Definition

Um über **skalierbare Web-Architekturen** sprechen zu können, muss erst definiert werden, was *Skalierbarkeit* allgemein bedeutet. Skalierbarkeit beschreibt wie die Leistung eines Softwaresystems durch Hinzufügen von Ressourcen und/oder weiterer Knoten (Rechner) gesteigert werden kann. Die Definition durch *The Linux Information Project* (LINFO) lautet:

„Scalable refers to the situation in which the throughput changes roughly in proportion to the change in the number of units of or size of the inputs." (The Linux Information Project (LINFO) 2006)

Wobei der *throughput* die Menge an Arbeit oder Ausgabe bedeutet, die in einer vorgegebenen Zeit vom System berechnet werden kann.

Es existieren zwei Methoden zur Leistungssteigerung, jede Methode besitzt Vor- und Nachteile:

- **Vertikale Skalierung (*scale up*)** – Die Leistung des bestehenden Systems wird durch Hinzufügen von Ressourcen (schnellere CPU, mehr/schnelleres RAM/HDD, etc. pp.) gesteigert. Großer Vorteil der vertikalen Skalierung ist die Tatsache, dass an der Software keinerlei Veränderungen vorgenommen werden müssen. Der größte Nachteil sind die hohen Kosten und die Tatsache, dass sehr schnell ein Limit erreicht wird, da Hardwareleistung nicht in die Unendlichkeit reicht (auch wenn sie sich ständig weiter entwickelt).
- **Horizontale Skalierung (*scale out*)** – Die Leistung des bestehenden Systems wird durch Hinzufügen von Knoten (weiteren Rechnern) gesteigert. Großer Vorteil ist, dass dieser Art der Skalierung keine Grenzen gesetzt sind, es lassen sich immer wieder neue Knoten hinzufügen. Der größte Nachteil ist, dass die Software angepasst werden muss um parallelisierbar zu sein. Dabei lässt sich nicht jede Software parallelisieren, was unter Umständen ein Problem darstellen kann. Es ergeben sich weitere Nachteile, so steigt der Verwaltungsaufwand, da alle Knoten miteinander kommunizieren müssen. Dies wiederum führt auch dazu, dass das Netzwerk zwischen den verschiedenen Knoten besonderer Beachtung bedarf.

Der Kostenfaktor führt in der Regel dazu, dass Unternehmen sich für die vertikale Skalierung entscheiden, es handelt sich um eine einzelne Ausgabe die sehr einfach vorherzusehen ist. Virtualisierung kann jedoch eine horizontale Skalierung sehr viel billiger werden lassen als eine horizontale. Unternehmen wie Facebook und Google skalieren in ihren Systemen vertikal, die benötigte Rechenleistung könnte durch horizontale Skalierung niemals erreicht werden.

Für Web-Applikationen gibt es weitere Schlüsselfaktoren die beachtet werden müssen. Die folgende Liste beschreibt sechs Prinzipien die es bei einer verteilten Web-Applikation zu beachten gilt: (Matsudaira 2012)

- **Verfügbarkeit (*availability*)** – Die Uptime (Zeit in der eine Website erreichbar ist) ist absolut kritisch für die Reputation und Funktionalität vieler Unternehmen. Händler wie Amazon verlieren tausende Verkäufe wenn die Website auch nur für Sekunden nicht verfügbar ist. Hochverfügbarkeit erfordert Redundanz von Schlüsselkomponenten, schnelle Wiederherstellung bei (Teil-)Systemausfällen und Fehlertoleranz wenn Probleme auftreten.
- **Geschwindigkeit (*performance*)** – Die Geschwindigkeit beeinflusst die Benutzung, die Benutzerzufriedenheit und Suchergebnisplatzierung. Eine Web-Applikation muss deshalb für schnelle Antworten und niedrige Latenz ausgelegt werden.
- **Zuverlässigkeit (*reliability*)** – Eine Anfrage muss stets dieselbe Antwort liefern, werden Daten geändert oder aktualisiert muss die Antwort diese Änderungen berücksichtigen und die neue Version ausliefern. Benutzer müssen sich auf Konsistenz und Verhalten verlassen können.
- **Skalierbarkeit (*scalability*)** – Wird hier nochmals als extra Punkt aufgeführt, beeinflusst jedoch die anderen Punkte direkt bzw. sind die anderen Punkte Teil der Skalierbarkeit.
- **Verwaltungsaufwand (*manageability*)** – Ein System sollte einfach zu verstehen sein, Fehler sollten schnell gefunden werden, neue Benutzer sollten sich schnell zurecht finden. Um schnell und einfach skalieren zu können ist es sehr wichtig den Verwaltungsaufwand gering zu halten.
- **Kosten** – Die Kosten ziehen sich wieder durch alle anderen Punkte durch. Hardware, Software, Stundenaufwände und andere Kosten (z. B. Strom) müs-

sen hier beachtet werden. Geld das ausgegeben wird sollte natürlich auch wieder hereinkommen durch die Applikation (dies ist jedoch stark abhängig vom Geschäftsmodell).

