Auslagerung der Ausführung von Methoden der HYPRE Bibliothek in ein Cloudsystem Recherche und Literaturverzeichnis

Thomas Rückert

October 20, 2017

Abstract

abstract formulieren

Contents

List of Figures

List of Tables

Listings

Todo list

allgemeine einführung inklusive übersicht

1 Einführung in die relevanten Themen

1.1 Allgemeines und Recherche zum Thema Cloud

check das hier:

- nur innerhalb der cloud (ungeeignet): Light-weight remote communication for high-performance cloud networks `http://ieeexplore.ieee.org/document/6483669/`
- Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud `http://ieeexplore.ieee.org/abstract/document/5708447/`

1.1.1 Einführung und Grundbegriffe zu Cloudcomputing

Entwicklung von monolitischen Systemen zu verteilten Anwendungen (SOA, Client/Server). Ermöglicht Auslagerung kostenpflichtiger Berechnungen auf Server, 'schwacher' Client kein Problem mehr. Serverlast kann bei Anwendungen stark schwanken. Zum Beispiel periodische Schwankungen Tag vs Nacht. Klassischer Server muss die hohe Last stemmen, hat dann in anderen Perioden starken leerlauf. Einmalige, sehr hohe Last bei besonderen Situationen (zum Beispiel durch einmalige, nicht wiederkehrende Sportevents wie Olympia/WM). Klassischer Server könnte in diesem Zeitraum komplett ausfallen. Cloud soll dynamische Resource sein, die sich je nach Bedarf skalieren kann.

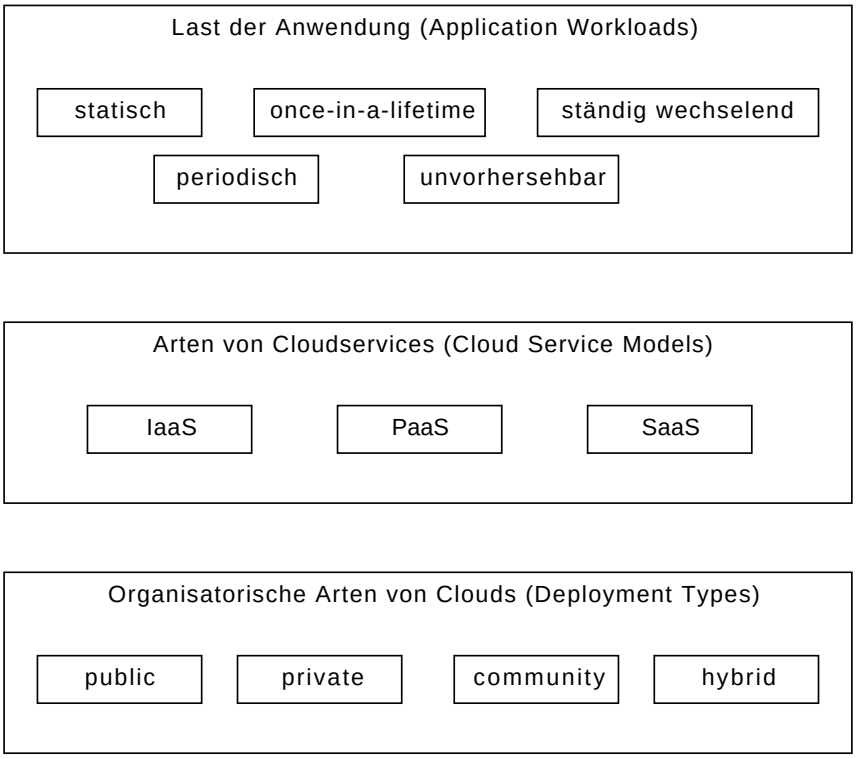
Im Folgenden werden die eben kurz angeschnittenen Eigenschaften von Cloudsystemen näher betrachtet. Ein Übersicht dazu ist in Abbildung 1.1 gegeben.

Verteilung der Last von Anwendungen Im Buch Cloud Computing Patterns von Christoph Fehling wird die Verteilung von Last in die folgenden Kategorien eingeteilt:

- static workload
- periodic workload
- once-in-a-lifetime workload
- unpredictable workload
- continuously changing workload

Static workload beschreibt eine nicht oder nur minimal schwankende Last. **Periodic workload** hat dagegen wiederkehrende Schwankungen. Diese können zum Beispiel von der Tageszeit abhängig sein. Ein **Once-in-a-lifetime workload** ist eine einmalige Lastspitze. **Unpredictable workload** liegt vor, wenn sich die Last ständig, jedoch unregelmäßig und zufällig verändert, sodass diese nicht vorhersehbar ist. **Continuously changing workload** verändert sich in linear steigend oder fallend.

Figure 1.1: Übersicht Grundlagen Cloud Computing



NIST definition introduces five fundamental properties that characterize a cloud offering [CloudcomputingPatterns chap. 1.1]

- On-demand self-service
- Broad network access
- Measured service (pay-per-use)
- Resource pooling
- Rapid elasticity

On-demand self-service 'Provisioning' und 'decomissioning' als Aktivitäten zum Hinzufügen oder Entfernen von weiteren Ressourcen. Das kann durch Benutzer über grafische oder Kommandozeilenschnittstellen geschehen oder automatisiert über eine API.

Broad network access Ein starkes Netzwerk ist essentiell um eine Verbesser durch die Auslagerung von Berechnungen zu erreichen. So kann Zugriffszeit auf Daten weniger abhängig von ihrem pysikalischen Speicherort werden.

genauer definieren

Measured service (pay-per-use) Durch die Nutzung von Cloudsystemen kann man stark von der Flexibilität der Ressourcen profitieren. Diese Flexibilität muss sich auch im Bezahlmodell widerspiegeln.

Resource pooling Ein Cloudsystem benötigt einen (großen) Pool an Ressourcen. Nur so kann Flexibilität für die Nutzer gewährleistet werden. Um eine Austauschbarkeit der Ressourcen zu ermöglichen muss eine homogene Nutzung der Ressourcen existieren.

genauer definieren

warum? flexible Nutzung, Kosten

Rapid elasticity Elastizität von Cloudsystemen ermöglicht eine Effiziente Zuweisung von Ressourcen auf die Nutzer. Der Ressourcepool muss dynamisch unter den Nutzern aufgeteilt werden können.

IDEAL cloud-native applications [CloudcomputingPatterns chap. 1.2]

- Isolated state
- Distribution
- Elasticity
- Automated management
- Loose coupling

Isolated state Cloudanwendungen und ihre Komponenten sollten zustandslos sein. Jede Ressource die zustandslos ist kann deutlich einfacher entfernt oder hinzugefügt werden als eine Ressource mit einem Zustand. So können aufeinander folgende Interaktionen eines Nutzers beliebig auf verschiedene Ressourcen verteilt werden. Eine Ressource die beispielsweise die erste Interaktion getätigt hat wird für weitere Interaktionen nicht mehr benötigt.

Distribution Cloudsysteme können auf viele verschiedene Standorte verteilt sein. In jedem Fall bestehen sie aus vielen verschiedenen Ressourcen. Anwendungen sollten daher aus mehreren Komponenten bestehen, die auf verschiedene Ressourcen verteilt werden können.

Elasticity Horizontale Skalierung statt vertikaler Skalierung: Anwendung soll vom Hinzufügen weiterer Ressourcen profitieren können (horizontal). Es soll nicht nur die 'Verbesserung' einer Ressource eine bessere performance ermöglichen (vertikal). Die Stärke von Cloudsystemen ist die dynamische Zuweisung von Ressourcen. Cloudanwendungen müssen daher horizontal skalieren, also eine Parallelisierbarkeit vorweisen.

Automated management Durch die Elastizität können Ressourcen von Cloudanwendungen während der Laufzeit ständig hinzugefügt und entfernt werden. Diese Aktionen sollten aufgrund von Monitoring der Systemlast ausgelöst werden. Damit die Verwaltung der Ressourcen jederzeit schnell und entsprechend der aktuellen Lage stattfindet sollte sie automatisiert sein.

Loose coupling Da sich die verfügbaren Ressourcen der Anwendung während der Laufzeit ändern können sollten Komponenten möglichst unabhängig voneinander sein. Das reduziert die Fehleranfälligkeit für die Fälle in denen Komponenten kurzzeitig nicht verfügbar sind. Da verteilte Anwendung diese Eigenschaft aufweisen sind Technologien wie Webservices, SOA, asynchrone Kommunikation relevant für Cloudanwendungen.

wie?

Technologien etwas weiter ausführen

1.1.2 Kategorisierung von Cloudsystemen

Cloud Service Models

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)

Infrastructure as a Service gibt einem Zugang zu Netzwerk, Computern (unter umständen virtuell) und Speicher. Es ist daher sehr nah an den schon länger existierenden und bekannten Hosted Server Lösungen. **Platform as a Service** dageben entfernt die Notwendigkeit die unterliegende Infrastruktur zu verwalten. Das betrifft Hardware sowie die Betriebssystemebene. Man erhält eine Umgebung in der man seine Anwendung ausführen kann, ohne sich um Themen wie Updates, Kapazitäten von Speicher und anderen Ressourcen oder ähnliche typische Adminaufgaben kümmern zu müssen. **Software as a Service** stellt ein Cloudmodell dar, welches sich eher an Endnutzer richtet. Man erhält Zugriff auf eine Komplette Anwendung, wie zum Beispiel einen Mailserver. Bei dieser Variante muss sich der Nutzer um keinerlei Aufgaben der Verwaltung der Software kümmern.

wie zum beispiel, kurz erläutern

Organisatorische Arten von Clouds (Deployment Types)

- public
- private
- community
- hybrid

Eine **public cloud** ist für jeden verfügbar. Eine **private cloud** dagegen nur für ein einziges Unternehmen oder einen Nutzer (od. Interessengemeinschaft). Eine **community cloud** liegt zwischen den beiden ersten Varianten. Sie ist in der Regel für eine Menge von Unternehmen verfügbar. Das kann notwendig werden, falls die Unternehmen an einem gemeinsamen Projekt arbeiten. Bei **hybrid clouds** ist von mehreren Clouds die Rede. Diese können aus verschiedenen Arten bestehen und sind untereinander verbunden. So können sich unterschiedliche Anwendungen unter eigenständigen Umgebungen Informationen austauschen und interagieren.

1.1.3 Cloudsysteme

Vergleich in Tabelle? - Konflikt zu chpt. 'Gegenüberstellungen ...'

Verfügbare Cloudsysteme für public Clouds

Amazon EC2 `https://www.youtube.com/watch?v=jLVPqoV4YjU&index=3&list=WL` ssh Zugang siehe ab Minute 50

Instances mit verschiedenen verfügbaren Images = AMI (Linux, Windows). ELB als elastischer Loadbalancer. EBS - elastischer Blockstorage. CloudWatch zum Überwachen von Ressourcen und Applikationen. Kann zum automatischen Skalieren genutzt werden. EC2 Actions - Recover, Stop, Terminate. AWS CodeDeplay erlaubt deploying ohne downtime. Amazon Amazon EC2 Container Service ist ein Containermanagementservice zur Nutzung von Docker Containern.

Kostenloser Testzugang: `https://aws.amazon.com/free/`

Open Source Cloudsystem für private Clouds

- a guide to open source cloud `http://www.tomsitpro.com/articles/open-source-cloud-computing-software,2-754.html`
- 5 open source cloud platforms `http://solutionsreview.com/cloud-platforms/open-source-cloud-platforms-enterprise/`

Sandstorm Link: `https://sandstorm.io/`

Kann als private oder public cloud genutzt werden. Kann entweder im bestehenden Cloudsystem von Sandstorm genutzt werden oder selbst gehostet werden. Im Cloudsystem von Sandstorm stehen SaaS und PaaS bereit. Wenn man selbst hostet sind alle Service Modelle verfügbar, also zusätzlich auch IaaS.

Die verfügbaren Platformen stellen ein Linuxsystem bereit. Daher kann jede beliebige Sprache genutzt werden welche auf Linux läuft. Ist aber in erster Linie für SaaS-Apps gedacht.

Fazit: wohl ungeeignet

Openstack Link: `https://www.openstack.org/`
Openstack kostenlos testen: `http://trystack.org/`
devstack for getting started with openstack `http://docs.openstack.org/developer/devstack/`
install openstack on ubuntu `https://www.youtube.com/watch?v=jpk4i66-IU4`

why openstack? `https://www.youtube.com/watch?v=Bk4NoUsikVA`

Wird in erster Linie als IaaS verwendet. Man kann bestehende Hardware mit Cloudstack in ein Cloudsystem umwandeln. Es stehen verschiedene Services bereit, mit denen Kernanforderungen wie Speicher (Swift) oder Rechenleistung (Nova) befriedigt werden können. Es gibt darüber hinaus weitere optionale Services für beispielsweise Datenbanken (Trove), Messaging (Zaqar), Container (Magnum) und viele Weitere.

Beispielkonfiguration: `https://www.openstack.org/software/sample-configs#high-throughput-computing`
GLANCE - Image Service 'Stores and retrieves virtual machine disk images. OpenStack Compute makes use of this during instance provisioning.'
NOVA - Compute 'Manages the lifecycle of compute instances in an OpenStack environment. Responsibilities include spawning, scheduling and decomissioning of machines on demand.'
KEYSTONE - Identity 'Provides an authentication and authorization service for other OpenStack services. Provides a catalog of endpoints for all OpenStack services.'
CINDER - Block Storage 'Provides persistent block storage to running instances. Its pluggable driver architecture facilitates the creation and management of block storage devices.'

Fazit: modulare Services passend für Anforderungen

Apache Cloudstack

Definition

Apache Cloudstack `https://cloudstack.apache.org/`

'Apache CloudStack is an open source Infrastructure-as-a-Service platform that manages and orchestrates pools of storage, network, and computer resources to build a public or private IaaS compute cloud.

