zugegriffen werden, wenn eine Referenz eingesetzt wird. Diese drei Referenztypen sind die „*Var*“, das „*Atom*“, der „*Agent*“ und die „*Ref*“. Dabei ist die *Var* der Mechanismus um threadübergreifend Konstanten zu teilen. Dieser Mechanismus ermöglicht das Ändern des Inhaltes, wird aber üblicherweise nicht für inhaltliche Änderungen zur Laufzeit eingesetzt. *Vars* werden als Konfigurationsvariablen eingesetzt, bieten aber keinen Laufzeitschutz vor getrenntem Zugriff. Der Mechanismus, der dies ermöglicht, ist das *Atom*. Das Atom wird für unkoordinierte, synchrone Updates verwendet. Unkoordiniert bedeutet hier, dass ein Zugriff unabhängig von anderen Zugriffen stattfindet und synchron heißt, dass ein Aufruf erst nach erfolgreicher Änderung der Variablen zurückkehrt. Für unkoordinierte und asynchrone Updates werden die *Agents* eingesetzt. Sie bieten die gleiche Funktionalität wie *Atoms*, kehren aber nach einen Aufruf zur Veränderung des Inhalts gleich zurück. Senden mehrere Threads gleichzeitig Anfragen zur Änderung an den *Agent*, werden diese der Reihe nach verarbeitet. Es besteht die

Möglichkeit auf eine Gruppe von Aufrufen zu warten. Der vierte und komplexeste Mechanismus ist die *Ref*. Dieser Zugriff ist synchron und koordiniert. Zugriffe auf *Refs* können nur in Transaktionen stattfinden. Innerhalb einer Transaktion können mehrere Funktionen ausgeführt und Variablen verändert werden. Sollte ein Teil der Transaktion nicht erfolgreich beendet werden, wird die gesamte Transaktion wiederholt.

Der große Vorteil der Mechanismen in Clojure ist, dass man diese verwenden muss. Der Entwickler hat keine Möglichkeit versehentlich ungeschützt schreibend auf Variablen zuzugreifen oder eine Locking-Reihenfolge nicht zu beachten. Durch die Automatisierung auf Sprachebene wird der Entwickler von vielen Verantwortungen freigesprochen und kann sich auf die Funktionalität der Anwendung konzentrieren.

Im Vergleich zu Clojure wurde in [12] ein Überblick zu anderen Technologien gegeben. Einleitend wurden zwei Kategorien erläutert, um die Mechanismen zur Lösung von Parallelisierungsproblemen in Multicore- und Multinode zu unterteilen. Anschließend konnten zwei Technologien aus verschiedenen Sprachen für die Multicore- und eine für die Multinodekategorie erarbeitet werden. Untersucht wurden Scala Aktoren, die STM.Net und Googles MapReduce.

Das Aktormodell wurde schon 1973 zur Unterstützung der Forschung auf dem Gebiet der künstlichen Intelligenz entwickelt, vgl. [7]. Auf Basis von leichtgewichtigen Threads werden Aktoren als kleinste nebenläufige Einheit erstellt. Aktoren können Nachrichten versenden, neue Aktoren erstellen und auf empfangene Nachrichten reagieren. Aktoren haben den Vorteil, dass sie die „share nothing“ Strategie verfolgen und damit keine Synchronisationsmechanismen benötigen. In Scala werden Aktoren in JVM-Threads instantiiert. Durch die Beschränkung der Threadmenge durch das Betriebssystem bietet Scala jedoch auch einen eventbasierten Ansatz an, vgl. [15].

Die STM.Net bietet einen Software Transactional Memory für die .Net Plattform an, vlg. [5]. Das Konzept war hier ähnlich dem von Clojure. Mit der Einführung eines neuen Codeblocks wurden geschützte Zugriffe auf geteilte Ressourcen realisiert. Dazu wurde die Common Language Runtime (CLR) so modifiziert, dass diese mit dem CLR-Just In Time Compiler erkennen konnte, welche Objekte in einem geschützten Abschnitt modifiziert werden. Für diese Objekte wurden Schattenkopien und feingranulare Locks angelegt. Die STM.Net wurde allerdings nicht in die .Net Plattform integriert, sodass sie ein experimentelles Stadium nicht überwinden konnte.

