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

Thomas Rückert

February 1, 2017

Abstract

Contents

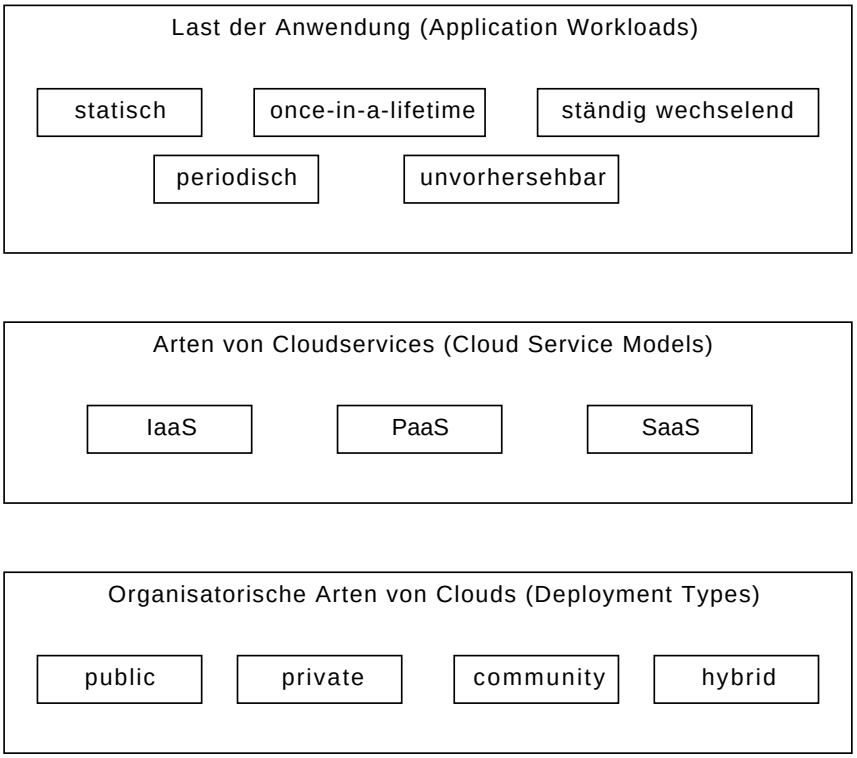
1 Informationssammlung und Einführung in die relevanten Themen

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

Verfügbare Cloudsysteme Vergleich in Tabelle

Open Source Cloudsystem für private Clouds

- openstack `https://www.openstack.org/`
- apache cloudstack `https://cloudstack.apache.org/`
- 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/

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

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

....
....
.....

Müssen wir structs übertragen?????????

Reicht es inbuild data types zu übertragen und structs werden nur auf dem server erstellt?

Beeinflusst die Entscheidung der Wahl des Frameworks.

....
....
.....

- 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 RPC

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

1.3.2 Service

1.3.3 Socket

2 Vergleich verschiedener Technologien und Werkzeuge für den Einsatz bei der Implementierung

2.1 '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?)

2.2 Technologie

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

2.3 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

2.4 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

3 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)