`https://cloudstack.apache.org/`

With CloudStack you can:

Set up an on-demand elastic cloud computing service. Allow end-users to provision resources' [http://docs.cloudstack.apache.org/en/latest/concepts.html]

private IaaS

1.1.4 Ressourcen

- amazon aws `https://aws.amazon.com/types-of-cloud-computing/`
- BOOK: cloud computing patterns `https://katalog.bibliothek.tu-chemnitz.de/Record/0012763915`
- raspberry pi private cloud `https://sc5.io/posts/a-private-raspberry-pi-cloud-with-arm-docker/`
- more raspberry `https://www.raspberrypi.org/forums/viewtopic.php?f=36&t=54997`

1.2 Mathematische Grundlagen

Laplace-Gleichung Partielle Differentialegleichung. `https://www.youtube.com/watch?v=Z9NjA_f2VZw` `https://www.youtube.com/watch?v=-D4GDdxJrpg`

Grundlagen für HYPRE: 2D Laplace

1.3 Parallele Programmierung

Die absolute Ausführungszeit ist ein Hauptkriterium für die Performanz. Sie sollte möglicht gering sein. Ab einem gewissen Punkt kann die sequentielle Ausführung jedoch nicht mehr wesentlich optimiert werden. Sollte es im Algorithmus jedoch unabhängige Berechnungen geben kann eine parallele Ausführung eine deutlich Verbesserung der Ausführungszeit erreichen. Parallele Programme werden immernoch auf einem einzelnen Computer ausgeführt. An der Ausführung sind mehrere Prozesse beteiligt. Diese kommunizieren über einen gemeinsamen Speicher.

Grundbegriffe Paralleler Ausführung Die Ausführungszeit läuft bei einer parallelen Berechnung ab dem Start des Programmes bis die Ausführung des letzten teilnehmenden Prozessors beendet ist(Rauber & Rünger, 2013). Diese Zeit nennt sich parallele Ausführungszeit $T_p(n)$, wobei n die Problemgröße beschreibt. Kriterien zur Bewertung eines parallelen Programmes sind Kosten, Speedup und Effizienz.

Die Kosten (Rauber & Rünger, 2013, 180) $C_p(n)$ ergeben sich aus dem Produkt der parallelen Ausführungszeit $T_p(n)$ und der Anzahl der Prozessoren p:

$$C_p(n) = T_p(n) * p$$

Die Kosten geben so die insgesamt notwendige Arbeit an.

Der Speedup (Rauber & Rünger, 2013, 180-182) gibt die Verbesserung der parallelen Implementierung im Vergleich zur sequentiellen Implementierung an. Er wird durch $S_p(n)$ beschrieben und ergibt sich aus der Division von der Ausführungszeit der besten sequentiellen Lösung $T^*(n)$ durch die parallele Ausführungszeit $T_p(n)$:

$$S_p(n) = \frac{T^*(n)}{T_p(n)}$$

Der Speedup gibt somit die relative Ersparnis der parallelen Ausführung im Vergleich zur Sequentiellen an.

Die Effizienz (Rauber & Rünger, 2013, 182-183) $E_p(n)$ ist ein Maß für die Performanz eines parallelen Programms. Sie kann auf verschiedene Arten bestimmt werden: Die Division besten sequentiellen Lösung $T^*(n)$ durch die Kosten $C_p(n)$. Oder der Division des Speedup $S_p(n)$ durch die Anzahl der Prozessoren p.

$$E_p(n) = \frac{T^*(n)}{C_p(n)} = \frac{S_p(n)}{p}$$

Eine optimale Effizienz wird durch den Wert 1 beschrieben $E_p(n) = 1$. Diese wird erreicht wenn $S_p(n) = p$ ist.

Verteilte Ausführung Verteilte Systeme bestehen im Unterschied zur parallelen Programmierung. Hierbei wird nicht mehr nur auf einem Computer gerechnet. Es befinden sich stattdessen mehrere autonome in einem Netzwerk(DeNero, 2017). Diese kommunizieren über den Austausch von Nachrichten. Ein Standard für den Nachrichtenaustausch im verteilten sowie parallelen Rechnen ist das Message-Passing Interface - MPI(MPI-Forum, 2017).

MPI MPI wird beschrieben als "message-passing library interface specification" ((Forum, 2015, 1)).

MPI definiert einen Standard für message passing Bibliotheken, also für Nachrichtenübertragung. Der Standard wird vom MPI Forum spezifiziert. Er existiert in verschiedenen Versionen. 1994 wurde die erste Version veröffentlicht: MPI-1.0(Forum, 2015, ii). Die derzeit aktuellste Version ist MPI-3.1, welche seit 2015 existiert.

Es existiert eine Reihe von Implementierungen. Drei frei verfügbare Beispiele sind MPICH, LAM/MPI und OpenMPI[1](Rauber & Rünger, 2013, 228). Wobei LAM/MPI einer mehrerer Vorgänger von OpenMPI ist.

Nutzt MPI für Parallelisierbarkeit: MPI - Message Passing Interface.

1.4 HYPRE - Überblick über die Bibliothek

HYPRE ist eine freie Software die vom Center for Applied Scientific Computing(CASC, 2017a) entwickelt wird. Dieses ist ein Teil der Organisation Lawrence Livermore National Laboratory(LLNL, 2017), einer der wichtigsten staatlichen Forschungseinrichtungen der USA(CASC, 2017b).

[1]MPICH: `http://www.mpich.org`, LAM/MPI: `http://www.lam-mpi.org`, OpenMPI: `http://www.open-mpi.org`

HYPRE ist unter der GNU Lesser General Public License (Free Software Foundation) Version 2.1 lizensiert. Diese ermöglicht die Benutzung der Bibliothek HYPRE sowohl in freier als auch proprietär lizensierter Software. Der Funktionsumfang von HYPRE wird beschrieben als 'Scalable Linear Solvers and Multigrid Methods'.

Die aktuellste Version 2.11 ist seit dem 09.06.2016 verfügbar. Die Bibliothek steht in dieser Version für die Sprachen C (nativ), C++ und FORTRAN bereit. Bis zur vorletzten Minorversion 2.10 wurde HYRPE auch mit dem Babel Interface angeboten. Dieses bot die Möglichkeit HYPRE von anderen Sprachen als C und FORTRAN zu nutzen. So wurde im User Manual die Verwendung aus der Sprache Python beschrieben. Seit Version 2.11.1 steht jedoch keine Implementierung mehr von HYPRE mit Babel Interface zur Verfügung.

1.4.1 Funktionsumfang (und deren Funktionsweise)

Wie bereits erwähnt wird der Funktionsumfang von HYPRE als 'Scalable Linear Solvers and Multigrid Methods' beschrieben. Zur detaillierteren Erklärung von HYPRE kann man diese Beschreibung genauer Betrachten:

Begriffsklärung Der Begriff **Solver** beschreibt ein Stück Software, dieses kann beispielsweise als Bibliothek implementiert sein. Mit Hilfe dieses Stücks Software kann man ein mathematisches Problem lösen. Für ein Problem kann es verschiedene Implementierungen von **Solvern** geben. Diese können bei unterschiedlichen Faktoren der Eingabedaten Vor- und Nachteile haben. Diese können beispielsweise in Performance oder Genauigkeit liegen.

Linear beschreibt die Art von Problemen die HYPRE lösen kann. Es handelt sich dabei um die lineare Algebra. Ein wichtiger Teil der linearen Algebra sind lineare Gleichungssysteme. erweitern

Das zweite Adjektiv zum Begriff Solver ist **scalable**, also skalierbar. Es gibt also einen Hinweis darauf, dass die Implementierungen der Solver in HYPRE auch oder besonders bei großen Eingaben eine gute Performanz aufweisen. Für die Bewertung wird bei einer wachsenden Problemgröße ebenso die Rechenkapazität erhöht. Skalierbarkeit einer Funktion liegt dann vor, wenn die nötige Ausführungszeit dabei konstant gehalten werden kann.(Henson & Yang, 2000) Dabei ist es in der Regel hilfreich, dass rechenintensive Teile parallel ausgeführt werden können. Man spricht in diesem Fall von horizontaler Skalierung, also dem hinzufügen weiterer Berechnungsknoten. Das können beispielsweise weitere Rechner sein die einem Cluster hinzugefügt werden. Um dabei eine verbesserte Leistung zu erhalten muss die Software eine entsprechende Skalierung unterstützen. Die Berechnungen müssen also in voneinander unabhängige Teile gespalten werden können. Im Gegensatz dazu steht die vertikale Skalierung. Hier hat man eine einzige Ressource deren Rechenleistung erhöht wird. Diese ist für eine Software in der Regel einfacher zu nutzen, stößt allerdings schneller an Grenzen. So kann die die Signalfrequenz einer CPU zum Beispiel nicht beliebig weit erhöht werden.

Der zweite Teil der Beschreibung lautet **Multigrid Methods** steht für Mehrgitterverfahren. Diese stellen eine Approximationslösung für Differentialgleichungen dar. Dafür werden die Eingabedaten auf eine Weise ausgedünnt, bei der sich die Lösung so wenig wie möglich verändert. Eine solche Verkleinerung der Daten ist bei Problemen mit bis zu Milliarden von Unbekannten wichtig.

So kann die Komplexität von Problemen erheblich reduziert werden. HYPRE verwendet beispielsweise Algebraic Multigrid Methods (AMG).

Eine parallele Implementierung davon ist **BoomerAMG**. Dieser wurde bereits für Probleme mit dutzenden Millionen Unbekannten auf Maschinen mit über Tausend Prozessoren eingesetzt.(Henson & Yang, 2000) Mehrgitterverfahren können als sogenannte Preconditioner, oder Vorkonditionierer, eingesetzt werden. Dabei werden die Eingabedaten noch vor dem Lösen optimiert. BoomerAMG kann beispielsweise direkt als Solver verwendet werden. Ebenso kann es aber auch als Preconditioner verwendet werden und anschließend ein anderer Solver zum Einsatz kommen. Beispielsweise bei der Verwendung des Krylow-Unterraum-Verfahren ist eine Vorkonditionierung wichtig. Dieses ist auf das Lösen dünnbesetzter Gleichungssysteme optimiert. Wenn die Eingabedaten diesem Kriterium nicht entsprechen sollten sie also zunächst mit einem anderen Verfahren vorkonditioniert werden.

Im wesentlichen stellt HYPRE also Funktionen zum Lösen linearer Algebra bereit. Diese sind soweit optimiert, dass sie skalieren. Die Performanz der Berechnungen kann durch Mehrgitterverfahren verbessert werden. BoomerAMG ist ein Beispiel für ein solches Verfahren welches parallelisierbar ist.

Performanz Nutzt MPI für Parallelisierbarkeit: MPI - Message Passing Interface.

performance

Conceptual Interfaces

Solver Strategies

Preconditioner(s) Preconditioning (Fritzsche, 2010)

1.4.2 Verwendung und Implementierung

Installation Um HYPRE zu installieren sind einige Abhängigkeiten zu erfüllen. Die nötigen Schritte in einem Ubuntu sehen wie folgt aus:

Listing 1: Installation HYPRE in Ubuntu

```
cd /vagrant

//yaourt mpich2
//yaourt openmpi
sudo apt-get install mpich

//Download https://github.com/LLNL/hypre
git clone https://github.com/LLNL/hypre

//Innerhalb des Verzeichnis muss `configure`
//gefolgt von einem `make` ausgef hrt werden.
cd hypre/src

./configure
```

```
//Alternativ kann auch das build tool CMake
//eingesetzt werden.
make
```

Implementierung

Checkliste Die folgenden Punkte beschreiben grob die notwendigen Schritte eines Workflows für die Implementierung.

1. Build any necessary auxiliary structures for your chosen conceptual interface. / Datenstrukturen für Gitter (Grid) und Schablone (Stencil) aufbauen, abhängig vom gewählten Interface.
2. Build the matrix, solution vector, and right-hand-side vector through your chosen conceptual interface. / Matrix, Lösungsvektor, right-hand-side Vektor aufbauen. Diese können über verschiedene HYPRE-calls mit den jeweiligen Informationen gefüllt werden.
3. Build solvers and preconditioners and set solver parameters (optional). / Solver und preconditioner aufbauen und deren Parameter setzen. Je nach conceptual interface sind verschiedene solver verfügbar.
4. Call the solve function for the solver. / Solver aufrufen damit das Problem berechnet wird.
5. Retrieve desired information from solver. / Lösung vom solver abrufen.

Zunächst müssen die Datenstrukturen für die In- und Outputdaten definiert werden. Dies können beispielsweise Matrizen oder Vektoren sein. Die Entscheidung über das Inputformat ist abhängig vom gewählten Input-Interface.

Die gewählten Datenstrukturen müssen als nächstes mit den Inputdaten gefüllt werden. HYPRE stellt dafür verschiedene Calls bereit. beispiele

Der nächste Schritt ist das Anlegen des gewünschten Solvers. Es stehen verschiedene Solver bereit(CASC, 2017d), beispielsweise SMG (a parallel semi-coarsening multigrid solver) oder BoomerAMG (parallel implementation of the algebraic multigrid method).

1.4.3 Beispielhafter Einblick

In der HYPRE-Bibliothek steht eine Reihe von Beipielen für die Benutzung zur Verfügung. An dieser Stelle wird das Beispiel 5 näher betrachtet. Dieses löst ein zweidimensionales Laplaceproblem (nxn) ohne Randbedingungen. Die Anzahl der Unbekannten beträgt daher N=n. Die komplette Implementierung des Beispiel 5 ist im Anhang zu finden.

Vorbereitungen Da HYPRE für die Parallelisierung auf die MPI Bibliothek zurückgreift muss dieses auch zu Beginn der Anwendung initialsiert werden. Dafür muss durch

```
MPI_INIT()
```

eine grundsätzliche MPI-Session gestartet werden. Weiterhin müssen für die Prozesse ein Kommunikator bestimmt und die eigene id sowie die gesamte Anzahl der Prozesse gelesen werden. Dafür werden die Methoden

`MPI_Comm_rank()`

und

`MPI_Comm_size()`

aufgerufen.

Listing 2: HYPRE Nutzung: Vorbereitungen

```
//init mpi
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myid);
MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
```

Matrix anlegen (analog zu Vektor) Matrizen und Vektoren anlegen. Es stehen analoge Methoden für Matrizen und Vektoren bereit. Im folgenden Listing 3 sind die wichtigen Methoden daher lediglich für das Beispiel der Matrix aufgeführt.

