Scalable Web Architectures: Task 1

Markus Deutschl, BSc

Definition

„**Skalierbarkeit** ist die Fähigkeit eines Systems, sich wachsenden Ansprüchen an die Leistungsfähigkeit anzupassen. Systeme werden sowohl auf Hardware- als auch auf Softwareebene skaliert.“ (Beren 2011). Das bedeutet, dass sowohl auf der Hardware- als auch auf Softwareseite Mechanismen implementiert werden können, um mit höheren, aber auch niedrigeren Anforderungen an die **Leistungsfähigkeit** des Systems umgehen zu können. Hierbei müssen mitunter schon bei der Entwicklung bzw. der Auswahl der verwendeten Software Vorbereitungen getroffen werden, um diesen Aufbau skalieren zu können.

Im Allgemeinen gibt es bei der Skalierung auf der Hardwareseite zwei Ansätze, die hier nachfolgend kurz erklärt werden sollen.

Vertikale Skalierung (scale-up): Die Leistungsfähigkeit des Systems wird hier durch die Verwendung von leistungsstärkerer Hardware gesteigert. Es werden z.B. mehr CPUs oder mehr RAM eingebaut um den Rechner, auf dem die Software läuft, aufzuwerten. Der Vorteil dieses Ansatzes ist, dass die Leistung ohne Änderungen des Applikationscodes erhöht werden kann. Die vertikale Skalierung ist meistens mit relativ hohen Kosten verbunden, wenn sie eine gewisse Leistungsgrenze übersteigen soll. Außerdem ist sie nicht unbegrenzt möglich, da irgendwann die beste Hardware, die sich am Markt befindet, verbaut ist und die Maschine keinen Platz für weitere Teile bietet. [vgl. (Straube 2011)]

Horizontale Skalierung (scale-out): Im Gegensatz zur vertikalen Skalierung wird hier die Leistungsfähigkeit durch das Hinzufügen von zusätzlichen Rechnern zum System erhöht. Dies ist meist billiger, da hier auch sogenannte „Commodity Hardware“ verwendet werden kann. Allerdings kommt es hier sehr stark auf die Software an, wie stark die Leistung gesteigert werden kann. Diese muss von Anfang an auf Verteilung bzw. Parallelisierung ausgelegt sein, da die einzelnen Rechenaufgaben sonst sehr schwer über mehrere Rechner verteilt werden können. [vgl. (Straube 2011)]

Des Weiteren gibt es mehrere **Dimensionen**, in denen die Skalierbarkeit gemessen werden kann.

Administrative Skalierbarkeit: Sie bezeichnet die Fähigkeit, mit steigenden Benutzerzahlen umgehen zu können, die sich ein einzelnes, verteiltes System teilen. [vgl. (Bill 2012) S.8]

Funktionale Skalierbarkeit: Ein System wird als gut funktional skalierbar bezeichnet, wenn das Hinzufügen neuer Funktionalität mit geringem Aufwand möglich ist. Dies kann durch ein

gutes Software-Design erreicht werden, das auf eine robuste und modulare Code-Struktur setzt.

Geographische Skalierbarkeit: Bezeichnet die Fähigkeit eines Systems, die Performance, den Nutzen oder die Usability unabhängig von der Expansion zu einem geographisch verteilten Ansatz zu wahren. [vgl. (Bill 2012) S.8]

Lastskalierbarkeit: Die Reaktionsfähigkeit eines verteilten Systems, die Ressourcen an größere oder kleinere Last bzw. Anzahl der Eingaben anpassen zu können. Alternativ kann hier auch betrachtet werden, wie einfach ein System bzw. eine Komponente hinzugefügt, entfernt oder verändert werden kann, um sich der sich ändernden Last anpassen zu können. [vgl. (Gudars 2008)]

Bei **Web-Projekten** bezieht sich Skalierbarkeit in den meisten Fällen darauf, mehr Benutzeranfragen pro Tag bzw. Stunde performant verarbeiten zu können. Hierbei gibt es unterschiedliche Subsysteme der Anwendung, bei denen man die Performance erhöhen kann. Ein großer Schwachpunkt sind meistens die **Datenbankzugriffe**, die die Anwendung deutlich ausbremsen können. Die Vermeidung dieser kann bereits erhebliche Einsparungen zur Folge haben. Dies lässt sich durch den Einsatz von Caching und durch die Verwendung nativer Konnektoren zur Datenbank und durch Replikation erreichen.