Das MapReduce Modell verfolgt einen anderen Ansatz bezüglich der Parallelisierung. Aus der Kategorie Multinode stammend, werden hier tausende von Computern zur parallelen Berechnung eingesetzt, vgl. [4]. Entwickelt wurde diese Technologie um ungenutzte Ressourcen in Rechenzentren zugänglich zu machen. In Verbindung mit dem Google File System (GFS) werden die zwei Funktionen Map und Reduce eingesetzt um verteilte Berechnungen durchzuführen. Beide Funktionen müssen seiteneffektfrei und auf alle Daten anwendbar sein. Die Map-Funktion wird auf die zu verarbeitenden Daten angewandt, um Schlüssel/Werte-Paare zu bilden. Zu einem Schlüssel werden mehrere Wertepaare gebildet und dann mit der Reduce-Funktion reduziert, sodass ein verteiltes Endergebnis entsteht. Bei abgeschlossener Berechnung steht dann zu jedem Schlüssel ein Wert als Teilergebnis zur Verfügung.

Die aus den Vorarbeiten gewonnenen Erkenntnisse aus dem Bereich der Parallelisierung und Skalierung sollen in der zu verfassenden Masterarbeit die Grundlage der Untersuchungen bilden. Einblicke in verschiedene Bereiche des Kontextes werden die Untersuchungen fachlich unterstützen und früh auf bereits bekannte Probleme aufmerksam machen.

2.2 Zielsetzung

Ziel der Masterarbeit ist die Untersuchung von unterschiedlichen Bereichen der Informatik auf die Anwendbarkeit im Bezug auf Skalierungsprobleme. Den Ausgangspunkt der Untersuchungen wird der Living Place Hamburg mit seiner Middleware stellen, vgl. [13], [14]. Die Architektur dieser Middleware soll auf Skalierungspotential untersucht werden. Inhalte, die über den Living Place hinaus verwendbar sind, werden auf verschiedenen Skalierungsebenen erarbeitet. Untersuchungen in den Bereichen Monitoring, Programmiersprache, Kommunikationsschnittstelle, Verteilung von Jobs und Ausfallreaktion sollen einen Rahmen zur Themenwahl der Masterarbeit bilden. Im Folgenden werden die Teilaspekte dieser Bereiche erläutert die einen Beitrag zur Lösung von Skalierungsproblemen leisten können.

Monitoring

Die Aufgabenverteilung setzt voraus, dass es Ressourcen im Einzugsbereich einer Anwendung gibt, auf die eine Aufgabe verteilt werden kann. Diese Information wird mittels Monitoring der Ressourcen ermittelt. Eine Möglichkeit des Monitorings ist die Auswertung der CPU, RAM und Festplattenauslastung. Eine Untersuchung soll zeigen, ob Metainformationen helfen können, die Ressourcen unter Betrachtung von z.B. zeitlichen Abhängigkeiten zu optimieren. Sollte ein Teilnehmer der Infrastruktur zum Beispiel eine Berechnung durchführen, von der er bestimmen kann, zu welchem Zeitpunkt diese abgeschlossen ist, würde diese Information einer koordinierenden Instanz die Möglichkeit geben, diese Ressourcen in eine Verteilung einzuplanen.

Des Weiteren soll untersucht werden, wie diese Informationen ermittelt werden können. Eine Möglichkeit ist das Auslesen von Informationen mit SNMP (Simple Network Management Protocol). Eine Plattform, die eventuell erweitert werden kann, um einen Ressourcenkoordinator automatisch über freie Ressourcen im System zu informieren, wäre Nagios, vgl. [9].

Programmiersprache

Die meisten objektorientierten Programmiersprachen schützen geteilte Ressourcen über Locks. Dieser Mechanismus erfordert eine hohe Disziplin und eine ausführliche Dokumentation, um mit einem großen Entwicklerteam Software zu entwickeln, die Multicore CPUs

voll ausnutzt. Durch die Komplexität von feingranularem Locking, entstehen Raceconditions und infolgedessen oft auch Deadlock, Lifelocks und nicht reproduzierbare Systemzustände. Programmiersprachen, die versuchen das Problem des Lockings zu umgehen, sind meist funktional oder nutzen einen automatischen Locking-Mechanismus. Objektorientierte Sprachen bieten meist nicht die Flexibilität um automatische Mechanismen auf Sprachlevel zu integrieren, oder sind zu alt um solche unter Gewährleistung von Abwärtskompatibilität einzubauen. Andere Ansätze verfolgen das Aktormodell um Flexibilität über Prozessorauslastung mit leichtgewichtigen Prozessen sicherzustellen, vgl. [3].

