Skalierbarkeit

Wenn ein System aus der Infrastruktur herauswächst, in der es ursprünglich installiert wurde, gibt es die Möglichkeit, durch Skalierung dem wachsenden Datenvolumen und Ansprüchen zu begegnen. Hierbei unterscheidet man zwischen **vertikaler Skalierung** (**scale up**) und **horizontaler Skalierung** (**scale out** oder **Sharding**). Das bedeutet eine Steigerung von Kapazität und Leistung entweder durch das Hinzufügen schnellerer Komponenten (meist CPU oder Speicher) in eine bestehende Servereinheit (scale up) oder die Vergrößerung eines Server-Clusters durch das Hinzufügen zusätzlicher Knoten (scale out).

Vertikale Skalierung

Vertikale Skalierung (**Scale Up**) meint ein Aufrüsten oder Ersetzen der bestehenden Hardware eines Servers durch leistungsfähigere Komponenten, also mehr CPUs und mehr Speicher, wobei der Trend dahingeht, die Daten vorwiegend im Hauptspeicher zu halten. Das hat den Vorteil, dass so langwierige Disk-I/O-Zugriffe entfallen. Ein Upgrade der Hardware kann sich neben dem Performancegewinn auch in der Zuverlässigkeit widerspiegeln, da Hardwareteile nur eine begrenzte Lebensdauer haben und ein neuer Ersatz Ausfällen und dadurch einen etwaigen Datenverlust vorbeugen kann.

Beim Aufrüsten stößt man jedoch sehr bald an die Grenzen, sowohl was die Leistung der Hardwarekomponenten angeht, als auch deren Preis/Bezahlbarkeit.

Relationale Datenbanksysteme sind hervorragend für die vertikale Skalierung geeignet, da sie für den Einsatz auf einem zentralen Server entwickelt wurden, was auch den strengen ACID-Richtlinien entgegenkommt. Dies macht jedoch eine Verteilung der Lasten auf ein Cluster schwer. Sie eignen sich nicht gut für Outscaling, da die Wahrung von Integrität und Konsistenz der Daten immer auch mit Sperren von bearbeiteten Ressourcen und Kommunikations- und Synchronisationsprozessen verbunden ist. Das bedeutet also mehr Aufwand bei der Verteilung auf mehr Maschinen und somit auch erheblicher Performanceverlust.

Bestrebungen danach, relationale Datenbanksysteme zu schaffen, die ebenfalls horizontal skalierbar sind, finden sich in der NewSQL-Bewegung.

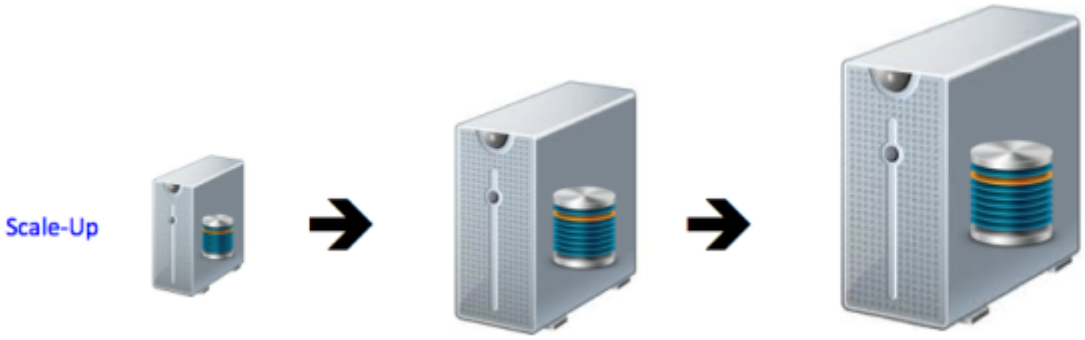
Vorteile

- Geringerer Stromverbrauch im Vergleich zu Scale Out
- Zuverlässigere Hardware
- Geringere Software-Lizenzkosten
- Einfach umzusetzen
- Hilft Clustergröße in Scale-Out-Systemen zu verringern