Listing 3: HYPRE Nutzung: Matrix anlegen (analog dazu Vektor)

```
//create matrix / vector

HYPRE_IJMatrix A;
int ilower;
int iupper;

//...

/* Create the matrix.
  Note that this is a square matrix, so we indicate the
     row partition
  size twice (since number of rows = number of cols) */
HYPRE_IJMatrixCreate(MPI_COMM_WORLD, ilower, iupper,
    ilower, iupper, &A);

/* Choose a parallel csr format storage (see the User's
    Manual) */
HYPRE_IJMatrixSetObjectType(A, HYPRE_PARCSR);

/* Initialize before setting coefficients */
HYPRE_IJMatrixInitialize(A);

//in einem loop:
HYPRE_IJMatrixSetValues(A, 1, &nnz, &i, cols, values);
```

```
/* Assemble after setting the coefficients */
HYPRE_IJMatrixAssemble(A);

/* Get the parcsr matrix object to use */
HYPRE_IJMatrixGetObject(A, (void**) &parcsr_A);
```

Solver anlegen und aufrufen Solver erstellen und aufrufen.

Listing 4: HYPRE Nutzung: Solver anlegen und aufrufen

```
//create solver
//(Boomer)

/* Create solver */
HYPRE_BoomerAMGCreate(&solver);

/* Now setup and solve! */
HYPRE_BoomerAMGSetup(solver, parcsr_A, par_b, par_x);
HYPRE_BoomerAMGSolve(solver, parcsr_A, par_b, par_x);

/* Destroy solver object */
HYPRE_BoomerAMGDestroy(solver);
```

mit Preconditioner Solver mit anderem Preconditioner.

Listing 5: HYPRE Nutzung: Solver mit Preconditioner

```
//erstelle PCG mit AMG als preconditioner

/* Create solver */
HYPRE_ParCSRPCGCreate(MPI_COMM_WORLD, &solver);

/* Now set up the AMG preconditioner and specify any
   parameters */
HYPRE_BoomerAMGCreate(&precond);

/* Set the PCG preconditioner */
HYPRE_PCGSetPrecond(solver, (HYPRE_PtrToSolverFcn)
   HYPRE_BoomerAMGSolve,
                       (HYPRE_PtrToSolverFcn)
                          HYPRE_BoomerAMGSetup,
                          precond);

/* Now setup and solve! */
HYPRE_ParCSRPCGSetup(solver, parcsr_A, par_b, par_x);
HYPRE_ParCSRPCGSolve(solver, parcsr_A, par_b, par_x);

/* Destroy solver and preconditioner */
HYPRE_ParCSRPCGDestroy(solver);
HYPRE_BoomerAMGDestroy(precond);
```

weitere Parameter Vor dem Ausführen des solvers können weitere Parameter gesetzt werden.

Listing 6: HYPRE Nutzung: Solver mit Preconditioner

```
/* Set some parameters (See Reference Manual for more
    parameters) */
HYPRE_BoomerAMGSetPrintLevel(solver, 3);  /* print solve
    info + parameters */
HYPRE_BoomerAMGSetOldDefault(solver); /* Falgout
    coarsening with modified classical interpolaiton */
HYPRE_BoomerAMGSetRelaxType(solver, 3);   /* G-S/Jacobi
    hybrid relaxation */
HYPRE_BoomerAMGSetRelaxOrder(solver, 1);   /* uses C/F
    relaxation */
HYPRE_BoomerAMGSetNumSweeps(solver, 1);   /* Sweeeps on
    each level */
HYPRE_BoomerAMGSetMaxLevels(solver, 20);  /* maximum
    number of levels */
HYPRE_BoomerAMGSetTol(solver, 1e-7);      /* conv.
    tolerance */
```

Ergebnis auswerten Nach dem Lösen des Problems können die Ergebnisse gelesen werden.

Listing 7: HYPRE Nutzung: Solver mit Preconditioner

```
/* Run info - needed logging turned on */
HYPRE_BoomerAMGGetNumIterations(solver, &num_iterations);
HYPRE_BoomerAMGGetFinalRelativeResidualNorm(solver, &
    final_res_norm);
```

Alles zusammenfügen Im Anhang ist das komplette Beispiel 5 zu sehen, welches zuvor in Ausschnitten erläutert wurde. Im folgenden soll der Zusammenhang der einzelnen Methoden näher erläutert werden. (CASC, 2017c)

näher erläutern, paragraphen oben drüber: methoden etwas genauer erklären

1.4.4 Ressourcen

- offiziell: `http://computation.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-methods`
- Übersicht Publikationen: `http://computation.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-methods/publications`
- Pursuing scalability for hypre's conceptual interfaces `http://dl.acm.org/citation.cfm?doid=1089014.1089018`
- (mehr) beispiele `https://redmine.scorec.rpi.edu/anonsvn/fastmath/docs/ATPESC_2013/Exercises/hypre/examples/README_files/c.html`

1.5 Webtechnologien und Werkzeuge für die verteilte Ausführung

1.5.1 RPC

RPC steht für Remote Procedure Call. In klassischen Programmen werden Funktionsaufrufe (oder Prozeduraufrufe) auf einem einzelnen Gerät mit einem einzigen Prozess durchgeführt. Ein Schritt zur parallelen Ausführung ist die Interprozesskommunikation, bei der verschiedene Prozesse eines Systems miteinander kommunizieren können. RPC geht noch einen Schritt weiter. Die kommunizierenden Prozesse befinden sich hier in verschiedenen Adressräumen, können sich also auf verschiedenen Geräten befinden. Der besondere Gedanke hinter RPC ist der, dass diese Aufrufe entfernter Programme Funktions- oder Prozedurbezogen erfolgen sollen. Anstatt also eine Funktion aufzurufen und lokal auszuführen, soll sie auf einem entfernten System ausgeführt werden. Die Implementierung mit RPC soll sich nicht weiter vom Programmablauf mit lokalen Funktionen unterscheiden. Dennoch sind einige Unterschiede notwenig, um die Verbindung aufzubauen und zu verwalten. Es unterscheidet sich hier von anderen Kommunikationsprotokollen wie REST, welches eine ressourcenbezogene Kommunikation verwendet.

was ist rest

1.5.2 Service/Webservice

SOAP

erklärung service kommunikation allgemein

1.5.3 Socket

erklärung socket allgemein

2 Implementierungen von Webtechnologien

2.1 RPC

rpc aufräumen

RPC steht für Remote Procedure Call. Sie bieten die Möglichkeit von Funktionsaufrufen in verteilten Systemen. Im Gegensatz zu Protokollen wie REST sieht RPC wie ein Methodenaufruf aus. Das Ziel ist die Ausführung einer bestimmten Prozedur, nicht das verwalten einer bestimmten Ressource. Protokolle sind zum Beispiel XML-RPC und Json-RPC.

- understanding rest and rpc for http `https://www.smashingmagazine.com/2016/09/understanding-rest-and-rpc-for-http-apis/`
- Implementing remote procedure calls `http://dl.acm.org/citation.cfm?id=357392`
- c++ json-rpc lib `https://github.com/cinemast/libjson-rpc-cpp/`

REST vs RPC

skizze rpc call (vs skizze rest call)

warum rest ungeeignet

XML-RPC XML-RPC ist ein Protokoll für Remote Procedure Calls. In diesem wird XML zur Formatierung der übertragenen Daten verwendet. Wie in RPC besteht auch in XML-RPC der Bezug auf Methoden. Darin besteht der Unterschied zu anderen Protokollen, welche sich auf die Manipulation von Daten beziehungsweise Ressourcen beziehen. Es gibt in verschiedenen Sprachen Implementierungen von Server-Client-Frameworks die das XML-RPC Protokoll zur Kommunikation verwenden.

xml-rpc XML-RPC Bibliothek für C und C++ `http://xmlrpc-c.sourceforge.net/`.

Installation:

Listing 8: XML-RPC Installation

```
./configure
make
make install
//And then, if Linux:
ldconfig
```

Diese Bibliothek bietet ein performantes Framework zur Implementierung von XML-RPC Anwendungen. Sie bietet sicher Kommunikation per HTTPS. Es können größere Datenmengen per Packet Stream übertragen werden. Die Verwendung der Frameworkmethoden erfolgt sehr transparent und modular, wodurch man leicht Einfluss auf die funktionsweise nehmen kann. Das führt allerdings auch dazu, dass im Vergleich zu modernen Frameworks recht viel boiler plate code erforderlich ist.

JSON-RPC JSON-RPC ist wie XML-RPC ein Protokoll für Remote Procedure Calls. Dieses verwendet für die Formatierung der Daten das Json-Format, was den wesentlichen Unterschied zum XML-RPC darstellt. Das Json-Format benötigt im Vergleich zu XML wesentlich weniger Zeichen für die Strukturierung der Daten. Durch die unterschiedliche Effizienz in der Repräsentation der Daten ergeben sich bei JSON-RPC bessere Übertragungsgeschwindigkeiten. Die Auswahl an JSON Bibliotheken welche in C und C++ implementiert sind ist groß. Im folgenden github-Projekt wurde eine große Auswahl miteinander verglichen: `https://github.com/miloyip/nativejson-benchmark`.

liste durchgehen

- information: `http://www.simple-is-better.org/json-rpc/`
- wiki: `https://en.wikipedia.org/wiki/JSON-RPC`
- json rpc server: `https://github.com/hmng/jsonrpc-c`
- tcp client: `http://jsonrpc-cpp.sourceforge.net/index.php?n=Main.HomePage`
- rpc client?: `https://groups.google.com/forum/#!topic/json-rpc/9O1dbQehU04`

Im folgenden Abschnitt wird eine Auswahl von JSON-RPC Bibliotheken vorgestellt:

json-rpc json-rpc-libs:

- `https://github.com/yeryomin/libjrpc`
- `https://github.com/jhlee4bb/jsonrpC`
- `https://github.com/hmng/jsonrpc-c`
- `https://github.com/pijyoi/jsonrpc`

yeryomin/libjrpc `https://github.com/yeryomin/libjrpc` install:

Listing 9: yeryomin/libjrpc Installation

```
git clone git@github.com:yeryomin/libjrpc.git
git clone git@github.com:yeryomin/libipsc.git
git clone git@github.com:yeryomin/libfmt.git
git clone git@github.com:zserge/jsmn.git
git clone git@github.com:yeryomin/liba.git

cd libipsc
make
sudo ln -s /path/to/libipsc.h /usr/include
cd ..

cd jsmn
make
sudo ln -s /path/to/jsmn.h /usr/include
```

```
cd ..

cd libfmt
make
sudo ln -s /path/to/libfmt.h /usr/include
cd ..

cd libjrpc
make
sudo ln -s /path/to/libjrpc.h /usr/include
```

so many unresolved dependencies..... wow

jhlee4bb/jsonrpC (needs libwebsockets installed: ‘yaourt libwebsockets‘ etc.)

Listing 10: jhlee4bb/jsonrpC Installation

```
git clone git@github.com:jhlee4bb/jsonrpC.git
cd jsonrpC
mkdir build
cd build
cmake ..
make
build output left in jsonrpc-x.y
```

völlig veraltet - websocket-lib ist inziwschen komplett inkompatibel

client and server stubs: https://github.com/masroorhasan/RPC

rpc for php and c: https://github.com/laruence/yar

2.2 Bibliotheken/Frameworks/Implementierungen

Mercury Homepage: `https://mercury-hpc.github.io/` Dokumentation: `http://mercury-hpc.github.io/documentation/` High-level RPC Layer: `http://mercury-hpc.github.io/documentation/#high-level-rpc-layer` Mailing-List: `https://lists.mcs.anl.gov/mailman/private/mercury/`

Für HPC optimierte c-Bibliothek. Bietet außerdem MPI-Unterstützung. (Etwas) Dokumentation vorhanden.

Trios/Nessie `https://software.sandia.gov/trac/nessie/wiki/WikiStart`

Portals `https://www.researchgate.net/publication/221201996_Efficient_Data-Movement_for_Lightweight_IO`

DART `http://coewww.rutgers.edu/www4/cacweb/TASSL/Papers/dart_hpdc.pdf`

2.3 andere

Python in C Ausführen

ressourcen https://docs.python.org/2.5/ext/callingPython.html https://www.codeproject.com/articles/11805/embedding-python-in-c-c-part-i
better
http://www.linuxjournal.com/article/8497 https://www.codeproject.com/articles/820116/embedding-python-program-in-a-c-cplusplus-code

basics some text

Listing 11: Python in C ausführen

```
#include <python3.6m/Python.h>

int main()
{
  Py_Initialize();
  PyRun_SimpleString(
    "print('Hello World from Embedded Python.')"
  );
  Py_Finalize();
}
```

kompilieren von c-code mit python3.6

Listing 12: Python in C kompilieren

```
cc prog.c -o prog.o -I/usr/include/python3.6m -lpython3.6
   m
  -lm -L/usr/lib/python3.6
```

spring-server framework https://github.com/bartobri/spring-server

Listing 13: Installation bartobri/spring-serve

```
git clone git@github.com:bartobri/spring-server.git
```

2.4 Serialisierung

https://www.quora.com/Is-there-a-C-struct-to-JSON-generator-library

Protocol Buffers

protobuf überarbeiten und nach oben zu grpc verweisen

protobuf: https://github.com/protobuf-c/protobuf-c-rpc
https://developers.google.com/protocol-buffers/ Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data. Implementierung für C: https://github.com/protobuf-c/protobuf-c https://github.com/protobuf-c/protobuf-c-rpc

Fazit Generiert den Code für structs selbstständig. Für den Anwendungsfall bestehen die Strukturen jedoch bereits und müssen umgewandelt werden. Ist daher ungeeignet.

2.5 Beispielinstallation OpenStack: Devstack

OpenStack Projekte https://www.openstack.org/software/project-navi

Nova, Horizon ...

Nova

Glance

Cinder

Neutron

Horizon

Heat

DevStack DevStack ist ein Projekt von OpenStack, welches eine Grundinstallation einer OpenStack-Umgebung bereitstellen soll. Es soll helfen eine Entwicklungsumgebung für OpenStack mit wenigen Schritten einzurichten. DevStack ist nicht für den Produktiveinsatz vorgesehen.