Um nun eine Web-Applikation skalierbar zu gestalten existieren diverse Techniken, die folgende Liste soll ein paar der wichtigsten kurz aufführen: (Matsudaira 2012) (Henderson 2007)

- Caching
- Proxies
- Load Balancing
- Queues
- Asynchrone Systeme
- Replikation
- Content Delivery Networks
- Cloud Computing
- …

Omicron.at

Während meines Bachelorpraktikums habe ich bei Omicron [www.omicron.at] gearbeitet. Die Website ist **nicht** von mir umgesetzt worden, ich habe lediglich daran während meines Praktikums gearbeitet.

Die Architektur der Omicron-Website ist sehr einfach und wie zu erwarten:

- Virtueller Server mit Debian 6
- Apache 2.2.15 Webserver
- PHP APC als OPC-Cache
- PHP 5.3.3 via `mod_php`
- TYPO3 4.5.x
- MySQL 5-Datenbank

Ich habe damals während meines Praktikums diverse Bereiche der Website optimiert. So habe ich die Webserverkonfiguration vereinheitlich und vereinfacht. Konfiguration von ETags, far future expires headers und GZIP-Komprimierung um die Frontend-Geschwindigkeit ohne Softwareänderungen zu optimieren. Eine weitere Maßnahme war die Erstellung einer CSS-Sprite für alle Bilder die im Design als Hintergrundgrafik mehr als einmal vorkommen. Der YSlow-Score vor meinen Änderungen lag irgendwo zwischen 50 und 60, nach meinen Änderungen (und bis heute) wird ein Score von 86 erreicht. Mehr ist *dank* TYPO3 nicht einfach umsetzbar.

Der VPS der von Omicron eingesetzt wird hat Reserven ohne Ende. Da es sich um ein großes Unternehmen handelt wurde direkt in den, zum Zeitpunkt der Anmietung, größten VPS investiert. Der Server hat eine sehr schnelle Intel Xeon CPU und über 12 GB Arbeitsspeicher. Für vertikale Skalierung ist entsprechend bereits im Vorfeld gesorgt worden.

Die genaue Anzahl an Zugriffen ist mir nicht bekannt, die folgenden Aussagen sind entsprechend Schätzungen von mir und entbehren jeglicher messbarer Grundlagen. 100 × mehr tägliche Zugriffe sollte der Server gerade noch vertragen, 100 × mehr Zugriffe pro Sekunde dürften jedoch bereits ein Problem darstellen. Dasselbe gilt für 10.000 × mehr Zugriffe pro Tag. 10.000 × mehr Zugriffe pro Sekunde kann der Server nicht verarbeiten da hier das C10K-Problem zum Tragen kommt. (Kegel 2006)

Schlechte Skalierbarkeit ergibt sich bereits aus der Architektur der Applikation. Apache ist ein Webserver der sehr viele Funktionen bietet, außer den Standardfunktionen eines jeden Webservers benötigt die Omicron-Website jedoch keine der Funktionen. In Sachen Performance ist der Apache das Schlusslicht aller bekannter Webserver. Der Einsatz von nginx oder lighttpd würde hier sofort sehr einfach Abhilfe schaffen. PHP über `mod_php` anzusprechen ist ein weiteres großes Problem. Die Verwendung von php-fpm würde hier die Prozesse (Apache und PHP) entkoppeln und für eine enorme Leistungssteigerung sorgen. Ein anderes großes Problem stellt TYPO3 dar. Das CMS ist bekannt dafür sehr träge und langsam zu sein und eine Verteilung ist mehr oder weniger gar nicht möglich aufgrund des Aufbaus des CMS.