Nachteile

- Kostspielige Hardware
- Stößt schnell an seine Grenzen

- Hardwareausfall kann Systemstillstand bedeuten



(Grafik-Quelle: Dhandala 2015)

Horizontale Skalierung

Horizontale Skalierung (**Scale Out**) meint das Hinzufügen neuer Server in ein bestehendes Cluster. Die Idee dabei ist, dass ein einzelner Server irgendwann nicht mehr in der Lage ist, das Datenvolumen und die eingehenden Anfragen darauf alleine zu stemmen. Daher wird die Last auf ein Server-Cluster verteilt.

Während man bei der vertikalen Skalierung irgendwann an die Grenzen stößt, was Hardware oder deren Preis angeht, gibt es bei der horizontalen Skalierung hingegen theoretisch keine Begrenzung, da stets neue Rechner hinzugefügt werden können. Dabei müssen diese nicht einmal besonders leistungsstark sein (vgl. Chang et al. 2006: S. 1), was ein kostengünstiges Erweitern ermöglicht. Denkbar wäre auch eine Auslagerung auf Cloud-Services.

Je größer ein solches System wird, desto höher wird auch die Wahrscheinlichkeit, dass ein Hardwaredefekt auftritt, besonders wenn es sich um kostengünstige Low-End-Hardware handelt. Um nun also Datenverlust zu vermeiden und eine stete Verfügbarkeit des Systems zu gewährleisten, werden Daten nicht nur auf einem Knoten gehalten, sondern mehrfach redundant auf verschiedene Knoten verteilt.

Zudem bedeutet das Hinzufügen neuer Knoten auch einen höheren Wartungsaufwand und benötigt komplexere Programme zur Verwaltung und Parallelisierung. Eine Option, dem zu begegnen, ist eine vertikale Skalierung der Knoten eines verteilten Systems, was die Installation zusätzlicher Kapazitäten überflüssig machen kann und so helfen, die Größe eines Clusters zu reduzieren.

Im Gegensatz zu den RDBMS wurden die meisten NoSQL-Systeme von Anfang an für das verteilte Rechnen konzipiert und sind daher bestens für horizontale Skalierung geeignet, können dabei jedoch nicht alle ACID-Eigenschaften wahren und garantieren keinen Transaktionserfolg oder sofortige Aktualisierung der Daten (Eventual Consistency/CAP-Theorem). Viele der großen NoSQL-Systeme sind als Open-Source-Projekte frei verfügbar, was sie für den Einsatz auf vielen Maschinen besonders interessant macht.

Replikation

Wenn ein System durch *Lesezugriffe* zu stark belastet wird, bietet sich eine **Master-Slave-Replikation** als Lösungsansatz für eine Entlastung an. Bei einem solchen Aufbau wird die Datenbank

auf einem Master-Server und einer Reihe von Slave-Servern repliziert. Der Master behandelt die Schreibzugriffe und die Slave-Knoten die Lesezugriffe. Fallen mehr Lesezugriffe an, kann das System durch horizontales Skalieren entlastet werden, indem neue Slave-Knoten hinzugefügt werden. Ein weiterer Vorteil ist, dass die Datenbank erreichbar bleibt, auch wenn ein oder mehrere Slave-Knoten ausfallen. Fällt jedoch der Master aus, ist kein schreibender Zugriff auf die Datenbank mehr möglich, bis er wiederhergestellt oder ersetzt wurde (vgl. Sadalage/Fowler 2012: S. 40). Zwar werden Slaves regelmäßig aktualisiert, jedoch herrscht in der Zwischenzeit eine Inkonsistenz, solange die Updates noch nicht weitergegeben wurden.