Zu untersuchen ist die Anwendbarkeit von Mechanismen wie dem Software Transactional Memory oder dem Actor Model in anderen Bereichen der Skalierbarkeit. Der Multi Version Concurrency Control (MVCC) Ansatz ließe sich eventuell auch auf Bereiche der verteilten Berechnung übertragen. Denkbar ist zudem eine Kombination von einer verteilten virtuellen Maschine mit solchen Mechanismen. Der geringe Entwicklungsaufwand durch die Automatisierung von Locking-Mechanismen in Verbindung mit dem Zugriff auf verteilte Systeme könnte eine deutliche Steigerung der Ressourcenauslastung herbeiführen.

Kommunikationsschnittstelle

Um eine skalierbare Infrastruktur für ein verteiltes System realisieren zu können, werden flexible Lösungen zur Kommunikation benötigt. Der Einsatz einer Middleware kann helfen die Auffindbarkeit eines Dienstes zu erhöhen. Im Living Place Hamburg wird ein Message Broker eingesetzt, der als zentraler Kommunikationsmittelpunkt fungiert, vgl. [14]. Nachrichten werden über das JSON-Datenformat ausgetauscht und sind damit an textuelle Übertragungen gebunden. Werden die Nachrichten komplexer und die Grenzen des Nachrichtenformats überschritten, sollte die Common Object Request Broker Architecture (CORBA) als Erweiterung der Kommunikationsinfrastruktur untersucht werden. CORBA bildet seit langem einen Kommunikationsstandard um über Plattformen und Netzwerkgrenzen hinweg Objekte zu instantiieren und auszutauschen, vgl. [6].

Verteilung

Die Verteilung von Aufgaben auf mehrere Computer oder ganze Cluster erfordert nicht nur einen Mechanismus der diese Möglichkeit bietet, sondern auch den Cluster selbst. In privaten Haushalten werden selten mehr als zwei Computer betrieben, in einem Wohnhaus allerdings deutlich mehr. Unter dem Gesichtspunkt der Verteilung ist zu untersuchen, ob eine lokale Verteilung in einem eventuellen Hausnetzwerk Sinn macht. Wenn innerhalb eines privaten Wohnhauses ein vom privaten Internetzugang getrenntes LAN existiert, so würden rechenintensive Aufgaben auf Computer im ganzen Haus verteilt werden können. Welche Software zum Einsatz kommen müsste, um dieses System auf einen Stadtteil auszuweiten und inwiefern die privaten Computer geschützt bleiben können, ist ein weiterführender Aspekt.

In einem System wie dem von Google eingesetzten MapReduce wird, zur Verfügbarkeit von Rechenleistung und Arbeitsspeicher, auch die Lokalität der Daten mit einbezogen.

Dieser Mechanismus ermöglicht dem System das Netzwerk zu entlasten, indem möglichst viele Zugriffe auf Daten, die zur Berechnung eines Jobs notwendig sind, lokal stattfinden, vgl. [4]. Ob das Konzept der Datenlokalität im Kontext von Skalierbarkeit den Datentransfer innerhalb der Ressourcenverteilung minimieren kann, bleibt zu untersuchen.

Ausfallreaktion

In verteilten Systemen gibt es zwei Arten von Ausfallsicherheit. Zum einen den *Hot-Standby* und zum anderen den *Cold-Standby*, vgl. [8]. Der Cold-Standby Server ist ein System das nach dem Ausfall des Hauptservers von einer zweiten Instanz gestartet wird. Der Hot-Standby Server hingegen läuft synchron auf gleichem Hirarchielevel mit dem Hauptserver. Zu untersuchen ist eine Failoverstrategie mit Virtuellen Maschinen (VM). Auch ob es im Rahmen der Virtualisierung eine Möglichkeit gibt das Konzept der Replikatsserver in unterschiedlichen Hypervisoren herzustellen, indem VMs den selben Systemzustand verfolgen, gilt es zu untersuchen.

2.3 Abgrenzung

Das Umfeld, in dem die Untersuchungen stattfinden, bietet viele Möglichkeiten um in einer komplexen Architektur Lösungen auf ihre Anwendbarkeit zu prüfen. Je mehr Projekte im Living Place realisiert werden, desto mehr wird sich die Laborumgebung an ein reales Umfeld anpassen. Damit sinkt die Chance Ausnahmefälle außer Acht zu lassen, die in einer Laborumgebung zunächst nicht auftreten würden. Durch den direkten Zugriff auf die meisten Projekte können Einzelkonzepte und Ideen nachvollzogen und im praktischen Einsatz verglichen werden. Die Untersuchungen werden keine Gesamtlösung hervorbringen, sondern viel mehr den Einblick in die Problematik schärfen und ein Problembewusstsein schaffen, das bei folgenden Entwicklungen hilfreich sein wird.