Um die Skalierbarkeit zu verbessern müsste zuerst einmal die Software angepasst werden. Der Einsatz eines CMS ist oftmals bereits hinderlich wenn es um Verteilung geht, da sie nicht für die parallele Verarbeitung konzipiert werden und davon ausgehen, dass die Applikation auf einem Server läuft. Alle Anfragen müssen vom selben Server beantwortet werden insofern Sessions ins Spiel kommen. Natürlich gibt es auch hier Methoden diesem Umstand entgegen zu wirken, diese sind jedoch meist mit einem sehr hohen Verwaltungsaufwand verbunden. Dies würde sich für Omicron nicht rechnen. Eine eigens entwickelte, einfache Software wäre entsprechend die kostengünstigste Lösung. Danach müsste die Architektur angepasst werden. Schnellerer Webserver, php-fpm als Prozessmanager und z. B. der Einsatz von CloudFlare als CDN und Proxy.

Bachelor 2

In meiner zweiten Bachelorarbeit habe ich damit experimentiert selbst eine Architektur zu erarbeiten die eine extrem hohe Anzahl von Anfragen mit sehr kostengünstiger Hardware verarbeiten kann. Der Aufbau war dabei wie folgt:

- KVM VPS (EDIS KVM ADVANCED plus)
  - Intel basiert (2 Kerne)
  - 1.536 MB Arbeitsspeicher
  - 22 GB SAS-RAID
  - GB-Anbindung
- Debian GNU/Linux 6 "Squeeze" x64
- `sysctl.d`-Optimierung (Fussenegger 2012)
- nginx als Webserver
- PHP APC als OPC-Cache
- PHP 5.4.5 via php-fpm
- Drupal 7.15
- MariaDB 5.5.25

Die finalen Benchmarks haben ergeben, dass der Server über 10 Millionen Anfragen pro Tag beantworten kann. Hierbei war jedoch ein großes Problem, dass die Server die verwendet wurden um die Anfragen zu generieren, gar nicht mehr Anfragen generieren konnten. Sehr wahrscheinlich könnte der Server noch mehr Anfragen pro Tag beantworten.

Die gesamte Arbeit mit genauem Testaufbau und den durchgeführten Benchmarks kann über folgenden Link abgefragt werden: bobdo.net/theses/bachelor-2.pdf

In der Arbeit beschreibe ich auch Möglichkeiten wie die Leistung weiter gesteigert werden kann. Die Architektur meines Masterprojektes MovLib basiert auf diesen Erfahrungen und dort werde ich gemeinsam mit meinen Teammitgliedern weiter eruieren wie die Leistung noch weiter gesteigert werden kann.

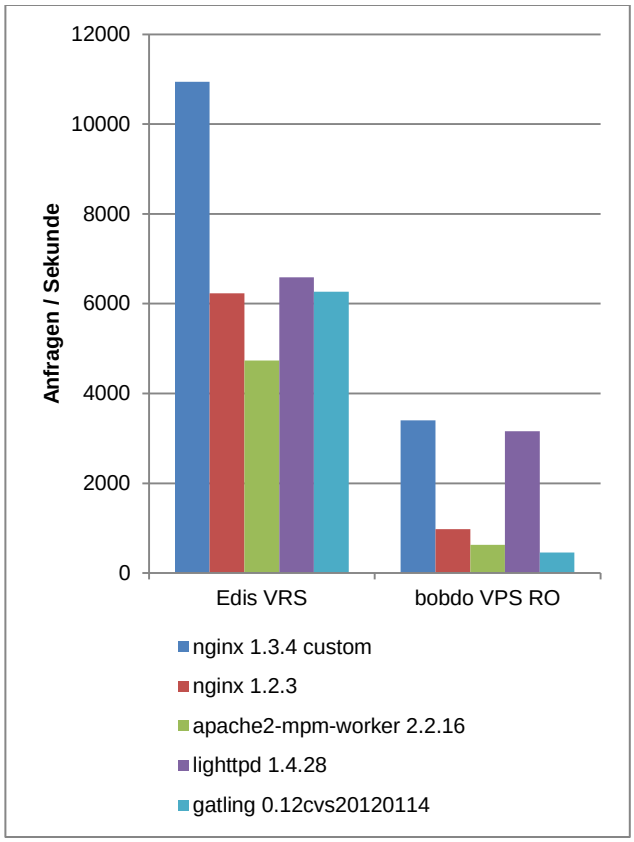
Ausgehend von 10 Millionen Anfragen pro Tag und einer Zunahme um den Faktor 100 (also 1 Milliarde Anfragen pro Tag, was circa der Anzahl an Anfragen pro Monat der deutschen Wikipedia entsprechen würde (Wikimedia Analytics Team 2013)) müssten erst gar keine Messungen durchgeführt werden, um zu wissen, dass der Server diese niemals beantworten könnte. Bei einem Anstieg um den Faktor 10.000 (also 100 Milliarden Anfragen pro Tag, im Vergleich: Facebook verarbeitet mehr als 1 Billion Anfragen pro Monat (Indvik 2011)) würde das Ganze natürlich noch fataler aussehen.