Weitere Punkte:

- Content Delivery Networks (CDNs)
- Auslagern des Datenbanksystems auf externe Maschinen
- Clustering und Replikation des Datenbanksystems
- Cloud Computing
- Reverse Proxies
- Load Balancing durch Einsatz mehrerer Webserver

Evaluierung von zwei Web Projekten

HD+ BSS

Während meiner zwei Praktika bei der Infonova GmbH habe ich am BSS und der Call Center Software für HD+ mitgearbeitet. Ich darf hier zwar nur begrenzt Auskunft geben und weiß auch nicht über die gesamte Architektur und Konfiguration bzw. Skalierbarkeit Bescheid, aber ich werde versuchen, die geforderten Punkte trotzdem zu beantworten.

Architektur

- Server (Web und DB-Server):
  - Betriebssystem: Solaris
  - Hardware: unbekannt (Da in Kundenhand)
- Webserver: JBoss 4.2.2
- Datenbanksystem: Oracle Enterprise 10g
- Programmiersprache: Java EE
- ORM: Hibernate
- Workflow-Engine: JBPM (Version unbekannt)
- Frameworks:

    - Active Controls
    - JSP
    - WebAC
    - Freemarker Template Engine
- Build Tool: Apache Maven 2.x
- Continuous Integration Server: Apache Continuum
- Test-Frameworks:
    - Junit
    - TestNG
    - Selenium

Erhöhung der Zugriffe

Im besagten System muss zwischen verschiedenen Arten von Zugriffen unterschieden werden.

- Zugriffe auf das Web-Frontend (von Call Center Operatoren)
- Anfragen aus dem externen Webshop (startet Workflows)
- Import/Export Aufgaben von Files → DB und umgekehrt (von externen Vertragspartnern)

Diese drei verschiedenen Arten von Zugriffen werden nun als getrennte Szenarien betrachtet und bei der Erhöhung der Zugriffe beschrieben.

Frontend

Da das Web-Frontend nur von Call Center Mitarbeitern verwendet wird, ist es nicht realistisch, dass die Zugriffe so exponentiell steigern. Ein hundertfacher Anstieg würde jedoch auf keinen Fall ein Problem für den Server darstellen. Dies zeigte sich bereits durch Tests auf der viel schwächer dimensionierten Testmaschine. Bei der 10.000 fachen Anzahl von Zugriffen würde es jedoch auf jeden Fall zu Problemen kommen, da die Inhalte der Website sehr oft dynamisch aus der Datenbank geladen werden müssen, da die Operatoren immer die aktuellsten Daten benötigen. Der Webserver alleine würde diese Belastung leicht bewältigen können, jedoch wäre die Latenz durch die ständigen Datenbankzugriffe sehr hoch. Aber wie bereits angesprochen ist dieses Szenario für ein Call Center nicht realistisch.

Webshop

Hier kommen Anfragen aus dem Webshop auf der HD+-Website für Bestellungen zum Server und müssen durch Workflows abgearbeitet werden. Eine Anforderung des Kunden war es, etwa drei Millionen solcher Anfragen pro Tag verarbeiten zu können, was zu meiner Zeit bei Infonova auch durch Tests bestätigt wurde. Würde die Anzahl allerdings drastisch erhöht werden, würde ein Bottleneck durch die Datenbank entstehen, da die Workflow-Engine dort bei jedem einzelnen Schritt für jeden einzelnen Workflow Zugriffe machen muss.

Import/Export

Auf dem Server werden sporadisch Dateien von Vertragspartnern via FTP abgelegt, die dann entsprechend weiterverarbeitet werden müssen. Hierbei werden etwa E-Mails versendet, oder Workflows freigegeben. Da diese Operationen an sich wenig komplex sind, würde auch die hundertfache Anzahl dieser auch keine wesentlichen Auswirkungen auf das System haben. Bei der 10.000 fachen Anzahl könnten jedoch Zugriffe auf das Backend nicht funktionieren bzw. die Datenbank etwas überlastet werden. Durch robuste

Fehlermechanismen sollte diese Menge an Anfragen jedoch trotzdem verarbeitet werden können.