In der Grundstruktur verschiedener Cloudsysteme gibt es starke Überschneidungen. Es gibt eine Menge von Hardwareressourcen die zur Verfügung stehen. Dazu zählen Beispielsweise CPUs, RAM oder Speicher. Es gibt weiterhin Instanzen von virtuellen Containern. Diese Container beinhalten alles notwendige, um Software ausführen zu können. Sie sind sehr leichtgewichtig und schlank. Das liegt daran, dass sie kein komplettes eigenes Betriebssystem mitbringen. Dies stellt den Hauptunterschied zu herkömmlichen Virtuellen Maschinen dar.

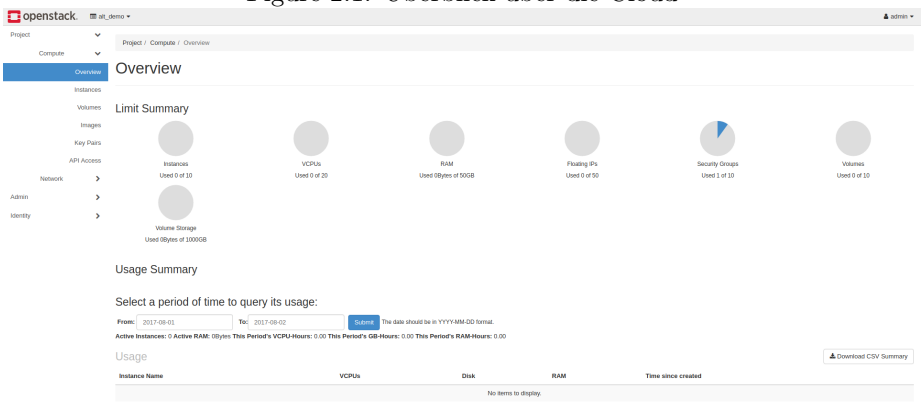
evtl docker container ausführlicher

http://docs.openstack.org/developer/devstack/
https://www.youtube.com/watch?v=jpk4i66-IU4
http://ronaldbradford.com/blog/setting-up-ubuntu-on-virtualbox-for-devstack-2016-03-30/
http://ronaldbradford.com/blog/setting-up-ubuntu-using-vagrant-2016-04-01/
http://ronaldbradford.com/blog/downloading-and-installing-devstack-2016-04-02/

Automatisches Setup: https://github.com/lorin/devstack-vm

Grafische Oberfläche Die Oberfläche besteht aus drei großen Bereichen: Project, Admin und Identity. Diese können am linken Rand ausgewählt werden. Wichtig ist der Project-Bereich, in dem die Projekteigenschaften abgerufen und geändert werden können. Der Bereich des Projekts ist noch einmal unterteilt in Compute und Network. Die Unterpunkte beider Bereiche werden im folgenden Abschnitt kurz erläutert.

Figure 2.1: Überblick über die Cloud



Overview Eine gute Übersicht über den aktuellen Zustand des Projekts bietet hier die Seite 'Overview'. In Abbildung 2.1 ist ein Screenshot zu sehen. Diese zeigt einen Überblick über Instances, VCPUs, RAM, Floating IPs, Security Groups, Volumes und Volume Storage.

Hier ist eine kurze Beschreibung der Punkte:

- Instances - die aktuell geladenen Instanzen, diese werden unten genauer beschrieben
- VCPUs - die virtuellen CPUs, Instanzen werden für die Ausführung auf diese aufgeteilt
- RAM - der RAM wird ebenso wie die CPUs unter den Instanzen aufgeteilt
- Floating IPs - diese können Instanzen zugewiesen werden, damit diese von außerhalb der Cloud erreichbar sind
- Security Groups - Die security groups bieten die Möglichkeit ip-Filter für ein Projekt anzulegen. Jedes Projekt erhält nach dem Erstellen standardmäßig einen Filter der alle IPs blockiert. "Security groups are sets of IP filter rules that are applied to an instance's networking." `https://www.mirantis.com/openstack-portal/express-openstack-portal/manage-openstack-security-groups-via-horizon/`
- Volumes - Diese verwalten den verfügbaren Festplattenspeicher. Es können Volumen mit einer bestimmten Speicherkapazität erstellt werden und Instanzen zugewiesen werden. Jede Instanz benötigt mindestens ein Volume.
- Volume Storage - dieses Diagram bietet einen Überblick über den Speicherplatz den die Volumes belegen.

Instances Hier können Instanzen überblickt und verwaltet werden. In Abbildung 2.3 ist die Seite ohne Instanzen zu sehen. Beim Erstellen einer neuen Instanz ist ein Wizard behilflich, siehe Abbildung 2.4. In diesem können fol-

gende Informationen angelegt werden: Details, Source, Falvor, Networks, Network Ports, Security Groups, Key Pair, Configuration, Server Groups, Scheduler Hints und Metadata. In Abbildung 2.2 sieht man eine Instanz die gerade geladen wird.

Figure 2.2: Erste Instanz lädt nach dem Anlegen

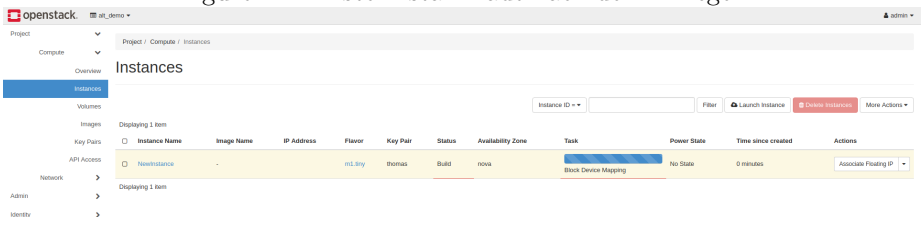


Figure 2.3: Leere Übersicht über Instanzen

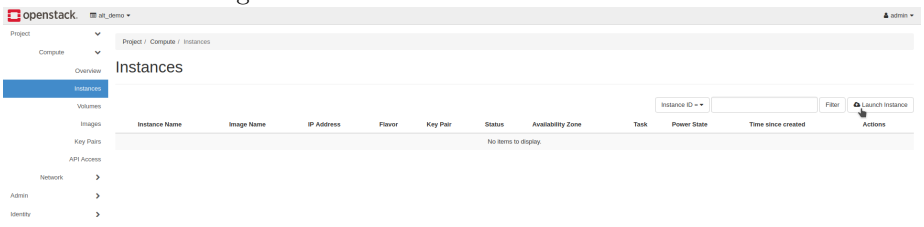
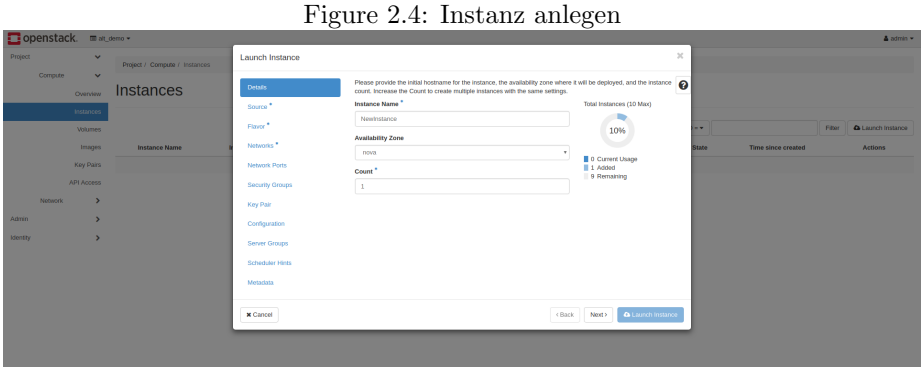


Figure 2.4: Instanz anlegen



Volumes Von hier können die Volumes gesteuert werden. Es gibt einen Überblick über alle bestehenden Instanzen. In der Abbildung 2.5 existiert ein Volume. Es hat den Namen 'New Volume' und ist 4GB groß.

Images Im Tab Images ist die Übersicht über die Images gegeben. Es existiert ein Image für CirrOS. CirrOS ist eine minimale Linux Distribution. `https://docs.openstack.org/image-guide/obtain-images.html` Es kann aber auch ein beliebiges anderes virtual machine image benutzt werden.

Figure 2.5: Volumes des Projekts

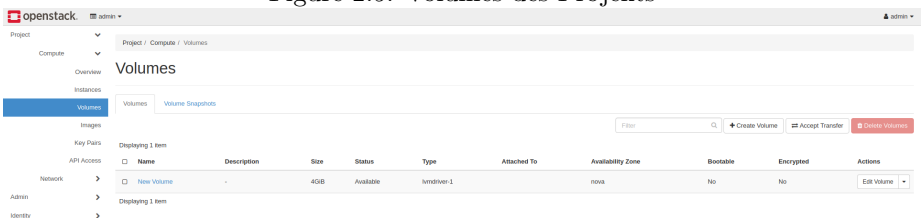
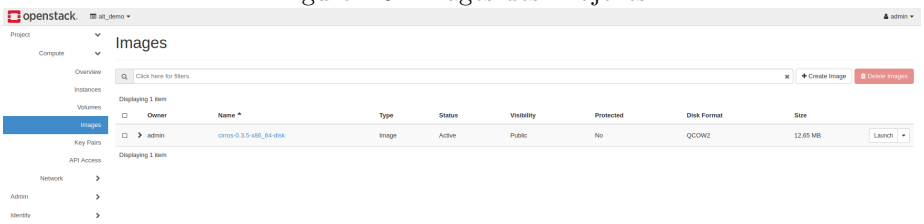
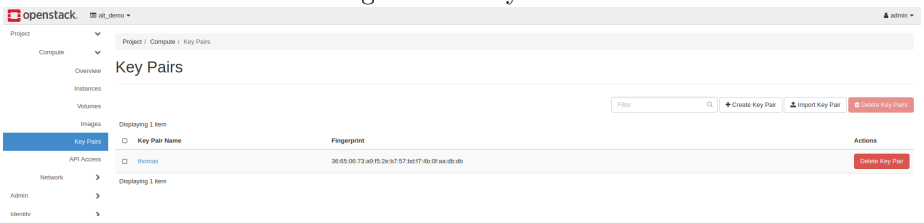


Figure 2.6: Images des Projekts



Key Pairs Es können ssh-Schlüsselpaare benutzt werden, um den Zugriff auf Images auf sichere Art zu ermöglichen. In diesem Menü kann neues Schlüsselpaar erzeugt werden. Wird das Paar beim Erstellen einer neuen Instanz ausgewählt wird der public key in diesem hinterlegt. Der erzeugte private key kann vom anderen Ende der ssh-Verbindung genutzt werden.

Figure 2.7: Key Pairs



API Access In der Abbildung 2.8 sind einige Elemente des Menüs API Access zu sehen. Die Liste zeigt das Mapping von Services auf Endpunkte. Alle Endpunkte sind in diesem Fall auf die ip 172.28.128.3 gemapped. Dies ist die ip, über die auch die Grafische Oberfläche in diesem Fall erreichbar ist.

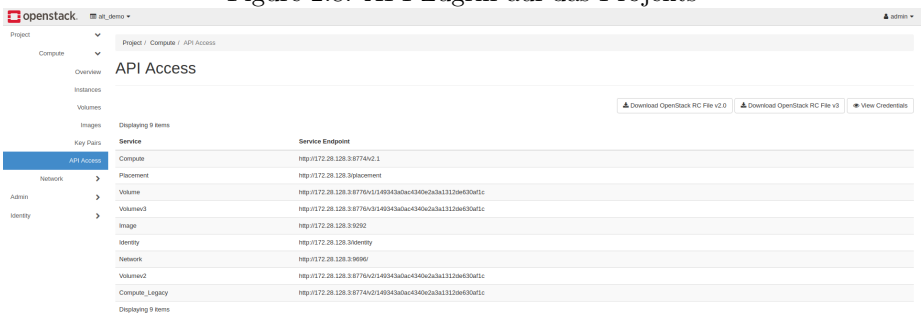
Bereich Network Im Bereich Network befinden sich die Unterpunkte Network Topology, Networks, Routers, Security Groups und Floating IPs.

Vorbereitung für die Installation

Ubuntu 16.04 in Vagrant aufsetzen

1. `vagrant init bento/ubuntu-16.04`

Figure 2.8: API Zugriff auf das Projekts



2. Vagrantfile anpassen. Es sollten mindestens 4GB RAM zugewiesen werden, damit OpenStack performant läuft.

Listing 14: Installation Ubuntu 16.04 in Vagrant

```
Vagrant.configure(2) do |config|
  config.vm.box = "bento/ubuntu-16.04"
  config.vm.network "private_network", type: "dhcp"

  config.vm.provider "virtualbox" do |v|
    v.memory = 4096
  end
end
```

3. `vagrant up`
4. `vagrant ssh`

Installation

Devstack installieren

1. `git clone https://git.openstack.org/openstack-dev/devstack`
2. `cd devstack`
3. Bei Bedarf zu einem (älteren) stable release wechseln:

   `git checkout stable/ocata`

   oder

   `git checkout stable/mitaka`

   Die master-branch kann auch genutzt werden.
4. `ifconfig enp0s8 | grep addr`
5. `inet addr kopieren`

6. `cp samples/local.conf .`
7. `HOST_IP="xxx.xxx.xxx.xxx"`
8. `echo "HOST_IP=${HOST_IP}" >> local.conf`
9. Es muss ein user mit sudo-rechten (ohne passwort) existieren, der nicht root ist. Der Standarduser vagrant ist dafür geeignet.
10. `./stack.sh`

Am Ende der erfolgreichen Installation erscheint die folgende Ausgabe in etwa so. Von hier können die nötigen Informationen für das weitere Vorgehen entnommen werden.

Listing 15: Ausgabe von DevStack

```
=========================
DevStack Component Timing
=========================
Total runtime          3644

run_process              87
test_with_retry           7
apt-get-update           12
pip_install             881
restart_apache_server    20
wait_for_service         61
git_timed               355
apt-get                 457
=========================



This is your host IP address: 172.28.128.3
This is your host IPv6 address: ::1
Horizon is now available at http://172.28.128.3/dashboard
Keystone is serving at http://172.28.128.3/identity/
The default users are: admin and demo
The password: nomoresecret
```

Starten

Openstack (Horizon) starten Die folgenden Informationen sind abhängig von der Konfiguration. Für dieses Beispiel sind folgende Daten notwendig.

