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

Thomas Rückert

March 8, 2017

Abstract

Contents

Todo list

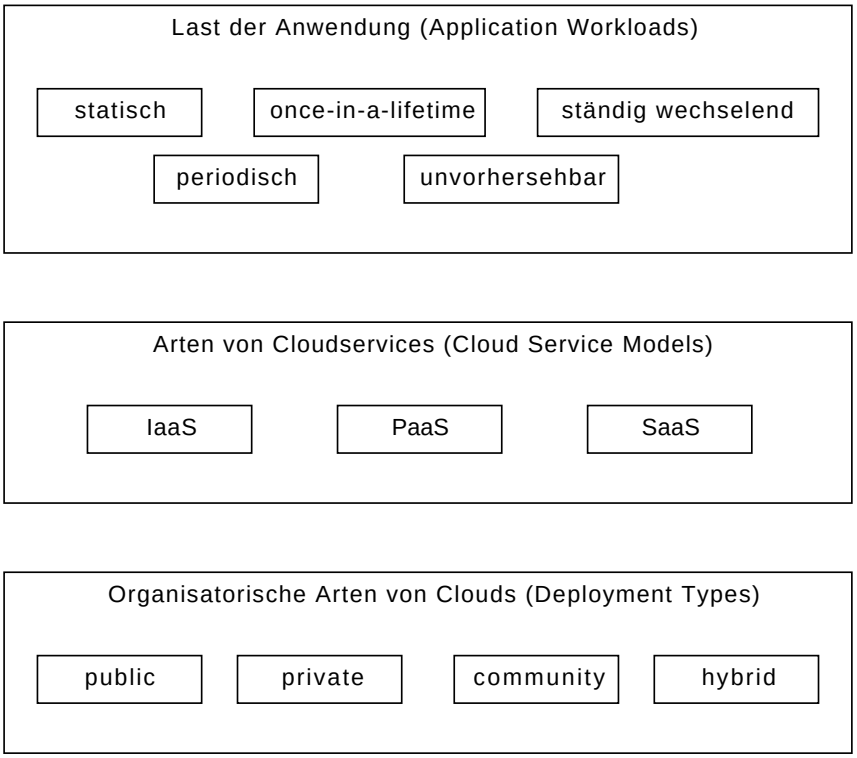
1 Informationssammlung und Einführung in die relevanten Themen

1.1 Recherche zum Thema Cloud

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.

Figure 1: Übersicht Grundlagen Cloud Computing



Im Folgenden werden die eben kurz angeschnittenen Eigenschaften von Cloudsystemen näher betrachtet. Ein Übersicht dazu ist in Abbildung 1 gegeben.

Verteilung der Last von Anwendungen Im Buch Cloud Computing Patterns von Christoph Fehling wird die Verteilung von Last in die folgenden Kat-

egorien 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.

1.1.2 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 [genauer definieren] ist essentiell um eine Verbesserung durch die Auslagerung von Berechnungen zu erreichen. So kann Zugriffszeit auf Daten weniger abhängig von ihrem pysikalischen Speicherort werden.

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 [genauer definieren]) 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. [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.

1.1.3 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. [wie?] Da verteilte Anwendung diese Eigenschaft aufweisen sind Technologien wie Webservices, SOA, asynchrone Kommunikation relevant für Cloudanwendungen. [Technologien etwas weiter ausführen]

1.1.4 Arten von Cloud

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. [wie zum beispiel, kurz erläutern] **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.

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.

1.1.5 Cloudsysteme

Vergleich in Tabelle

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-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

1.1.6 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-dock`
- more raspberry `https://www.raspberrypi.org/forums/viewtopic.php?f=36&t=54997`

1.2 HYPRE - Überblick über die Bibliothek

HYPRE ist eine freie Software von Lawrence Livermore National Laboratory. Es ist unter der GNU Lesser General Public License (Free Software Foundation) Version 2.1 lizensiert. Der Funktionsumfang von HYPRE umfasst 'Scalable Linear Solvers and Multigrid Methods'. Es steht als Bibliothek für die Sprachen C (nativ) und FORTRAN bereit. Die aktuellste Version 2.11.1 ist seit dem 09.06.2016 verfügbar. Bis zur vorletzten Version 2.10.1 wurde HYRPE auch mit dem Babel Interface bereit. 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.