Features

Die Applikation an sich kann leider nur zu einem gewissen Grad skalieren. Aufgrund des verwendeten Datenbanksystems kann der Datenbestand nur innerhalb der Limitierungen von Oracle 10g skaliert werden (hauptsächlich vertikal).

Eine Möglichkeit zur Skalierung besteht bei den Workflows. Diese können auf einer eigenen Instanz des Jboss Webservers deployed werden, die nur dafür zuständig ist und eine eigene Datenbank benötigt. Somit könnte der Haupt-Webserver und das Datenbanksystem entlastet werden. Auch der Import/Export-Service könnte auf einem eigenen Server + Datenbank getrennt vom Restsystem laufen.

Abgesehen dieser Möglichkeiten bleibt nur noch die vertikale Skalierung als Möglichkeit, da im Restsystem zu viele Abhängigkeiten existieren, die nur durch eine komplette Neuimplementierung behoben werden können.

MovLib

Da ich außer dem HD+-Projekt nicht wirklich eine Applikation habe, die ich analysieren könnte werde ich MovLib als zweites Projekt heranziehen. Das Projekt gibt es zwar noch nicht, jedoch ist die Systemarchitektur schon festgelegt und es existieren bereits Messwerte zur Performance der Konfiguration.

Architektur

- KVM-Root Server:
  - 2 CPU-Kerne @ 2,5GHz
  - 2048 MB RAM
  - 45 GB SAS-RAID
- Webserver: nginx 1.3.x
- Datenbanksystem: MariaDB 10.0.x
- Programmiersprache: PHP 5.5
- Schnittstelle: FastCGI
- Prozessverwalter: php-fpm
- OPC-Cache: PHP APC
- DB-Konnektor: mysqlnd (nativer Konnektor)

Erhöhung der Zugriffe

Eine Abschätzung der Zugriffszahlen ist sehr schwierig, da MovLib sehr stark von seiner User-Basis abhängt und diese erst aufbauen muss. Um auf eine Zugriffsrate wie die des größten Konkurrenten imdb.com (etwa 28 Millionen pro Tag) müsste schon eine extrem große Benutzeranzahl existieren, die die Website aktiv benutzt.

Was jedoch bereits bekannt ist, dass die Verarbeitung von etwa 10 Millionen HTTP-GET-Anfragen sehr wohl realistisch ist. Diese Zahl wurde bereits von Richard Fussenegger in seiner Bachelorarbeit „High Performance und Hardening“ erhoben, in der sich durch starke Konfigurationsanpassungen der oben aufgezählten Systeme diese Zahl durch Benchmarks ergab. Es ist zu erwarten, dass die Anzahl der bearbeitbaren Anfragen beim Aufbau von

MovLib noch höher wäre, da der verwendete Server mehr Leistung besitzt als bei den durchgeführten Benchmarks. [vgl. (Fussenegger 2012)]

Wenn man nun von etwa 100.000 Anfragen pro Tag, was in etwa doppelt so viel ist wie die Anfragen des Konkurrenten ofdb.de, als Nennwert ausgeht, wäre der Faktor 100 für den Server kein Problem. [vgl. (4seohunt 2013)] Hier wären wir bei genau 10 Millionen Anfragen, was ca. dem Limit des Servers entspricht.

Zieht man nun den Faktor 100.000 in Betracht, so würden sich etwa 10 Milliarden Anfragen pro Tag ergeben. Hier ist auch ohne Messungen klar, dass diese nicht verarbeitet werden können. Diese riesige Anzahl würde sogar den täglichen Traffic von wikipedia.org klar übertreffen, welcher in einer Spitzenzeit im Jahr 2009 bei ca. 7 Milliarden lag [vgl. (iwebtool 2013)].

Um einer so riesigen Anzahl von Anfragen Herr zu werden müssten definitiv viel mehr Server rund um die Welt in Betrieb genommen werden, die über Load Balancing angesteuert werden, um die Last zu verteilen. Des Weiteren müsste der Datenbestand durch Replikation aufgeteilt werden, um auch hier Engpässe zu vermeiden. Idealerweise würde hier jeder Webserver zumindest einen Datenbankserver zur Verfügung gestellt bekommen.