1. `http://172.28.128.3/dashboard` in einem Browser öffnen
2. Benutzername admin oder demo
3. Password nomoresecret

Reboot `https://ask.openstack.org/en/question/5423/rebooting-with-devstack/` Wenn die Vagrantbox herunter gefahren wurde muss devstack nach einem erneuten Start ebenfalls neu gestartet werden. Ansonsten können die verschiedenen Services nicht genutzt werden. Das Webfrontend Horizon kann in diesem Fall keinerlei Ressourcen laden. Zum erneuten Erstellen der openstack Umgebung können die folgenden Befehle genutzt werden.

```
./unstack.sh
./stack.sh
```

Bei einem erneuten Ausführen von stack.sh würden jedoch alle Datenbanken neu erstellt werden. Das würde zu einem kompletten Datenverlust führen. Dies ist aber nicht in jedem Fall gewünscht. Dafür steht eine screen-Konfiguration bereit mit der die notwendigen Services gestartet werden können.
`http://stackoverflow.com/questions/36268822/no-rejoin-stack-sh-script-in-my-setup`

```
screen -c stack-screenrc
```

So kann eine bestehende openstack-Konfiguration auch nach einem Neustart weiter genutzt werden.

SSH-Verbindung zur Cloud http://openstack-cloud-mylearning.blogspot.de/2015/02/openstack-how-to-access-vm-using-ssh.html

3 Gegenüberstellung verfügbarer Werkzeuge und Varianten

gesamt etwa 15-20 seiten

vergleiche kapitel ausarbeiten

beschreibung als prosatext zu vergleichen in tabellen

evtl hybrid ergänzen

3.1 'private' und 'public' Cloud

Es kann zwischen private und public Clouds unterschieden werden. Im folgenden Abschnitt werden beide Arten einander gegenübergestellt.

Table 3.1: Private vs. Public Cloud

	private	public
Drittanbieter beteiligt	☐	☑
Daten bei Drittanbieter	☐	☑
Privatsphäre	kein Zugriff für Dritte	abhängig vom Cloudanbieter
Sicherheit	in eigener Verantwortung	Verantwortung liegt beim Cloudanbieter
Infrastruktur selbst verwaltet	☑	☐
Plattform selbst verwaltet	☑	☑
Skalierbarkeit (der Infrastruktur)	hängt vom genutzten Rechenzentrum ab	große Cloudanbieter können nahezu beliebig skalieren
Skalierbarkeit (nach unten)	Maximalanforderung muss immer vorhanden sein (echte Hardware)	bei geringer/keiner Auslastung herunter fahrbar - Kosten Einsparung
spezielle Hardware	spezielle Hardware kann ausgewählt werden (zB HPC, Networking)	je nach Verfügbarkeit beim Anbieter

(Stiel, 2011) (onlinetech, 2017)

Der wesentliche Unterschied zwischen private und public Cloud liegt in der Beteiligung eines Drittanbieters. Dieser ist lediglich bei der public Cloud vorhanden. Er stellt das Cloudsystem für seine Nutzer zum Zugriff auf dessen Infrastruktur zur Verfügung. Die Daten liegen also auf dem System des Drittanbieters. Selbst ohne dauerhaft genutzte Festplatte müssen zumindestens Daten für den Zeitraum der Verarbeitung an das Cloudsystem übertragen werden. Das

kann bei sensiblen Daten relevant sein. Dies betrifft auch die Punkte Sicherheit und Privatsphäre. Man muss dann dem Anbieter der public Cloud Vertrauen schenken und hat keinen direkten Einfluss. In einer private Cloud ist dagegen kein Drittanbieter involviert. Diese wird auf der eigenen Infrastruktur installiert. Man ist also selbst für Daten und deren Sicherheit sowie Privatsphäre verantwortlich.

Das führt auch zum nächsten Unterschied. Die Infrastruktur muss in der private Cloud selbst verwaltet werden. In der public Cloud dagegen nicht.

Verwaltung von Plattform sowieso Software hängen vom Cloudtyp ab. In der Regel ist bei einem Cloudsystem von einem IaaS die Rede. In dem Fall spielt es keine Rolle ob sie private oder public ist. Bei beiden müssen Plattform und Software selbst verwaltet werden.

Die eigene Verwaltung der Infrastuktur hat daraus folgende Effekte auf die Eigenschaften der Cloud. Die Skalierbarkeit einer privaten Cloud wird in der Regel eher an Grenzen stoßen. Die Ressourcen für sind im Gegensatz zur public Cloud nicht flexibel. Sollten diese nicht mehr den Anforderungen genügen muss das Rechenzentrum erweitert werden, sobald dessen Grenzen erreicht sind. In der public Cloud können jederzeit neue Ressourcen hinzugefügt werden. Dies kann automatisiert und mit geringem Zeitaufwand erfolgen. Theoretisch wird der Nutzer hier nie an die Grenzen des Rechenzentrums stoßen.

Besonders problematisch sind Anwendungen mit stark schwankender Auslastung in privaten Clouds. In Phasen mit geringer Last wird Hardware unter Umständen nicht effizient ausgelastet. Während solcher Zeiträume können in public Clouds Ressourcen deaktiviert werden. Dies kann eine erhebliche Kosteneinsparung erzeugen.

3.2 Technologie

tabelle/vergleiche zwischen technologien: rpc,socket,service etc

Es gibt verschiedene Technologien und Werkzeuge um eine Kommunikation zwischen zwei Endpunkten zu ermöglichen. Es sollen in diesem Abschnitt drei davon betrachtet werden: RPC, Sockets und Services. Dabei werden jedoch Methoden verglichen, die durchaus auch nebeneinander verwendet werden können. So könnte beispielsweise die Verbindung einer RPC-Kommunikation über Sockets stattfinden. Der Vergleich bezieht sich daher eher auf die reine Verwendung der jeweiligen Technologie, unabhängig von möglichen Frameworks darüber oder Techniken im Hintergrund. Für Sockets in der Sprache C könnte somit beispielsweise die Bibliothek sys/socket zum Einsatz kommen:

```
#include <sys/socket.h>
```

Für RPC beispielsweise die Bibliothek xmlrpc

```
#include
```

Services stehen repräsentativ für eine serviceorientierte Architektur (SOA). Diese könnte zum Beispiel REST verwenden.

- welche Werkzeuge für die kommunikation zwischen client und cloud
- rpc, socket, service ...
- abwägen zwischen performance, aufwand ...

- welche framework könnten genutzt werden

Table 3.2: Vergleich von Technologien

	RPC	Socket	Service
Server zu Client	1:n	1:1	1:n
Bezug	Methoden	-	Daten (deren Manipulation)
performance	...	...	...
aufwand	...	...	...
frameworks	...	...	...

Am Ende wird ein Framework die Beste Lösung sein.

3.3 Service Modelle

Übersicht über die verfügbaren Service Modelle von Clouds:

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

Eine Cloud kann Nutzern Zugang zu Softwareanwendungen geben. Diese befinden sich im Gegensatz zu traditionellen Anwendungen nicht auf dem lokalen Gerät des Nutzers, sondern laufen auf einem entfernten System, dem Cloudsystem. In diesem Fall spricht man von Software as a Service, SaaS.

Weiterhin kann eine Cloud Zugang zu einer weitestgehend fertigen Entwicklungsbeziehungsweise Laufzeitumgebung bieten. Hier können Nutzer ohnes eigenhändiges Installieren und Pflegen von Betriebssystemen, Bibliotheken, Datenbanken oder Ähnlichem ihre Software ausrollen. In diesem Fall spricht man von Platform as a Service, PaaS.

Schließlich kann eine Cloud auch lediglich Zugang zu Rechen- und Speicherkapazität geben. Hier können Nutzer eigene Systeme installieren. Das Planen, Installieren und Verwalten eines eigenen Rechenzentrums und dessen Hardware ist nicht nötig. In diesem Fall spricht man von Infrastructure as a Service, IaaS.

wird evtl schon teilweise in abschnitt 1 (allgemeines zur cloud) abgedeckt

Figure 3.1: Aufgabenverteilung der Service Modelle

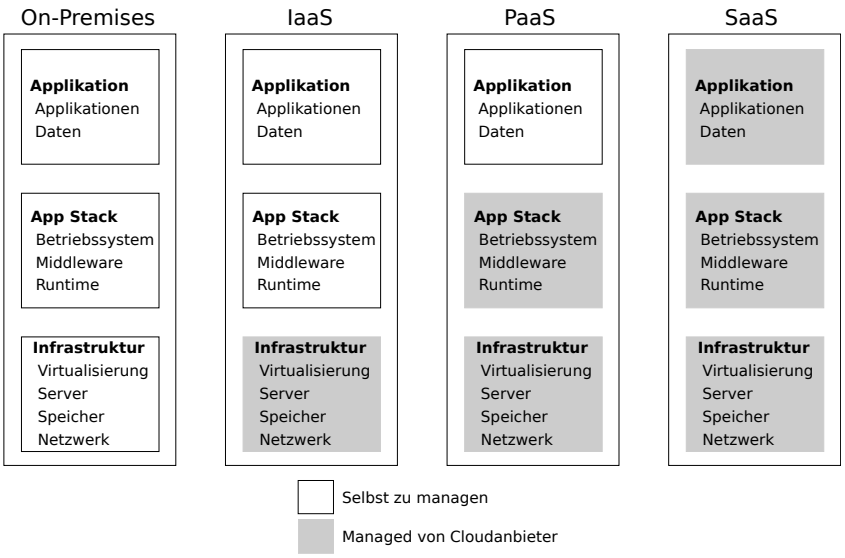


Table 3.3: Vergleich von Modellen

	Datacenter	IaaS	PaaS	SaaS
Infrastruktur	☑	☐	☐	☐
Application Stack	☑	☑	☐	☐
Application	☑	☑	☑	☐
User	☑	☑	☑	☑

"The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications and possibly limited control of select networking components (e.g., host firewalls)." (NIST, 2012)

(Kavis, 2014)

3.4 Sprache

- hardware-nah: C/C++ (bessere performance)
- vs netzwerknah: (bessere möglichkeiten die kommunikation zu Implemen-

tieren)

- kann eine hybridform eingsetzt werden? zB (micro-)service: bib in c kommuniziert mit client-backend in php, dieses führt die calls zum server aus

Table 3.4: Vergleich von Technologien

	C/C++	Python	Hybrid
erste Version	1972	1991	
Level	Hochsprache	Hochsprache	...
Interpreter/Compiler	Skriptsprache	kompliliert	...
	Performance	vorgefertigter Code, wenig Boilerplate	...
Typisierung	streng	schwach/dynamisch	
	Sicherheit		

3.5 Vergleich ausgewählter Cloudsysteme in einer Tabelle

Im folgenden Abschnitt werden einige bestehende Cloudsysteme miteinander verglichen. Dazu werden relevante Kriterien heran gezogen. Im Vergleich werden sowohl Clouds von kommerziellen als auch gemeinnützigen Anbietern berücksichtigt.

from see also section in `https://en.wikipedia.org/wiki/Amazon_Elastic_Compute_Cloud` validate list

- AppScale
- Bitnami
- CopperEgg
- ElasticHosts
- Eucalyptus (computing)
- FlexiScale
- FUJITSU Cloud IaaS Trusted Public S5
- GoGrid
- Google App Engine
- Google Compute Engine
- GreenQloud
- Internap
- Linode
- Liquid Web

- Lunacloud
- Microsoft Azure
- Nimbula
- OpenShift
- OrionVM
- OVH
- Rackspace Cloud
- RightScale
- Savvis
- TurnKey Linux Virtual Appliance Library
- Zadara Storage

Meta `cloud.com` and cloudstack, azure und openstack beteiligt.[2] `https://aws.amazon.com/ec2/details/`

fix this footnote

[2] test

Table 3.5: Meta zu Cloudanbietern

	Amazon EC2	Apache CloudStack	Microsoft Azure	OpenStack
Anbieter	Amazon	Apache Software Foundation	Microsoft	OpenStack Foundation [3]
Initialer Release	2006	2010	2010	2010
Lizenz	proprietär	Apache License 2	Plattform ist Closed source, Client SDKs Open Source	Apache License 2.0
Entwickelt in	?	Java	?	Python
Homepage	`https://aws.amazon.com/ec2/`	`https://cloudstack.apache.org`	`https://azure.microsoft.com`	`https://openstack.org`
Pricing	verschiedene Modelle, zB. on-demand oder reserviert	kostenfrei, weil selbst gehosted und deployed	pay for use	kostenfrei wenn selbst gehosted und deployed, sonst abhängig von einem der zahlreichen Partner
Bekannte Nutzer	Netflix, Expedia, airbnb	Dell, Huawai, SAP	Daimler, BMW, Lufthansa	NASA, Intel, PayPal

Die vier Cloudsysteme werden von vier verschiedenen Anbietern entwickelt. Bei EC2 und Azure handelt es sich um die Unternehmen Amazon und Microsoft. Apache CloudStack und OpenStack werden von den Organisationen Apache Software Foundation beziehungsweise der OpenStack Foundation entwickelt.

Die Amazon Cloud stellt einen der ersten Vertreter von echten Cloudsystemen dar. Mit 'echten' Cloudsystemen sind die modernen Clouds gemeint, wie wir sie heute kennen. Schon vorher gab es große Datencenter, welche beispielsweise über VPN erreichbar waren. Sie sind allerdings nur als indirekter Vorläufer heutiger Cloudsysteme zu sehen. Diese wurde 2006 in einer ersten Beta für die Öffentlichkeit zugänglich. Erste Versionen von Apache CloudStack, Microsoft Azure und OpenStack wurden allesamt im Jahr 2010 veröffentlicht.

Die Cloudsysteme von Amazon und Microsoft stehen unter einer proprietären Lizenz. Dagegen sind CloudStack und OpenStack Open Source Clouds.