Die Zeitplanung für die Voruntersuchungen der Masterarbeit umfasst drei Monate. Von März bis Mai 2011 sollen die Theorien aus Abschnitt 2.2 untersucht werden, vgl. Abb. 2.1. Dabei sieht die Reihenfolge vor, dass Erkenntnisse aus dem Bereich des Monitorings Vorteile für Untersuchungen in Bereichen der Verteilung und Ausfallreaktion bringen werden. Bei Neuentwicklungen sollen nach einer Einarbeitung in das Monitoring, Programmiersprachen verwendet werden die automatische Synchronisationsmechanismen realisieren. Die Kommunikationsschnittstelle wird im Anschluss auf Replikationsfähigkeit und Erweiterbarkeit geprüft. Mit den konkreten Ergebnissen über die Lastverteilung des verteilten Systems kann im Anschluss die Untersuchung weitergeführt werden.

2.4 Chancen und Risiken

Die zu untersuchenden Konzepte werden auf verschiedenen Gebieten Erkenntnisse erzeugen, die eventuell miteinander kombiniert werden können. Nachdem die ersten Kon-

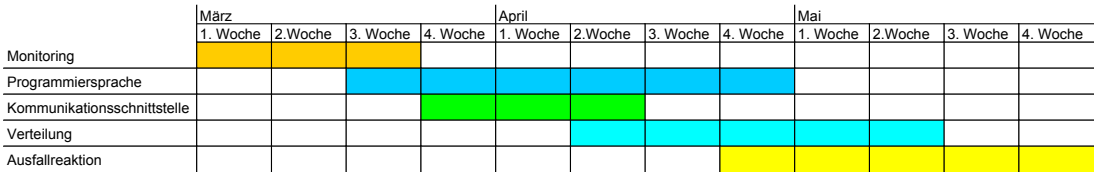

Abb. 2.1: Zeiplanung zu Voruntersuchungen der Masterarbeit

zepte getestet wurden, können Erkenntnisse eventuell auch auf andere Teilbereiche übertragen werden und somit eine Wechselwirkung bilden, in der sich unterschiedliche Themenbereiche gegenseitig mit Lösungsansätzen helfen. Durch die enge Zusammenarbeit mit anderen Masterprojekten werden die hier genannten Ansätze weiterentwickelt und eventuell neue Anregungen für Folgeuntersuchungen gegeben.

Die Untersuchungen zu den in Abschnitt 2.2 genannten Themengebieten werden sehr umfangreich und damit einhergehend komplex. Zum einen stellt das ein Risiko dar, da der Überblick über die Erkenntnisse schnell verloren werden kann. Zum anderen kann es vorkommen, dass für bestimmte Untersuchungen das Umfeld nicht ausreicht. Zum Beispiel im Bezug auf die Skalierbarkeit von verteilten Systemen kann nicht im voraus festgelegt werden, ab wie vielen Computern der Performancegewinn dem Overhead unterliegt. Eventuell kann auch nicht jede Untersuchung den exakt gleichen Rahmenbedingungen ausgesetzt werden, da sich die Projekte im Living Place ständig weiterentwickeln.

3 Zusammenfassung

Computer entwickeln sich stetig weiter und folgen dem Moor'schen Gesetz nachdem sich die Anzahl der Transistoren alle 12 bis 18 Monate verdoppelt. Auf die damit einhergehende Leistungssteigerung muss die Software reagieren. Auch im Living Place Hamburg werden viele Computer zum Einsatz kommen, die mehr Rechenleistung bereitstellen als ihre Projekte benötigen. Dieser Umstand macht das Labor zu einem idealen Umfeld für Untersuchungen in Bezug auf Skalierbarkeitstheorien und parallele Datenverarbeitung. Die Vorarbeiten in diesem Bereich konnten erste Erkenntnisse erzeugen und lassen somit Folgerungen zu die es zu Untersuchen gilt. Das hochparallele Umfeld bietet reale Bedingungen für Tests. Systematisch sollen Teilbereiche der Informatik auf Konzepte zur Steigerung der Effizienz untersucht werden und gegebenenfalls neue Erkenntnisse zur Weiterentwicklung vorhandener Konzepte beisteuern. Praktische Tests werden zeigen, ob die aufgestellten Theorien praxisrelevante Ergebnisse liefern können.

3.1 Ausblick

Eine Anstellung als Werkstudent bei der Firma Fast Lane GmbH wird in Zukunft zu weiteren Erkenntnissen auf dem Gebiet der Virtualisierung und Infrastruktur beitragen. Die dort eingesetzten Technologien zur Schulung von Virtualisierunglösungen, werden Einblicke in benötigte Software, Hardware und Vorwissen geben. Diese Lösungen können im Living Place bei verteilten Anwendungen oder beim Zusammenfassen von Projekten auf virtuelle Maschinen Einsatz finden.