Außerdem müsste ein CDN zu Hilfe genommen werden, um statische Dateien wie Bilder, Stylesheets und JavaScript effizient auszuliefern und somit auch die Serverlast zu reduzieren. Eine aggressive Caching-Strategie mit Reverse Proxies und memcached könnte noch für weitere Lastreduzierung sorgen.

Features

PHP als Programmiersprache wird nur immer auf einem Server ausgeführt und deshalb können einzelne Anfragen nicht parallelisiert werden. Allerdings können einzelne Anfragen sehr gut auf viele Server durch Load Balancing verteilt werden, solange keine Sessions verwendet werden.

MariaDB kann durch Replikation auf viele Server verteilt werden und kann so die lesenden Zugriffe beschleunigen. Schreibende Zugriffe werden jedoch nur auf einem Master-Server erlaubt, wodurch möglicherweise ein Bottleneck entstehen könnte. Diesem Problem könnte man aber mit einem Cluster-Setup beikommen.

Weitere Beschleunigungen können durch die Konfigurationen von nginx, MariaDB, APC und des Kernels erlangt werden. Dies wird im Rahmen des Masterprojekts auch eine Rolle spielen.

Vorkenntnisse und Wünsche

Durch Lehrveranstaltungen im Bachelor-Studium habe ich bereits Erfahrungen mit Clustering gesammelt. Installation und Konfiguration von Linux-Programmen sind mir auch geläufig, jedoch habe ich noch nichts in Richtung Optimierung gemacht.

Mich würde persönlich die Skalierung von Datenbanksystemen interessieren, da ich mir dank meiner zweiten Bachelorarbeit schon einiges an theoretischem Wissen aneignen konnte, allerdings dieses Wissen in der Praxis noch nie angewandt habe.

Außerdem interessiere ich mich noch für den effektiven Einsatz von Caching-Mechanismen, um die Performance von Anfragen zu steigern.

Die Implementierung von verteilten Anwendungen mit unterschiedlichen Subsystemen und die Kommunikation zwischen diesen wäre auch noch interessant. Auch im Hinblick auf den Code-Aufbau und die Robustheit der Kommunikationsstrukturen.

Literaturverzeichnis

4seohunt. *Ergebnisseite für ofdb.de.* 2. März 2013. http://4seohunt.com/www/ofdb.de (Zugriff am 2. März 2013).

Beren, Oliver. „Skalierbarkeit kurz und gut.“ *Uni Hannover Website.* 29. November 2011. http://www.se.uni-hannover.de/pub/File/kurz-und-gut/ws2011-labor-restlab/RESTLab-Skalierbarkeit-Oliver-Beren-Kaul-kurz-und-gut.pdf (Zugriff am 1. März 2013).

Bill, Martin. „Konzeption und Analyse einer skalierbaren Architektur für Social Media Streams.“ *TU Dresden Website.* 30. Juni 2012. http://www.rn.inf.tu-dresden.de/uploads/Studentische_Arbeiten/Diplomarbeit_Martin_Bill.pdf (Zugriff am 1. März 2013).

Fussenegger, Richard. „High Performance und Hardening.“ *Bobdo Website.* 2. September 2012. http://bobdo.net/theses/bachelor-2.pdf (Zugriff am 1. März 2013).

Gudars. *Über die Skalierbarkeit und große Datenmengen.* 15. Januar 2008. http://leaspro.blogspot.co.at/2008/01/skalierbarkeit-und-groe-datenmengen.html (Zugriff am 1. März 2013).

iwebtool. *Alexa Traffic Rank Ergebnis für wikipedia.org.* 27. Februar 2013. http://www.iwebtool.com/alexa_traffic_rank?domain=wikipedia.org&data=p&range=max (Zugriff am 2. März 2013).

Straube, Georg. „Cloud Computing zur Optimierung einer Netzwerkpartitionierung.“ *Uni Rostock Website.* 15. August 2011. http://wwwiuk.informatik.uni-rostock.de/fileadmin/wwwiuk/download/studarbeiten/bsca_georg_straube.pdf (Zugriff am 1. März 2013).