[3] initiiert von Rackspace Hosting und NASA

Für die verschiedenen Clouds stehen ganz unterschiedliche Preismodelle zur Verfügung. Sollte man die beiden Open Source Clouds als private Cloud nutzen wollen, fallen keine extra Kosten an. Dafür ist in diesem Fall ein erhöhter Integrations- und Pflegeaufwand nötig.

Amazon selbst bietet verschiedene Modelle. Eine günstige Variante für den Einstieg kann On-Demand sein. Man zahlt im Nachhinein für Ressourcen nur genau in dem Maße wie man sie genutzt hat. Das ist vor allem bei stark variierender Last vorteilhaft. Ein weiteres Modell sind die Reserved Instances. Diese werden fest reserviert. Sollte man eine anhalten hohe Auslastung haben kann dieses Modell besser geeignet sein. Die zwei weiteren Modelle lauten Spot-Instances (flexible Reservekapazität) und Dedicated Hosts (physikalischer Server). Die Nutzungsdauer wird sekundengenau abgerechnet.

Die Kosten für Microsofts Azure hängen von verschiedenen Faktoren ab. Zum einen bezahlt man für jedes Produkt den man Nutzt, das kann zum Beispiel eine Compute- oder Storageeinheit sein. Der Preis ist zusätzlich von der Qualität des Produkts abhängig, ein größerer Speicher kostet also mehr. Faktor drei ist die Nutzungsdauer des Produkts. Das Modell ähnelt damit dem On-Demand Modell von Amazon.

Alle vier Cloudsysteme haben sehr namhafte Vertreter vorzuweisen. Die Amazon EC2 wird beispielsweise von Netflix, Expedia und airbnb verwendet. Daimler, BMW und Lufthansa setzen dagegen auf Microsofts Azure. Die Open Source Cloud CloudStack wird von Dell, Huawai und ebenfalls Daimler eingesetzt. Die NASA, Intel und PayPal zählen zu den Nutzern von OpenStack.

Table 3.6: Vergleich von Cloudanbietern

	Amazon EC2	Apache CS	Azure	Openstack
private	☐	☑	☐	☑
public	☑	(☑)	☑	(☑)
hybrid	☐	☑	☐	☑
Service Modell	Iaas	(Iaas)	Iaas	(Iaas)
elastic	☑	(☑)	☑	(☑)
self deployed	☐	☑	☐	☑
hosted	☑	☑	☑	☑
open source	☐	☑	☐	☑

Apache CS und OpenStack werden als private Clouds entwickelt. Es gibt jedoch Unternehmen, die diese auf Ihrer eigenen Infrastruktur auch als public Clouds anbieten. Sie stellen in diesem Fall also einen Hybridtyp zwischen private und public Cloud dar.

Im Falle von OpenStack existiert ein Marktplatz auf dem nach zertifizierten Abietern von public OpenStack Clouds gesucht werden kann (siehe `https://www`

.openstack.org/marketplace/public-clouds/). Dort ist Beispielsweise die Deutsche Telekom vertreten (siehe https://www.openstack.org/marketplace/public-clouds/deutsche-telekom/open-telekom-cloud).

Dadurch können auch alle dieser Cloudsysteme als IaaS verwendet werden. Apache CS und Openstack bieten jedoch keine IaaS wenn man sie als private Cloud verwendet. In diesem Fall muss eine eigene Infrastuktur verwendet werden. Um diese trotzdem als IaaS-Cloud zu nutzen kann man jedoch eine Hosted-Lösung von einem Drittanbieter nutzen.

Der Gedanke einer elastischen Cloud ist vor allem bei den beiden public Clouds wirksam. Deren Anbieter besitzen große Rechenzentren, die die Anforderungen eines einzelnen Projekts weit überschreiten. Diese werden nur durch eine hohe Anzahl von Nutzern ausgelastet. Bei einer wechselhaften Auslastung können Ressourcen zwischen verschiedenen Nutzern clever aufgeteilt werden. Dadurch reduziert sich deren Leerlauf.

Da private Clouds in der Regel nur einen Nutzer haben kann die Elastizität weniger gut genutzt werden. Besonders wenn lediglich ein einzelnes oder wenige Projekte auf dem privaten Cloudsystem laufen. Sollten diese gerade eine geringe Last verursachen ist ein Leerlauf des Cloudsystems nicht zu vermeiden. Bei großen Unternehmen mit sehr vielen Anwendungen und Zugriffen kann allerdings ein ähnlicher Effekt wie bei public Clouds entstehen.

Nur die privaten Clouds können selbst deployed werden. Dagegen können alle vier Cloudsysteme als hosted Lösung genutzt werden.

Apache CS und Openstack stehen unter einer Open Source Lizenz. Amazon EC2 und Azure sind dagegen proprietäre Software. Sollte man selbst Änderungen an seinem Cloudsystem vornehmen wollen kommen also nur die Ersteren in Frage.

4 Auswertung der Vergleiche und Auswahl der Werkzeuge

Vergleiche verschiedener Technologien und Werkzeuge. Auswahl für den Einsatz bei der Implementierung.

gesamt etwa 5-10 seiten

4.1 gRPC

`http://www.grpc.io/`
Googles open source RPC framework. Dokumentation: `http://www.grpc.io/docs/`.

Installation Installiere vagrant ubuntu 16.04. Setup von gRPC nach der Installation:

Listing 16: Installiere vagrant ubuntu 16.04

```
cd /vagrant
sudo apt-get install build-essential autoconf libtool
sudo apt-get install libgflags-dev libgtest-dev
sudo apt-get install clang libc++-dev
git clone -b $(curl -L http://grpc.io/release) \\
https://github.com/grpc/grpc
cd grpc
git submodule update --init
make && sudo make install
cd third_party/protobuf
sudo apt-get install unzip
./autogen.sh
./configure
make && sudo make install
cd -
cd examples/cpp/helloworld/
sudo apt-get install pkg-config
make
```

kurze Einführung

grpc genauer beschreiben (auch protobuf)

Benutzt Protobuf zum erstellen von Services und Nachrichtenobjekten (stubs). Offizieller support für C++ vorhanden. Implementierung für c vorhanden `https://github.com/protobuf-c/protobuf-c`. Das Basisframework von gRPC ist ebenfalls in nativem C geschrieben. Die Highlevel-Framework-API ist jedoch nicht in C verfügbar.

Da die Definitionen der stubs nicht sprachgebunden erfolgen ist eine (spätere) Umstellung auf eine andere Sprache einfacher. Es müssen die stubs für die neue Sprache lediglich aus den gleichen Definitionen neu generiert werden.

Listing 17: Beispiel grpc

```
package helloworld;
```

```
// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

Überblick Funktionsumfang Ein definierter Service steht nach dem generieren der Implementierung in der gewünschten Sprache als Interface bereit. Dieses kann alternativ auch als AsyncService implementiert werden.

Listing 18: grpc Service Definition

```
//synchron
class X : public Greeter::Service {}
//oder asynchron
class X : public Greeter::AsyncService {}
```

Listen von Request- oder Responseobjekten können in Form eines Streams übertragen werden.

Listing 19: grpc Funktion Definition

```
rpc ListFeatures(Rectangle) returns (stream Feature) {}
```

4.2 Ressourcen

- HPC in der Cloud `http://grids.ucs.indiana.edu/ptliupages/publications/cloud_handbook_final-with-diagrams.pdf`

4.3 Roadmap Implementierung

roadmap checken/erweitern/abhaken

Was muss zwischen Client und Server übertragen werden: Anzahl der calls zwischen Client und Server sollte so gering wie möglich gehalten werden. Der Aufbau der Datenstrukturen sollte daher komplett innerhalb des Clients passieren. Sobald alle Eingabedaten erstellt sind muss ein Solver aufgerufen werden. Die Solver sollten auf dem Server ausgeführt werden.

(später:) Eventuell kann sogar der Aufwand der Berechnung abgeschätzt werden. Somit könnten einfache Berechnungen innerhalb des Client ausgeführt werden. Lediglich komplexe Berechnungen an den Server übermittelt werden.

Fazit 1. Möglichkeit: Es müssen (alle) HYPRE-Datentypen übertragen werden können. Dazu muss ein Mapper existieren. Dieser sollte die Datenstrukturen in ein Datenformat umwandeln welches übertragen werden kann (json). Er muss außerdem die übertragbaren Daten wieder in die Datenstrukturen zurück umwandeln können.

2. Möglichkeit: Alle Eingabeparameter und Funktionsaufrufe werden gesammelt. Auf Seite des Servers werden diese nacheinander ausgeführt um die Datenstrukturen zu bilden. Danach wird der Solver ausgeführt. Am Ende die lediglich die Antwort umgewandelt und übertragen.

Client

Anforderungen Bibliothek. Alle HYPRE-Calls implementieren die einen solver ausführen. Statt der direkten Ausführung müssen die Input-Daten an den Server übertragen werden. Außerdem muss der Call-Name übertragen werden. Nach Ausführung auf dem Server muss die Antwort wiederum in die entsprechenden Datentypen umgewandelt werden.

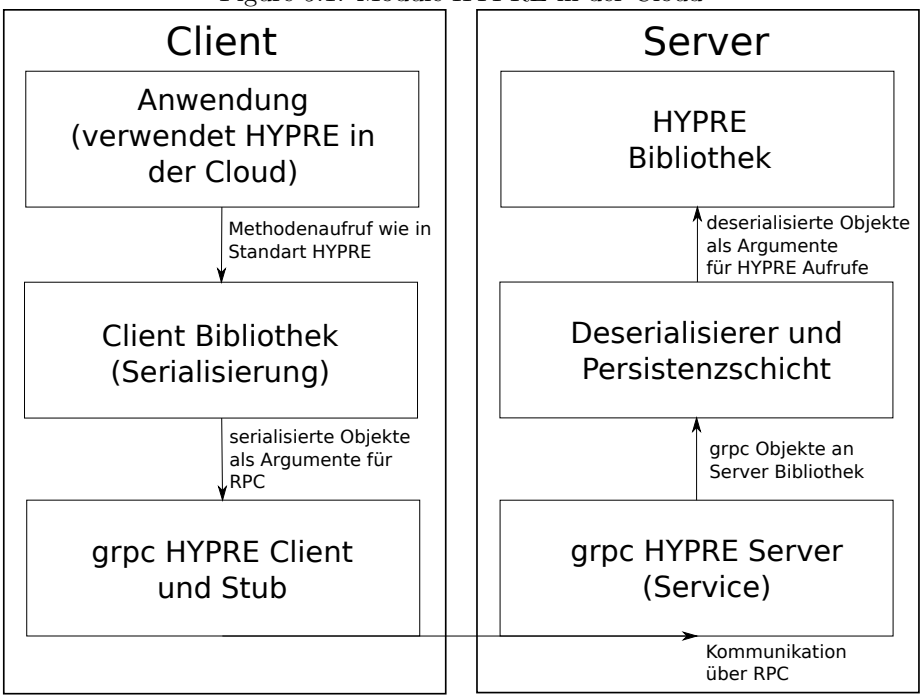
Kernmodule Einige HYPRE-Methoden überschreiben. Mapping HYPRE-Datentypen zu JSON. Mapping JSON zu HYPRE-Datentypen. Webservices implementieren für Komunikation mit dem Server.

5 Verteilte Ausführung von HYPRE - Planung

5.1 HPC in der Cloud

5.2 Überblick

Figure 5.1: Module HYPRE in der Cloud



fix figure: convert into nice pdf

5.3 Planung der Bibliothek - Client

Umschreiben sodass es einer Planung entspricht und nicht den status quo beschreibt

gesamt etwa 20-25 seiten

5.3.1 Allgemein

Für die verteilte Ausführung von HYPRE muss zunächst die Kommunikation mit einem externen Server möglich sein. Um diese zu Implementieren kommt grpc zum Einsatz. Client und Server kommunizieren also über Remote Procedure Calls. Es existiert ein proto-File, welches die Definition des HYPRE-Servers enthält.

Dafür ist ein eigenes grpc Package entstanden, welches einen Service definiert. Dieser Service beinhaltet die verfügbaren HYPRE Methoden. Es sind also beispielsweise Methoden vorhanden zum Verwalten der Matrizen und Vectoren für die Eingabedaten. Diese ermöglichen das Erstellen, Setzen von Eigenschaften oder Löschen. Ebenso gibt es Methoden zum Bearbeiten von Solvern.

Es sind neben den Methoden auch Nachrichten, Messages definiert. Diese werden als Argumente für die Methoden verwendet. Sie verhalten sich als dy-

namischer Datentyp. Grpc erlaubt eine Reihe von Standardtypen, die zu Messages zusammengefügt werden können. Nach der Übersetzung in die jeweilige Sprache sind die Messages in der Regel Klassen.

Es gibt HYPRE-Methoden in denen Objekte als Argumente übergeben werden. Für die verteilte Ausführung werden diese Objekte jedoch nicht zwischen Client und Server übertragen. Es werden die Methoden zum Erstellen und Setzen der Eigenschaften vom Client aufgerufen. Der Client erhält dabei einen Identifikator unter dem der Server das Objekt speichert.

Die Definition des HYPRE-rpc Services im proto-File ermöglicht das generieren von C++ Code. Aus den Messages und aus dem Service hypreSrv werden Klassen erzeugt. Die darin enthaltenen Variablen werden zu Klassenvariablen. Die Eigentliche Logik zur Kommunikation wird vom grpc-Framework bereit gestellt. Es müssen also lediglich die Messages erstellt und RPC-Methoden aufgerufen werden.

5.3.2 Kompatibilitätsschicht HYPRE - rpcHYPRE

Die Verwendung der RPC Aufrufe unterscheidet sich von der Verwendung der HYPRE Methoden. Das betrifft zum einen die Benennung, aber besonders die Argumente der Methoden. Die verteilte Ausführung von HYPRE sollte idealer Weise analog zur lokalen Variante verwendbar sein. Um dies zu Realisieren ist eine weitere Schicht zwischen bestehenden HYPRE Programmen und der RPC Implementierung notwendig. Diese kann die offizielle HYPRE-Bibliothek ersetzen. Dafür müssen alle Aufrufe, Klassen und Definitionen aus der HYPRE-Bibliothek auch in der neuen Bibliothek vorhanden sein. Diese sind dann gegenenfalls anders implementiert, die Schnittstellen sollten sich jedoch nicht unterscheiden.

das ist nur geplant, bisher nicht implementiert

5.4 In der Cloud - Server

Das Ziel der verteilten Ausführung ist es, rechenintensive Teile auf ein System externes auszulagern. Kriterien für die Auslagerung sind Ausführungszeit, Rechenlast und Lösbarkeit.

Die Zeitersparnis der Berechnungen sollte mindestens gleich groß zu den Einbußen durch die Übertragung sein. Ist dies nicht der Fall würde die gesamte Ausführungszeit steigen.

Selbst wenn keine relevante Zeitersparnis erzielt werden kann ist es dennoch möglich zumindestens die Last auf dem lokalen Rechner zu reduzieren.

Ein weiteres Kriterium ist die Lösbarkeit. Die Berechnung von sehr großen Problemen könnte die Kapazitäten eines Rechners komplett übersteigen. Selbst bei unendlicher Zeit wäre eine Berechnung nicht möglich. In diesem Fall wäre eine externe Berechnung unvermeidbar.

Es muss festgestellt werden ob die externe Ausführung in einem Cloudsystem sinnvoll ist. Es sollte auch ein Vergleich zu einem HPC-System stattfinden. Dabei ist zum einen entscheidend ob oder wie sich Implementierung auf den jeweiligen Systemen unterscheidet. Zum anderen ist besonders interessant, ob ein Cloudsystem eine ausreichend schnelle Berechnung durchführen kann. Cloudsysteme sind zwar hervorragend skalierbar, jedoch nicht unbedingt auf High Performance Computing ausgelegt. Dabei gibt es jedoch Unterschied zwischen den verschiedenen verfügbaren Clouds. Die Verwendung

hpc begriff klären

von Linux-Images in der Cloud ermöglicht beispielsweise die Verwendung von herkömmlichen HPC-Werkzeugen wie MPI. Private Cloudsysteme wie OpenStack können zudem auf eigener Infrastruktur zum Einsatz kommen. Sie können also auch auf eigenen HPC Systemen eingesetzt werden.

`https://wiki.openstack.org/wiki/HPC` hpc openstack

5.4.1 Umsetzung

`https://cyclecomputing.com/running-mpi-applications-in-amazon-ec2/` mpi in der cloud

6 Verteilte Ausführung von HYPRE - Beispielhafte Implementierung

Implementierung beschreiben

6.1 Benchmarks

nur planung, evtl in weiterführende arbeiten

Um zu überprüfen ob oder ab wann die Ausführung auf einem externen Rechner Vorteile bringt sind Benchmarks erforderlich. Es müssen verschieden Komplexe Probleme gemessen werden. Ebenso müssen für die Ausführungen auf verschiedenen Systemen gemessen werden.

7 Zukünftige, weiterführende Arbeiten

eventuell werden Teile von hier in die aktuelle Arbeit verschoben

- Skalierbarkeit der Cloud (einsetzen)
- Benchmarking lokal vs. remote HPC vs. cloud vs. ...
- daraus ableiten: Vorteile der Auslagerung:
  - Performance
  - Speicher
  - Lösbarkeit (nur remote überhaupt lösbar)
- Beispiel in nutzbares Framework umwandeln
- Beispiel nicht nur lokal sonder auch in Cloudumgebung
- auf remote Seite (effektiv) parallelisieren

References

CASC. (2017a). *Homepage center for applied scientific computing.* online. Retrieved from https://computation.llnl.gov/casc (zuletzt besucht am 28.06.2017)

CASC. (2017b). *Hypre software releases.* online. Retrieved from https://computation.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-methods/software (zuletzt besucht am 28.06.2017)

CASC. (2017c). *Reference manual hypre.* online. Retrieved from https://computation.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-methods/software

CASC. (2017d). *User's manual hypre.* online. Retrieved from https://computation.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-methods/software

DeNero, J. (2017). *Distributed and parallel computing.* online. Retrieved from http://wla.berkeley.edu/~cs61a/fa11/lectures/communication.html (zuletzt besucht am 10.07.2017)

Forum, M. P. I. (2015). *Mpi: A message-passing interface standard.* online. Retrieved from http://mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf

Fritzsche, D. (2010). *Overlapping and nonoverlapping orderings for preconditioning.* Wuppertal: Universitätsbibliothek Wuppertal.

Henson, V. E., & Yang, U. M. (2000). Boomeramg: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, *41*, 155–177.

Kavis, M. (2014). *Architecting the cloud: Design decisions for cloud computing service models (saas, paas, and iaas).* online. Retrieved from http://proquest.tech.safaribooksonline.de/9781118826461

LLNL. (2017). *Homepage lawrence livermore national laboratory.* online. Retrieved from https://www.llnl.gov/ (zuletzt besucht am 28.06.2017)

MPI-Forum. (2017). *Mpi forum homepage.* online. Retrieved from http://mpi-forum.org (zuletzt besucht am 10.07.2017)

NIST. (2012). *Cloud computing synopsis and recommendations.* online. Retrieved from http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-146.pdf

onlinetech. (2017). *Public vs. private cloud computing.* online. Retrieved from http://www.onlinetech.com/resources/references/public-vs-private-cloud-computing (zuletzt besucht am 23.08.2017)

Rauber, T., & Rünger, G. (2013). *Parallel programming: For multicore and cluster systems* (2nd ed.). Berlin: Heidelberg: Springer.

Stiel, H. (2011). Das unterscheidet private und public cloud. *Computerwoche*, *09*.

A
HYPRE

A.1 HYPRE Example 5