Nutzt MPI für Parallelisierbarung.

-performance?

1.2.1 Funktionsumfang allgemein

Wie bereits erwähnt wird der Funktionsumfang von HYPRE als 'Scalable Linear Solvers and Multigrid Methods' beschrieben. -was heist das? was genau kann man lösen?

Conceptual Interfaces

Solver Strategies

Preconditioner(s)

1.2.2 Funktionsweise

wie löst sie die probleme (groben einblick in die funktionsweise der methoden, falls möglich)

-wie funktioniert das? (lässt sich das beantworten)

1.2.3 Verwendung

Installation Download von `http://computation.llnl.gov/projects/hypre-scalable-linear-solver-software` oder `https://github.com/LLNL/hypre`. Innerhalb des Verzeichnis muss 'configure' gefolgt von einem 'make' ausgeführt werden. Alternativ kann auch das build tool CMake eingesetzt werden.

Vorbereitungen vor einer Implementierung

erste Implementierung liste siehe manual:

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.

Ausführung, Test 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.

1.2.4 Ressourcen

- offiziell: `http://computation.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-m`
- Übersicht Publikationen: `http://computation.llnl.gov/projects/hypre-scalable-linear-solve 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`

1.3 Werkzeuge für die verteilte Ausführung

- c/c++ json lib benchmarks `https://github.com/miloyip/nativejson-benchmark`
- 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/`

1.3.1 Using RPC inside of C

RPC steht für Remote Procedure Call. Sie bieten die Möglichkeit von Funktionsaufrufen in verteilten Systemen. Im Gegensatz zu Protokollen wie HTTP oder 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/`

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

Installation: In the simplest case, it's just a conventional

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

json-rpc 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
rpc client?: https://groups.google.com/forum/#!topic/json-rpc/9O1dbQehU04
tcp client: http://jsonrpc-cpp.sourceforge.net/index.php?n=Main.HomePage
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:

```
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.)

```
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

json vs xml

complete RPC libs

- client and server stubs: `https://github.com/masroorhasan/RPC`
- hpc optimized: `https://github.com/mercury-hpc/mercury`
- protobuf: `https://github.com/protobuf-c/protobuf-c-rpc`
- rpc for php and c: `https://github.com/laruence/yar`

1.3.2 Calling Python from C

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`

```
#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

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

1.3.3 spring-server framework

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

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

1.3.4 Service

1.3.5 Socket

1.4 Serialisierung

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

Protocol Buffers `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`

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

2 Vergleiche und Vorbereitungen für die Implementierung

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

2.1 Ressourcen

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

2.2 'private' vs 'public' Cloud

- gibt es open source cloud systeme? welche?
- welche public clouds gibt es (zB aws, windows azure, adobe creative)
- was sind unterschiede (neben dem access, zB performance?)

2.3 Technologie

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

2.4 Cloudtyp

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

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)
- aber entscheidend: wie eignen sich diese typen für 'unsere' Implementierung

2.5 Sprache

- hardware-nah: C/C++ (bessere performance)
- vs netzwerknah: (bessere möglichkeiten die kommunikation zu Implementieren)
- kann eine hybridform eingsetzt werden? zB (micro-)service: bib in c kommuniziert mit client-backend in php, dieses führt die calls zum server aus

2.6 Vergleich ausgewählter Cloudsysteme in einer Tabelle

anhand relevanter (für das Projekt) Eigenschaften

Table 1: My caption

Cloudsystem	private	PaaS	...
Openstack	☑		
Apache CS	☑		
Amazon AWS	☐	☑	

2.7 Roadmap Implementierung

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.

3 (Test-)Installation OpenStack

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

Ubuntu 16.04 in Vagrant aufsetzen

1. `vagrant init bento/ubuntu-16.04`
2. Vagrantfile anpassen. Es sollten mindestens 4GB RAM zugewiesen werden, damit OpenStack performant läuft.

```
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`

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.

```
=========================
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
```

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.

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

4 Zukünftige, weiterführende Arbeiten

eventuell werden Teile von hier in die aktuelle Arbeit verschoben

- Skalierbarkeit der Cloud (einsetzen)
- Vorteile der Auslagerung:
  - performance
  - speicher
  - lösbarkeit (nur remote überhaupt lösbar)