Literaturverzeichnis

[1] AMAZON WEB SERVICES LLC: *Amazon Elastic Compute Cloud (Amazon EC2)*. Amazon Web Services LLC. 2010. – URL http://aws.amazon.com/ec2/. – [Online; Stand 25. Februar 2011]

[2] CHANG, Fay ; DEAN, Jeffrey ; GHEMAWAT, Sanjay ; HSIEH, Wilson C. ; WALLACH, Deborah A. ; BURROWS, Mike ; CHANDRA, Tushar ; FIKES, Andrew ; GRUBER, Robert E.: *Bigtable: A Distributed Storage System for Structured Data*. Google, Labs. 2006. – URL http://labs.google.com/papers/bigtable.html. – [Online; Stand 21. Mai 2011]

[3] CHAROUSSET, Dominik: Softwarearchitekturen für die Entwicklung verteilter Anwendungen. In: *Ausarbeitung Anwendungen 2 SoSe 2009* (2009)

[4] DEAN, Jeffrey ; GHEMAWAT, Sanjay: *MapReduce: Simplifed Data Processing on Large Clusters*. Google, Labs. 2010. – URL http://labs.google.com/papers/mapreduce.html. – [Online; Stand 23. Mai 2010]

[5] GROFF, Dana: *STM.NET DevLab Incubation Complete*. 2010. – URL http://blogs.msdn.com/b/stmteam/archive/2010/05/12/stm-net-devlab-incubation-complete.aspx. – [Online; Stand 28. August 2010]

[6] GROUP, Object M.: *The OMGs CORBA Website*. 2011. – URL http://www.corba.org/. – [Online; Stand 24. Februar 2011]

[7] HEWITT, Carl ; BISHOP, Peter ; STEIGER, Richard: A universal modular ACTOR formalism for artificial intelligence. In: *IJCAI'73: Proceedings of the 3rd international joint conference on Artificial intelligence*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1973, S. 235–245

[8] KLAUS IRMSCHER, Prof. Dr.-Ing. habil.: Scriptum zur Lehrveranstaltung Verteilte Systeme. (2011). – URL http://www.informatik.uni-leipzig.de/~irmscher/lehre/skripte/VerteilteSystemeScriptum.pdf

[9] NAGIOS ENTERPRISES: *Nagios*. Nagios Enterprises. 2011. – URL http://www.nagios.org/. – [Online; Stand 25. Februar 2011]

[10] NVIDIA CORPORATION: *nVidia CUDA*. NVIDIA Corporation. 2010. – URL http://www.nvidia.de/object/cuda_research_de.html. – [Online; Stand 21. Februar 2011]

[11] OTTO, Kjell: Clojure - concurrency revisited. In: *Hochschule für Angewandte Wissenschaften Hamburg* (2010). – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-aw1/otto/bericht.pdf. – [Online; Stand 05. Februar 2010]

[12] OTTO, Kjell: Concurrent Programming Models - Related Work. In: *Hochschule für Angewandte Wissenschaften Hamburg* (2010). – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2010-aw2/otto/bericht.pdf. – [Online; Stand 21. Februar 2011]

[13] OTTO, Kjell ; VOSKUHL, Soeren: Entwicklung einer Architektur für den Living Place Hamburg. (2010). – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2010-proj1/otto_voskuhl.pdf

[14] OTTO, Kjell ; VOSKUHL, Soeren: Entwicklung einer Architektur für den Living Place Hamburg. (2011). – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master10-11-proj2/otto_voskuhl.pdf

[15] SCALA-LANG.ORG: *Scala Actors: A Short Tutorial*. 2010. – URL http://www.scala-lang.org/node/242. – [Online; Stand 19. Mai 2010]

[16] THE APACHE SOFTWARE FOUNDATION: *Hadoop Distributed File System*. The Apache Software Foundation. 2011. – URL http://hadoop.apache.org/hdfs/. – [Online; Stand 21. Februar 2011]

[17] VISION LAB: *FASTRA II*. Department of Physics University of Antwerp (CDE). 2010. – URL http://fastra2.ua.ac.be/. – [Online; Stand 21. Februar 2011]

[18] VOLKMANN, Mark: *Software Transactional Memory*. 2010. – URL http://java.ociweb.com/mark/stm/article.html. – [Online; Stand 14. Februar 2010]