```
/*
   Example 5

   Interface:    Linear-Algebraic (IJ)

   Compile with: make ex5

   Sample run:   mpirun -np 4 ex5

   Description:  This example solves the 2-D Laplacian
      problem with zero boundary
                 conditions on an n x n grid.  The number
                     of unknowns is N=n^2.
                 The standard 5-point stencil is used,
                    and we solve for the
                 interior nodes only.

                 This example solves the same problem as
                    Example 3.  Available
                 solvers are AMG, PCG, and PCG with AMG
                    or Parasails
                 preconditioners.  */

#include <math.h>
#include "_hypre_utilities.h"
#include "HYPRE_krylov.h"
#include "HYPRE.h"
#include "HYPRE_parcsr_ls.h"

#include "vis.c"

int hypre_FlexGMRESModifyPCAMGExample(void *precond_data,
     int iterations,
                                       double
                                          rel_residual_norm
                                          );


int main (int argc, char *argv[])
{
   int i;
   int myid, num_procs;
   int N, n;
```

```
int ilower, iupper;
int local_size, extra;

int solver_id;
int vis, print_system;

double h, h2;

HYPRE_IJMatrix A;
HYPRE_ParCSRMatrix parcsr_A;
HYPRE_IJVector b;
HYPRE_ParVector par_b;
HYPRE_IJVector x;
HYPRE_ParVector par_x;

HYPRE_Solver solver, precond;

/* Initialize MPI */
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myid);
MPI_Comm_size(MPI_COMM_WORLD, &num_procs);

/* Default problem parameters */
n = 33;
solver_id = 0;
vis = 0;
print_system = 0;


/* Parse command line */
{
   int arg_index = 0;
   int print_usage = 0;

   while (arg_index < argc)
   {
      if ( strcmp(argv[arg_index], "-n") == 0 )
      {
         arg_index++;
         n = atoi(argv[arg_index++]);
      }
      else if ( strcmp(argv[arg_index], "-solver") ==
         0 )
      {
         arg_index++;
         solver_id = atoi(argv[arg_index++]);
      }
      else if ( strcmp(argv[arg_index], "-vis") == 0 )
      {
         arg_index++;
```

```
            vis = 1;
         }
         else if ( strcmp(argv[arg_index], "-print_system
            ") == 0 )
         {
            arg_index++;
            print_system = 1;
         }
         else if ( strcmp(argv[arg_index], "-help") == 0
            )
         {
            print_usage = 1;
            break;
         }
         else
         {
            arg_index++;
         }
      }

      if ((print_usage) && (myid == 0))
      {
         printf("\n");
         printf("Usage: %s [<options>]\n", argv[0]);
         printf("\n");
         printf("  -n <n>              : problem size in
            each direction (default: 33)\n");
         printf("  -solver <ID>        : solver ID\n");
         printf("                        0  - AMG (
            default) \n");
         printf("                        1  - AMG-PCG\n")
            ;
         printf("                        8  - ParaSails-
            PCG\n");
         printf("                        50 - PCG\n");
         printf("                        61 - AMG-
            FlexGMRES\n");
         printf("  -vis                : save the
            solution for GLVis visualization\n");
         printf("  -print_system       : print the matrix
             and rhs\n");
         printf("\n");
      }

      if (print_usage)
      {
         MPI_Finalize();
         return (0);
      }
   }
```

```
/* Preliminaries: want at least one processor per row
   */
if (n*n < num_procs) n = sqrt(num_procs) + 1;
N = n*n; /* global number of rows */
h = 1.0/(n+1); /* mesh size*/
h2 = h*h;

/* Each processor knows only of its own rows - the
   range is denoted by ilower
   and upper.  Here we partition the rows. We account
      for the fact that
   N may not divide evenly by the number of processors
      . */
local_size = N/num_procs;
extra = N - local_size*num_procs;

ilower = local_size*myid;
ilower += hypre_min(myid, extra);

iupper = local_size*(myid+1);
iupper += hypre_min(myid+1, extra);
iupper = iupper - 1;

/* How many rows do I have? */
local_size = iupper - ilower + 1;

/* Create the matrix.
   Note that this is a square matrix, so we indicate
      the row partition
   size twice (since number of rows = number of cols)
      */
HYPRE_IJMatrixCreate(MPI_COMM_WORLD, ilower, iupper,
   ilower, iupper, &A);

/* Choose a parallel csr format storage (see the User'
   s Manual) */
HYPRE_IJMatrixSetObjectType(A, HYPRE_PARCSR);

/* Initialize before setting coefficients */
HYPRE_IJMatrixInitialize(A);

/* Now go through my local rows and set the matrix
   entries.
   Each row has at most 5 entries. For example, if n
      =3:

   A = [M -I  0; -I M -I; 0 -I M]
   M = [4 -1  0; -1 4 -1; 0 -1 4]
```

```
   Note that here we are setting one row at a time,
      though
   one could set all the rows together (see the User's
       Manual).
*/
{
   int nnz;
   double values[5];
   int cols[5];

   for (i = ilower; i <= iupper; i++)
   {
      nnz = 0;

      /* The left identity block:position i-n */
      if ((i-n)>=0)
      {
         cols[nnz] = i-n;
         values[nnz] = -1.0;
         nnz++;
      }

      /* The left -1: position i-1 */
      if (i%n)
      {
         cols[nnz] = i-1;
         values[nnz] = -1.0;
         nnz++;
      }

      /* Set the diagonal: position i */
      cols[nnz] = i;
      values[nnz] = 4.0;
      nnz++;

      /* The right -1: position i+1 */
      if ((i+1)%n)
      {
         cols[nnz] = i+1;
         values[nnz] = -1.0;
         nnz++;
      }

      /* The right identity block:position i+n */
      if ((i+n)< N)
      {
         cols[nnz] = i+n;
         values[nnz] = -1.0;
         nnz++;
      }
```

```
      /* Set the values for row i */
      HYPRE_IJMatrixSetValues(A, 1, &nnz, &i, cols,
         values);
   }
}

/* Assemble after setting the coefficients */
HYPRE_IJMatrixAssemble(A);

/* Note: for the testing of small problems, one may
   wish to read
   in a matrix in IJ format (for the format, see the
      output files
   from the -print_system option).
   In this case, one would use the following routine:
   HYPRE_IJMatrixRead( <filename>, MPI_COMM_WORLD,
                       HYPRE_PARCSR, &A );
   <filename> = IJ.A.out to read in what has been
      printed out
   by -print_system (processor numbers are omitted).
   A call to HYPRE_IJMatrixRead is an *alternative* to
      the
   following sequence of HYPRE_IJMatrix calls:
   Create, SetObjectType, Initialize, SetValues, and
      Assemble
*/


/* Get the parcsr matrix object to use */
HYPRE_IJMatrixGetObject(A, (void**) &parcsr_A);


/* Create the rhs and solution */
HYPRE_IJVectorCreate(MPI_COMM_WORLD, ilower, iupper,&b
   );
HYPRE_IJVectorSetObjectType(b, HYPRE_PARCSR);
HYPRE_IJVectorInitialize(b);

HYPRE_IJVectorCreate(MPI_COMM_WORLD, ilower, iupper,&x
   );
HYPRE_IJVectorSetObjectType(x, HYPRE_PARCSR);
HYPRE_IJVectorInitialize(x);

/* Set the rhs values to h^2 and the solution to zero
   */
{
   double *rhs_values, *x_values;
   int    *rows;
```

```
      rhs_values = (double*) calloc(local_size, sizeof(
         double));
      x_values = (double*) calloc(local_size, sizeof(
         double));
      rows = (int*) calloc(local_size, sizeof(int));

      for (i=0; i<local_size; i++)
      {
         rhs_values[i] = h2;
         x_values[i] = 0.0;
         rows[i] = ilower + i;
      }

      HYPRE_IJVectorSetValues(b, local_size, rows,
         rhs_values);
      HYPRE_IJVectorSetValues(x, local_size, rows,
         x_values);

      free(x_values);
      free(rhs_values);
      free(rows);
   }


   HYPRE_IJVectorAssemble(b);
   /*  As with the matrix, for testing purposes, one may
       wish to read in a rhs:
       HYPRE_IJVectorRead( <filename>, MPI_COMM_WORLD,
                                       HYPRE_PARCSR, &b );
       as an alternative to the
       following sequence of HYPRE_IJVectors calls:
       Create, SetObjectType, Initialize, SetValues, and
          Assemble
   */
   HYPRE_IJVectorGetObject(b, (void **) &par_b);

   HYPRE_IJVectorAssemble(x);
   HYPRE_IJVectorGetObject(x, (void **) &par_x);