Eine weitere Replikationsart ist die **Peer-to-Peer-Replikation**. Hier wurde das Master-Slave-Prinzip aufgehoben, da der Ausfall des Masters unter Umständen zu einer Unerreichbarkeit der Datenbank für Schreibzugriffe führen kann. Fällt in dieser Topologie ein Server aus, bleibt das System trotzdem voll erreichbar, da nun jeder Knoten ist nun ein „Master“ ist und sowohl *Lese-*, als auch *Schreibzugriffe* verarbeiten kann. Dabei kommunizieren die Server bei Schreibzugriffen untereinander um die Daten entsprechend aktuell zu halten. Bei gleichzeitigen Schreibzugriffen auf ein Record kann es jedoch zu Konflikten kommen, wobei unter Umständen ein Update auch verloren gehen kann. Dies lässt sich unter Inkaufnahme erhöhter Netzwerkauslastung durch Koordination unter den Knoten bei Schreibzugriffen vermeiden (Sadalage/Fowler 2012: S. 43).

Sharding

Sharding hilft bei der Entlastung des Systems durch eine Verteilung der Daten in einem Server-Cluster. So können auch Datenvolumen verarbeitet werden, die zu groß für einen einzelnen Server wären. Dabei werden die Daten nicht wahllos zerteilt, sondern idealerweise so, dass Daten, auf die oft zusammen zugegriffen wird, auch auf demselben Server liegen. Tabellen können dabei anhand eines Shard-Keys (ein oder mehrere statische Attribute) in Untertabellen aufgespalten werden (Microsoft o. J.).

Es kann auch sinnvoll sein, Daten, auf die aus einer bestimmten Region zugegriffen wird, auch auf Servern vor Ort oder in der Nähe zu speichern, um so die Antwortzeiten zu verkürzen. Man spricht dabei auch von **Regionalisierung**. Bei Graphdatenbanken kann das manchmal auch einen Anhaltspunkt für eine sinnvolle Schnittstelle bei der Partitionierung sein. So eine Trennung kann manchmal auch gesetzlich verlangt werden oder eine Compliance-Anforderung sein, sodass z.B. Kundendaten aus Europa auch auf europäischen Servern gespeichert werden.

Der Server, auf dem diese Daten liegen, ist dann auch für deren Verwaltung zuständig, ein zentraler Master, der Updates koordiniert und zu Inkonsistenzen führen kann, ist also überflüssig. Das erlaubt eine hohe Verfügbarkeit durch parallele unabhängige Zugriffe und zudem eine hohe Ausfalltoleranz, da bei einem Serverversagen nur ein Teil der Daten davon betroffen ist (durch zusätzliche Replikation der Daten kann zudem ein Datenverlust vermieden werden) und die restliche Datenbank verfügbar bleibt.

Die meisten NoSQL-Systeme wurden für eine solche Verteilung konzipiert, ermöglichen automatisiertes Sharding und verwalten ihre Datensätze ohnehin in Aggregaten, die zusammengehörig sind und auf die in atomarer Weise zugegriffen werden kann und bei Verteilung auch nicht getrennt werden (vgl. Sadalage/Fowler 2012: S. 39). Auf diese Weise sind schnelle Antwortzeiten der Server möglich. Allgemein empfiehlt es sich, auf verteilten Systemen mit kurzen Anfragen zu arbeiten, die möglichst nur einen Knoten benötigen, da ansonsten die Performance durch Kommunikations- oder Synchronisations-Overhead darunter leidet (Stonebraker/Cattell 2011).

Vorteile

- Geringere Anschaffungskosten im Verglich zu Scale Up
- Hardwareausfall kann besser kompensiert werden
- Hohe Verfügbarkeit durch Lastenverteilung und Replikation
- Einfach aufzurüsten
- Theoretisch unendlich skalierbar

Nachteile

- Höhere Stromkosten (Serverbetrieb, Kühlung, etc.)
- Komplexer Verwaltungsaufwand
- Komplexere Software
- Eventuell höhere Software-Lizenzkosten

(Grafik-Quelle: Dhandala 2015)