Um mit dieser immensen Anzahl an Anfragen umgehen zu können müssten Server um die den ganzen Globus zur Verfügung stehen, alle Daten müssten redundant zur Verfügung stehen und ein ausgeklügeltes Cachingsystem müsste sicherstellen, dass so wenig Anfragen wie möglich überhaupt erst einen Applikationsserver treffen (um die Last so gering wie möglich zu halten).

Ein entscheidender Faktor beim Verarbeiten so vieler Anfragen ist die Art der Applikation. Facebook hat hier z. B. einen großen Nachteil gegenüber vieler anderer Anwendungen, da sie ständig mit Echtzeitdaten arbeiten müssen und Änderungen sofort global zur Verfügung stehen müssen. Ein sehr interessanter Artikel über die größten technischen Errungenschaften von Facebook wurde unlängst bei golem.de veröffentlicht: www.golem.de/1301/97179

Interessant bei diesen Überlegungen ist, dass selbst für durchschnittlich große Websites ein kleiner VPS eigentlich ausreichend ist. Eine solche Website muss vielleicht 1.000.000 Anfragen pro Tag (das entspricht circa der Anzahl an Anfragen von DerStandard.at (4seohunt.com 2013)) beantworten, das ist extrem viel, aber für den hier vorgestellten Server immer noch machbar.

Was hier bei diesen Benchmarks jedoch wichtig ist, ist die Tatsache, dass es sich lediglich um HTTP-GET-Anfragen handelt. Die Performance von anderen Anfragen wurde bewusst nicht berücksichtigt, der Grund dafür ist denkbar einfach. Über 90 % aller Anfragen die an eine Website gerichtet sind, sind nun mal HTTP-GET-Anfragen. Bei einer realen Applikation müsste jedoch auch darauf geachtet werden, dass der Server die anderen Anfragen (die dann auch eine serverseitige Verarbeitung nach sich ziehen) auch schnell und effizient bearbeiten kann. Genau hier würde bei einem echten Server das Problem entstehen.



Vorwissen und Wünsche

Im Bereich Linux-Optimierung und Optimierung der selbst eingesetzten Software mache ich selbst sehr viel. Auch die Verwendung und Konfiguration von Cache-Software und Proxies ist mir geläufig. Mit Clustering habe ich durch diverse Lehrveranstaltungen während eines Bachelorstudiums bereits Erfahrung.

Themen die mich persönlich interessieren, bzw. wo ich gerne noch mehr lernen würde umfassen im Besonderen Datenbanken. Datenbanken sind extrem komplex und benötigen immer besonderes Monitoring. Gerne würde ich auch einmal ein Datenbankcluster aufbauen und/oder ein verteiltes Datenbanksystem aufbauen.

Das Thema „Messaging“ was von Videla Alvaro vorgestellt werden wird interessiert mich auch sehr. Da durch Messaging interoperabilität zwischen vielen verschiedenen Programmiersprachen und Software umgesetzt werden kann. Dies entspricht meiner Auffassung, dass eine Programmiersprache bzw. ein Programm niemals die Lösung für ein Problem darstellen kann (besonders wenn es um Performance geht).

Literaturverzeichnis

Fussenegger, Richard. *sysctl.d.* 16. August 2012. https://github.com/Fleshgrinder/sysctl.d.

Henderson, Cal. *Scalable Web Architectures: Common Patterns & Approaches.* 21. April 2007. http://www.slideshare.net/techdude/scalable-web-architectures-common-patterns-and-approaches.

Kegel, Dan. *The C10K problem.* 2. September 2006. http://www.kegel.com/c10k.html.

Matsudaira, Kate. *Scalable Web Architecture and Distributed Systems.* 15. Juni 2012. http://www.aosabook.org/en/distsys.html.

The Linux Information Project (LINFO). *Scalable Definition.* 7. März 2006. http://www.linfo.org/scalable.html.