  /*  Print out the system  - files names will be IJ.out.
     A.XXXXX
       and IJ.out.b.XXXXX, where XXXXX = processor id */
   if (print_system)
   {
      HYPRE_IJMatrixPrint(A, "IJ.out.A");
      HYPRE_IJVectorPrint(b, "IJ.out.b");
   }
```

```
/* Choose a solver and solve the system */

/* AMG */
if (solver_id == 0)
{
   int num_iterations;
   double final_res_norm;

   /* Create solver */
   HYPRE_BoomerAMGCreate(&solver);

   /* Set some parameters (See Reference Manual for
      more parameters) */
   HYPRE_BoomerAMGSetPrintLevel(solver, 3);  /* print
      solve info + parameters */
   HYPRE_BoomerAMGSetOldDefault(solver); /* Falgout
      coarsening with modified classical interpolaiton
       */
   HYPRE_BoomerAMGSetRelaxType(solver, 3);   /* G-S/
      Jacobi hybrid relaxation */
   HYPRE_BoomerAMGSetRelaxOrder(solver, 1);   /* uses
      C/F relaxation */
   HYPRE_BoomerAMGSetNumSweeps(solver, 1);   /*
      Sweeeps on each level */
   HYPRE_BoomerAMGSetMaxLevels(solver, 20);  /*
      maximum number of levels */
   HYPRE_BoomerAMGSetTol(solver, 1e-7);      /* conv.
      tolerance */

   /* Now setup and solve! */
   HYPRE_BoomerAMGSetup(solver, parcsr_A, par_b, par_x
      );
   HYPRE_BoomerAMGSolve(solver, parcsr_A, par_b, par_x
      );

   /* Run info - needed logging turned on */
   HYPRE_BoomerAMGGetNumIterations(solver, &
      num_iterations);
   HYPRE_BoomerAMGGetFinalRelativeResidualNorm(solver,
       &final_res_norm);
   if (myid == 0)
   {
      printf("\n");
      printf("Iterations = %d\n", num_iterations);
      printf("Final Relative Residual Norm = %e\n",
         final_res_norm);
      printf("\n");
   }

   /* Destroy solver */
```

```
   HYPRE_BoomerAMGDestroy(solver);
}
/* PCG */
else if (solver_id == 50)
{
   int num_iterations;
   double final_res_norm;

   /* Create solver */
   HYPRE_ParCSRPCGCreate(MPI_COMM_WORLD, &solver);

   /* Set some parameters (See Reference Manual for
      more parameters) */
   HYPRE_PCGSetMaxIter(solver, 1000); /* max
      iterations */
   HYPRE_PCGSetTol(solver, 1e-7); /* conv. tolerance
      */
   HYPRE_PCGSetTwoNorm(solver, 1); /* use the two norm
       as the stopping criteria */
   HYPRE_PCGSetPrintLevel(solver, 2); /* prints out
      the iteration info */
   HYPRE_PCGSetLogging(solver, 1); /* needed to get
      run info later */

   /* Now setup and solve! */
   HYPRE_ParCSRPCGSetup(solver, parcsr_A, par_b, par_x
      );
   HYPRE_ParCSRPCGSolve(solver, parcsr_A, par_b, par_x
      );

   /* Run info - needed logging turned on */
   HYPRE_PCGGetNumIterations(solver, &num_iterations);
   HYPRE_PCGGetFinalRelativeResidualNorm(solver, &
      final_res_norm);
   if (myid == 0)
   {
      printf("\n");
      printf("Iterations = %d\n", num_iterations);
      printf("Final Relative Residual Norm = %e\n",
         final_res_norm);
      printf("\n");
   }

   /* Destroy solver */
   HYPRE_ParCSRPCGDestroy(solver);
}
/* PCG with AMG preconditioner */
else if (solver_id == 1)
{
   int num_iterations;
```

```
double final_res_norm;

/* Create solver */
HYPRE_ParCSRPCGCreate(MPI_COMM_WORLD, &solver);

/* Set some parameters (See Reference Manual for
   more parameters) */
HYPRE_PCGSetMaxIter(solver, 1000); /* max
   iterations */
HYPRE_PCGSetTol(solver, 1e-7); /* conv. tolerance
   */
HYPRE_PCGSetTwoNorm(solver, 1); /* use the two norm
    as the stopping criteria */
HYPRE_PCGSetPrintLevel(solver, 2); /* print solve
   info */
HYPRE_PCGSetLogging(solver, 1); /* needed to get
   run info later */

/* Now set up the AMG preconditioner and specify
   any parameters */
HYPRE_BoomerAMGCreate(&precond);
HYPRE_BoomerAMGSetPrintLevel(precond, 1); /* print
   amg solution info */
HYPRE_BoomerAMGSetCoarsenType(precond, 6);
HYPRE_BoomerAMGSetOldDefault(precond);
HYPRE_BoomerAMGSetRelaxType(precond, 6); /* Sym G.S
   ./Jacobi hybrid */
HYPRE_BoomerAMGSetNumSweeps(precond, 1);
HYPRE_BoomerAMGSetTol(precond, 0.0); /* conv.
   tolerance zero */
HYPRE_BoomerAMGSetMaxIter(precond, 1); /* do only
   one iteration! */

/* Set the PCG preconditioner */
HYPRE_PCGSetPrecond(solver, (HYPRE_PtrToSolverFcn)
   HYPRE_BoomerAMGSolve,
                       (HYPRE_PtrToSolverFcn)
                          HYPRE_BoomerAMGSetup,
                          precond);

/* Now setup and solve! */
HYPRE_ParCSRPCGSetup(solver, parcsr_A, par_b, par_x
   );
HYPRE_ParCSRPCGSolve(solver, parcsr_A, par_b, par_x
   );

/* Run info - needed logging turned on */
HYPRE_PCGGetNumIterations(solver, &num_iterations);
HYPRE_PCGGetFinalRelativeResidualNorm(solver, &
   final_res_norm);
```

```
      if (myid == 0)
      {
         printf("\n");
         printf("Iterations = %d\n", num_iterations);
         printf("Final Relative Residual Norm = %e\n",
            final_res_norm);
         printf("\n");
      }

      /* Destroy solver and preconditioner */
      HYPRE_ParCSRPCGDestroy(solver);
      HYPRE_BoomerAMGDestroy(precond);
   }
   /* PCG with Parasails Preconditioner */
   else if (solver_id == 8)
   {
      int    num_iterations;
      double final_res_norm;

      int      sai_max_levels = 1;
      double   sai_threshold = 0.1;
      double   sai_filter = 0.05;
      int      sai_sym = 1;

      /* Create solver */
      HYPRE_ParCSRPCGCreate(MPI_COMM_WORLD, &solver);

      /* Set some parameters (See Reference Manual for
         more parameters) */
      HYPRE_PCGSetMaxIter(solver, 1000); /* max
         iterations */
      HYPRE_PCGSetTol(solver, 1e-7); /* conv. tolerance
         */
      HYPRE_PCGSetTwoNorm(solver, 1); /* use the two norm
          as the stopping criteria */
      HYPRE_PCGSetPrintLevel(solver, 2); /* print solve
         info */
      HYPRE_PCGSetLogging(solver, 1); /* needed to get
         run info later */

      /* Now set up the ParaSails preconditioner and
         specify any parameters */
      HYPRE_ParaSailsCreate(MPI_COMM_WORLD, &precond);

      /* Set some parameters (See Reference Manual for
         more parameters) */
      HYPRE_ParaSailsSetParams(precond, sai_threshold,
         sai_max_levels);
      HYPRE_ParaSailsSetFilter(precond, sai_filter);
      HYPRE_ParaSailsSetSym(precond, sai_sym);
```

```
      HYPRE_ParaSailsSetLogging(precond, 3);

      /* Set the PCG preconditioner */
      HYPRE_PCGSetPrecond(solver, (HYPRE_PtrToSolverFcn)
          HYPRE_ParaSailsSolve,
                          (HYPRE_PtrToSolverFcn)
                              HYPRE_ParaSailsSetup,
                              precond);

      /* Now setup and solve! */
      HYPRE_ParCSRPCGSetup(solver, parcsr_A, par_b, par_x
          );
      HYPRE_ParCSRPCGSolve(solver, parcsr_A, par_b, par_x
          );


      /* Run info - needed logging turned on */
      HYPRE_PCGGetNumIterations(solver, &num_iterations);
      HYPRE_PCGGetFinalRelativeResidualNorm(solver, &
          final_res_norm);
      if (myid == 0)
      {
         printf("\n");
         printf("Iterations = %d\n", num_iterations);
         printf("Final Relative Residual Norm = %e\n",
             final_res_norm);
         printf("\n");
      }

      /* Destory solver and preconditioner */
      HYPRE_ParCSRPCGDestroy(solver);
      HYPRE_ParaSailsDestroy(precond);
   }
   /* Flexible GMRES with  AMG Preconditioner */
   else if (solver_id == 61)
   {
      int    num_iterations;
      double final_res_norm;
      int    restart = 30;
      int    modify = 1;


      /* Create solver */
      HYPRE_ParCSRFlexGMRESCreate(MPI_COMM_WORLD, &solver
          );

      /* Set some parameters (See Reference Manual for
          more parameters) */
      HYPRE_FlexGMRESSetKDim(solver, restart);
```

```
HYPRE_FlexGMRESSetMaxIter(solver, 1000); /* max
    iterations */
HYPRE_FlexGMRESSetTol(solver, 1e-7); /* conv.
    tolerance */
HYPRE_FlexGMRESSetPrintLevel(solver, 2); /* print
    solve info */
HYPRE_FlexGMRESSetLogging(solver, 1); /* needed to
    get run info later */


/* Now set up the AMG preconditioner and specify
    any parameters */
HYPRE_BoomerAMGCreate(&precond);
HYPRE_BoomerAMGSetPrintLevel(precond, 1); /* print
    amg solution info */
HYPRE_BoomerAMGSetCoarsenType(precond, 6);
HYPRE_BoomerAMGSetOldDefault(precond);
HYPRE_BoomerAMGSetRelaxType(precond, 6); /* Sym G.S
    ./Jacobi hybrid */
HYPRE_BoomerAMGSetNumSweeps(precond, 1);
HYPRE_BoomerAMGSetTol(precond, 0.0); /* conv.
    tolerance zero */
HYPRE_BoomerAMGSetMaxIter(precond, 1); /* do only
    one iteration! */

/* Set the FlexGMRES preconditioner */
HYPRE_FlexGMRESSetPrecond(solver, (
    HYPRE_PtrToSolverFcn) HYPRE_BoomerAMGSolve,
                      (HYPRE_PtrToSolverFcn)
                         HYPRE_BoomerAMGSetup,
                         precond);


if (modify)
/* this is an optional call - if you don't call it
    , hypre_FlexGMRESModifyPCDefault
   is used - which does nothing.  Otherwise, you
      can define your own, similar to
   the one used here */
   HYPRE_FlexGMRESSetModifyPC( solver,
                               (
                                  HYPRE_PtrToModifyPCFcn
                                  )
                                  hypre_FlexGMRESModifyPCAMGExample
                                  );


/* Now setup and solve! */
HYPRE_ParCSRFlexGMRESSetup(solver, parcsr_A, par_b,
     par_x);
```

```
      HYPRE_ParCSRFlexGMRESSolve(solver, parcsr_A, par_b,
           par_x);

      /* Run info - needed logging turned on */
      HYPRE_FlexGMRESGetNumIterations(solver, &
          num_iterations);
      HYPRE_FlexGMRESGetFinalRelativeResidualNorm(solver,
           &final_res_norm);
      if (myid == 0)
      {
         printf("\n");
         printf("Iterations = %d\n", num_iterations);
         printf("Final Relative Residual Norm = %e\n",
             final_res_norm);
         printf("\n");
      }

      /* Destory solver and preconditioner */
      HYPRE_ParCSRFlexGMRESDestroy(solver);
      HYPRE_BoomerAMGDestroy(precond);

   }
   else
   {
      if (myid ==0) printf("Invalid solver id specified.\
          n");
   }

   /* Save the solution for GLVis visualization, see vis/
       glvis-ex5.sh */
   if (vis)
   {
      FILE *file;
      char filename[255];

      int nvalues = local_size;
      int *rows = (int*) calloc(nvalues, sizeof(int));
      double *values = (double*) calloc(nvalues, sizeof(
          double));

      for (i = 0; i < nvalues; i++)
         rows[i] = ilower + i;

      /* get the local solution */
      HYPRE_IJVectorGetValues(x, nvalues, rows, values);

      sprintf(filename, "%s.%06d", "vis/ex5.sol", myid);
      if ((file = fopen(filename, "w")) == NULL)
      {
```

```
         printf("Error: can't open output file %s\n",
            filename);
         MPI_Finalize();
         exit(1);
      }

      /* save solution */
      for (i = 0; i < nvalues; i++)
         fprintf(file, "%.14e\n", values[i]);

      fflush(file);
      fclose(file);

      free(rows);
      free(values);

      /* save global finite element mesh */
      if (myid == 0)
         GLVis_PrintGlobalSquareMesh("vis/ex5.mesh", n-1)
            ;
   }

   /* Clean up */
   HYPRE_IJMatrixDestroy(A);
   HYPRE_IJVectorDestroy(b);
   HYPRE_IJVectorDestroy(x);

   /* Finalize MPI*/
   MPI_Finalize();

   return(0);
}

/*--------------------------------------------------------------------------

   hypre_FlexGMRESModifyPCAMGExample -

    This is an example (not recommended)
   of how we can modify things about AMG that
   affect the solve phase based on how FlexGMRES is doing
      ... For
   another preconditioner it may make sense to modify the
       tolerance..

 *--------------------------------------------------------------------------*/


int hypre_FlexGMRESModifyPCAMGExample(void *precond_data,
    int iterations,
```

```
                                        double
                                            rel_residual_norm)
{


   if (rel_residual_norm > .1)
   {
            HYPRE_BoomerAMGSetNumSweeps((HYPRE_Solver)
               precond_data, 10);
   }
   else
   {
      HYPRE_BoomerAMGSetNumSweeps((HYPRE_Solver)
         precond_data, 1);
   }


   return 0;
}
```